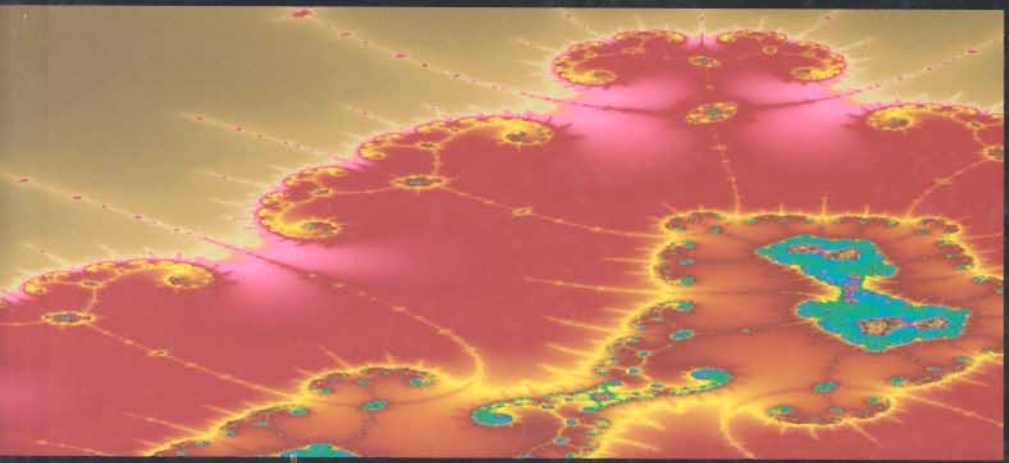
INCLUYE
CD-ROM



PROGRAMACIÓN

JAVA 2
J2SE 1.4

JOHN ZUKOWSKI

SYBEX

ANAYA MULTIMEDIA

004.1
Z1K
J

INFORMATICA
PROGRAMACION
INTERNET
JAVA 2
J2SE 1.4

Todos los nombres propios de programas, sistemas operativos, equipos hardware, etc. que aparecen en este libro son marcas registradas de sus respectivas compañías u organizaciones.

Reservados todos los derechos. El contenido de esta obra esta protegido por la ley, que establece penas de prision y/o multas, además de las correspondientes indemnizaciones por daños y perjuicios, para quienes reprodujeren, plagiaren, distribuyeren o comunicasen públicamente, en todo o en parte, una obra literaria, artistica o cientifica, o su transformación, interpretacibn o ejecucion artistica fijada en cualquier tipo de soporte o comunicada a traves de cualquier medio, sin la preceptiva autorizacibn.

Copyright © 2003 by Anaya Multimedia, S.A.
Original English language edition copyright © 2003 by SYBEX Inc.
All rights reserved including the right of reproduction in whole or in part in any form.
This edition published by arrangement with the original publisher, SYBEX Inc.

Edición española:

© EDICIONES ANAYA MULTIMEDIA (GRUPO ANAYA, S.A.), 2003
Juan Ignacio Luca de Tena, 15. 28027 Madrid
Deposito legal: M-27.751-2003
ISBN: 84-415-1559-X
Printed in Spain
Imprime: Artes Graficas Guemo; S.L.
Febrero, 32. 28022 Madrid

Agradecimientos

Desarrollar una obra es siempre una tarea agradable. Llevarla a buen tkrmino puede convertirse, sin embargo, en un viaje largo y dificil. Despuks de haber pasado por esto ocho veces, uno piensa que va a ser más sencillo.

Como siempre, hay muchas personas que han ayudado a convertir mi horrible manuscrito original en el contenido de cuidada forma que tiene ahora en sus manos. En primer lugar, me gustaría darle las gracias a la gente de Sybex: a la editora de producción Kylie Johnston, la editora de adquisiciones y desarrollo Diane Lowery y a la editora Carol Henry por llegar conmigo hasta el final. A mi editor tkcnico Rob Castaneda de Customware por subsanar todos los problemas tkcnicos de los ejemplos. Kevin Ly y Dan Mummert hicieron un trabajo estupendo con el contenido del CD. Otras contribuciones de gran importancia fueron las del desarrollador Maureen Forys y del equipo de revisores.

No puedo olvidarme de Terence Parr por escribir el prólogo. Ter parece no recordar que si no llego a alquilar ese Jaguar, tendría que haber ido a buscarme al aeropuerto a las tres de la mañana. Además, como Brian Christeson, que tambikn trabajó para jGuru, le habrii contado, el asiento del copiloto sólo podia desplazarse hacia adelante, por lo que Brian tenia que sentarse con las rodillas tociindole casi la barbilla. Y para colmo, como llovió toda la semana, ¡ni siquiera pude bajar la capota!

Algunas personas que no estuvieron directamente involucradas en la creación de este libro merecen tambikn ser mencionadas en esta sección. Enhorabuena a Melissa y Steve Pelton por decidir terminar con su época de libertad y por su inminente "atadura" que les harii muy felices. Gracias a Matt Groening por crear los Simpsons, por todo el entretenimiento que proporcionan y por conseguir apartarme unos momentos de mi ordenador. Por ultimo, me gustaría saludar a mi prima Rachel que decidió no regresar a Boston; mis mejores deseos para ti y tu futuro marido, Todd. Gracias tambikn a Heinz, por crear ketchup de color verde y morado; y apostar por lo diferente.

Y, por supuesto, Gracias a mi esposa, y a mi perro Jaeger, ¡que puede que pronto tenga un hermanito!

John Zukowski

Índice de contenidos

Prólogo **24**

Introducción **28**

Un mapa de carreteras 29
Para aquellos que no sean programadores en C/C++/C# 29
Para los programadores en C/C++/C# 30
Para los programadores en Java 30
Para todo el mundo 30
Estructura y propiedades del libro 30
Normas usadas en este libro 31
Ayuda técnica de Java 31

Parte I. Bases de Java **33**

1. Introducción a Java **34**

Historia de Java 101 36
Examinando la arquitectura Java 37
Simple 37
Orientado a objetos 38
Distribuido 39
Interpretado 39
Robusto 39
Seguro 40

De arquitectura neutral 41
Portátil 41
Alto rendimiento 42
Multitarea 42
Lenguaje dinfimico 42
El punto de vista de Microsoft 43
Resumen 44

2. Comenzar con Java 2 SDK 46

Applets contra aplicaciones 48
Entorno de ejecución de Java 48
Applets y aplicaciones: ¿Cuál es la diferencia? 49
Cómo trabajar con Java 2 SDK 50
Herramientas SDK 51
Descargar e instalar SDK 53
Descargar SDK 53
Instalación de SDK 53
Comprender el CLASSPATH 55
Cómo crear aplicaciones Java 57
La aplicación Hello World 57
Cómo compilar el código fuente de la aplicación 58
Cómo ejecutar la aplicación 59
Documentando la aplicación 61
Integración con código C/C++ 63
Cómo depurar su aplicación 64
Cómo desensamblar sus clases 64
Cómo crear applets Java 65
Applet HelloWorld 65
Cómo compilar el código del applet 67
Cómo crear el cargador HTML 68
Ejecutando el applet 69
Cómo documentar el applet 70
Cómo depurar su applet 70
Resumen 71

3. Cómo trabajar con objetos 72

Introducción a la POO 74
Desde objetos hasta clases 74
Campos y métodos 75
Cómo diseñar clases 77
Abstracción 77
Encapsulación 77
Ocultación de información 78
Generalización 78

Abstracción, segunda toma 80
Asociación 81
Agregación 82
Resumen 8 3

4. Tipos de datos, modificadores y expresiones 86

Comentarios 88
Un ejemplo de comentarios javadoc 89
Palabras clave 92
Variables 93
Tipos de datos 94
Literales 95
Cómo inicializar variables con literales 96
Literales de punto flotante especiales 98
Instrucciones y expresiones 99
Operadores 100
Operadores aritméticos 101
Operadores relacionales 102
Operadores booleanos 104
Operadores bitwise 105
Operadores de cadena 107
Operadores de asignacion 108
Operadores especiales 109
Conversion de datos 109
El operador instanceof 110
El operador new 110
Modificadores 110
Almacenamiento y duración de los modificadores 110
El modificador abstract 111
El modificador static 111
El modificador synchronized 114
El modificador native 114
El modificador volatile 114
El modificador transient 115
El modificador final 115
El modificador Strictfp 115
Modificadores de accesibilidad 116
Resumen 1 1 8

5. Clases de paquetes y acceso a interfaces 120

Casting o conversion de tipos de datos 121
Conversión entre tipos primitivos 122
Conversión entre referencias 123

Uso de this y super con las referencias al constructor y los miembros 124
Para referencias al constructor 124
Para referencias de los miembros 126
El modelo de memoria de objetos de Java 127
Interfaces de Java 130
Definir una interfaz 131
Cláusula implements 131
Trabajo con clases internas e interfaces 133
Clases superiores e interfaces 134
Clases de miembro 135
Clases locales 136
Clases anónimas 136
Los paquetes de Java 138
La declaración import 138
Resolución de los nombres de las clases 139
Paquetes y directorios 140
Declaración package 141
Compilar y ejecutar código empaquetado 141
Convenciones recomendadas 142

6. Arrays y declaraciones de control de flujo 144

Arrays en Java 145
Acceder a los elementos de un array 146
Declarar y crear arrays 147
Declaración 147
Creación e inicialización 147
Uso de la memoria 148
Copiar y clonar arrays 150
Ordenar y buscar arrays 152
Uso de matrices multidimensionales 153
Uso de declaraciones para el control de flujo 154
Declaraciones condicionales 155
Declaración if 155
Declaraciones switch 157
Declaraciones para bucles 159
Declaraciones for 159
Declaraciones while 161
Declaraciones do-while 162
Declaraciones para romper el flujo 163
Declaraciones break 164
Declaraciones continue 164
Declaraciones de etiqueta: 165
Declaraciones return 165
Resumen 166

7. Control de excepciones y aserciones 168

Constructores de control de excepciones de Java 169
Jerarquia de excepciones 170
Un ejemplo de control de excepciones 172
Mktodos Throwable comunes 173
Cómo controlar mktodos que arrojan excepciones 174
Cómo arrojar excepciones por uno mismo 176
Cómo crear clases de excepción personalizadas 177
Encadenar excepciones 177
Lectura de un volcado Stack 178
Trabajar con la cláusula finally 179
Comprobar aseveraciones 181
Uso de la palabra clave assert 182
Activar el control de aseveraciones 183
Activar el control de aseveraciones en tiempo de ejecución 184
Resumen 185

8. Subprocesos y multitarea 186

Bases de la multitarea 187
Crear y ejecutar un thread 188
Usar una subclase de la clase Thread 188
Implementar la interfaz Runnable 189
Mktodos de control de thread 189
El ciclo vital de un subproceso 192
Thread recikn creado 193
Thread ejecutable 193
Thread bloqueado 193
Thread eliminados 194
Grupos de thread 194
Prioridades y planificación 195
Organización y espacio de tiempo 196
Valores de prioridad de thread 196
Obtener información sobre thread y grupos de thread 196
Ejecutar tareas periódicamente 197
Detener la repetición de tareas periódicas 200
Demostración de prioridades y temporización 200
Multitarea avanzada 207
Sincronización de thread 207
Modelo del monitor de Java para la sincronización 210
Diferencias en las tkcnicas de sincronización 213
Comunicaciones entre thread 215
Uso de wait(). notify() y notifyAll() 215
Thread variables locales 222

Thread demonio 225
Agrupar thread 226
Resumen 230

Parte II. Explorando las bibliotecas Java 233

9. Paquetes estándar de Java 234

Bibliotecas de utilidades 236
Paquete java.lang 236
Paquete java.util y java.math 237
Paquete java.io y java.nio 238
Paquete java.text 238
Paquete java.beans 239
Paquetes de seguridad 239
Bibliotecas multimedia 240
Paquete java.awt 240
Paquete javax.swing 240
Paquetes java.awt.print y javax.print 241
Paquete javax.sound 242
Bibliotecas para programas distribuidos 242
Paquete java.net 242
Paquetes javax.xml, org.w3c.dom y org.xml.sax 242
Paquete java.sql y javax.sql 243
Paquetes java.rmi, javax.rmi y org.omg 243
Paquete javax.naming 243
Resumen 244

10. Animación e imagenes 246

Gráficos bisicos 247
Creación de un marco básico 249
Dibujar líneas y rectangulos 252
Dibujar poligonos 253
Dibujar óvalos 255
Dibujar arcos 256
Dibujar texto 257
Control del estado de gráficos 258
Trasladar un sistema de coordenadas 259
Especificar un Area de recorte 260
Color del dibujo 262
Modo de dibujo 264
Fuente actual 265
Trabajo con tamaño de fuentes 266
Bases de la animación 269
Trabajar con imágenes 270

Carga 270
Dibujar 271
ImageObserver 271
Juntiindolo todo 272
Modo de empleo de MediaTracker 274
Realizar animaciones 277
Operaciones de dibujo usando el buffer 280
Modificar la escala de imágenes 282
Recortar imágenes 285
Reproducir audio 286
Trucos para una animación más rápida 288

11. Gráficos e imhgenes 2D 290

Proceso gráfico 292
Un salto a Graphics2D 292
Iniciación a Graphics2D 295
Pariimetros configurables 296
Gradientes de pintura y rellenos con textura 300
Suavizar contornos 303
Dibujar curvas 307
Clases para curvas 307
Curvas GeneralPath 312
Transformaciones 313
Composición 316
Procesar texto 319
Ampliación del soporte para fuentes 319
Disefio avanzado de texto 320
Manipulador de imágenes 324
Formato flexible de imágenes 325
Operaciones avanzadas con imágenes 326
Codificación de imiigenes 329
Conexiones para el dispositivo gráfico 331
Configuración de gráficos 331
Modos de muestra de griificos 333
Dibujo a pantalla completa 334
Resumen 338

12. Administración de un diseiio GUI 340

Posición de Java respecto a GUI 341
Disefio de GUI con administradores de disefio 343
Las clases cliente Container y Component 343
El componente JFrame 344
El componente Container 345
FlowLayout 349

BorderLayout 352
CardLayout 355
JSplitPane 359
GridLayout 361
Administrador y cuadto BoxLayout 363
Alineación de los componentes Box 366
SpringLayout 368
GridBagLayout y GridBagContraints 373
Dimensiones de los componentes 373
Trabajar con GridBagLayout 374
Elementos internos de los administradores de diseño: la interfaz LayoutManager 378
Resumen 381

13. Componentes Swing para GUI 384

La superclase Component 386
Mktodos de control de eventos 393
Métodos para mover componentes y modificar su tamaño 394
Mktodos de consulta de ubicación y geometria 395
Mktodos grificos y de representación 395
Métodos de los administradores de diseño 396
Mktodos para la administración de focos 396
Mktodos de dibujo automitico 397
Mktodos para el cambio y la solicitud de estado 398
Mktodos relacionados con las imágenes 398
Mktodos relacionados con los mktodos de entrada 399
Mktodos de menu desplegable 399
Mktodos para la comunicación de igual a igual 399
Y el resto 400
La superclase Swing de las clases GUI 401
Agregar componentes a la interfaz GUI 406
Agregar botones 406
Mostrar imigenes en los botones 409
Definición de bordes y contornos para los botones 412
AbstractButton 415
Cambiar los iconos de un botón 415
Ubicación de texto e iconos 417
Configuración de los botones nemotécnicos 419
Trabajar con la información sobre herramientas 421
Configuración del administrador de información sobre herramientas 422
Uso de JPanel para agrupar elementos 422
Agregar etiquetas a los elementos 424
Agregar botones de activación 425
Agregar casillas de verificación 428

Aiiadir botones de opción en grupos de botones 429
Agregar listas desplegables 431
Aiiadir paneles deslizantes 433
Agregar listas 434
Suministrar barras de desplazamiento 436
Agregar campos de texto 439
Ocultar campos de contraseñas 440
Ubicación de las áreas de texto 441
Mostrar HTML 443
Rotar campos de entrada 444
Formato de campo de texto 446
Barras de herramientas 447
Trabajar con marcos internos 449
Configuración de tablas 450
Compatibilidad con tecnologias de ayuda a discapacitados 451
Creación de nuevos componentes 452
Componentes del sistema de menu 454
Agregar barras de menus 454
Agregar menus 454
Agregar elementos de menu 455
Agregar menus emergentes 456
Un programa para construir una barra de menus 457
Resumen 458

14. Control de eventos 462

Control de eventos basado en delegación 463
Tipos de eventos y oyentes 467
Eventos de acción 469
Eventos de ajuste 471
Eventos padre 474
Eventos de componente 476
Eventos de contenedor 478
Eventos focales 480
Eventos de hipervínculo 482
Eventos de marcos internos 484
Eventos de elemento 486
Eventos de tecla 489
Eventos de menu 493
Eventos de ratón 495
Eventos de rueda de ratón 498
Eventos de menu desplegable 501
Eventos de ventana 502
Adaptadores 504
Resumen 505

15. Capacidades de Swing avanzadas 508

Manejador de eventos Swing avanzado 509
Administración de listas de Listener 510
Abstraction Action Listening 511
Interacción con el subsistema de focos 516
La clase KeyboardFocusManager 517
Desplazamiento del foco 520
Desplazamiento programable 522
Trabajando con Thread multiples 523
Cuadros de diilogo para usuarios 524
Cuadros de diilogo para mensajes 524
Cuadros de diálogo para mensajes 529
Cuadros de diilogo de confirmación 530
Cuadros de diálogo para la entrada de datos 531
Cuadros de diilogo para opciones 532
Cuadros de selección de colores 533
Cuadros de selección de archivos 536
Arquitectura Modelo/Vista/Controlador 546
Utilización del modelo de botón 547
Objeto delegate de botones 548
Look and Feel modificable 549
Componentes Swing avanzados 550
Cuadros de lista y listas desplegables 550
ListModel/ComboBoxModel 551
ListCellRenderer 554
ListSelectionModel 555
Tablas 559
TableModel 565
TableColumnModel 569
ListSelectionModel 571
TableCellRenderer 573
TableCellEditor 576
Árboles 578
TreeModel 579
TreeNode 581
TreeSelectionModel 584
TreePath 585
TreeCellRenderer 586
Propiedades del cliente 589
Resumen 589

16. Transferencia de datos 592

Objetos transferibles 593
Dataflavor 594

La clase StringSelection 597
Transferencia de datos mediante el portapapeles 597
Transferencia de texto 599
El EditorKit y las acciones de texto 600
Transferencia de imágenes 603
La clase ImageSelection 607
Arrastrar y soltar 609
Resumen 612

17. Utilidades de Java 614

El marco de las Colecciones de Java 2 616
Vectores. Stack y enumeración 616
Trabajando con vectores 616
Pushing Stack 621
Desplazamiento por las enumeraciones 622
Diccionarios. tablas Hash y propiedades 625
Hojeando diccionarios 626
Tablas Hash 626
Examen de propiedades 630
Trabajo con Preferencias 632
BitSet 635
Collections e Iterators 637
Set 641
List y ListIterator 642
ListIterator 645
Map 646
Sincronización y legibilidad 649
Algoritmos y organización 651
Comparable y Comparator 651
Utilidades variadas 654
Biblioteca de matemáticas de Java 654
La clase java.lang.Math 655
El paquete java.math 657
BigInteger 657
BigDecimal 658
Formato del resultado numérico 660
Fechas y calendarios 662
La clase Date 662
La clase Calendar 663
Formato de las fechas de entrada y salida 665
Resumen 667

18. Programación avanzada de Applet 668

Interfaces GUI exclusivas para Applet 669
Ejemplo: Calculadora basada en stack 670

Resumen de los bloques de una calculadora de stack 672
El panel del teclado de la calculadora 673
La pantalla de la calculadora 679
El Stack de la calculadora 682
El subsistema de digitos de siete segmentos 684
Implantación de Applets 691
El periférico de Java 692
Obtención e instalación del periférico de Java 692
Java Web Start 695
Ejecución de las aplicaciones demo 695
Funcionamiento de Web Start 700
Optimización del tiempo de inicio 702
Mantener ejecutables pequeños 702
Minimizar el numero de clases 703
Utilizar archivos JAR 704
Reducir el tiempo de inicio 704
Ofuscar código 705
Resumen 706

19. Streams y programación E/S 710

Métodos de gestión de archivos de Java 712
La clase File de Java 712
Asignación de nombres de archivo 714
Cuestionamiento de atributos de archivos 715
Manipulación de directorios 716
Manipulación de archivos temporales 717
La clase RandomAccessFile de Java 717
Métodos de manipulación de Stream de E/S 723
InputStream y Reader 724
FileInputStream y FileReader 726
ByteArrayInputStream 729
CharArrayReader y StringReader 730
SequenceInputStream 730
PipedInputStream y PipedReader 731
ObjectInputStream 736
InputStreamReader 737
FilterInputStream and FilterReader 737
BufferedInputStream y BufferedReader 737
DataInputStream 739
LineNumberReader 740
PushbackInputStream y PushbackReader 742
InflaterInputStream and Children 742
Tokenizing Input Streams 745
StreamTokenizer 746

Output Streams y Writers 746
ByteArrayOutputStream y CharArrayWriter 748
FilterOutputStream y FilterWriter 750
BufferedOutputStream y BufferedWriter 750
DataOutputStream 751
ObjectOutputStream 752
PrintStream y PrintWriter 752
DeflaterOutputStream y variables 754
Persistencia de objetos y Serialización 755
Criterio para la serialización 756
La interfaz Serializable 756
La interfaz Externalizable 757
Creación de streams de salida para serializar 757
Uso de streams de entrada para "deserialización" 759
Serialización de excepciones 761
Lectura y escritura de un stream de objeto 761
Un ejemplo de serialización 762
transient y ObjectStreamField 765
Código XML y decodificación 766
El nuevo paquete E/S 768
Búferes NIO, canales y grupos de caracteres 769
Expresiones normales 772
Registro 775
Resumen 777

20. Trabajos de impresión 778

Historia del soporte de impresión 779
Resumen de las funciones de impresión de Java 1.4 782
Ejemplo sencillo de impresión 787
Cómo mostrar un cuadro de diálogo 788
Utilización de atributos 791
Utilización de StreamPrintService 795
Impresión de pantallas 797
Resumen 799

21. JavaBeans 800

Modelo de los componentes JavaBeans 802
Los Bean de Java 802
Arquitectura de los Beans 803
El modelo de eventos JavaBean 804
Fuentes y destinatarios de eventos 805
Propiedades de los Beans 807
Propiedades simples 807
Propiedades indexadas 807

Propiedades dependientes 808
Propiedades restrictivas 808
Los métodos Bean 809
Grabación de Beans 810
Revisión y personalización de Beans 810
El proceso de exploración de Beans 810
La interfaz BeanInfo 811
Persistencia del Bean 812
El taller de programación de Beans (BDK) 812
Instalación del BDK 813
Uso del BDK BeanBox 814
Los Beans de muestra 815
Conexión de Beans 817
Grabación y restauración de las aplicaciones Bean 819
Cómo crear un nuevo Bean 820
El Bean TrafficLight 821
El BeanInfo de TrafficLight 825
Compilación del Bean TrafficLight y el BeanInfo 826
Cómo incluir el nuevo Bean en el BeanBox 827
Prueba del nuevo Bean 827
Resumen 828

22. Programación en Red 830

Protocolos de Java: TCP/IP 831
IP: El corazón de las comunicaciones de datos en Internet 832
TCP: Para garantizar la recepción 833
Gestión de direcciones de Internet 835
Busqueda de direcciones textuales 837
Obtención de direcciones numéricas 838
Comunicación con sistemas remotos 839
Comunicación a bajo nivel mediante UDP 839
Conexión a servidores mediante TCP 842
Conexión con un servidor de correo SMTP 845
Un ejemplo de cómo enviar un correo electrónico 846
Conexión a un servidor HTTP de Web 849
Un ejemplo: Descarga de una pagina Web 850
Cómo obtener otros recursos de la Web 852
Operaciones en URL 853
Control de vinculos HTTP 856
Uso de la identificación por contraseñas 859
Lectura de canales de socket 862
Lecturas Nonblocking 864
Programación de sistemas de servidor 869
Un programa simple de servidor 871

El cliente para el servidor de muestra 873
Servidores Nonblocking 875
Las factory 877

23. Bibliotecas de extensiones de Java 880

La edición estándar de Java (J2SE) 881
XML 882
Transformación de documentos XML mediante XSLT 882
Análisis mediante SAX y DOM 884
Seguridad de Java 886
Conectividad de las bases de datos de Java (JDBC) 889
Los componentes API 890
Drivers JDBC 891
Nivel de la aplicación 892
Invocación de método remoto (RMI). CORBA. y Java IDL 894
RMI 894
Un ejemplo de RMI 896
Paso 1: Definir las interfaces para las clases remotas 897
Paso 2: Crear y compilar las clases de implementación 898
Paso 3: Crear clases stub y skeleton 901
Paso 4: Crear y compilar la aplicación del servidor 901
Paso 5: Comenzar el registro RMI y la aplicación del servidor 904
Paso 6: Crear y compilar el programa cliente 904
Paso 7: Comprobar el cliente 906
La interfaz nativa de Java (JNI) 907
Edición Enterprise de Java 2 907
Servlets 907
La interfaz API de Servlet 909
Ejemplo de un Servlet sencillo 910
Páginas de JavaServer (JSP) 916
JavaMail 918
La interfaz de directorios e identificación de Java (JNDI) 921
Servicio de mensajes de Java (JMS) 921
JavaBeans Enterprise (EJB) 923
Comunicaciones de Java (COMM) 924
Resumen 927

Parte III. Apendices 929

Apéndice A. Sintaxis del lenguaje Java 930

Indicadores de Applet 947
Resumen de los componentes de Swing 949
Asistencia para las herramientas de líneas de comandos 956

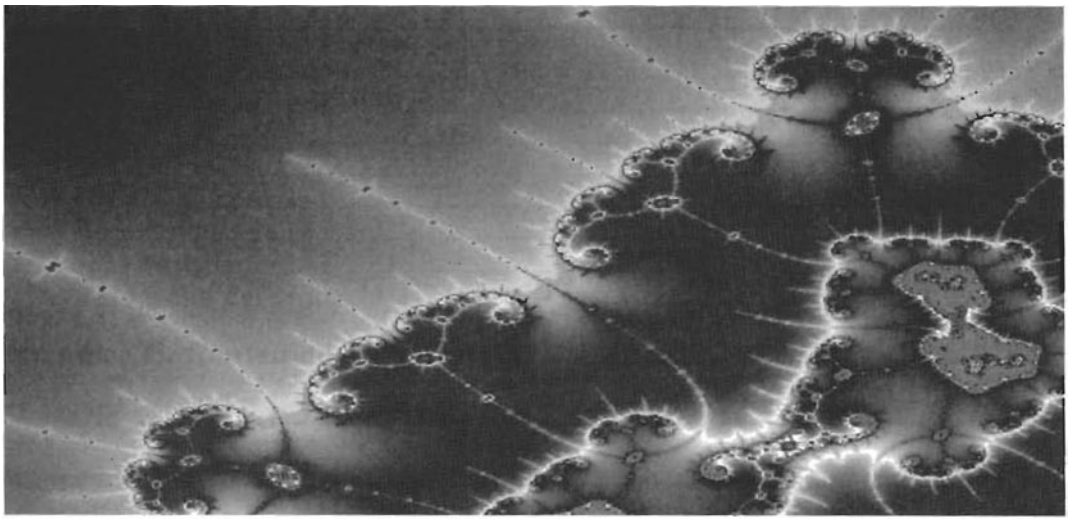
Apéndice B. Recursos en la Red 962

Cómo encontrar respuestas 963
Preguntas mas frecuentes 964
Foros 964
Listas de correo 965
Grupos de noticias 965
Recursos generales 967
Grupos de usuarios 968
Ayuda tecnica del producto 968

Apéndice C. Contenido del CD-ROM 970

Glosario 974

Indice alfabético 995



Prólogo

JAVA se ha consolidado firmemente como el lenguaje de programación más utilizado en la actualidad por las aplicaciones network-aware de detección de redes y ha demostrado ser un lenguaje muy efectivo en programación general. Su gran popularidad se debe especialmente a que proporciona un ambiente de programación seguro, transferible y de gran expresividad que además soporta la distribución de software de una manera invisible y sin interrupciones a través de la Red.

Durante los ultimos 20 aiios, me he encontrado con gran variedad de problemas relacionados con el desarrollo de programas que Java elimina o mejora drásticamente. Por ejemplo, como diseiiador del tan utilizado compilador intérprete ANTLR, me enfrente a serios problemas de portabilidad con C/C++, en cuanto a variaciones en las bibliotecas y el tamaño de un integer, o variable de tipo entero. En muchas ocasiones, tuve que rehacer de nuevo todo el trabajo por no disponer de una biblioteca o infraestructura con la información necesaria. Perdi mucho tiempo creando y limpiando código para comunicarme con equipos remotos y en localizar fallos de memoria en lenguajes que no eran *garbage-collected* y, que, por lo tanto, no liberaban memoria de manera automática. En una ocasión en la que necesitaba una interfaz de limpieza remota, ¡tuve que limpiar un programa mediante un LED (diodo emisor de luz) y un osciloscopio!

La gran portabilidad de Java es legendaria entre los desarrolladores de aplicaciones server-side, que corren en el servidor. Como el principal desarrollador de la aplicación de la página jGuru.com, tuve que preparar el programa para Linux, PC y Mac OS X en las fases de desarrollo, prueba y utilización. Estoy ya tan acostumbrado a la portabilidad de Java que nunca me cuestiono si quiera la posibilidad de

ejecutar la aplicación en cualquier sistema operativo que desee. Y no sólo es el código Java server-side el que se puede ejecutar en cualquier sistema; he utilizado además un IDE (entorno de desarrollo integral) basado en Java en las tres plataformas sin ningun problema. Increible, ¿verdad?

Los programadores resultan mucho más productivos en Java que en cualquier otro lenguaje. El simple hecho de no tener que escribir código para localizar y reasignar memoria es ya una gran ventaja. Java dispone de un simple pero efectivo modelo integrado de tareas, que facilita enormemente la construcción de servidores, servicios centralizados en red y de interfaces GUI. La envergadura y profundidad de las bibliotecas estándares de Java son realmente impresionantes. Es bastante dificil obtener resultados satisfactorios en C cuando la biblioteca estándar contiene unicamente el comando `printf()`.

Java se diseñó basándose en la idea de que el código se transmite a través de una red hasta llegar a equipos de todo tipo y procedente de fuentes que pueden ser fiables o no. Por lo tanto, Java se construyó en base a una infraestructura para identificar y validar código, que comprueba la perfecta ejecución de dichos códigos y que incluso le protege de los errores que usted mismo pueda cometer. En C/C++, por ejemplo, puede sobrescribir buferes que anulan otras variables y que causan comportamientos extraños. Este exceso de buferes, además de ser muy dificil de localizar, es el culpable de la mayoría de las brechas en la seguridad de los programas. En Java, todos los códigos se ejecutan en una zona de pruebas con lo que se evita que el código malintencionado pueda ejecutar métodos privilegiados, saturando los buferes o cualquier otra cosa que pudiera poner en peligro la integridad del sistema.

Java, además, le permite enviar datos a través de Internet, asi como el codigo necesario para interpretar dicha información. Antes de la implementación de Java en 1995, las páginas web sólo podian enviar información al buscador de su equipo. Sin embargo, en la actualidad, un servidor es capaz de enviar un applet de Java sin interrupciones y sin señal visible al buscador, desde donde se empieza a ejecutar como cualquier otra aplicación de cliente, pero con limitaciones de seguridad que se activan automáticamente. La capacidad de instalar de manera automática la ultima actualización de un programa en el ordenador de un cliente reduce enormemente el coste adicional de enviar y actualizar dicho programa. La gran parte de las grandes compaiiias emplean multitud de administradores de sistemas para actualizar los conversores internos de tecnologia más antigua. En el caso de los ordenadores personales, la gran mayoría se encuentran obsoletos ya que las actualizaciones implican un esfuerzo y un coste muy elevados.

El mercado de shareware de Java sigue su expansión, ayudado por los numerosos manuales, grupos de noticias, listas de correo y páginas web, tales como Java Developer Connection de Sun, developerWorks de IBM, y <shameless-plug>jGuru.com</shameless-plug>. Y lo que es más importante, recuerde que para conseguir la fuente de todas las bibliotecas, el código en si constituye la documentación.

En lo que respecta a John Zukowski, es la persona que, sin duda alguna, mejor conoce el mundo de Java, incluyendo sus bibliotecas. Es también la persona que

más Coca Cola consume, y el que está casado con la mujer más amable. En 1995, jGuru (*MageLang* Institute, por ese entonces) rastreó todas las listas de correo, buscando un gurú en Java. John escribia sistemáticamente unos comentarios de gran alcance y, además, su nivel de actividad indicaba que no dormia mucho. Es además una persona con gran sentido del humor. Recuerdo el dia que apareció en un retiro de la compaiiia conduciendo un impecable Jaguar alquilado. ¡Hasta la fecha, sigue asegurando que es el único que tenian en la agencia de alquiler!

John ha estado trabajando con Java toda su vida y sabe exactamente cómo debe enseiiarse. Ha mejorado sus ejemplos durante mucho tiempo hasta dar con los que muestran exactamente lo que usted necesita saber, evitándole lo que no tiene importancia. Las instrucciones de John ofrecidas en jGuru han sido siempre las mas populares entre nuestros estudiantes y lectores.

En cuanto al CD que se incluye en este libro, contiene todo lo que necesita para empezar a construir sus programas, desde algo muy sencillo hasta servidores de Internet. Asi que, ¡no lo dude! ¡Este manual será una buena inversión para su futuro!

Terence Parr
Cofundador y Jefe de investigación de jGuru.com

Introducción

Bienvenido a Programación en Java 2, J2SE 1.4, el libro que le proporcionará una completa cobertura sobre la ultima versión de Java. Aqui encontrará temas sencillos, medios y avanzados con los que podrá convertirse en un programador en Java.

Un mapa de carreteras

¿Para quién es esta obra? Aunque el diseño del libro tiene una secuencia lógica, la verdad es que habrá pocos lectores que se lean el libro de portada a contraportada. Dependiendo de la idea que tenga en mente, las siguientes secciones le mostraran cómo puede aprovechar el contenido de esta obra.

Para aquellos que no sean programadores en C/C++/C#

Si es la primera vez que se ve las caras con Java y con el estilo de programación de C/C++/C#, es muy probable que no tenga más remedio que leer todo el libro. Algunos conceptos se parecerán a otros lenguajes de programación, pero para comprender cómo funciona Java, tendrá que recorrer todos los ejemplos del libro. Le aconsejamos que antes de pasar de una sección a otra, se tome un tiempo de descanso para asegurarse de que ha asimilado correctamente todo lo leído.

Para los programadores en C/C++/C#

Debido a las similitudes que hay entre C/C++/C# y Java, los programadores de C/C++/C# se podrán saltar varios capitulos y centrarse únicamente en los especificos de Java. Los programadores en C pueden saltarse la mayor parte de los capitulos 4 y 6; los programadores en C++ pueden saltarse también el capitulo 3, la sección sobre clases del capitulo 5 y, si tienen experiencia en control de excepciones, el capitulo 7.

Aunque nunca viene de mal repasar estos capitulos, especialmente la utilidad de declaración del capitulo 7, no requieren, sin embargo, la misma atención que el material nuevo del manual.

Para los programadores en Java

Si ya cuenta con cierta experiencia en el mundo de la programación en Java y ha decidido que ya va siendo hora de comprarse un libro en condiciones que le enseiie todo lo que todavia no sabe, conviene que repase los primeros capitulos de la obra para refrescar conceptos y que luego se centre en otros como la multitarea o el control de excepciones.

También seria conveniente que revisara temas como las clases de Java en el capitulo 5, las declaraciones en el 7, asi como las tareas con temporizador en el capitulo 8.

Para todo el mundo

Consulte la tabla de contenidos para ver qué temas son los que más le interesan. En cada capitulo encontrara una serie de ejemplos gracias a los cuales se verán todos los conceptos estudiados. Encontrará también el código dentro del CD-ROM que acompafia a la publicación, así que no tendrá que escribir nada.

Estructura y propiedades del libro

La finalidad de esta obra es muy sencilla: tratar de aplicar las mejoras de Java tan pronto como sea posible. Este libro tiene gran cantidad de información. Use el índice para localizar la que necesite. Aqui encontrará una breve descripción del contenido de las distintas partes de este libro.

Parte I. Bases de Java: En la primera parte del libro le presentamos Java: su historia, el lenguaje y conceptos sobre la programación.

Parte II. Explorando las bibliotecas Java: En la siguiente parte del libro examinaremos las bibliotecas (o librerias, como también se las conoce) de Java.

Normas usadas en este libro

A lo largo del libro encontrard unos recthngulos sombreados que resaltan la información especial o importante, estos son los siguientes:

Advertencia

Indica un procedimiento que, en teoría, podría causar dificultades o incluso la perdida de datos; preste especial atencion a estos elementos para evitar los errores de programacion mas comunes y los que no lo son tanto.

Nota

Resalta la informacion interesante o adicional y suele contener pequeños trozos extra de informacion tecnica sobre un tema.

Truco

Llaman la atencion sobre hábiles sugerencias, pistas recomendables y consejos utiles.

Ademhs en este libro se usan las siguientes convenciones tipogrdficas:

- Los códigos de ejemplo aparecen en un tipo de letra `Courier`.
- Las opciones de menus se indican en orden jerárquico, con cada instrucción de men6 separada por el signo "mayor que" y en un tipo de letra Arial. Por ejemplo, Archivo>Abrir quiere decir hacer clic en el comando Archivo en la barra de men6 y luego seleccionar Abrir.

El codigo del programa se rige por los convenios estdndar para la utilización de mayúsculas que se sigue en la API de Java. Por ejemplo, en los nombres de las clases, la inicial de cada palabra aparecerá en mayúsculas, mientras que en el nombre de las funciones únicamente llevará mayúsculas la inicial de la primera palabra.

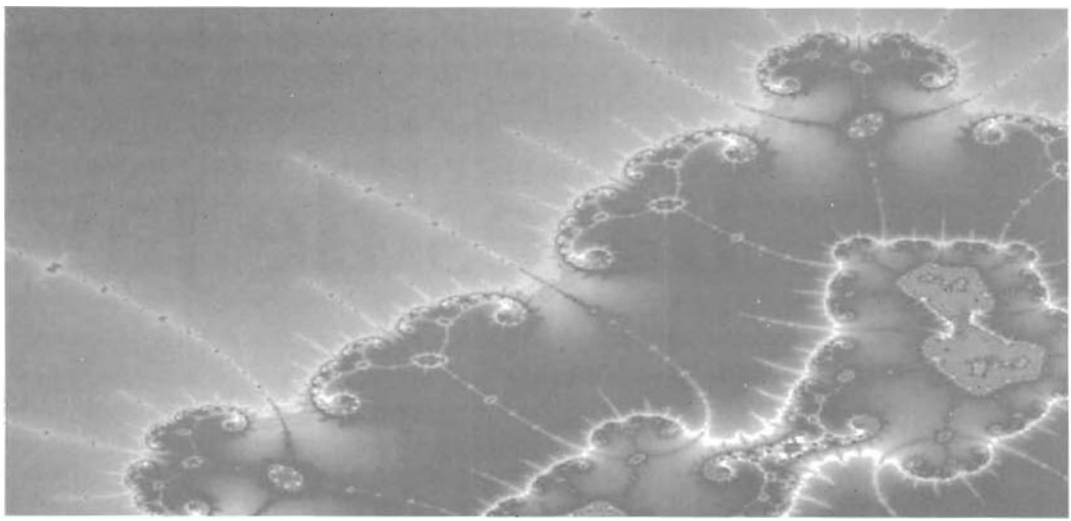
Además encontrará tablas donde se detallan los miembros de las diferentes clases. Si el nombre aparece subrayado significa que es estático, mientras que si aparece en cursiva, el miembro estd protegido.

Ayuda técnica de Java

En el apéndice B encontrarh un listado de las preguntas más frecuentes, foros, listas de correo, grupos de noticias, y grupos de usuarios.

Parte I

Bases de Java



troducci

a Java

Érase hace mucho tiempo una isla en Indonesia conocida por el nombre de Java. La gente de la isla vivió en paz, preocupiindose de sus negocios y sus rutinas diarias durante miles de aiios. Un dia, a principios de los aiios 90, una compaiiia llamada Sun Microsystems creó un nuevo lenguaje de programación llamado Oak. Dado que Oak sólo se usaba de forma interna en Sun, el nombre no importaba demasiado y la gente de Java siguió con su feliz existencia, sin conocer lo que les esperaba.

Unos aiios despuks, Sun estaba preparada para convertir Oak en un autkntico producto. Desafortunadamente, el nombre de Oak no pas6 la prueba de marca. Otros nombres como Silk, Ruby y WRL (*WebRunner* Language), tampoco alcanzaron el nivel exigido, pero, finalmente, el nombre de Java triunfó y pas6 la revisión legal. (Por lo visto, tener el mismo nombre para una isla y un lenguaje de programación no es importante. ¿Qué se puede esperar de una compaiiia con nombre de estrella?) Asi pues, el 23 de mayo de 1995, se present6 Java en la conferencia anual SunWorld de Sun, como la 12ª de 20 presentaciones diferentes. Poco podian imaginarse los indonesios lo que iba a suceder en su pequeiia isla.

Java, la plataforma, comenzó a evolucionar y Microsoft pens6 que era lo bastante importante como para incorporarlo a su navegador Internet Explorer. En la isla de Java, los volcanes empezaban a inquietarse. Entonces, en 1997, Sun demand6 a Microsoft por algunas incompatibilidades con Java y el Krakatoa entró en erupción. (Por supuesto, Java tiene casi 10 grandes erupciones volciinicas cada aiio; podia haber sido una coincidencia.) A medida que el juicio avanzaba, la vida en la isla empeoraba. La inquietud económica de Yakarta hizo que se solicitara la dimisión del presidente Suharto y Timor Oriental exigió la independencia política. En el

frente Sun Java casi se producen sangrientas batallas, con un acuerdo eventual con Microsoft de 20 millones de dólares en 2001. ¿Coincidencia? Pregunte al presidente Wahid qué opina.

Historia de Java 101

De acuerdo, ya esta bien de la compleja historia de la isla de Java y de Java la plataforma. Y si, eso es Java, la plataforma. Lo que comenzó en 1995 como otro lenguaje de programación ahora es conocido formalmente como una plataforma. Desde que comenzó como un lenguaje de programación compatible con Internet y no asociado a una plataforma, Java ha evolucionado hasta convertirse en un medio de creación de programas para casi cualquier cosa. Regresemos al principio y veamos como sucedió.

En enero de 1991 comenzó un proyecto con el nombre de Green. El propósito del proyecto Green era descubrir un modo de controlar consolas de televisión, esas cajas para acceder a la televisión por cable. Como el hardware de los dispositivos para el consumidor siempre estaba cambiando, James Gosling, el "padre" de Java, pensó que C++ no era lo más apropiado para el trabajo. Como resultado, se creó el lenguaje llamado Oak; era menos susceptible a los errores y no bloqueaba todo el sistema. Se ejecutaba inicialmente en un dispositivo de tecnologia Hammer llamado *7 (Star7). Haciendo cabriolas por toda la pantalla táctil estaba ese pequeño personaje digital llamado Duke, un asistente que podia aparecer y realizar varias tareas. (Más tarde se convertiria en la mascota de Java.)

Nota

Si nunca ha visto a Duke, se parece a una mezcla entre un comunicador de Star Trek y un diente dado la vuelta con un sombrero de bruja, una gran nariz roja y brazos como los de los ***Screen Beans.*** (Los ***Screen Beans*** son esos personajes graficos que se suelen ver moviendose por toda la pantalla en las presentaciones PowerPoint.)

Oak se us6 para solicitar un proyecto sobre un sistema operativo para consolas de televisión. La solicitud no prosper6 y Sun rechazó una oferta de compra de Trip Hawkins (entonces el jefe oficial ejecutivo de Gamemaker/3DO), de modo que Sun tenia este desecho y no sabia que hacer con él. Por suerte, la revolución de Internet habia comenzado con el lanzamiento de Mosaic, el predecesor del navegador Web Netscape Navigator. En 1994 se le busca una nueva utilidad a Oak para la Web y se crea el navegador llamado WebRunner (anteriormente renombrado a HotJava) para exhibir la reciente tecnologia de lo que terminaria convirtiéndose en Java.

Para entonces era el momento de hacer público Java. En la ahora infame conferencia SunWorld, John Gage de Sun y Marc Andreesen de Netscape presentaron

Java al mundo. Netscape se comprometio a incorporar Java en la siguiente versión de su navegador y al final del aiio, compañías como IBM, Oracle, Borland, Adobe, Macromedia, Lotus, Spyglass e Intuit habian incorporado Java a sus productos. Incluso Microsoft autorizó Java, comprometiéndose a incorporar Java a sus productos, sistemas operativos y herramientas de desarrollo.

Esta versión inicial de Java era tan pequeiia que cabia en un disquete. El interprete del nucleo tenia casi 100 Kb. La biblioteca matemática sumaba otros 20 Kb. El código para la integración con bibliotecas C eran otros 50 Kb. La mayoría de las bibliotecas de clase cabian en 375 Kb, dejando sólo las bibliotecas de gráficos especificas de la plataforma que variaban en tamaiio. En conjunto, Java solamente tenia alrededor de un megabyte y podia ejecutarse en un equipo con 1Mb de ROM y 1Mb de RAM. (Aunque *7 tenia 4 Mb.)

Saltamos seis aiios hacia adelante y Java ha evolucionado considerablemente. Ya no está restringido a ejecutarse en el navegador, encontrará programas de Java en cualquier parte, desde servidores Web hasta joyeria con luces parpadeantes o tostadoras inteligentes e incluso en las consolas de television para las que fue diseñado en un principio. Además, las bibliotecas de núcleo han aumentado considerablemente y en la ultima versión. (Java 2 Standard Edition, version 1.4) sólo el entorno de ejecución ocupa entre 30 y 40 Mb. Por supuesto, Java ahora puede hacer mas cosas y esa es la razón por la que usted tiene Java Platform, que es la combinación del lenguaje de programación, las bibliotecas estándar y el entorno de ejecución. Y, en este libro, estudiaremos la plataforma.

Examinando la arquitectura Java

De acuerdo con el libro blanco de Java (disponible en http://java.sun.com/people/jag/OriginalJavaWhitepaper.pdf), los objetivos del diseiio de Java eran ser "un lenguaje sencillo, orientado a objetos, distribuido, interpretado, robusto, seguro, de arquitectura neutral, portátil, de gran rendimiento, multitarea y dinámico". Al examinar lo que representan estos cliches, podemos ver lo que Java estaba intentado probar entonces y que todavia puede aplicarse.

Simple

En la primera conferencia C++, Bill Joy, cofundador y ahora cientifico jefe y jefe oficial ejecutivo de Sun, dijo que "C++ es demasiado complicado, lo que yo quiero es C++ ++ - -". En Java, eso es lo que 61 y todo el mundo obtiene. Java incorpora nuevas tareas como un recolector de elementos no utilizados automático (el ++) y elimina aspectos de C++ confusos y muy poco utilizados como la sobrecarga de operadores (el - -).

Otro aspecto de la simplicidad de Java es que nada en Java es realmente nuevo. Si observa el conjunto de funciones de Java junto a la historia de la informbtica, descubrirá que todo procede de algun otro lugar.

- Las clases proceden de C++ y Smalltalk, aunque en Java están limitadas a una sola herencia de implementación.
- Las interfaces proceden de Objective-C y proporcionan a Java herencia de interfaces multiple.
- Los paquetes creados en Modula agregaron a Java el espacio de nombre jerdrquico y las unidades de desarrollo lógicas.
- La concurrencia estaba en Mesa y en Java trajo la compatibilidad con multitarea.
- El control de excepciones procedia de Modula-3 y agregó a Java mdtodos que declaraban lo que producian.
- Los vinculos dinámicos y la recogida automática de elementos no utilizados procedia de Lisp, lo que le concedió a Java la capacidad para cargar clases adicionales a medida que se necesitan y liberar memoria cuando no es necesaria.

Y mientras se agregan nuevas funciones a Java, también procederdn de prácticas usadas y probadas. Por ejemplo, en la versión 1.4 todavia falta la compatibilidad con tipos de parámetros, también conocidos como plantillas o genéricos. Cuando se añada, esta función agregará compatibilidad con colecciones de tipos seguros, una función que lleva tiempo siendo solicitada. Sin embargo, los que mandan no quieren incorporar todos los elementos; están intentando asegurarse de que todo lo que se agrega a Java se realice correctamente. Seguramente encuentre compatibilidad con gendricos en Java 1.5, suponiendo que todo estd decidido para entonces.

Orientado a objetos

La gente dice que Java está orientado a objetos, pero ¿qué significa eso exactamente? Esencialmente, la programación orientada a objetos (OOP) es un modo de desarrollar software describiendo problemas mediante el uso de elementos u objetos desde el espacio del problema y no mediante un conjunto de pasos secuenciales que se ejecutarán en el ordenador. Un buen diseño conlleva a componentes reutilizables, extensibles y sostenibles. Estos componentes son lo bastante flexibles como para controlar los cambios que se puedan producir en el entorno a lo largo del tiempo, dado que el trabajo principal de estos objetos es precisamente intercambiar mensajes entre si.

Esencialmente, los API Core Java son una colección de estos componentes, previamente creados, llamados bibliotecas de clase. En lugar de crearlo todo de nuevo con Java, se comienza con las bibliotecas. Estas bibliotecas estándar han evolucionado a lo largo del tiempo, obteniendo más componentes previamente creados con cada nueva edición de Java.

Para obtener una visión más profunda de la programación orientada a objetos, vea el capitulo 3.

Distribuido

Desde su comienzo en 1982, la vision de Sun ha sido "la red es el ordenador" con Java querian un lenguaje de programación que reconociera la red y que admitiera el acceso a objetos distribuidos tan ficilmente como a los objetos locales. Este objetivo, aunque presente desde el principio, ha cambiado por lo que respecta a las posibilidades de Java.

Inicialmente, todo lo que Java podia hacer era acceder a objetos distribuidos mediante protocolos estándar basados en TCP/IP como HTTP. Con la última versión de Java Platform, se pueden invocar metodos en un equipo remoto tan fácil e invisiblemente como podría hacerlo en su mismo espacio de ejecución, mediante protocolos comunes tales como CORBA (Arquitectura intermediaria para solicitar objetos comunes) y RMI (Invocación de métodos remotos), además de los recién incorporados servicios Web. Para cada protocolo, el sistema se encarga de toda la conversión y transporte automáticamente.

Interpretado

Los programas de Java son interpretados. En lugar de ser compilado en ejecutables nativos, el código de Java es traducido en códigos de bytes no asociados a una plataforma.

Estos códigos de bytes se pueden transferir a cualquier plataforma que tenga Java Runtime Environment (JRE), que consiste en una Máquina Virtual de Java (JVM) y de este modo pueden ejecutarse sin volver a compilarlos o revincularlos. Este proceso puede sugerir que Java es intrinsecamente lento, pero, como se explicará en la siguiente sección "Rendimiento", esto no tiene por qué ser cierto. Los códigos neutrales de bytes en realidad contienen información adicional que puede usarse para optimizar la ejecución en tiempo de ejecución, según decisiones que no pueden realizarse en tiempo de compilación.

Robusto

La robustez es la medida de la fiabilidad de un programa. Java contiene varias funciones integradas que mejoran la fiabilidad de un programa:

- Java es un lenguaje muy basado en tipos: El compilador y el cargador de clases aseguran la corrección de todas las llamadas de método, lo que evita las diferencias implicitas entre tipos y las incompatibilidades entre versiones.
- Java no tiene punteros: No se puede hacer referencia a un puntero en la memoria, corrompiendo accidentalmente la memoria o pasarse el final de una pila. En Java hay referencias en lugar de punteros y no se puede mover su referencia para manipular directamente el espacio en memoria. (La excep-

ción es la memoria de video, a la que se puede controlar mas directamente desde Java 1.4.)

- Java realiza automaticamente la recolección de elementos no utilizados: Los programadores no pueden olvidarse accidentalmente de liberar memoria y no tienen que preocuparse por decidir las partes de la memoria que deben liberarse.
- Java fomenta el uso de interfaces en lugar de clases: Cómo aprenderá en el capitulo 5, las interfaces definen un conjunto de comportamientos y las clases implementan estos comportamientos. En lugar de pasar clases, se pasan interfaces, ocultando asi las implementaciones. Si la implementación cambia, como suele ocurrir, mientras la nueva clase implemente la antigua interfaz, todo lo demás funcionará perfectamente.

Seguro

Java tiene muchas funciones de seguridad integradas, y añade incluso alguna más con la versión 1.4. Estas funciones trabajan en diferentes apartados. Muchas de las cosas que hacen que Java sea robusto también garantizan la seguridad del programa que se esta ejecutando (por ejemplo, al no tener punteros de memoria, no se corrompe la memoria). En el modelo de ejecución hay integradas otras funciones de seguridad: el verificador de código de bit, el cargador de clases y el gestor de seguridad. Esto garantiza la seguridad del código que se esta ejecutando y evita que el código no seguro realice operaciones seguras, como leer el disco duro. El ultimo aspecto de la seguridad tiene que ver con la autenticación, autorización y encriptación para proteger la privacidad y asegurar la integridad de los datos.

El verificador de código de bytes proporciona el primer nivel de defensas en el frente de seguridad. Basicamente, dice "no confies en el compilador". Antes de cargar una clase en el entorno de ejecución, su código de bytes debe ser verificado. Si alguien usase un editor de niveles de byte en el archivo y modificase los bytes para acceder a algo a lo que no deberia poder acceder, el verificador lo detectará y evitará la carga de esa clase y la ejecución del programa. De hecho, los programas que se ejecutan en el JRE, ni siquiera tienen por qué estar escritos en el lenguaje de programación Java.

En la categoria de seguridad, a continuación está el cargador de clases. Como su nombre indica, el cargador de clases es el responsable de cargar las clases. En el frente de seguridad, el cargador de clases es el responsable de cargar las clases y mantenerlas apartadas de miradas poco fiables. En primer lugar, el cargador de clases mantiene las clases locales y las clases cargadas de la red en diferentes espacios de nombres, lo que asegura que las clases cargadas desde la red no pueden reemplazar las clases de sistema local. En segundo lugar, las clases cargadas desde diferentes posiciones de la red también se mantienen separadas. Esto garantiza que una clase que no es de sistema nunca se confundira con una clase de sistema; o que nunca acogerá una clase cargada desde otro servidor Web.

El tercer componente de seguridad del entorno de ejecucion es el administrador de seguridad, que refuerza las normas de seguridad para el entorno de ejecucion. Todas las tareas que requieren un permiso de seguridad para realizar alguna acción pasan por el administrador de seguridad. Si la tarea no puede obtener el permiso del administrador de seguridad, la tarea no se ejecuta. Esto es especialmente importante para el codigo que se carga a través de Internet desde páginas Web que no son dignas de confianza. Este código que se ejecuta en el navegador recibe el nombre de applet. Se ejecutan en una zona del navegador apodada "la caja de arena" (una zona segura en la que se ejecuta, pero sin acceso al entorno del cliente).

Desde Java 1.1 y hasta Java 1.4 inclusive, se han ido agregando más funciones de seguridad al conjunto Core Java API. Se permite a los programadores firmar las clases Java con una firma digital, lo que permite a los usuarios confiar lo suficiente en su seguridad como para sacarlas de la caja de arena. Se permite generar digests para mensajes para asegurar su integridad. Encontrard compatibilidad para administrar certificados, acceder a listas de control, SSL/TLS (Nivel de seguridad en las conexiones/Transport Layer Security) e incluso una biblioteca de encriptación exportable.

Todas estas funciones relacionadas con la seguridad estdn presentes para asegurar la correcta construcción y ejecución de programas distribuidos de red.

De arquitectura neutral

La "neutralidad arquitectónica" hace referencia a los códigos de bytes neutrales de Java. En lugar de compilarlos en binarios especificos de cada plataforma, los programas de Java se realizan para que se ejecuten en cualquier equipo sin recompilarlos o revincularlos. Si una compafiia desarrolla nuevo hardware, no tiene que desechar sus inversiones en software, sino que sólo debe colocar un JRE en la nueva plataforma. Y si una nueva compafiia desarrolla hardware completamente nuevo o un nuevo sistema operativo, no tienen que comenzar desde cero sin ningun software para el producto. Con tan sólo agregar el JRE, la plataforma recién diseñada puede ejecutar todos los programas de Java existentes.

De hecho, esto es exactamente lo que hizo una compañía llamada SavaJe (www.savaje.com). SavaJe creo un nuevo sistema operativo para Compaq iPAQ y otras aplicaciones de información basadas en StrongARM de 32 bits. En lugar de tener ensefiar a los programadores a crear aplicaciones para el nuevo sistema operativo, fueron capaces de tomar las aplicaciones existentes basadas en Java y simplemente colocarlas en los dispositivos y se ejecutaron perfectamente (aunque probablemente tendrian una pantalla mds pequefia de lo que esperaban).

Portátil

Quizás haya oido el mantra de Java: Escribir una vez, Ejecutarse en cualquier sitio (EE). El objetivo de Java es ser capaz de tomar el mismo programa y ejecutarlo en cualquier arquitectura. Aunque esta portabilidad no se consigue solamente con

los códigos de bytes de plataforma neutral. En lugar de confiar en detalles específicos de la implementación, tales como el tamaño de un entero, todas las representaciones numéricas en tamaiio, orden de byte y manipulación, estiin definidas en Java Language Specification (http:/ / java.sun.com/ docs/ books/jls/).

Alto rendimiento

Podría pensar que incluir las palabras interpretado y alto rendimiento en la misma frase es un oximorón. No obstante, los códigos de bytes de plataforma neutral realmente pueden convertirse en tiempo de ejecución en código maquina especifico de la CPU, ejecutiindose casi tan rápidamente (si no más) como código compilado de forma nativa C y C++. Hay dos herramientas de traducción incluidas con Java que lo hacen automáticamente: La herramicnta de primera generación recibe el nombre de compilador Justo a tiempo (JIT). La herramienta de segunda generación de Sun se llama HotSpot. Esencialmente, HotSpot y el compilador JIT hacen lo mismo: traducir en tiempo de ejecución al conjunto de instrucciones nativas. Sin embargo, HotSpot también supervisa el código mientras se ejecuta y optimiza la velocidad en los bloques mas frecuentemente ejecutados, en lugar de en todo.

Nota

Si quiere ver una comparacion de las diferentes plataformas de ejecucion de Java, estudie el informe Volano (www.volano.com/report/).

Multitarea

Puede imaginar un subproceso como un contexto de ejecución. Por ejemplo, cuando navega por Internet, su navegador mostrara una nueva página Web mientras sigue descargando imágenes en segundo plano. Cada una de estas tareas puede realizarse en un subproceso separado. Cuando un programa ejecuta varias tareas o subprocesos simultáneamente, se dice que el programa es multitarea.

Los programas multitarea comparten memoria y deben tener comunicación entre los subprocesos. El lenguaje de programación de Java y las bibliotecas estándar incluyen muchos recursos para ayudarle en este proceso de comunicación, asegurando la seguridad de los subprocesos. Aprenderá más sobre la creación de programas multitarea con subprocesos seguros en el capitulo 8.

Lenguaje dinámico

El ultimo cliché caracteristico de Java indica que Java es un lenguaje dinamico. Esto no quiere decir que el lenguaje esté continuamente evolucionando; la frase

tiene más que ver con las bibliotecas. En concreto, significa que aunque las bibliotecas están cambiando continuamente, los programas no deben volver a vincularse. Los primeros códigos de bytes de plataforma neutral seguirán funcionando despuks de que aparezcan nuevas bibliotecas, bien porque Sun presente una nueva version de Java o porque alguna otra compañía cree alguna biblioteca. Mientras las partes de la biblioteca empleadas no desaparezcan, los programas seguiran funcionando, incluso aunque se agreguen nuevos elementos a las bibliotecas. Esta naturaleza dinamica también se cumple en la preferencia de Java por las interfaces sobre las clases, como se mencionó al hablar de la robustez.

El punto de vista de Microsoft

Un libro sobre Java parece necesitar algunas palabras sobre cómo encaja Java en los planes de Microsoft. Basicamente, más alla de un doble intento de que los programadores de Visual J++ avancen, no encaja. El litigio entre Sun y Microsoft comenzó en enero del 2001. Cómo parte del acuerdo, Microsoft puede seguir puede seguir vendiendo el ultimo Java Runtime Environment de Microsoft, si quiere. Han surgido tres cuestiones: En primer lugar, el ultimo entorno proporcionado por Microsoft está basado en Java 1.1.4 (si, tres hace versiones) y es incluso más antiguo si tiene en cuenta el hecho de que la ultima version de 1.1 de Sun es la 1.1.8. En segundo lugar, como Windows XP, el entorno de ejecución de Java de Microsoft ya no se emite con el sistema operativo. En tercer lugar, Microsoft ofrece Visual J#.NET, una herramienta basada en Java 1.1.4 que es compatible con el lenguaje de programación Java pero no con la plataforma, admitiendo en su lugar la arquitectura .NET de Microsoft .

¿Importa? Tkcnicamente, depende. Por supuesto, es agradable tener todos los entornos de ejecución en cualquier equipo. Sin embargo, usar una versión antigua e incompatible de JRE causaba muchos problemas y dolores de cabeza. Sun continuará proporcionando el ultimo entorno de ejecución para las plataformas Windows gratis. Los usuarios sólo tienen que descargarlo de forma separada, a menos que Sun pueda convencer a los vendedores de hardware para que lo preinstalen. (Para Compaq y algunos otros vendedores de hardware, esto ya ha sido anunciado; Compaq adjuntará el entorno de ejecución de Sun con todas las cajas XP.) La decisión de empresa de Microsoft fue simplemente abandonar la compatibilidad del Java estandar con la plataforma Windows. Si le gustaban sus extensiones de Visual J++ y queria una buena integración de Windows, mejor pásese a C# en lugar de moverse al ultimo derivado de Java.

Nota

C#, **pronunciado "C almohadilla", es la creación de Anders Hejlsberg para trabajar con la plataforma .NET de Microsoft.**

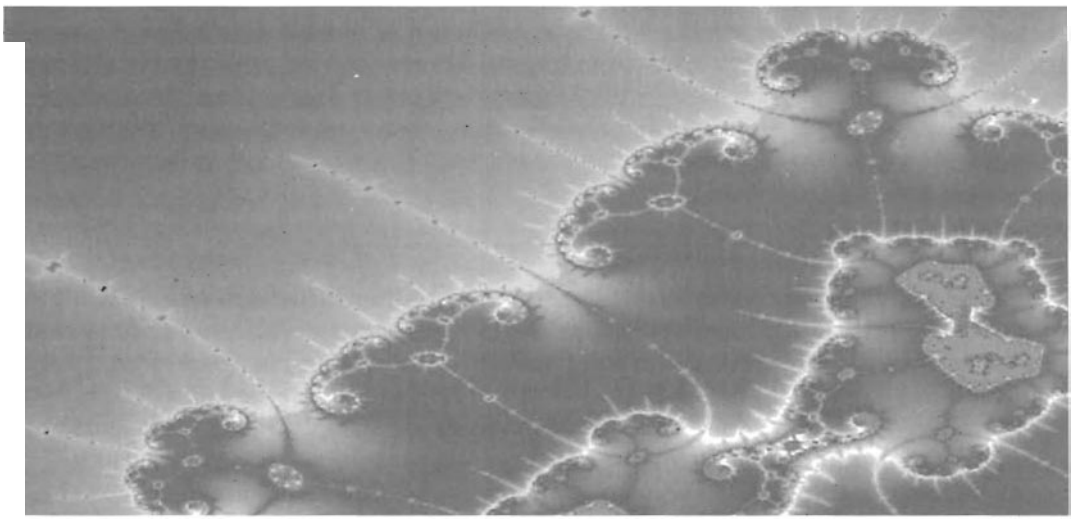
Por lo que respecta a Java como servidor, la falta de un tiempo de ejecución de Microsoft carece de importancia. Los servidores de aplicaciones tienden a tener su propio entorno de ejecución y es sencillo actualizarlos si no le gusta la que le proporcionan.

Resumen

En el verano del 2001, DevX public6 un análisis del estado de Java tras seis aiios de existencia (www.devx.com/judgingjava/articles/sixyears). A medida que avanza en el camino hacia el reino de los programadores productivos de Java, este articulo sera util. Puede leer sumarios de algunos de los hallazgos de DevX, basándose en una encuesta a mas de 2,600 programadores de Java.

En primer lugar, las herramientas son un componente importante del desarrollo de Java. No obstante, nadie está completamente feliz con lo que existe. En casi todos los casos, menos del 50 por ciento de los desarrolladores estaban demasiado satisfechos con cualquiera de las herramientas. Sin embargo, si le gusta seguir las modas, las seis herramientas mas usadas con alto grado de satisfacción en la epoca del articulo eran JBuilder de Borland, WebLogic de BEA, VisualAge para Java de IBM, WebGain Studio/Visual Café de WebGain, CodeWarrior de MetroWorks y Forte para Java de Sun. Mas alla de estas, hay un evidente vacio en la parte del mercado de las herramientas satisfactorias.

La segunda estadistica interesante del estudio de DevX trata sobre las tendencias de la tecnología/lenguaje. De todas las aplicaciones creadas, DevX dice que se cree que el 43 por ciento se construyeron usando tecnologia Java. Visual Basic ocupa la segunda posición con el 25 por ciento y C++ alrededor del 22 por ciento. (C# tenia el 2.8 por ciento para el desarrollo de futuros proyectos, pero ninguno actual.) Me gustaría pensar el 43 por ciento de Java es un numero que deberia animar a la gente a aprender Java y abandonar Visual Basic o C++, pero en realidad debo pensar sobre el valor de este numero como un simple valor estadistico. A simple vista, los desarrolladores encuestados ya eran programadores de Java. Su compaiiia al menos ya habia decidido dar un paso decisivo. Esto significa que las compaiiias sin proyectos de desarrollo Java probablemente no tenian gente para responder las encuestas. Aunque, en comparación, Bloor Interactive dijo en febrero del 2000 que "Java supera a C++ como el conocimiento de programación más demandado". En esa declaración, se fijaron en las ofertas de empleo. Java y C++ corren parejos en casi el 36 por ciento de las casi 350.000 ofertas de empleo en Internet recogidas en un periodo de 12 meses y VB era casi un 10 por ciento inferior. Asi, con una diferencia de más de un aiio entre los dos informes, el 43 por ciento de DevX no parece realmente mal encaminado. El informe de DevX concluye con una lista de áreas con demanda de conocimientos y en donde se necesitan más esos conocimientos. A medida que conoce más sobre Java, piense en adquirir un mayor conocimiento de las áreas de la tecnologia Java más necesitadas: acceso de datos, middleware, XML y programación de servidores.



2

Comenzar con Java 2 SDK

Los programas de Java vienen en varias formas. En principio, sólo habia dos: applets, que son los programas de Java que se ejecutan en un navegador Web y aplicaciones, que no son ejecutables en un navegador Web. Desde entonces, muchas otras variedades de programas de Java se han unido a sus filas. Los servlets son programas de Java que se ejecutan en servidores Web; midlets se ejecutan en dispositivos de información móviles; y hay muchos otros tipos. Este capitulo describe las diferencias especificas entre applets y aplicaciones y describe como crearlos con Kit de desarrollo de software de Java.

Conocido originalmente como Kit de desarrollo de software de Java (JDK), el Kit de desarrollo de software de Java 2 (SDK) es lo que se usa para crear programas para Java 2 Platform, Standard Edition (J2SE). SDK incluye todas las herramientas y bibliotecas estfindar de Java necesarias para crear applets y aplicaciones. Si quiere crear programas de servidor, servlets y otros programas de servidor, necesitarfi tener Java 2 Enterprise Edition (J2EE), que funciona sobre J2SE. Y para el desarrollo de midlets y aplicaciones inalfimbricas tiene Java 2 Micro Edition (J2ME).

SDK está lleno de herramientas de linea de comandos para la creación y ejecución de programas de Java. Encontrará un compilador para convertir el código fuente Java en programas ejecutables. También encontrarfi el modo de documentar, eliminar errores e integrar programas de Java con programas basados en C/C++, además de aplicaciones CORBA. En este capitulo aprendera a preparar su entorno de trabajo inicial y a construir applets y aplicaciones bfisicas. Probará por primera vez un código fuente de Java y aprenderfi a ejecutar programas en Java Runtime Environment. Desde editar hasta compilar y ejecutar, todas estas tareas de desarro-

llo pueden hacerse gratis; ninguna de ellas requiere coste alguno, al menos por lo que respecta a las plataformas de referencia de Sun.

Nota

Las plataformas de referencia disponibles de Sun son Solaris SPARC/x86, Linux x86 y Microsoft Windows. Para otras plataformas, probablemente necesitara visitar al vendedor de su SO y/o hardware.

Applets contra aplicaciones

La palabra applet suele hacernos pensar en una pequeiia aplicación. En realidad, no existe tal restricción respecto al tamaño de los applets; el factor que las define realmente es que el applet se inicia en un navegador Web cargando un archivo HTML. No obstante, dado que los applets se descargan de Internet, los desarrolladores no suelen hacer sus applets muy grandes, teniendo en cuenta el retraso de descarga para recibir los archivos desde el servidor de Internet y luego ejecutarlos en el navegador.

Por otra parte, las aplicaciones se instalan en el cliente, se inician desde la linea de comandos y no tienen limitaciones de descarga. Con las aplicaciones, la versión de Java que se esté ejecutando en su navegador carece de importancia.

Nota

Para obtener mas información sobre la copia en cache de applets para no tener que descargarlos cada vez que se necesiten, vea el capitulo 18. Describe varias opciones de uso, como el Java Plug-in y Java Web Start.

Entorno de ejecución de Java

Antes de pasar a las diferencias entre applets y aplicaciones, es importante asegurarse de que comprende el concepto de Entorno de ejecucidn de Java (JRE).Como su nombre sugiere, JRE es el entorno en el que se ejecutan los programas Java. Para los applets, el navegador proporciona ese entorno. Aunque hablaremos sobre reemplazar ese entorno por Java Plug-in en el capitulo 18, en la mayor parte de los casos, usaremos el navegador para ejecutar applets tomados de Internet. Por otra parte, para las aplicaciones, puede ofrecer JRE con sus programas o, puede confiar en que el usuario ya lo tiene instalado. Las aplicaciones pueden reemplazar JRE facilmente y podrá asegurarse de que el usuario esta usando la version con la que probo el programa.

llo pueden hacerse gratis; ninguna de ellas requiere coste alguno, al menos por lo que respecta a las plataformas de referencia de Sun.

Nota

Las plataformas de referencia disponibles de Sun son Solaris SPARC/x86, Linux x86 y Microsoft Windows. Para otras plataformas, probablemente necesitara visitar al vendedor de su SO y/o hardware.

Applets contra aplicaciones

La palabra applet suele hacernos pensar en una pequeña aplicación. En realidad, no existe tal restricción respecto al tamaño de los applets; el factor que las define realmente es que el applet se inicia en un navegador Web cargando un archivo HTML. No obstante, dado que los applets se descargan de Internet, los desarrolladores no suelen hacer sus applets muy grandes, teniendo en cuenta el retraso de descarga para recibir los archivos desde el servidor de Internet y luego ejecutarlos en el navegador.

Por otra parte, las aplicaciones se instalan en el cliente, se inician desde la linea de comandos y no tienen limitaciones de descarga. Con las aplicaciones, la versión de Java que se esté ejecutando en su navegador carece de importancia.

Nota

Para obtener mas información sobre la copia en cache de applets para no tener que descargarlos cada vez que se necesiten, vea el capitulo 18. Describe varias opciones de uso, como el Java Plug-in y Java Web Start.

Entorno de ejecución de Java

Antes de pasar a las diferencias entre applets y aplicaciones, es importante asegurarse de que comprende el concepto de Entorno de ejecución de Java (JRE).Como su nombre sugiere, JRE es el entorno en el que se ejecutan los programas Java. Para los applets, el navegador proporciona ese entorno. Aunque hablaremos sobre reemplazar ese entorno por Java Plug-in en el capitulo 18, en la mayor parte de los casos, usaremos el navegador para ejecutar applets tomados de Internet. Por otra parte, para las aplicaciones, puede ofrecer JRE con sus programas o, puede confiar en que el usuario ya lo tiene instalado. Las aplicaciones pueden reemplazar JRE fácilmente y podrá asegurarse de que el usuario está usando la version con la que probó el programa.

Puede descargar JRE de forma gratuita desde http://java.sun.com/j2se/1.4/jre/. No necesita tenerlo para crear programas de Java y la distribución es gratuita cuando se trata de implementar sus aplicaciones.

Aquí viene el punto problemático para los lectores: Aunque este libro trata de desarrollar programas para la plataforma Java 1.4, la version de JRE que aparece en la mayoría de los navegadores compatibles con Java es sólo Java 1.1.x. De modo que si intenta usar cualquiera de las nuevas e interesantes funciones agregadas a las bibliotecas desde Java 1.1, el applet resultante no funcionará en cualquier version de Internet Explorer. Por lo que resyecta a los navegadores de Netscape, el applet no funcionará en Netscape Navigator/Communicator 4.x y sólo tendrá alguna posibilidad de funcionar con Netscaye 6, ya que este navegador confia en el anteriormente mencionado Java Plug-in para su JRE, aunque no parte con ninguno.

Nota

Es muy importante tener en cuenta que, aunque el JRE que aparece en los navegadores esta basado en Java 1.1 todavia se pueden ejecutar applets en los navegadores creados a partir de la version 1.4 de Java 2 SDK. Simplemente, no puede usar las funciones que no estaban en la epoca de Java 1.1 ¡La clave esta en conocer cuales son las diferencias! El modo mas sencillo de conocer las diferencias es estudiar la documentación de la antigua version del tiempo de ejecucion Java en http://java.sun.com/products/jdk/1.1/docs.html, aunque este usando el compilador mas moderno.

Applets y aplicaciones: ¿Cuál es la diferencia?

Piense en dónde se ejecuta un applet y dónde se ejecuta una aplicación y coinprendera casi todas las diferencias entre ellas. Los applets se ejecutan en los navegadores y las aplicaciones no.

Cuando tenga clue usar un apylet, yiense siempre en descargar un programa desde Internet. Como un apylet se puede descargar cada vez que visita una página web, los navegadores deben proporcionar un entorno muy controlado para ejecutar ese ayplet. Habrá oido hablar de esa zona llamada zona de seguridad. La zona de seguridad proporciona un entorno seguro que evita que el applet pueda acceder a cualquier parte del equipo cliente y solamente permite al ayylet cornunicarse con el servidor desde el que procede. Además, el navegador dice al applet cuándo debe inicializarse, cuándo debe ejecutarse, cuándo activarse o desactivarse, además de cuándo descargarse. Por último, el navegador proporciona una zona dentro de una página HTML para que se muestre el applet y esa es la zona que el applet usa para dibujar.

Las aplicaciones, al contrario, no tienen estas restricciones o zonas de muestra automáticas. Como las aplicaciones se ejecutan de forma local, pueden hacer cual-

quier cosa. Un applet tiene una zona dentro del navegador para mostrarse pero, una aplicación no proporciona este entorno gráfico por defecto. De acuerdo, puede crear uno, pero no está alli automáticamente. Y un applet puede tener entradas desde el exterior procedentes de su propia página HTML, mientras que una aplicación obtiene las entradas desde la linea de comandos.

De modo que cuando llega la hora de escribir un programa, ¿cuál crear? Bien, depende. No hay una respuesta clara. Si quiere usar cualquier función de Java 1.4, 1.3 o incluso 1.2, estará obligado a trabajar con aplicaciones. No obstante, si no le importa instalar Java Plug-in, puede usar estas versiones posteriores en applets. Tenga en cuenta que el plug-in no es pequeiio: Está entre los 9Mb para Windows, los 21Mb para Linux y hasta los casi 30Mb para Solaris de 64 bits, de modo que piense en las consecuencias de usarlo. Para una intranet, el plug-in es una opción viable, pero para un applet basado en Internet, probablemente no lo sea. Como las redes corporativas pueden controlar la configuración del sistema y suelen tener conexiones de red de alta velocidad, el plug-in puede ser una opción adecuada.

Si le gusta la idea de entrega por Internet con poco o ningún coste de instalación, no olvide automaticamente las aplicaciones. Puede usar herramientas como InstallShield o InstallAnywhere, que crean aplicaciones multiplataforma instalables que pueden distribuirse por la red. Y Java Web Start es una opción estandar para distribuir aplicaciones, pero ejecutarlas en un entorno parecido a la zona de seguridad. Todas estas opciones se tratarán con mayor profundidad en el capitulo 18, junto con Java Plug-in.

Cómo trabajar con Java 2 SDK

Cuando comienza a crear programas en Java, muchos desarrolladores comienzan con el equipo de desarrollo de Sun, el SDK. Lo mejor es el precio (es gratuito) y siempre puede trabajar con la ultima version (siempre que su plataforma sea Windows, Solaris o Linux). SDK es un conjunto de herramientas de linea de comandos para crear programas Java, tanto applets como aplicaciones. Todo lo que tiene que hacer es proporcionar su editor preferido y ya tiene todo lo que necesita.

Como parte del equipo de desarrollo, obtiene JRE y un conjunto de bibliotecas para sus programas. La combinación del entorno de ejecucion y las bibliotecas suele denominarse Java Platform. Con cada nueva version de Java, el conjunto de bibliotecas cambia, pero el entorno de ejecucion subyacente permanece inalterado.

Las bibliotecas que acompafian a cada nueva version de Java son lo que realmente diferencia a Java de la mayoría de los otros lenguajes de programacion. Cuando conoce su version de Java, conoce las bibliotecas que puede esperar encontrar. Esto es cierto sin importar qué plataforma esté usando, tanto si es una de las implementaciones de referencia de Sun o una version externa de IBM o similar. Sun tiene un exhaustivo paquete integrado de prueba que se asegura de que todo lo que se llame "Java" supere su prueba de compatibilidad. De modo que si una implementación externa no lo pasa, no es Java. Como sus propios programas pue-

den convertirse de esta manera, Sun ha acuiiado el término "Escribir una vez, Ejecutarse en cualquier sitio (EE)" para describir los programas Java.

Las bibliotecas disponibles con la plataforma Java le permiten hacer casi cualquier cosa. Hay ayudas para graficos, trabajo en red, estructuras de datos, procesos XML y de E/S de disco, entre muchas otras tecnologias. Cuando se mueve a otros tipos de entornos de ejecución, como tarjetas inteligentes o dispositivos insertados en lugar de ordenadores personales, el conjunto de API principal será el mismo. Y Sun mantiene subconjuntos bien definidos de todas las bibliotecas para estos otros entornos, de modo que sabrá lo que no está disponible. Por ejemplo, no hay un entorno gráfico para las tarjetas inteligentes, de modo que las bibliotecas graficas no están disponibles. Mientras se ajuste a lo que esta disponible en los diferentes entornos de ejecución, los mismos programas Java se ejecutarán alli también (aunque normalmente conocerá el equipo en el que se piensa ejecutar el programa antes de comenzar).

Herramientas SDK

Antes de descargar e instalar Java 2 SDK, veamos las herramientas que acompañan al paquete y como funcionan en conjunción. Hay **23** utilidades y usará las siguientes ocho en este libro:

- javac: El compilador de Java. Convierte el código fuente en códigos de byte.
- java: El lanzador de la aplicación Java. Ejecuta códigos de byte para una aplicación.
- appletviewer: El applet lanzador de Java. Ejecuta códigos de byte para un applet.
- javadoc: La herramienta de comentarios de Java. Genera documentación HTML a partir de archivos fuente de Java.
- jdb: El depurador de Java. Se mueve a través de los programas a medida que se ejecutan, establece puntos de interrupción y supervisa el comportamiento.
- javap: El desensamblador de Java. Examina los códigos de byte para mostrar información.
- jar: Gestor Java Archive (JAR). Combina y comprime varios archivos y directorios en uno.
- javah: Generador de encabezados de archivo. Se usa cuando se combina Java con programas CIC++.

Suponiendo que tenga un editor de texto que permita guardar archivos fuente de Java como texto puro (es decir, sin el tipo de información de formato que se encuentra en los archivos .doc de Microsoft Word) puede crear y ejecutar programas Java con estas herramientas SDK. El proceso de crear aplicaciones aparece en

la figura 2.1 y la figura 2.2 muestra las pequeñas diferentes tareas implicadas en la creación de un applet que funciona. Tras descargar SDK, siga los pasos que aparecen en estas figuras para crear su primera aplicación y applet Java.

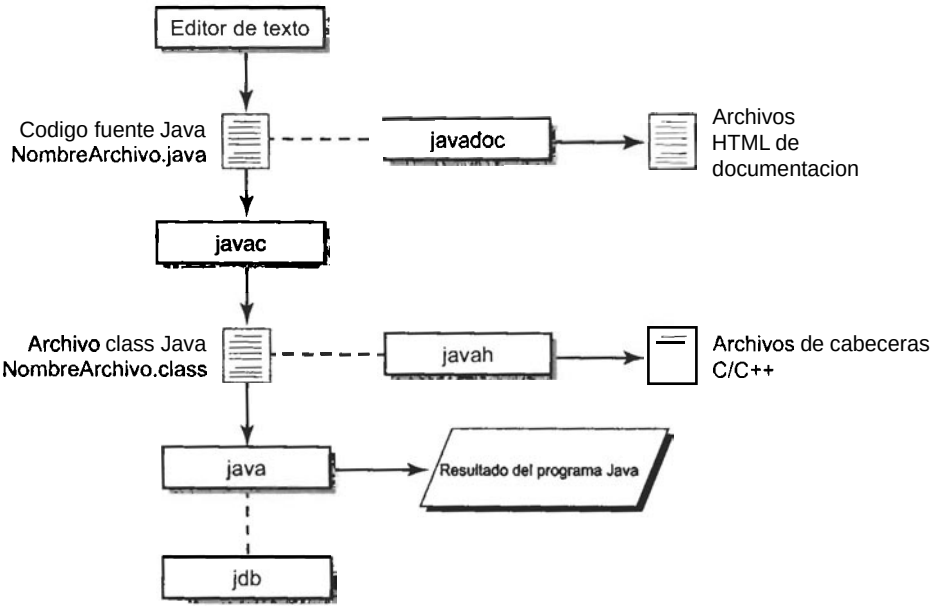


Figura 2.1. Cómo se crean aplicaciones Java con SDK

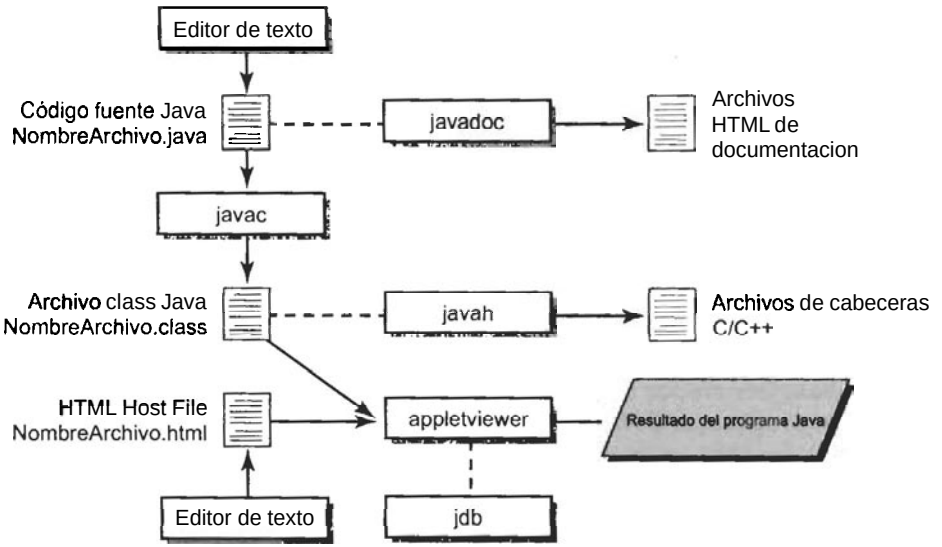


Figura 2.2. Cómo se crean applets Java con SDK

A continuación tiene las 15 herramientas SDK restantes, enumeradas en grupos lógicos:

- javaw: Lanzador de aplicaciones sin consola. Ejecuta códigos de byte para aplicaciones pero se ejecuta en una consola sin ventanas.
- extcheck: Comprobador de conflictos Java Archive (JAR).Busca conflictos de version en los archivos JAR.
- native2ascii: Conversor Unicode. Convierte archivos de un formato de codificación a otro.
- jarsigner, keytool, policytool: Estas herramientas de gestión de seguridad le permiten crear applets que salen de la zona de seguridad.
- rmic, rmid, rmiregistry, serialver: Herramientas para trabajar con Remote Method Invocation (RMI).
- idlj, orbd, servertool, tnameserv: Herramientas para trabajar con CORBA, IDL e IIOP.
- unregbean: Herramientas para desregistrar los componentes JavaBeans que se usan con ActiveX.

Descargar e instalar SDK

Para empezar a trabajar con Java, deberá descargar SDK para Java 1.4. Tras instalarlo en su equipo, podrá trabajar con los ejemplos de este libro.

Descargar SDK

Puede obtener la versión 1.4 de SDK para Microsoft Windows 95/98/Me/NT4/2000, para Solaris SPARC/x86, o para Linux en el CD de este libro, o directamente en la pagina de Internet de Sun en http://java.sun.com/j2se/1.4/. Para otras plataformas, deberá visitar http://java.sun.com/cgi-bin/java-ports.cgi para encontrar una lista de puertos externos. Las nuevas versiones de Java no están inmediatamente disponibles para otras plataformas, de modo que puede haber algun retraso en su disponibilidad.

Instalación de SDK

Las instrucciones para instalar SDK son diferentes para cada plataforma. Si tiene algun problema, siga siempre el vínculo **Installation** de la página de descarga de Sun para encontrar ayuda.

Instalación de Windows

El paquete de instalación de Windows es un instalador InstallShield. Simplemente ejecute el archivo j2sdk-1_4_0-win.exe, siga las indicaciones y tendrá todo ubicado en C:\j2sdk1.4.0 por defecto. No tiene que instalar los Native Interface

Header Files, Demos, o Java Sources, pero al menos deberá instalar los dos ultimos. Tras terminar con el instalador, actualice el PATH del sistema. Este es un conjunto de directorios que el sistema usa para encontrar los programas que debe ejecutar. Para que funcionen las herramientas Java en cualquier sitio, agregue:

```
C:\j2sdk1.4.0\bin
```

al PATH. Si ya hay varios elementos en el PATH, use un punto y coma (;) como separador.

El modo de actualizar el PATH del sistema depende de la versión de Windows que tenga. Por ejemplo, con Windows 95/98, puede agregar lo siguiente como la ultima linea del archivo C:\autoexec.bat file:

```
SET PATH=C:\j2sdk1.4.0\bin;%PATH%
```

donde %PATH% significa la actual configuración de ruta y simplemente anexione el recién instalado directorio contenedor SDK. Para Windows NT/2000, use el icono Sistema en el Panel de Control para cambiar el PATH. Y Windows Me tiene una utilidad de configuración de sistema enterrada en Accesorios>Herramientas de sistema>Información de sistema para hacer lo mismo.

En la mayoría de los casos, tras cambiar el PATH necesitará reiniciar el quipo para que los cambios tengan efecto, aunque quizás quiera esperar a reiniciar su equipo tras leer la próxima sección "Comprender el CLASSPATH".

Instalación de Linux

La versión Linux de Sun se ejecuta en plataformas Intel que ejecutan el kernel 2.2.12 y la biblioteca glibc 2.1.2-11 o posterior. La versión oficialmente compatible de Linux es RedHat Linux 6.2. Puede obtener el binario auto-extraible (j2sdk-1_4_0-beta-linux-i386.bin) o el archivo RedHat RPM (j2sdk-1_4_0-beta-linux-i386-rpm .bin).

- Si descarga el binario auto-extraible, tras ejecutar la secuencia de comandos, SDK estará instalado en el directorio j2sdk1.4.0 bajo el directorio en el que ejecutó la secuencia de comandos. En primer lugar, haga el archivo ejecutable mediante `chmod a+x` y ejecutelo. Sólo asegurese de que tiene permiso de escritura para cualquier directorio que esté usando.
- Con el archivo RPM, SDK se instalará en el directorio /usr/java/j2sdkl.4.0. Al igual que hizo con el binario auto-extraible haga este archivo ejecutable y ejecutelo. Sin embargo, al ejecutarlo sólo creará el archivo j2sdk-1.4.0.i386.rpm. Necesitará entonces ejecutarlo como raiz con `rpm -iv` para instalar SDK.

No se puede cambiar la posición de RPM, de modo que si prefiere instalarlo en /usr/local/, no use el archivo RPM.

Cuando haya instalado SDK, agregue el directorio contenedor en el directorio de instalacidn a su PATH para que las herramientas de Java estén disponibles.

Instalación de Solaris

Para las plataformas Solaris 2.6, 7 y 8, necesitarci disponer del archivo empaquetado apropiado (j2sdk-1_4_0-beta-solsparc.sh para Solaris/SPARC o j2sdk-1_4_0-beta-solx86.sh para Solaris/x86) o paquetes (j2sdk-1_4_0-beta-solsparc.tar.Z o j2sdk-1_4_0-beta-solx86.tar.Z). Puede ocurrir que tambien deba descargar parches para su sistema operativo. Combinados, el paquete y los parches pueden ocupar hasta 100Mb (comprimidos). Asegúrese de descargar e instalar los parches en primer lugar.

- Si descarga el archivo empaquetado, tras ejecutar la secuencia de comandos tendrá instalado SDK en el directorio j2sdk1.4.0. Simplemente haga ejecutable la secuencia de comandos mediante chmod +x y luego ejecute la secuencia de comandos desde el directorio en el que desea crear j2sdk1.4.0. Siempre podrá mover el directorio posteriormente.
- Si descarga los paquetes, necesitard descomprimir los archivos .tar.Z con zcat, eliminar los paquetes 1.3 SDK con pkgrm si instal6 1.3 (SUNWj3dmo, SUNWj3man, SUNWj3dev y SUNWj3rt) y agregue los nuevos paquetes 1.4 con pkgadd. Para SPARC, los nuevos paquetes son SUNWj3rt, SUNWj3dev, SUNWj3man, SUNWj3dmo, SUNWj3rtx y SUNWj3dvx. Para Intel, deje los dos ultimos. Terminará con SDK instalado en el directorio /usr/j2se.

Cuando haya instalado SDK, agregue el directorio contenedor a j2sdk1.4.0 o agregue /usr/j2se a su PATH de modo que las herramientas Java estén disponibles sin tener que especificar su ruta completa cuando necesite ejecutarlas.

> **Nota**
>
> Por motivos de conveniencia, todos los nombresde directoriosdurante el resto del libro estaran en formato DOS (Windows), excepto cuando sea necesario señalar alguna diferencia especifica entre plataformas.

Comprender el CLASSPATH

Además de la variable de entorno PATH, Java deyende de otro valor de variable: CLASSPATH. El entorno de ejecución de Java buscara elementos de usuario para ejecutar (clases de Java) en la variable de entorno CLASSPATH. Si no se establece adecuadamente, tendrá muchos problemas y le costara mucho resolverlos. El sistema sabe ddnde buscar las clases de sistema, de modo que no puede interferir con esto cuando experimente con la variable CLASSPATH.

Bajo circunstancias normales, no tendrá que hacer nada con el valor de su CLASSPATH. Si no está establecido, JRE buscará clases de usuario en el directorio en

el que se encuentre. Si CLASSPATH está establecido, JRE buscará en el directorio en el que se encuentra solamente si le indica que debe hacerlo y examinari solamente las posiciones especificadas por el valor de la variable. Normalmente, querrá que JRE busque en el directorio en el que se encuentre, porque ahi es donde puede encontrar las clases asociadas al programa que está desarrollando.

A veces, programas instalados anteriormente pueden establecer el valor de CLASSPATH. Si ha ocurrido, necesitari volver a agregar manualmente el directorio actual a la ruta de búsqueda. Un punto (.) en CLASSPATH representa el directorio con el que se está trabajando actualmente, de modo que agréguelo a la variable.

En Windows, las entradas CLASSPATH están separadas por punto y coma. Al igual que PATH, establecer esto depende de su plataforma. El nuevo valor buscará algo parecido a lo siguiente, aunque dependera de que otros programas tenga instalados para las otras entradas. La parte importante, en cualquier caso, son el punto y coma y el punto finales, como se muestra aquí:

```
SET CLASSPATH=C:\foo\bar;D:\bar\foo.jar;
```

Bajo Unix (Solaris o Linux), las entradas CLASSPATH están separadas por dos puntos. El modo exacto de establecer su ruta clepende de su intérprete de comandos. Aquí, la parte importante son los dos puntos y el punto al final:

```
setenv CLASSPATH /usr/foo:.  (Para intérpretes c)

CLASSPATH=/usr/foo:.   (Para intérpretes bourne)
export CLASSPATH
```

Truco

Para mas información sobre la configuración de CLASSPATH acuda a las siguientes URL: http://java.sun.com/j2se/1.4/docs/tooldocs/win32/classpath.html y http://java.sun.com/j2se/1.4/docs/tooldocs/solaris/classpath.html.

Cuando tenga todo preparado y haya reiniciado su equipo, si es necesario, puede ejecutar el comando `java -version` para comprobar si su PATH es correcto y asegurarse de que está usando la versión apropiada de Java SDK. Suponiendo que el comando le indica que está usando la versión 1.4, estará todo preparado. En caso contrario, tendrá que volver a comprobar sus configuraciones. Dado que el directorio contenedor de Java se agregó en la parte delantera del valor PATH, el sistema no encontrará ninguna otra version en otra posición PATH.

Cómo descargar la documentación

Además del mismo SDK, deberia descargar la documentación SDK desde http://java.sun.com/j2se/l.4/. El archivo comprimido de 31Mb (160Mb

descomprimido) incluye documentacion de uso y API para ayudarle a convertirse en un programador productivo. Aunque siempre puede buscar informacion en la red en http://java.sun.com/j2se/l.4/docs/, es mas rapido tener la documentacion instalada de forma local. Tras desempaquetar, asegúrese de señalar el archivo index.html en el directorio de documentos de nivel superior.

Cómo crear aplicaciones Java

Esta sección le llevará a través de todos los pasos que aparecen en la figura 2.1, desde la edición hasta la ejecución, pasando por la compilación y mucho más.

La aplicación Hello World

Para comenzar con el desarrollo con Java, necesitará un editor. Los usuarios de Unix encontrardn utiles los emacs o vi y los usuarios de Windows pueden usar el programa Bloc de notas estdndar. Escoja cualquier editor con el que esté familiarizado, mientras pueda guardar sus archivos fuente como texto.

Un truco para los usuarios de Windows: Si estd usando el Bloc de notas de Windows como editor de texto, descubrird que guarda los archivos .java con la extensión añadida .txt (por ejemplo, guarda Foo.java como Foo.java.txt). Una solución a corto plazo es escribir el nombre de archivo entre comillas ("Foo.java") para guardarlo. A largo plazo, tendrd que dirigirse al Explorador de Windows y asociar los archivos .java al Bloc de notas.

Para los principiantes, creard un programa que escribirá un mensaje en la pantalla. Este pequeño programa suele llamarse Hello World porque ese es el mensaje que muestra. Coloque el código fuente que aparece en el listado 2.1 en un archivo llamado HelloWorld.java. Java diferencia entre minúsculas y mayúsculas, de modo que deberá asegurarse de usar correctamente las mayúsculas y las minúsculas en el nombre del archivo. Debe coincidir con lo que se encuentra en el código fuente.

Listado 2.1. El programa Hello World

```
public class HelloWorld {
    public static void main(String args[]) {
        System.out.println("Hello, World");
    }
}
```

Nota

El codigo fuente de todos los ejemplos del libro aparece en el CD-ROM que acompaña al libro.

No se preocupe si todavia no comprende todos los detalles del programa. Aprenderá más a medida que avanza por el resto del libro. Básicamente, este archivo contiene la definición de una clase llamada `HelloWorld`. La clase contiene un método llamado `main()`.

Cuando intenta ejecutar una aplicación en Java, el entorno de ejecución busca un método `main()`. El método `main()` debe tener las palabras clave `public`, `static` y `void` asociadas a él. Además, debe aceptar una matriz de elementos String como argumento.

Nota

La implementación de referencia de Sun ejecutara el programa incluso si falta la palabra clave public para `main()`. Sin embargo, quizás no funcione con otros entornos de ejecución, de modo que habitúese a incluir las palabras clave necesarias.

La parte `String args[]` de la declaración del método `main()` hace referencia a cualquier argumento de linea de comandos que pase al programa Java. Esto es igual que la declaración de C/C++

```
int main(int argc, char *argv[]);
```

excepto que en C/C++, es necesario un segundo argumento para pasar el número de elementos de la matriz, En Java, las matrices conocen su longitud, de modo que en este caso el primer argumento no es necesario.

Aparte de las diferencias entre mayúsculas y minúsculas, la declaración del método de `main()` de Java es practicamente idéntica al mecanismo de C# para pasar argumentos de linea de comando:

```
public static void Main(string[] args)
```

La línea restante `System.out.println("...");` del listado2.1 indicaquese debe enviar el mensaje entre comillas a `System.out`, que es la ventana de la consola desde la que inició el programa.

Cómo compilar el código fuente de la aplicacion

Tras la ruta principal de la figura 2.1, puede ver que tras guardar su programa con el editor, el siguiente paso es compilarlo con el compilador (javac). Para compilar el programa, simplemente escriba `javac` seguido por el nombre del archivo fuente, respetando las mayúsculas y las minúsculas. Si no la tiene ya, necesitará tener una ventana de comandos emergente y estar en el directorio en el que guard6 su archivo. Entonces escriba

```
javac HelloWorld.java
```

Si el código fuente no tiene errores, javac genera un archivo llamado HelloWorld. class y regresa inmediatamente a la ventana de comandos.

Si estii interesado en seguir los entresijos de la compilación, puede usar la opción -verbose con javac. En una sola clase, los resultados no son muy interesantes, pero apreciarii lo que el compilador necesita para verificar que el programa es válido. Si ejecuta este comando:

```
javac -verbose HelloWorld.java
```

se generarii el siguiente resultado:

```
[parsing started HelloWorld.java]
[parsing completed 203ms]
[loading C:\j2sdk1.4.0\jre\lib\rt.jar(java/lang/Object.class)]
[loading C:\j2sdk1.4.0\jre\lib\rt.jar(java/lang/String.class)]
[checking HelloWorld]
[loading C:\j2sdk1.4.0\jre\lib\rt.jar(java/lang/System.class)]
[loading C:\j2sdk1.4.0\jre\lib\rt.jar(java/io/PrintStream.class)]
[loading C:\j2sdk1.4.0\jre\lib\rt.jar(java/io/FilterOutputStream.class)]
[loading C:\j2sdk1.4.0\jre\lib\rt.jar(java/io/OutputStream.class)]
[wrote HelloWorld.class]
[total 640ms]
```

Si revisa el código fuente de su programa y mira en la documentación para las clases, descubririi por qué cada una de las clases es necesaria. A menos que se indique lo contrario, las clases se crean a partir de la clase `Object`. El método `main()` del programa recibe un argumento `String`. Dentro de `main()`, hay una referencia a la clase `System`. La variable `out` de `system` resulta ser del tipo `PrintStream`. Las dos últimas clases cargadas hacen referencia a `PrintStream`, un tipo de `FilterOutputStream`, que es un tipo de `OutputStream` (que es un tipo de `Object`, pero que ya se ha cargado). Con todas estas clases verificadas y sin errores en el código fuente, la compilación deberia realizarse satisfactoriamente y se generaria el archivo .class.

El archivo .class generado es un binario formal de plataforma neutral que entiende la Miiquina Virtual de Java (JVM). JVM es lo que ejecuta los códigos de byte en JRE.

Si estii interesado en aprender más sobre el formato del archivo, puede leer las especificaciones de la Miiquina Virtual de Java en http://java.sun.com/docs/books/vmspec/2nd-edition/html/VMSpecTOC.doc.html.

Cómo ejecutar la aplicación

Tras compilar el código fuente, puede ejecutar el programa. Use el lanzador de aplicaciones Java, `java`, para hacerlo. Introduzca este comando:

```
java HelloWorld
```

que mostrará el resultado "Hello, World" como muestra la figura **2.3.**

Figura 2.3. Una sesion de Windows para compilar y ejecutar la clase HelloWorld

Truco

Cuando introduzca el comando `java`, asegúrese de no especificar .class al final del nombre de la clase que quiere ejecutar. Sólo especifique el nombre de la clase, pero no el nombre del archivo.

Hay algunas opciones de línea de comandos que pueden resultar interesantes cuando ejecute sus programas. Para ver una lista de las opciones, introduzca `-?` como opción de línea de comandos, como aparece a continuación:

```
java -?
```

Devolverá una lista de opciones que puede introducir en el lanzador, como se muestra aquí (no necesitará la mayoría de ellas):

Modo de empleo: `java` [-opciones] clase [argumentos...]:
(para ejecutar una clase)

o `java -jar` [-opciones] archivo jar [argumentos...]:
(para ejecutar un archivo jar)

Entre las opciones se incluyen:

`-hotspot`: Para seleccionar el "hotspot" VM.

`-server`: Para seleccionar el "servidor" VM.

Si se emplea, la opción para seleccionar el VM debe ir en primer lugar. El VM por defecto es `-hotspot`.

`-cp -classpath` <directorios y archivos zip/jar separados por ;>: Establece la ruta de búsqueda para las clases y recursos de la aplicación.

`-D`<nombre>=<valor>: Establece una propiedad de sistema.

-verbose[:clase | gc | jni]: Activa la salida profusa.

-version: Escribe la version del producto y finaliza.

-showversion: Escribe la version del producto y continua.

-?-help: Este mensaje de ayuda.

-X: Escribe la ayuda sobre las opciones no estdndar.

El listado 2.2 muestra algunas otras opciones no estdndar ocultas bajo la opción -X. Estas opciones no son estdndar y estdn sujetas a cambio sin aviso. Es decir, algunas de ellas en realidad se usan con mas frecuencia que las opciones estdndar. Por ejemplo, si ha creado un programa que requiere mucha memoria, puede necesitar aumentar el espacio de pila de Java con la opción -Xmx.

Listado 2.2. Opciones no estandar para el comando java

-Xmixed: Ejecucion en modo mixto (por defecto).
-Xint: Solamente modo interpretado.
-Xbootclasspath:<directorios y archivos zip/jar separados por ;>: Establece la ruta de busqueda para las clases y recursos bootstrap.
-Xbootclasspath/a:<directorios y archivos zip/jar separados por ;>: Anexiona al final de la ruta de clase bootstrap.
-Xbootclasspath/p:<directorios y archivos zip/jar separados por ;>: Anexiona al principio de la ruta de clase bootstrap.
-Xnoclassgc: Deshabilita la recolección de elementos no utilizados de clase.
-Xincgc: Habilita la recoleccion de elementos no utilizados creciente.
-Xconcgc: habilita la recoleccion de elementos no utilizados concurrente en su mayor parte
-Xloggc:<file>: Conecta el estatus GC a un archivo con marca de hora.
-Xbatch: Desactiva la cornpilacion de fondo.
-Xms<tamaño>: Establece el tamafio inicial de la pila de Java.
-Xmx<tamaño>: Establece el tamafio mdximo para la pila de Java.
-Xss<tamaño>: Establece el tamafio de pila de subproceso de java.
-Xprof: Muestra los datos de descripción de la cpu.
-Xrunhprof[:help] | [:<opción>=<valor>, ...]:Realiza la descripción de la pila JVMPI, cpu o monitor.
-Xdebug: Activa la depuración remota.
-Xfuture: Activa las pruebas más estrictas, anticipando los valores futuros por defecto.
-Xrs: Reduce el uso de Java/VM de sefiales del SO (ver documentación).

Documentando la aplicación

Desplazándonos un poco por el diagrama de flujo nos encontramos con el comando javadoc y la importante tarea de comentar el código. SDK para Java inclu-

ye una utilidad para permitir la documentación automática de su código fuente. Tras agregar comentarios especiales, a su código fuente, puede preparar estos comentarios para que sean transferidos automáticamente a sus archivos HTML para documentar su programa. Todos los comentarios del código fuente colocados entre /** y */ se consideran comentarios informativos y se agregan a los archivos generados. Si agregamos unas pocas líneas de comentarios al archivo fuente original Hello World, como aparece en el listado 2.3, probamos esta utilidad.

Listado 2.3. HelloWorld con varios comentarios

```
/** HelloWorld - Este es el primer programa de Java que he creado. */

public class HelloWorld {

/**
 * Ecte es el método inicial invocado por el lanzador de Java
 * Escribira un mensaje en la pantalla.
 */

 public static void main(String args[]) {
     System.out.println("Hello, World");
 }

}
```

Para generar la documentación, pase una lista de archivos fuente al comando javadoc para que las procese (en este ejemplo, sólo hay un archivo de código fuente):

```
javadoc HelloWorld.java
```

Esto mostrará un grupo de mensajes:

```
Loading source file HelloWorld.java...
Constructing Javadoc information...
Buiiding tree for all the packages and classes...
Building index for all the packages and classes...
Generating overview-tree.html...
Generating index-all.html...
Generating deprecated-list.html...
Building index for all classes...
Generating allclasses-frame.html...
Generating index.html...
Generating packages.html...
Generating HelloWorld.html...
Generating package-list...
Generating help-doc.html...
Generating stylesheet.css...
```

Todo este resultado le esta indicando, básicamente, que tiene siete nuevos archivos HTML y otro nuevo archivo para cada archivo fuente pasado al comando. También hay un archivo de Hojas de estilo en cascada (CSS) que contiene la infor-

mación de formato para las pdginas HTML. Si abre el archivo especifico de la clase, HelloWorld.html, verá los comentarios informativos que agregó. Esto se puede ver en la figura 2.4.

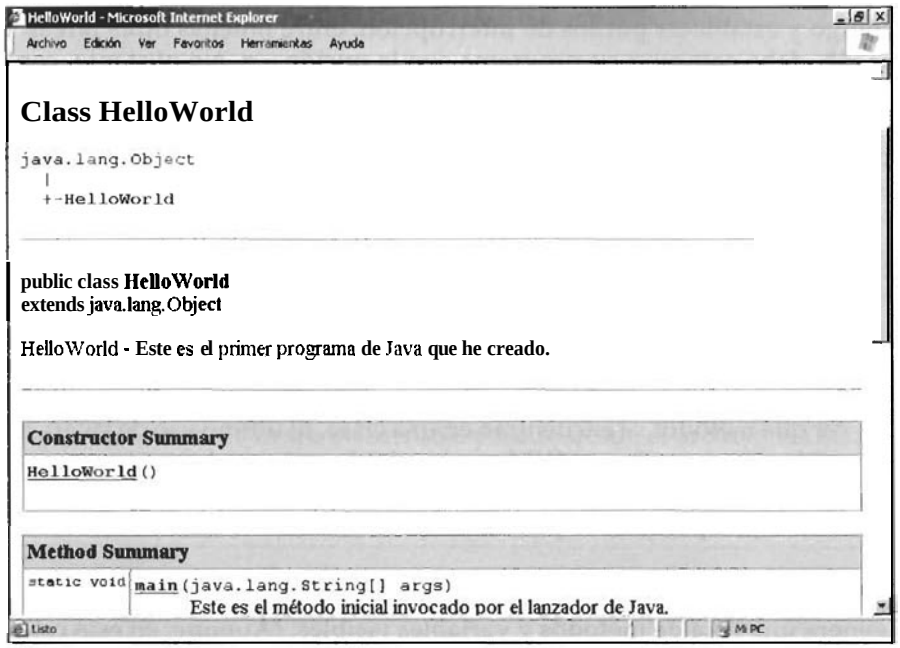


Figura 2.4. Vista del archivo HTML generado por javadoc a partir de los comentarios a la fuente HelloWorld.java

Integración con código C/C++

La herramienta javah le ayuda a conectar los programas Java con los programas C y C++. Esto se vuelve necesario cuando existen bibliotecas heredadas que no desea convertir a Java, pero que desea usar, o cuando no puede realizar algo con Java debido a algunas necesidades especificas de cierta plataforma. Como las bibliotecas estdndar han aumentado y como más caracteristicas se han convertido en estándar en Java, la necesidad de usar código nativo C/C++ ha disminuido enormemente. Si todavia necesita conectar, la herramienta javah estd alli para ayudarle. Ejecútela sobre un archivo de clase y generará los archivos de cabecera necesarios para conectar con la clase.

Nota

Si necesita conectar Java y codigo C/C++, revise el tutorial en línea en http://java.sun.com/docs/books/tutorial/native1.1/.

Cómo depurar su aplicación

SDK viene acompañado de un depurador de linea de comandos, jdb. Con jdb, puede ejecutar sus programas, examinar la configuración de variables, moverse por el código y establecer puntos de interrupción, entre muchas otras tareas. Si planea usar jdb, debe compilar su programa con la opción `-g`. No obstante, con entornos de programación integrados como Forte para Java (Community Edition) y JBuilder (Personal Edition) disponibles de forma gratuita, mejor consiga uno de ellos para depurar de forma visual sus programas. Estas herramientas hacen la tarea mucho más sencilla.

Cómo desensamblar sus clases

La herramienta javap le permite consultar cualquier clase y descubrir su lista de métodos (como `main()`) o descubrir los comandos ejecutados por la JVM subyacente. Descubrira que usa las consultas javap con frecuencia como un modo sencillo de ver qué métodos se encuentran en una clase. El ultimo uso de javap, comprobar la ejecución de comandos en JVM, es principalmente usada por los curiosos y poco frecuentemente.

Cuando a javap sólo se le pasa el nombre de la clase, asi:

```
javap HelloWorld
```

genera una lista de métodos y variables visibles. (Aunque, en este caso, no tenemos variables en la clase `HelloWorld`.)

```
Compiled from HelloWorld.java
public class HelloWorld extends java.lang.Object {
       public HelloWorld();
       public static void main(java.lang.String[]);
```

Aquí aparecen dos métodos: `HelloWorld()` y `main()`. La línea `HelloWorld()` es lo que se conoce por el nombre de constructor por defecto, un método especial que permite la creación de la clase. A menos que indique lo contrario, se crea automaticamente un constructor para una clase. El método `main()` es el que creó anteriormente.

Cuando emplea la opción `-c` con javap:

```
javap -c HelloWorld
```

se muestra la secuencia de los códigos de byte subyacente, ademds de la anterior lista de métodos. La clase `HelloWorld` aparece inmediatamente debajo. Básicamente, estos comandos se equiparan con el código fuente del archivo .java.

```
Compiled from HelloWorld.java
public class HelloWorld extends java.lang.Object {
       public HelloWorld();
```

```
        public static void main(java.lang.String[]);
}

Method HelloWorld()
       0 aload_0
       1 invokespecial #1 <Method java.lang.Object()>
       4 return

Method void main(java.lang.String[])
       0 getstatic #2 <Field java.io.PrintStream out>
       3 ldc #3 <String "Hello World">
       5 invokevirtual #4 <Method void println(java.lang.String)>
       8 return
```

Cómo crear applets Java

Ahora que ya ha visto cómo crear una aplicación, es el momento de aprender a crear un applet. Las tareas son casi idénticas, aunque para un applet debe crear un archivo HTML para cargarlo. No puede simplemente usar el lanzador de Java. El applet que va a crear mostrard un mensaje pasado desde el archivo HTML o "Hello, World" si no se proporciona un mensaje.

Applet HelloWorld

El proceso para crear el código fuente para el applet es idéntico al de crear una aplicación. Abra un editor de texto. Para el applet, el archivo del editor será HelloWorldApplet.java.

> **Nota**
>
> **No es un requisito del lenguaje que los nombres de las clases applet terminen con "Applet".**

El código fuente para la clase HelloWorldApplet se encuentra en el listado 2.4. Parece muy diferente del de una aplicación y lo estudiaremos en breve.

Listado 2.4. Clase HelloWorldApplet

```
import java.awt.Graphics;
import java.applet.Applet;

/** HelloWorldApplet - Este es el primer applet Java que creé. */
public class HelloWorldApplet extends Applet {

    String msg;

/**
    Este es el método inicial invocado por el lanzador de applets.
  */
```

```
  public void init() {
      msg = getParameter("message");
      if (msg == null) {
        msg = "Hello, World";
      }


/**
 * Este método mostrará un mensaje en el navegador
 */

    public void paint(Graphics g) {
        g.drawString(msg, 20, 30);
    }
}
```

Lo primero que apreciará es que no hay un método `main()`. En su lugar, los applets tienen mdtodos de ciclos biologicos. Los métodos de ciclos biológicos son los mdtodos de un applet que no son invocados por usted; el navegador los invoca. Y son opcionales, de modo que si no necesita que suceda algo especial, no tiene que proporcionar el método. Los mdtodos de ciclos biológicos aparecen en la tabla 2.1.

Tabla 2.1. Applet de metodos de ciclos biologicos

Método	Descripción
`public void init()`	Método de inicializacion: Invocado cuando se carga por primera vez un applet.
`public void start()`	Mdtodo de inicio: Se invoca cuando un navegador esta preparado para ejecutar un applet; por ejemplo, tras la inicializacion o cuando un usuario regresa a una pagina que tenia cargado el applet.
`public void stop()`	Mdtodo de desactivacion: Se invoca un navegador ha terminado de ejecutar un applet, como cuando un usuario abandona una pagina web.
`public void destroy()`	Mdtodo de finalización: Se invoca cuando un navegador está a punto de descargar un applet de la memoria.
`public void paint(Graphics g)`	Método de dibujo: Se invoca cuando parte de la zona de visualización del applet queda invalidada.

En el método `init()` de HelloWorldApplet, se comprueba si se encuentra presente un parámetro de mensaje en el cargador HTML. Si se encuentra presente, la variable `msg` toma el valor del parámetro:

```
String msg;
. . .
msg = getParameter["message");
```

Si no está establecido el pardmetro, se usa un mensaje por defecto:

```
it (msg == null) {
   msg = "Hello, World";
```

El método `paint()` es incluso más simple. Lo único que hace es mostrar el mensaje en la ventana. Los valores 20 y 30 representan las coordenadas de la pantalla. (Encontrard mds información sobre cómo dibujar en el capítulo 10.)

```
public void paint(Graphics g) {
    g.drawString(msg, 20, 30);
 }
```

Las restantes líneas también merecen una pequeña explicación, ademds. Una línea que comienza con `import` indica al compilador dónde buscar una clase. HelloWorldApplet usa las clases `Graphics` y `Applet`:

```
impart java.awt.Graphics;
import java.applet.Applet;
```

Nota

La aplicacion HelloWorld usó las ~lasede sistema **`string`** y **`System`** y no requiere una línea **`import`**. Estas dos clases estan en una ubicación especial conocida por el compilador, de modo que no se necesita la línea `import`.

Todos los applets deben "extenderse" a desde la clase `Applet`. El entorno de ejecución de applets (el navegador) necesita esto.

```
public class HelloWorldApplet extends Applet {
```

Cómo compilar el código del applet

El compilador javac no se preocupa de si el archivo fuente es para un applet o una aplicación. Sólo tiene que pasar el nombre de archivo al compilador.

Sin embargo, para asegurar que la clase generada se ejecutará en un navegador, emplee la opción de línea de comandos `-target 1.1`, como se indica a continuación:

```
javac -target 1.1 HelloWorldApplet.java
```

Si el archivo no tiene errores, se generará HelloWorldApplet.class.

Por defecto, los archivos de clase Java se generan para 1.2 JRE. Tras la versión 1.1, se realizaron algunos cambios al formato de archivos .class para optimizarlo.

Cómo crear el cargador HTML

Para ejecutar un applet, no puede simplemente invocar al lanzador Java, asi:

```
java HelloWorldApplet
```

Si lo hace, obtendrh un mensaje de error, porque el applet no incluye un método `main()`:

```
Exception in thread "main" java.lang.NoSuchMethodError: main
```

Dado que los applets se ejecutan en un navegador, necesitarh un archivo HTML para cargar el applet. La etiqueta empleada para cargar un applet es <APPLET>, de modo que si necesita colocar la etiqueta en un archivo HTML. Cualquier nombre de archivo servirá como nombre del archivo HTML porque no se requiere clasificación del archivo .class para el archivo .html file ya que se encuentra entre .java y .class. Ponga la etiqueta <APPLET> en funcionamiento estableciendo el valor de tres atributos: CODE, WIDTH y HEIGHT. El valor CODE es el nombre del applet que se va a ejecutar (en este caso, HelloWorldApplet) y WIDTH y HEIGHT (ambos en píxeles) son el espacio que el navegador reserva para el applet. Por ejemplo, el siguiente codigo cargará el applet en una zona de 300x200 y mostrará el mensaje por defecto:

```
<APPLET CODE=HelloWorldApplet WIDTH=300 HEIGHT=200>
</APPLET>
```

Nota

La etiqueta APPLET tiene muchos mas atributos. Aprendera mas sobre ellos y los usara a medida que lea este libro.

Entre las etiquetas <APPLET> de apertura y de cierre se pueden colocar diferentes elementos. Puede colocar aqui pardmetros, para que pasen al applet. Los parámetros se especifican con la etiqueta <PARAM>. Tienen dos atributos: el nombre del parámetro (NAME) y su valor (VALUE). No hay etiqueta de cierre </PARAM>. En HelloWorldApplet, puede pasar un mensaje diferente estableciendo el atributo de mensaje:

```
<APPLET CODE=HelloWorldApplet WIDTH=300 HEIGHT=200>
<PARAM NAME=message VALUE="Help Me">
</APPLET>
```

Otro elemento que puede encontrarse entre las etiquetas APPLET de apertura y de cierre es una indicación de lo que quiere mostrar cuando Java no está habilitado en el navegador (o simplemente no está disponible). En el siguiente código, especificamos el mensaje "Java is disabled".

```
<APPLET CODE=HelloWorldApplet WIDTH=300 HEIGHT=200>
Java is disabled.
</APPLET>
```

Ejecutando el applet

Una vez que tiene un archivo HTML para cargar el applet, puede usar la herramienta appletviewer que acompaiia a SDK para cargar el documento HTML (o puede usar su navegador).

```
appletviewer Hello.html
```

Cuando el archivo HTML está cargado, verd el applet, como se muestra en la figura 2.5 para appletviewer y la figura 2.6 para Netscape Communicator.

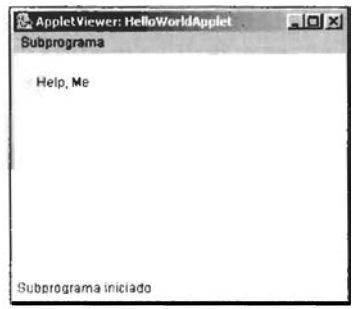


Figura 2.5. Corno usar appletviewer para rnostrar HelloWorldApplet con un rnensaje determinado

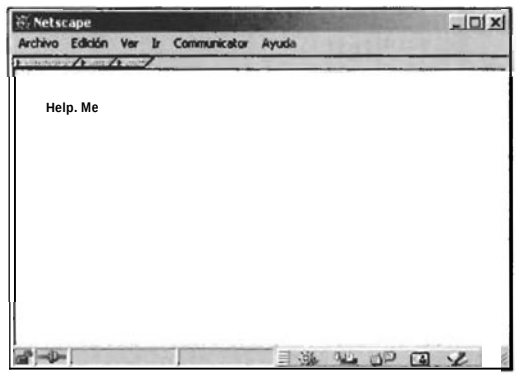


Figura 2.6. Corno usar Netscape Communicator para rnostrar HelloWorldApplet con un rnensaje deterrninado

Comparemos la figura 2.5 y la figura 2.6. Observe que hay algunas etiquetas HTML extra en el archivo cargado. Sin embargo, la herramienta appletviewer sólo comprende la etiqueta `<APPLET>`, de modo que appletviewer ignorard las demds etiquetas.

No obstante, los navegadores completos no ignorardn ninguna etiqueta válida. Ademds, si hay varias etiquetas `<APPLET>` en un sólo archivo HTML, appletviewer abrirá cada applet en su propia ventana, mientras que un navegador los mostrará todos en la misma página.

Cómo documentar el applet

Ejecutar javadoc en un applet no es diferente de hacerlo en una aplicación (simplemente pase el archivo fuente a la herramienta). Si pasa varios archivos fuente, los archivos comunes generados incluirán referencias a todos los archivos. Aqui tiene un ejemplo:

```
javadoc HelloWorld.java HelloWorldApplet.java
```

Ahora, si visualiza el archivo index.html, no solamente podrá ver los comentarios generados en HTML para cada clase individual, pero tambien tendrá un índice de todas las clases disponibles.

Observe la figura 2.7. Esto es más útil a medida que aumenta el número de clases y hay demasiadas interconexiones.

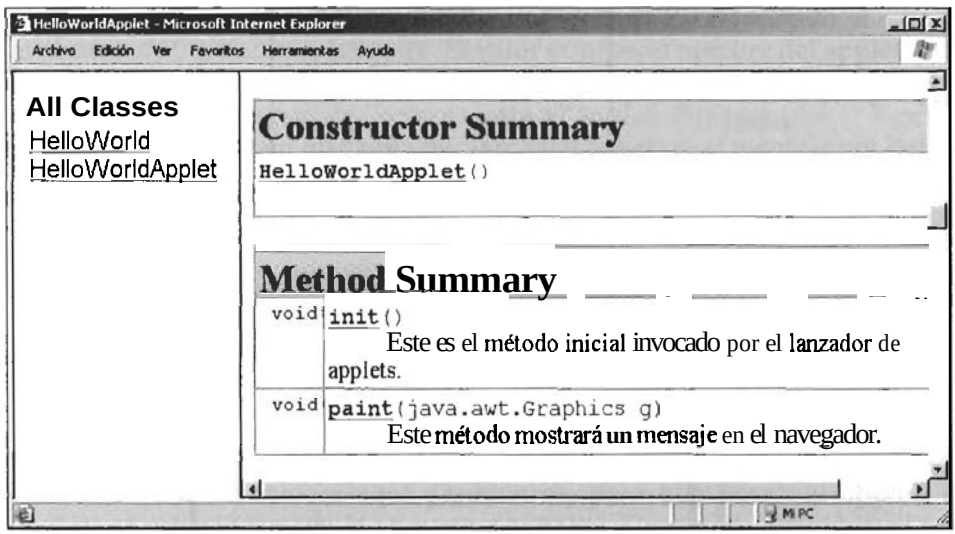


Figura *2.7.* Vista del archivo HTML generado por javadoc para el codigo HelloWorldApplet.java, con archivo de índice combinado

Cómo depurar su applet

Se pueden depurar applets con la ayuda del depurador jdb, al igual que se hacía con las aplicaciones. Para iniciar el depurador con un applet, necesitará pasar el indicador –debug a la linea de comandos de appletviewer:

```
appletviewer -debug Foo.html
```

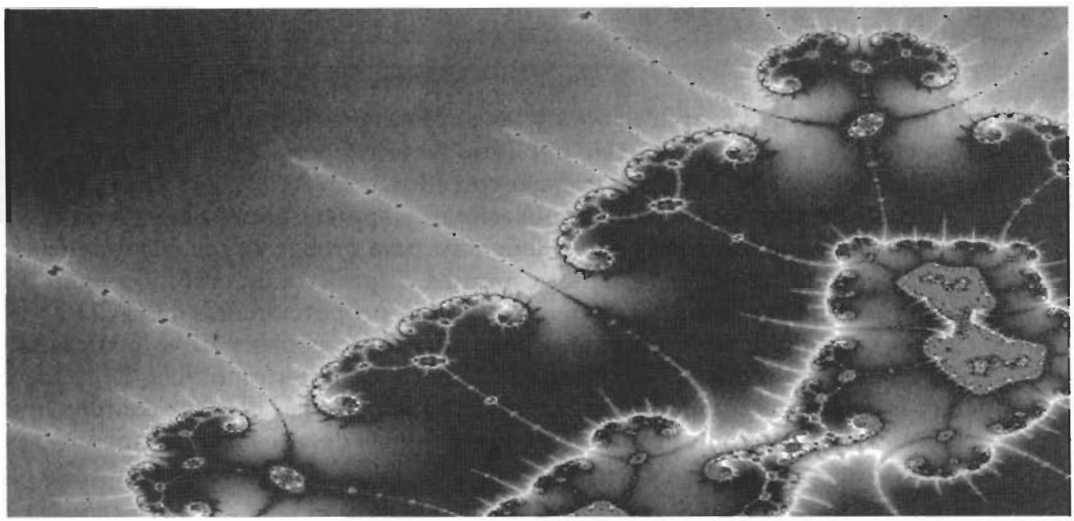
Aqui hemos de repetirnos, como se mencionó para las aplicaciones: Si necesita depurar de verdad su programa, mejor use un auténtico depurador en una IDE.

Resumen

Los aspectos de las aplicaciones y las applets descritos en este capitulo le ayudan a crear las bases para el resto del libro. Ahora deberia tener una ligera idea de sus diferencias y similitudes al trabajar con ellos.

Además de aprender sobre las aplicaciones y applets, ahora tiene un entorno de programacidn que funciona y que puede usar para probar y practicar con los ejemplos que encontrari en los restantes capitulos. Si prefiere un entorno integrado de programación, pruebe las herramientas JBuilder proporcionadas en el CD.

En el capitulo 3, se le preparará para aprender las bases de la programacidn orientada a objetos. Ya le hemos mostrado parte de la terminologia, de modo que palabras como clase, método y constructor no le deberian resultar completamente desconocidas. Trataremos estos elementos con más detalles en el siguiente capitulo, a partir de la base que ya tenemos.



3
Cómo trabajar con objetos

Programar con el lenguaje de programacidn de Java no es sólo sentarse y escribir cddigo. A menudo deberd apartarse de las tareas de escritura de cddigo para preocuparse de otros asuntos importantes. Deberá mirar con más atención el diseño del programa, asegurando su solidez y revisando las zonas de posible reutilizacidn de software.

El arquetipo asociado mds comunmente usado hoy en dia recibe el nombre de programacidn orientada a objetos (POO).

Este capitulo es una introducción a la terminologia y a los conceptos tras POO y cómo hacer funcionar estos conceptos en el diseño de su programa. Para ayudarle a identificar los conceptos, encontrará diagramas dibujados usando la notación de Lenguaje para modelos unificado (UML). Y, para mantener la continuidad con el cddigo de ejemplo que aparece en este capitulo, estudiaremos estos conceptos tal y como se aplican al diseño de un sistema escolar, con estudiantes, profesores y cursos como algunas de sus partes.

Nota

El material que aparece aqui debe presentarle los conceptos de programacion orientada a objetos. Si ya esta familiarizado con los conceptos de la POO, puede omitir este capitulo (aunque quizás quiera ver los ejemplos, que usan el lenguaje de programacion Java). Además, si la notación UML no le resulta conocida, quizás quiera seguir los diagramas UML.

¿Qué es UML?

Antesde empezar a describir las clases, observemos masde cerca el Lenguaje de modelado unificado. UML es una notacion estandar de notacion para el modelado de objetos del mundo real cuando se desarrolla un diseño orientado a objetos. Los expertos en el modelado de objetos Grady Booch, Jim Rumbaugh e Ivar Jacobson combinaron sus tres procesos de modelado distintos para crear un estandar común. En noviembre de 1997, el Grupo de gestion de objetos (http://www.omg.org) dio la bendicion a la version 1.1 de UML y UML se convirtio en el lenguaje de modelado orientado a objetos estandar. En otoño del 2001, la ultima version de UML era 1.4.

Este capitulo describe cómo leer los diagramas especificos de UML que se usan en este libro. Si lo desea, incluso puede descargar todas las especificaciones de UML de http://www.omg.org/technology/documents/formal/uml.htm.

Al igual que la mayoría de las tecnicas de modelado, UML proporciona una representación visual de una arquitectura de sistema. Usada basicamente durante las fases de análisis y diseño, los ingenieros utilizan UML para capturar y realizar anotaciones a los diagramas de flujo de modo que los programas puedan implementarse, probarse y usarse.

UML puede sonar como algo complicado, pero en realidad no es tan dificil de leer. Y, en realidad, la lectura de los diagramas UML es todo lo que necesita saber sobre UML para avanzar por este libro.

Introducción a la POO

La programación orientada a objetos es simplemente una manera de pensar sobre el sistema que se va a desarrollar. En lugar de pensar en los datos y la lógica de programación de forma separada, que es el enfoque de la programación estructurada, trata de dividir los programas en sus componentes y concentrarse en la interacción entre esas partes.

Estas partes reciben el nombre de objetos e incluyen los datos y lógica de programación necesarios para interactuar con los otros componentes del programa. Estos datos y lógica de programación son m6s conocidos como el estado y comportamiento del programa. La figura 3.1 muestra cómo interactuaria un sistema orientado a objetos.

Desde objetos hasta clases

La premisa básica en la programación orientada a objetos es que todos los elementos en el programa creado es un objeto. Lo que en realidad son estos objetos depende del programa que se está desarrollando. Por ejemplo, en el caso de un

sistema escolar, los objetos más probables serian estudiante, profesor, curso, horario, clase, escuela y todo lo necesario para que el sistema funcione.

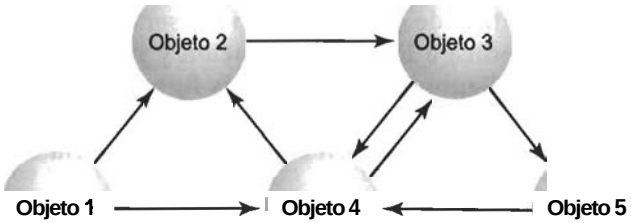


Figura 3.1. Interacciones dentro de un sisterna orientado a objetos

En Java, la representacidn de estos objetos de programa es la clase. La clase contiene el prototipo para la creacidn del objeto. Dentro de este prototipo, el creador de la clase incluye las estructuras de datos y los algoritmos necesarios para manipular la clase. Todo permanece dentro de la clase.

Cuando necesita crear un objeto dentro de su programa, crea una instancia de una clase. Esa instancia almacena informacidn especifica de estado sobre el objeto que se encuentra en las estructuras de datos precisadas en la definición de la clase.

Campos y métodos

Cuando se modelan objetos reales, se necesita describir las estructuras de datos y los algoritmos asociados a una clase. En Java, la estructura de datos de una clase es la clase campos y los algoritmos de la clase reciben el nombre de métodos. La mitad de la dificultad de aprender un nuevo lenguaje de programacidn es aprender su terminologia. Por ejemplo, puede haber usado anteriormente el término atributo para los campos y posiblemente funciones o procedimientos en lugar de métodos. Desde el punto de vista de la prograrnacidn de alto nivel, estos términos quieren decir lo mismo, es decir, campo, atributo y estructura de datos son lo mismo y método, funcidn y procedimiento también son lo mismo.

Para mostrarlo, imagine que cada estudiante en su sistema escolar tiene un nombre y un número identificativo. De modo que debe agregar un campo a su definicidn de clase para almacenar cada fragmento de datos. La figura 3.2 muestra el aspecto de la clase desde una perspectiva de modelado UML.

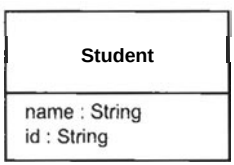


Figura 3.2. El modelo Student con atributos

El diagrama de clase de la figura **3.2** tiene dos partes. La parte superior del cuadro tiene el nombre de la clase `Student`. La parte inferior del cuadro tiene los atributos, nombre e identificación de la clase. El String junto a cada atributo representa su tipo de datos o el modo en el que se almacenan los datos. (Dependiendo de la fase del modelado, el tipo de datos puede o no, ser conocido y se puede dejar en blanco.) Ahora, si se dedica a la implementación y no al diseño de un proyecto, alguien podría darle el diagrama de clase de la figura **3.2** y sabria exactamente lo que se necesita para implementar la clase. Aunque aun no haya estudiado la sintaxis del lenguaje Java (lo estudiará en el capítulo 4), podría entender la siguiente definición de clase:

```
class Student {
    String name;
    String id;
}
```

Nada demasiado interesante todavia, pero esto define una clase llamada `Student` con dos atributos, nombre e identificación de datos, justo como indica el diagrama.

Cuando se diseñan clases, no sólo se describen los atributos de clase sino que también se le da a la clase algunas responsabilidades. Las responsabilidades representan los comportamientos que quiere que tenga la clase, para describir las interacciones con otros objetos o simplemente con la clase misma.

Considere el caso de un estudiante en un entorno escolar; una tarea necesaria es matricularse en un curso. Cuando agrega responsabilidades a nuestro diagrama de clase, agrega una tercera zona al diagrama, para operaciones. En la figura **3.3,** hemos agregado una operación de registro que realizarán los estudiantes.

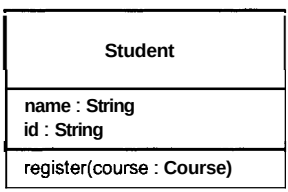


Figura 3.3. El modelo Student con atributos y una operación

Regresando de nuevo al código, sabe que necesita agregar un método `register()` a la definición de clase. En este punto, no hay suficiente información para definir todo el método, pero tiene el marco básico de la definición de la clase, como se indica a continuación:

```
class Student {
    String name;
    String id;

    void register(Course course) {

    }
}
```

el contexto escolar), los clientes y los escolares comparten el nombre e identificacibn de una persona. En el mundo UML, cuando conecta dos o más cuadros con una linea continua que tiene una flecha vacia en un extremo, está indicando esta relacibn generalización/herencia. La figura 3.5 muestra estas relaciones, incluyendo los atributos y operaciones de todas las clases involucradas.

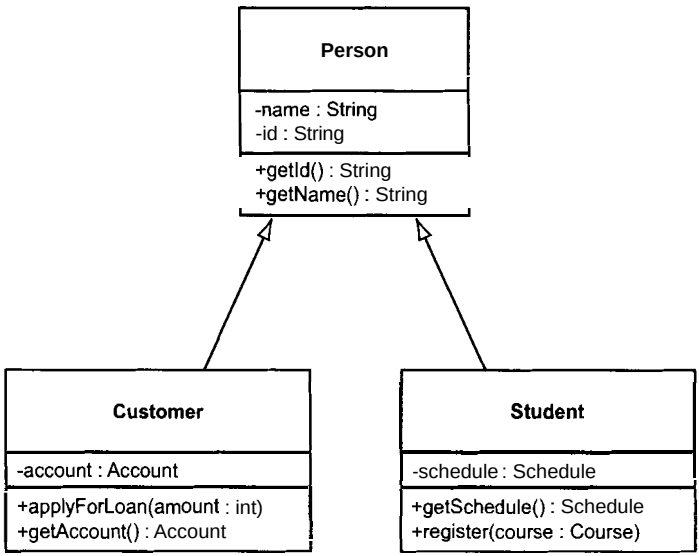


Figura 3.5. Las relaciones de herencia de Person

Como las flechas se dirigen al interior de la clase `Person`, en la figura *3.5*, `Person` es la superclase. `Customer` y `Student` son subclases. ¿Qué conseguimos con la herencia? Aunque el nombre y la identificación son atributos de `Person`, tanto `Customer` como `Student` poseen estos atributos mediante la herencia. Las subclases deberian ser colocadas debajo de su superclase en diagramas de relación UML, como aparece en la figura *3.5*. Esto incrementa el factor de entendimiento de cualquier persona que mire rápidamente el diagrama.

Ahora, volviendo de UML a Java de nuevo, las estructuras básicas de las clases son como se indica a continuacibn:

```
class Person {

}

class Customer extends Person {
  . . .
}

class Student extends Person {
  . . .
}
```

Asi, la figura **3.3** muestra que será necesario definir un objeto. A medida que programa un sistema y descubre los estados y comportamientos necesarios, conocera nuevos objetos necesarios para que su modelo de sistema esté completo.

Como diseñar clases

La programación orientada a objetos añade mucha terminologia nueva a su vocabulario. Tres nuevas palabras son abstracción, encapsulación y ocultación de información. En esta sección también estudiaremos generalización, abstracción, asociación y agregación.

Abstracción

Cuando define clases en un sistema, modela las caracteristicas de un objeto especifico del sistema en cuestión. El sistema escuela de nuestro ejemplo sólo requiere que un estudiante tenga un nombre y una identificación, aunque el mismo estudiante en un sistema bancario podría necesitar que le proporcionara mas información (cantidad prestada, condiciones de pago e interés, por ejemplo). Tanto en el sistema bancario como en el escolar, sigue tratando con estudiantes, pero cada zona en la que se emplea un estudiante requiere una abstracción diferente para ese entorno especifico.

Una abstracción en este contexto simplemente es el modo en que se considera un objeto para un entorno especifico.

Cada modelo en el que se emplea un objeto requiere una abstracción especial de ese objeto. En el mundo bancario, el banco no se preocupa por el horario estudiantil. En el mundo escolar, la escuela no se preocupa por las condiciones de pago, mientras le paguen.

Cuando se crean objetos, es importante diseñarlos con el conjunto de atributos especificos y operaciones necesarios para el sistema en cuestión. No intente agregar posibles atributos y operaciones futuros por anticipado. Los agregará más tarde siempre y cuando reutilice un objeto más tarde, usando los conceptos de encapsulación y ocultación de información.

Encapsulación

Donde la abstracción solicita programar sus objetos para la tarea especifica en cuestión, la encapsulación solicita dividir en compartimentos los objetos. Es decir, mantener todo junto. En los programas orientados a objetos, los objetos contienen su estado y su comportamiento. Al ocultar los detalles de implementación del comportamiento dentro de una clase, ha encapsulado el objeto. La adecuada encapsulación permite la implementación de cambios sin afectar a otros componentes del programa.

Ocultacion de informacion

No solamente debe ocultar los detalles de implementación sino que también debe restringir el acceso a la información del estado de un objeto. Esta restricción al acceso a atributos (y posiblemente, algunas de las operaciones del objeto) recibe el nombre de ocultación de información.

La ocultación de información permite al programador cambiar los detalles subyacentes de una definición de clase sin afectar al resto del programa. Por ejemplo, si alguien quiere el numero de identificación de un estudiante, ¿debería mirar en la cartera del estudiante y coger su carné para conseguir su identificación o deberia ser capaz de preguntarlo? Esconder la cartera que contiene la identificación para evitar que la roben es el equivalente a ocultar información. Esto hace que el unico modo de obtener la informacion sea preguntando.

En notación UML, los atributos y las operaciones en un diagrama de clase pueden tener indicadores de visibilidad. Un signo menos en nuestros diagramas de clase indica que el elemento sólo seria visible desde dentro de la clase, mientras que un signo mas designa la visibilidad desde fuera de la clase. Si agrega los métodos necesarios para acceder a los atributos, obtendrá la figura **3.4.** Entonces, en el momento de la programación, un programador tomaría el diagrama UML y agregaria los modificadores de acceso adecuados equivalentes a los indicadores de visibilidad. Aprenderá más sobre los modificadores de acceso en el capitulo 4.

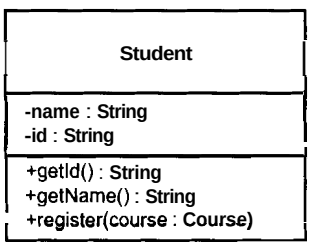


Figura 3.4. El modelo Student con indicadores de visibilidad

Generalización

La relación entre un objeto general y una versión más especifica de 61 recibe el nombre de generalización. La clase general es conocida como superclase. A la versión especifica se la llama subclase. En el entorno de programación orientada a objetos, esto también recibe el nombre de herencia. La subclase hereda los atributos y operaciones de la superclase.

En nuestros mundos escolar y bancario, el banco en realidad no quiere trabajar con un estudiante, propiamente dicho, sino con un cliente. Los clientes y los estudiantes comparten algunos atributos y operaciones. Todo el mundo tiene nombres y, hoy en dia, un numero identificativo de algún tipo. Aunque los clientes no tienen horarios (en el contexto del banco) y los estudiantes no tienen cuentas bancarias (en

En Java la palabra clave `extends` se usa para implementar herencia. Entonces, posteriormente en su programa, cualquier método que necesite un `Person` para realizar una tarea puede trabajar con un `Customer` o un `Student`.

Cuando estd programando clases, las relaciones de herencia suelen ser obvias y fáciles de identificar. Si observa similitudes de atributos u operaciones entre tareas, tendrd una oportunidad de crear una relación de herencia.

Abstracción, segunda toma

Las clases mostradas hasta ahora han sido lo que se llama *clases concretas.* Cada clase ha sido completa, con todo su estado y comportamiento contenido dentro de la clase. Lo contrario de una clase concreta es una *clase abstracfa.* En lugar de definir todo el comportamiento dentro de si misma, una clase abstracta describe la estructura de ese comportamiento, pero deja la autentica implementación para las subclases.

En los mundos de la banca y la escuela, el cliente y el estudiante deben pagar facturas. En el banco, la factura es el pago de un préstamo. En la escuela, la factura es la matricula. Para generalizar la situación, puede agregar un mktodo `payBill()` al objeto `Person`, sin ningun comportamiento, creando un método abstracto. A continuación, en cada subclase de `Person`, la subclase debe definir el comportamiento especifico para el mktodo abstracto. A continuación, en tiempo de ejecución, el tipo de `Person` determina especificamente cómo debe pagarse esa factura. La figura 3.6 ejemplifica las relaciones de abstracción.

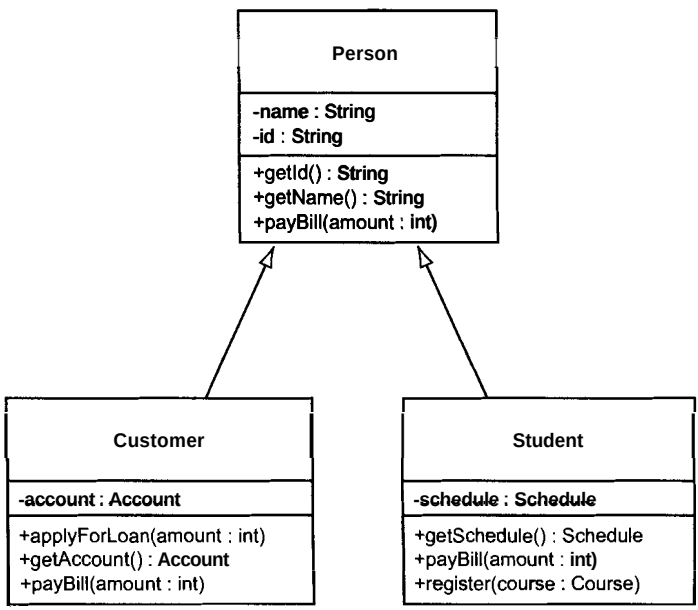


Figura 3.6. Las relaciones abstractas de Person

Al examinar la figura **3.6** aparecen dos cambios en el modelo que habiamos seguido hasta ahora. En primer lugar, el método `payBill()` se agrego a las tres clases. Sin embargo, el segundo cambio es el más importante. Para mostrar algo en el diagrama como abstracto, el nombre debe aparecer en cursiva en lugar de con un tipo de letra normal. Observe en la figura **3.6** que, el método `payBill()` de `Person` no es lo unico que aparece en cursiva, sino que también lo está el nombre de la clase. Cada vez que una clase contiene un método abstracto, automáticamente, la clase tambien se considera abstracta.

La relación abstracta significa que, en cualquier lugar del programa en el que se espera que aparezca `Person`, se invocará el método `payBill()` de la subclase especifica. Por ejemplo, suponga que hay un método `receiveBirthdayMoney()` en alguna parte que recibe pardmetros de Person, que a su vez recibe el dinero y la cantidad real. Puede invocar al método `payBill()` de esa persona y aunque no conozca qué tipo de persona recibird el dinero del cumpleaiios, se invocard el método `payBill()` adecuado.

Aqui tiene el aspecto que podría tener `receiveBirthdayMoney()`:

```
receiveBirthdayMoney(Person p, int amount) {
    p.payBill(amount);
```

Asociación

En las ilustraciones que hemos visto hasta ahora, las unicas relaciones que aparecian eran las de jerarquia de clase para la clase `Person`. No obstante, aqui hay otras relaciones. Cuando dos clases interactuan, se dice que tienen una relación de asociación. Por ejemplo, nuestra clase `Custorner` tiene una `Account`. El estudiante se matricula en un `Course` y tiene un `Schedule`.

Hacer anotaciones a estas relaciones nos lleva de nuevo a la notación UML. Las relaciones de asociación se dibujan con una linea continua que conecta las clases relacionadas. Se puede usar una punta de flecha abierta para mostrar la dirección de la relación y la linea puede estar identificada para dar nombre a la asociacion, pero ninguna de las dos cosas es indispensable.

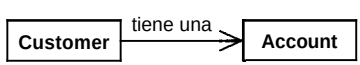


Figura 3.7. La relación de asociacion de un Customer

Además del rotulo, también se puede agregar orientación a uno o ambos extremos de la linea. Por ejemplo, un estudiante puede recibir una o más clases. Y una clase no necesita tener alumnos a una determinada hora; en otras palabras, puede existir sin estudiantes.

De este modo, la cardinalidad de `Student` a `Course` serd `1..*`, que significa 1 o más y la relación de `course` a `Student` seria `0..*`, que significa 0 o más. Esto se puede ver en la figura **3.8.**

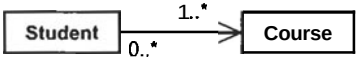


Figura 3.8. Las relaciones de asociacion de un Student, con indicaciones de numeración

Hay muchos indicadores de cardinalidad diferentes disponibles para cuantificar las relaciones en un diagrama UML. La tabla 3.1 enumera varios de los más comunes. Por ejemplo, si sabe que debe haber al menos tres relaciones entre dos clases, use el rango 3..n en la línea de relación.

Tabla 3.1. lndicadores de cardinalidad

Cantidad	Descripción
0..1	0 o 1
1	1
0..*	0 o mas
1..*	I o mas
n	Cualquier número positivo
0..n	De 0 a n
1..n	De1an

Cuando llega el momento de codificar sus relaciones de asociación, la cardinalidad determina cómo lo hará. En el caso de un número de relaciones pequeño o fijo normalmente agregará campos a la definición de clase para cada relación. Cuando la cantidad hace referencia a una cantidad de conexiones desconocida o importante, podría necesitar romper la relación y conservarla en su propia clase. Por ejemplo, la figura 3.5 muestra que un estudiante tiene un horario, no diferentes cursos. El horario debe mantener las relaciones con diferentes cursos, mientras que el estudiante sólo necesita una referencia al horario. Y aquí tiene el aspecto del código Java para esa disposición:

```
class Student {
        Schedule schedule;

}
```

Agregación

Cómo se acaba de describir, el ejemplo de un horario compuesto de cursos es una relación de agregación. La agregación asocia todo un objeto a sus partes, donde las partes son multiples del mismo tipo. La agregación sólo es una forma especiali-

zada de una relación de asociación. Debido a la frecuencia de la relación de agregación, la notación UML proporciona su propia sintaxis. Se dibuja un diamante en lugar de, o además de, la cardinalidad del lado "completo" de la relación, como muestra la figura 3.9.

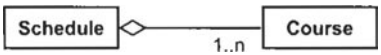


Figura 3.9. La relación de agregacion de una Schedule a Course

Nota

En la agregacion de modelado, el simbolo del diamante puede ser continuo o abierto. Cuando el diamante es continuo, el unico modo de acceder a las partes es a traves del todo y las partes no existen de forma independiente fuera del todo.

La codificación de relaciones agregadas en Java requiere trabajar con matrices o clases especiales que almacenan colecciones de elementos. Estas funciones se describira' posteriormente en los siguientes capitulos: las matrices en el capitulo 6 y las colecciones en el capitulo 17.

Resumen

Cada lenguaje de programación orientado a objetos admite las funciones descritas en este capitulo. Los conceptos generales no sera'n muy novedosos para ninguno de ustedes, pero para algunos, el uso de UML puede no ser muy familiar. Comprender la sintaxis para la creación de clases es la parte sencilla y se describe en el capitulo 5. Sin embargo, comprender los principios de la programación orientada a objetos requiere tiempo y pra'ctica.

Si la notación del Lenguaje para modelos unificado es algo nuevo para usted, hay muchos recursos diferentes para ayudarle a comenzar.

Si le gusta seguir un camino basado en las herramientas, Mastering UML with Rational Rose 2001 es un buen lugar para empezar; Rational Rose es una herramienta de modelado muy popular

Nota

¿Le interesa probar Rational Rose? Descargue una version de prueba desde http://www,rational.com/tryit/rose/. Otras herramientas populares disponibles son Together (acuda a http://www.togethersoft.com) y la herramienta de codigo abierto ArgoUML (vaya a http://argouml.tigris.org/).

La edición Enterprise de JBuilder 6 ahora viene acompaiiada por su propia herramienta para visualizar código UML. Puede probarla durante 30 dias con la versión de prueba que aparece en el CD de este libro. Por supuesto, con JBuilder no aprenderd el significado de los diagramas; simplemente deja a la herramienta que los cree por usted.

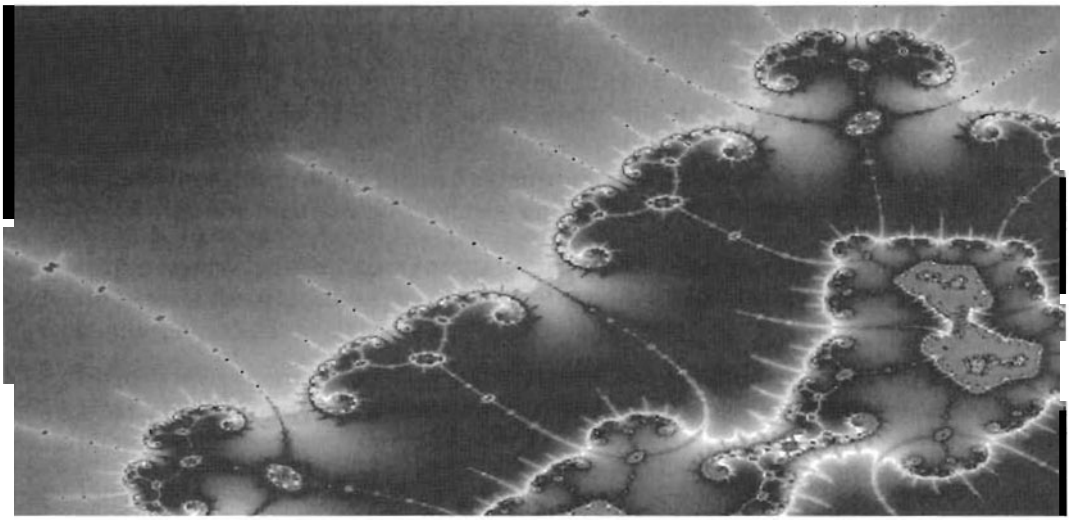
Para los que estén interesados en adentrarse aún mds en el trabajo con UML, aqui tienen algunos recursos adicionales, en Internet y fuera de él, para ayudarle:

http://www.celigent.com/omg/umlrtf/tutorials.htm ofrece transparencias de la serie de tutoriales Object Modeling with OMG UML, presentadas en las reuniones OMG de 2001.

http://www.cetus-links.org/oo_uml.html es la página de Cetus Links sobre UML. Encontrard una amplia colección de vinculos a tutoriales y articulos sobre este tema.

Fuera de Internet se han publicado muchos libros sobre UML, algunos de los creadores de UML. UML Explained por Kendall Scott (2001) es un buen principio. Ni siquiera debe ser un programador para comprender este texto. Otros dos recursos utiles son The Unified Modeling Language User Guide por Grady Booch, Ivar Jacobson y James Rumbaugh (1998) y UML Distilled por Martin Fowler y Kendall Scott (1999).

No crea que debe conseguir estos recursos ahora mismo; no son necesarios para su trabajo con el resto del libro. No obstante, es recomendable conocerlos si decide aprender mds sobre UML.



4

Tipos de datos, modificadores y expresiones

La plataforma Java está compuesta del lenguaje de programación y las bibliotecas. En este capitulo conocerá la sintaxis del lenguaje Java (dejaremos las bibliotecas para más adelante). La sintaxis es bastante similar a la de C, C++ y C# y hasta cierto punto, a la de Visual Basic. Las mismas piezas bdsicas están ahi, con pocas diferencias. La cantidad de diferencia que percibird dependerá del lenguaje del que proceda, si conocia alguno con anterioridad. Básicamente, este capitulo se ocupa de la fundación de la gramdtica de Java, la sintaxis usada para describir los elementos que constituyen programas de Java validos. Para aprender la sintaxis del lenguaje de programacidn Java, lo mejor es comenzar con lo básico y avanzar a partir de ahi. Con esto en mente, estudiaremos lo siguiente en este capitulo: Comentarios, palabras clave, variables, tipos de datos, literales, instrucciones y expresiones, operadores y modificadores. En los siguientes tres capitulos se describirdn otras partes de la gramática. Alli encontrara información sobre definición de clases, matrices, instrucciones de control de flujo y control de excepciones.

Nota

Si no esta interesado en la gramatica formal del lenguaje Java, la encontrara documentada en http://java.sun.com/docs/books/jls/second_edition/html/syntax.doc.html. Sin embargo, este documento que no ha sido actualizado para Java **1.4;** no incluye material sobre aserciones, y se presentaran en el capitulo 7 (junto con el control de excepciones).

Comentarios

Los comentarios son la información proporcionada por el programador para documentar el código fuente. Los programadores deberian agregar comentarios a su cddigo fuente Java de modo que cualquiera sepa la función para la que se program6 una clase, método, variable o bloque de código fuente, cuando se vuelva sobre ese elemento por alguna razón. Puede usar uno de tres tipos de comentario, cada uno de ellos con su propio propósito.

`// comentario` Todo el texto tras // y hasta el final de la línea se considera un comentario.

`/* comentario */` Todo el texto entre los caracteres /* y */ son el comentario. El comentario puede ocupar varias líneas.

Advertencia

Puede anidar comentarios `/* */`. En ese caso, el primer `*/` finalizara ambos comentarios, lo que hara que el resto del comentario que quede fuera sea tratado como codigo por el compilador.

`/** comentario */` Este es un comentario especial llamado comentario javadoc, llamado asi por la herramienta SDK. Funciona como un comentario `/**/`. Los comentarios javadoc se crean para clases, metodos y variables de nivel de clase. Entonces puede ejecutar la herramienta javadoc para generar un conjunto de páginas HTML que documentan el conjunto de clases utilizadas. La tabla 4.1 enumera las @etiquetas especiales que puede colocar en los comentarios, para usarlas en la herramienta javadoc cuando genera las páginas HTML.

Tabla 4.1. @etiquetas estandar de javadoc

Formato	Descripción
`@author`nombre	Usada para especificar el nombre del creador del elemento. Ejemplo `@author John Zukowski`
`@deprecated` texto	Usada para especificar que un elemento ya no se usa. El texto describe lo que se debe hacer, si es posible. Ejemplo: `@deprecated`Como el de la versión 2.3, `use {@link #getFoo() getFoo()} en su lugar`
`@exception`texto de la excepcion `@throws` texto de la excepcion	Usado para especificar que un método arroja una excepcion. Ejemplo:`@throws IllegalArgument-Exception` cuando el tipo no es válido

Formato	Descripción
{@link nornbre de la etiqueta)	Utilizada para insertar un vínculo a otra pagina javadoc. Disefiado para estar "en linea" con otra descripcion, Ejernplo: {@link #getFoo() getFoo()}
@param nornbre del texto	Usada para describir un pararnetro de rnetodo. Ejernplo: @param escribe el nuevo tipo de foo
@returntexto	Usada para describir el valor devuelto. Ejernplo: @returntipo foo actual
@see referencia	Utilizada para crear una referencia cruzada con otro paquete, clase, rnetodo u otro docurnento. Ejernplo: @see Bar#FOO_TYPE @see Bar#getFoo() @see #getFoo() @see "My Book" @see jGuru
@sincetexto	Usada para especificar el rnornento en que se afiadio algo. Ejernplo: @since V2.3
@serialtexto @serialField texto con nornbre de tipo @serialData texto	Usada para docurnentar la naturaleza serializable de una clase. Ejernplo: @serialField Foo int Tipo actual
@versiontexto	Usada para docurnentar la version de un elernento Ejernplo: @version 2.4

Not.

Para más informacion sobre cómo trabajar con comentarios javadoc, puede obtener mas informacion en:
http://java.sun.com/j2se/javadoc/writingdoccomments/.

Un ejemplo de comentarios javadoc

El listado 4.1 muestra los comentarios de una clase particular (Bar), con comentarios javadoc creados en los sitios adecuados. Aún no debe preocuparse sobre otra sintaxis aparte de la de los comentarios aqui estudiados. Aprenderá más a lo largo del resto del libro.

Listado 4.1. La clase Bar

```
/**
 * El componente <code>Bar</code> no hace demasiado
 * Sólo está para enseñarle javadoc
 * <p>
 * eversion 2.4
 * @authorJohn Zukowski
 */
public class Bar {
 /**
  * Un tipo de foo especlfico
  */
 public static final int FOO_TYPE = 0;
 /**
  * Otro tipo de foo especifico
  */
 public static final int ANOTHER—FOO—TYPE = 1;
 /**
  * El tipo de foo actual
  */
 private int type;
 /**
  * Crea una Bar
  */
 public Bar() {
     type = FOO_TYPE;
 }
 /**
  * Devuelve el tipo de foo actual
  *
  * @returntipo de foo actual
  * @deprecatedComo la versión 2.3, use {@link #getFoo() getFoo()} en su
  * lugar
  */
 public int foo() {
     return getFoo();
 }
 /**
  * Devuelve el tipo de foo actual
  *
  * @returntipo de foo actual
  * @since V2.3
  */
 public int getFoo() {
     return getFoo();
 }
 /**
  * Cambia el tipo de foo actual

  * @param escribe el nuevo tipo de foo
  * @throws IllegalArgumentException Cuando el tipo no es válido
  */
 public void setFoo(int newValue) {
     if ((newValue != FOO_TYPE) &&
```

```
        (newValue != ANOTHER_FOO_TYPE)) {
    throw new IllegalArgumentException("Bad type");

 type = newValue;
}
/**
 * Método marcador de posición para ver etiquetas
 * @see Bar#FOO_TYPE
 * @see Bar#getFoo()
 * @see #getFoo()
 * @see "My Book"
 * @see <a href="http://www.jguru.com>jGuru</a>
 */
public void myVoid() {
}
```

Nota

Observara que se ignoran los * al comienzo de cada línea de comentario javadoc. Como muestra la figura 4.1, los * no se agregan al resultado generado. Solo se usan para dar una mejor apariencia.

Con los comentarios colocados en su sitio, puede ejecutar la herramienta javadoc sobre el archivo fuente mediante el siguiente comando:

```
javadoc Bar.java
```

Esto generará la documentación HTML para la clase, como se indica a continuación:

```
Loading source file Bar.java...
Constructing Javadoc information...
Building tree for all the packages and classes.
Building index for all the packages and classes
Generating overview-tree.html...
Generating index-all.html...
Generating deprecated-list.html...
Building index for all classes...
Generating allclasses-frame.html...
Generating index.html...
Generating packages.html...
Generating Bar.html...
Generating package-list...
Generating help-doc.html...
Generating stylesheet.css...
```

Se generará un archivo HTML para cada clase, junto con varios archivos de índice que, en realidad, no son muy litiles, excepto cuando se generan mliltiples clases juntas. Abra el archivo Bar.html, por ejemplo, y obtendrá los comentarios al estilo javadoc de ese archivo HTML. La figura 4.1 muestra una pequeña parte del archivo generado.

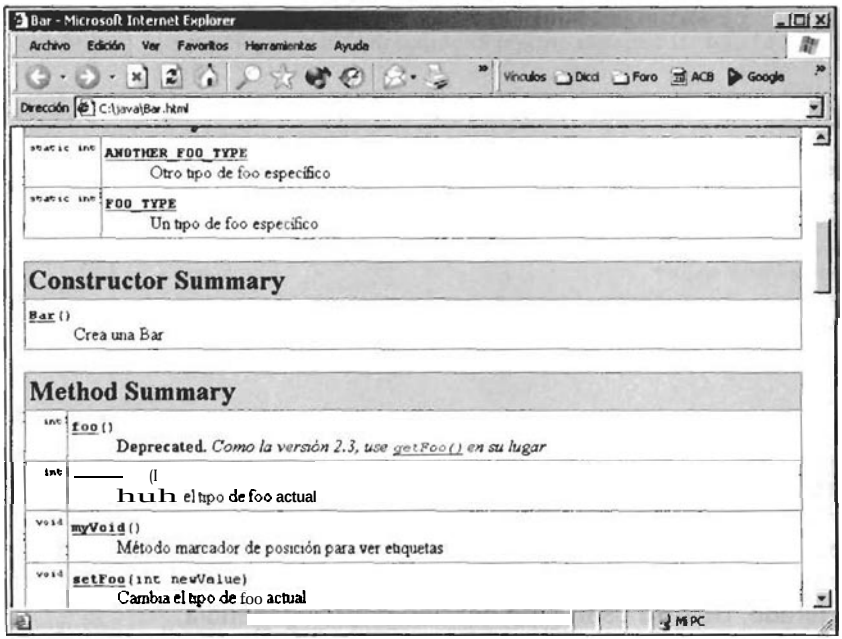


Figura 4.1. Ejemplo de documentación de clase generada mediante javadoc

Palabras clave

Antes de agregar comentarios a sus archivos fuente, probablemente use una gran cantidad de palabras clave (palabras especiales para el compilador de Java que indican al sistema lo que debe hacer). La lista completa de palabras clave de Java se encuentra en la tabla 4.2 y conocerá m6s sobre ellas a medida que avanza en el libro.

Tabla 4.2. Palabras clave de Java

`abstract`	`boolean`	`break`	`byte`
`case`	`catch`	`char`	`class`
`const*`	`continue`	`default`	`do`
`double`	`else`	`extends`	`final`
`finally`	`float`	`for`	`goto*`
`if`	`implements`	`import`	`instanceof`
`int`	`interface`	`long`	`native`
`new`	`package`	`private`	`protected`

`public`	`return`	`short`	`static`
`strictfp`	`super`	`switch`	`synchronized`
`this`	`throw`	`throws`	`transient`
`try`	`void`	`volatile`	`while`

* Termino reservado, aunque no se utiliza en Java.

Variables

Una variable es un comodin usado para almacenar información durante la ejecución del programa. El nombre dado a una variable especifica es su identificador. Hay restricciones a los caracteres que pueden formar un identificador, pero no hay límite a su longitud. Las reglas básicas son las siguientes: El primer carácter de un identificador debe ser una letra y se pueden usar letras o números para las demas posiciones. Estas reglas parecen sencillas, pero en realidad son un poco más complejas de lo que se pudiera pensar. En primer lugar, ¿qué es una letra? Las letras de Java son las típicas A-Z y a-z. Pero también incluyen, por motivos históricos, los caracteres $ y _. Y eso no es todo. Java comprende el conjunto de caracteres Unicode, de modo que puede agregar estos caracteres al conjunto de caracteres básicos ASCII. Sólo nos queda describir los numeros; éstos pueden ser cualquiera de los digitos numéricos, de 0 a 9.

Nota

El estandar Unicode describe todos los caracteres posibles en un solo mapa de 16 bits para la mayoría de la comunidad internacional. Encontrara mas inforrnacion sobre Unicode en Unicode Consortium en http://www,unicode.org.

La siguiente lista de nombres de variable válidos incluye algunos de los caracteres Unicode aceptados por el compilador:

- $help_me123
- façade
- café
- υττολογιοτηζ (Griego)

Nota

Para mas inforrnacion sobre lo que hace que un identificador sea valido, vea las descripciones de los metodos `isJavaIdentifierstart()` **e** `isJavaIdentifierPart()` **en la clase** `java.lang.Character`.

Tipos de datos

Cuando declara una variable, debe especificar su tipo. Las declaraciones de variable tienen la forma:

```
type name
```

El tipo de valor almacenado en la variable es el tipo de datos. Los tipos de datos pueden ser primitivos o de referencia. Un tipo primitivo contiene un valor apropiado para el tipo de datos, como el número 1 si el tipo es para números enteros. Por otro lado, los tipos de referencia señalan hacia el lugar donde se almacena el auténtico valor o conjunto de valores.

En el lenguaje de programación Java hay ocho tipos diferentes de tipos de datos primitivos (vea la tabla 4.3). Sus tamaños estdn predefinidos por la especificación del lenguaje, de modo que no tiene que preocuparse por las diferencias entre plataformas.

Tabla 4.3. Tipos de datos integrados en **Java**

Tipo de datos	Descripción	Tamaño
`boolean`	Un valor booleano	NIA (`true` o `false`)
`byte`	Entero con signo de complemento a dos	8 bits
`short`	Entero con signo de complemento a dos	16 bits
`int`	Entero con signo de complemento a dos	32 bits
`long`	Entero con signo de complemento a dos	64 bits
`float`	Punto flotante de precision simple IEEE 754	32 bits
`double`	Punto flotante de precision doble IEEE 754	64 bits
`char`	Un caracter	16 bits

El tipo `boolean` es especial. No hay un auténtico tamaiio definido para el tipo, sólo que el tipo sólo puede tener dos valores: `true` o `false`.

El uso que haga de los tipos de datos numéricos dependerá del tamaño que puedan tener los posibles valores. Cada uno de los tipos tiene un rango válido, que aparece en la tabla 4.4.

Si el rango de valores posibles para un tipo numérico particulares demasiado pequeño para lo que necesita, o necesita mayor precisión para sus resultados, considere el usar las clases `BigInteger` y `BigDecimal` que aparecen en el paquete matemdtico de java. Éstas ofrecen operaciones numéricas sin limite de tamaño, al precio de un peor rendimiento. Revise el capitulo 17 para conseguir mds detalles sobre su uso.

Tabla **4.4.** Rangos de los tipos de datos numericos

Tipo	Mínimo	Máximo
byte	-128	127
short	-32768	32767
int	-2147483648	2147483647
long	-9223372036854775808	9223372036854775807
float	Positivo: 1.40239846 x 10^{-45}	3.40282347 x 10^{38}
	Negativo: -3.40282347 x 10^{38}	
double	Positivo: 4.94065645841246544 x10^{-324}	1.79769313486231570 x10^{308}
	Negativo: -1.79769313486231570 x10^{308}	

El tipo de datos String: Hay otro tipo de datos que requiere atención especial: El tipo de datos String. La clase `java.lang.String` representa una secuencia inalterable de caracteres. La clase tiene lo que se podría considerar un status especial con el compilador; no obstante, es un tipo de referencia y no uno primitivo.

Literales

Cada variable tiene un valor, ¿pero cómo se especifica un valor determinado? Esto es el trabajo de un literal, la expresión constante usada para expresar un valor. Cada tipo de datos tiene su propio formato para especificar un literal, como se describe en la tabla 4.5.

Tabla **4.5.** Formatos literales para tipos de datos

Tipo	Formato
boolean	true o false
int	Decimal: Digitos 0-9 (no empieza con 0)
	Octal: Cero (0) seguido por los digitos 0-7, por ejemplo: 0431
	Hexadecimal: Ox o OX seguido por los digitos 0-9, A-F; por ejemplo: 0xCAFEBABE
long	Agrega I o L al final de un literal int, p.ej. 0xDECADAL
float	Agrega f o F al final de un literal double; reemplaza d/D si se encuentran presentes, por ejemplo: 3.14f3.14E+15F (3.14 x 10^{15} = 3140000000000000)0.314f

Tipo	Formato
`double`	Notación cientifica, corno en 1.345e+10, donde el exponente puede expresarse con una e o una E; puede encontrarse una d o D al final de un literal `double`
`char`	Un único caracter o secuencia de escape entre cornillas sencillas, por ejernplo: 'J', 'z'
`string`	Varios caracteres o secuencias de escape entre cornillas dobles. por ejernplo: "John"

Además de los tipos que aparecen en la tabla 4.5, los objetos de referencia disponen de un literal especial: `Null`. El literal `null` es básicamente una referencia a ningún sitio, lo que significa que el objeto de referencia aún no tiene valor. Entre los literales `char` y `string`, existen secuencias de escape especiales que pueden usarse para caracteres en lugar de caracteres individuales. Éstos aparecen en la tabla 4.6.

Tabla **4.6.** Secuencias escape de carácter

Tipo	Secuencia escape	Descripción
Especial	`\b`	Retroceso.
	`\t`	Tabulador horizontal
	`\n`	Nueva linea
	`\f`	Alirnentacion de pagina.
	`\r`	Retorno de carro
	`\"`	Dobles cornillas (").
	`\'`	Cornillas sencillas (').
	`\\`	Barra inversa.
Octal	`\DDD`	Representa `\u0000` - `\u00FF\` seguido por entre 1 y 3 digitos octales; si son tres digitos, el prirnero debe estar entre 0 y 3; por ejernplo, A es lo rnisrno que \101 (65).
Unicode	`\uHHHH`	`\u` seguido por cuatro digitos hexadecirnales. Cada digito hexadecimal puede estar entre 0-9, a-f, o A-F.

Cómo inicializar variables con literales

Ahora que sabe cómo especificar un literal, ¿qué puede hacer con ellos? Bien, una cosa que puede hacer es inicializar variables. Al agregar un signo igual tras una

declaración de variable, puede especificar un valor inicial para esa variable. De forma general, el formato es el siguiente:

```
type variableName = intialValue
```

Pasando de lo general a lo especifico, si quiere inicializar la variable `count`, que es de tipo `int`, con el valor 3, usará lo siguiente:

```
int count = 3
```

Además de los modificadores disponibles descritos en la posterior sección Modificadores, aqui hay uno que merece una atención especial: el modificador final. Use el modificador `final` para señalar una variable como constante, que no se puede modificar una vez inicializada. Por ejemplo, en la siguiente linea, no se puede modificar el valor `count` de *3:*

```
final int count = 3
```

También está el concepto de inicialización retardada. Si declara cualquier variable para que sea `final`, como en:

```
final int count;
```

entonces puede inicializarla posteriormente, pero sólo una vez:

```
count = 3;
```

Si intenta asignar un nuevo valor a una variable `final` inicializada, el compilador generará un error en tiempo de compilación:

```
// Esto no es válido
count = 4;
```

Todas las variables declaradas a nivel de clase reciben el nombre de variables de miembro. Las variables de miembro se inicializar con un valor por defecto. Puede inicializarlas, pero no es indispensable que lo haga. Observe los valores por defecto que aparecen el la tabla 4.7.

Tabla 4.7. Valores iniciales de variable

Tipo	Valor
boolean	false
byte	(byte)0
short	(short)0
int	0
long	0L

Tipo	Valor
`float`	0.0f
`double`	0.0d
`char`	'\u0000'
`Tipo de referencia`	null

Las variables declaradas dentro de los bloques de código se denominan variables locales. Éstas deben inicializarse, o al menos se las debe asignar un valor, antes de ser usadas. Si intenta acceder a una variable fuera del bloque o antes de asignarla un valor, el compilador protestará. Los bloques de código están definidos entre llaves, comenzando con { y finalizando con }.

```
{
 // Esto es un bloque de código
 // Aquí esta la variable local
 int count;

// La variable count no es visible fuera del bloque de código
// Este código producirá un error en tiempo de compilación
System.out.println(count);Floating-Paint
```

Literales de punto flotante especiales

Hay tres literales de punto flotante especiales, definidos por la especificación IEEE 754 para operaciones de punto flotante, que debemos mencionar:

- No numeral o NaN: Es el resultado de una operación matemática inválida, como dividir cero de punto flotante entre cero de punto flotante, o multiplicar infinito por cero.
- Infinito positivo: Es el resultado de dividir un numero positivo entre cero de punto flotante.
- Infinito negativo: Es el resultado de dividir un número finito negativo entre cero.

La tabla 4.8 le ayudará a comprender el origen de infinito y de NaN.

Tabla 4.8. Producir infinito y valores no numéricos a partir de operaciones de punto flotante

arg1	arg2	arg1/arg2	arg1%arg2
Finito	±0.0	±∞	NaN
Finito	±∞	0.0	X

arg1	arg2	arg1/arg2	arg1%arg2
0.0	0.0	NaN	NaN
±∞	Finito	±∞	NaN
±∞	±∞	NaN	NaN

Instrucciones y expresiones

Quizds haya observado un punto y coma final (;) en algunas de las líneas de codigo de este capitulo. Este cardcter se situa al final de las instrucciones de Java, que son acciones ejecutables individuales del programa Java. Por lo general, las instrucciones estdn escritas en una sola línea, con su punto y coma final, como en:

```
final int count = 3;
```

Sin embargo, el compilador de Java no se preocupa de donde coloca espacios en blanco (como tabuladores, espacios y retornos de carro) en su código fuente. La línea anterior es equivalente a lo siguiente:

```
final
   int
      count
       =
            3;
```

Si usa una forma larga o una forma corta dependerd de su estilo de programación. En cualquier caso, el compilador no protestard y tratard la(s) línea(s) como una sola instrucción. Las instrucciones suelen incluir expresiones, que son combinaciones de variables, palabras clave y literales que evalúan un valor. Por ejemplo, 3 es una expresión ya que evalda un valor. Por otro lado, `System.out.println(count)` no es una expresión porque no evalua ningdn valor.

`System.out.println()` es un ejemplo de invocación de método. Invocar un método puede dar como resultado una expresión si se devuelve algún valor, como en el siguiente:

```
Math.max(var1, var2)
```

Puede colocar un método que devuelve un valor en el lado derecho de una instrucción de asignación si desea realizar alguna operación con dicho valor posteriormente en el programa. Por ejemplo:

```
int max = Math.max(var1, var2);
```

La estructura general de una invocación de método incluye una referencia de objeto, seguido por un punto y luego un nombre de método y una lista de argumento:

```
object.method(argumentos)
```

Sin embargo, si el método es estático, no necesitará una instancia de objeto especifica y puede simplemente usar el nombre de la clase:

```
ClassName.method(argumentos)
```

En el anterior ejemplo, esto se realiza con Math y el método max().

Operadores

Las instrucciones en si mismas son bastante aburridas. Necesita usar operadores, como el operador de asignación (=), para hacer sus instrucciones más interesantes. Los operadores son simbolos especiales usados para combinar variables, literales, invocaciones de método y otros operadores. Un operador funciona como un operando. Por ejemplo, en la siguiente instrucción:

```
2 * 3
```

el operador es el simbolo de multiplicación y los operandos son los números 2 y 3. Si quiere guardar el valor, se agrega otro operador, el signo igual (=) para las asignaciones:

```
int product = 2 * 3;
```

Cuando hay varios operadores en una expresión, aparece el concepto de prioridad de operador, o el orden en el que se aplican los operadores. ¿Sobre qué operador se actua en primer lugar? En la siguiente instrucción hay tres operadores: =, + y *.

```
int result = 2 + 3 * 5;
```

Afortunadamente, la asignación siempre se realiza en ultimo lugar en Java, de modo que en este ejemplo no tiene que preocuparse si 2 se asigna al resultado antes de que suceda alguna otra cosa. Pero ¿esta operación da cómo resultado (2 + 3) * 5, que es 25, o da como resultado 2 + (3 * 5), que es 17? Hay una gran diferencia.

Sin los paréntesis en nuestra sentencia, el valor del resultado será 17. Si se hubiera escrito la instrucción con parhtesis, el orden especifico no habria sido ambiguo:

```
int result = (2 + 3) * 5;
```

De modo que sin paréntesis, debe confiar en acordarse de la prioridad de los operadores. Cuanto más bajo sea el numero de prioridad, antes se considerarin los operandos de un operador en la evaluación de la instrucción.

Además de la prioridad de operadores, las instrucciones de Java incluyen asociatividad. La asociatividad dirige el orden "de izquierda a derecha" o "de derecha a izquierda" de una evaluación de expresión. Por ejemplo, con la multiplicación 2 * 3 * 5, la multiplicación es asociativa por la izquierda, lo que quiere decir que se realiza antes 2 * 3 que 3 * 5. En esta expresión, los resultados serán los mismos sin

importar el orden seguido, pero no siempre es este el caso. Y la asignación es asociativa por la derecha. Asi, para que el valor del resultado sea 2 + 3 * 5, la parte derecha del operador (el signo igual) debe evaluarse en primer lugar, antes de que ese valor se asigne a la variable de la izquierda.

La asociatividad y la prioridad de operador se detallarán en el posterior estudio de cada operador.

Truco

Le recordamos que si quiere forzar un orden de evaluación, use los parentesis. Así se asegurara de que una expresion se evaluara como espera y nunca entorpeceran el rendimiento de ejecución de un programa.

Operadores aritméticos

Los operadores aritméticos de Java se resumen en la tabla 4.9. Estos operadores aceptan operandos enteros o de punto flotante y producen resultados enteros o de punto flotante. Los operadores de autoincremento y de autodecremento se incluyen como operadores aritmeticos.

Tabla 4.9. Operadores aritmeticos

Operador	Función	Prioridad (1 =la más alta)	Asociatividad
++, --	Autoincremento postfijo (var++), autodecremento postfijo (var--)	1	Izquierda
++, --	Autoincremento prefijo (++var), autodecremento prefijo (--var)	2	Derecha
+, -	Unario mas (+var), unario menos (-var)	2	Derecha
*	Multiplicación	4	Izquierda
/	Division	4	Izquierda
	Resto (division modular)	4	Izquierda
+, -	Suma, resta	5	Izquierda

Un apunte sobre el operador resto (%): Devuelve el resto de dividir el primer operando entre el segundo, de modo que 7 % 2.5 pregunta el resto que queda tras dividir 7 entre 2.5, es decir, 3. Y-7 % 2.5 es -2. Si esperaba que los resultados de 7 % 2.5 fuera +/- 0.5, ahies donde aparece el método `IEEEremainder` de la clase `Math`.

Esto funciona segun el estfindar IEEE 754; habria devuelto -0.5 para el primer [`Math.IEEEremainder(7, 2.5)`] y 0.5 para el segundo.

Operadores relacionales

Los operadores relacionales comparan dos cantidades para determinar si son iguales o si una es mayor que la otra. El operador encargado de comprobar la igualdad es ==. Si los operandos son de un tipo preconstruido, (`aritmetic`, `character` o `boolean`) el operador de igualdad devolverfi un valor booleano (`true` o `false`).

Si son variables, el operador devolverfi `true` si se refieren al mismo objeto (o son null) y `false` en cualquier otro caso.

Cuando los operandos pertenecen a algún tipo de datos preconstruido, su funcionamiento es como nos podriamos esperar:

```
t  lean a, b;

a = (2 == 2); // a será true
b = (2 == 3); // b será false
```

Por el contrario, dos objetos equivalentes serán iguales cuando tienen el mismo objeto fisico en la memoria:

```
boolean a, b;
Flight f1, f2;
// f1 y f2 son dos objetos Flight independientes
// con los mismos valores predeterminados
f1 = new Flight();
f2 = new Flight();
a = (f1 == f2); // a será false porque f1 y f2 se refieren
                // a dos casos distintos, aunque contienen
                // los mismos datos
f1 = f2;
b = (f1 == f2); // b será true porque f1 y f2 se refieren
                // al mismo caso
```

Truco

Como muchas veces necesitara comprobar si los contenidos de dos objetos son iguales, la mayoría de los objetos implementan el método `equals()`, que le permite comprobar la igualdad de objetos del mismo modo que hacía con los primitivos.

El operador de diferencia(`!=`) hace exactamente lo contrario que el operador de igualdad. Devuelve `true` cuando los operandos sean distintos.

Los operadores numéricos se pueden comparar unos con otros a través de los operadores mayor que (>), menor que (<), mayor o igual que (>=) y menor o igual que (<=):

```
boolean a, b, c, d;
a = (1 . 2);  // a es false
b = (1 < 2);  // b es true
c = (1 <= 2); // c es true
d = (1 >= 0); // d es true
```

Truco

Algunos objetos implementan el método `compareTo()` para asegurar la ordenacion, igual que el método **`equals()`**. Esto le permite comprobar el orden natural de los objetos y de los primitivos.

Java también trabaja con el operador condicional de C y C++ (?:) que trabaja con tres operandos (operador ternario). El prirnero será booleano, y los otros dos pueden ser de cualquier tipo.

Si el primero es `true`, el resultado será el segundo operando. Si es **`false`**, su resultado será el tercer operando:

```
boolean b;
int c;
b = true;
c = (b ? 1 : 2); // 1 se asignará a c porque b es true
b = false;
c = (b ? 1 : 2); // 2 se asignará a c porque b es false
```

Advertencia

Tenga cuidado con la operación condicional `?:`. Aunque es útil, puede hacer que su codigo sea menos legible.

Los operadores de comparación se suelen utilizar con las declaraciones condicionales. La tabla 4.10 resume todos los operadores relacionales.

Tabla 4.10. Operadores relacionales

Operador	Función	Prioridad	Asociatividad
>, <, >=, <=	Prueba la magnitud relativa	7	Izquierda
==	Prueba la igualdad	8	Izquierda
!=	Prueba la desigualdad	8	Izquierda
?:	Condicional; devuelve uno de dos operandos basados en un basados en un tercero	14	Izquierda

Operadores booleanos

Los operadores booleanos actúan sobre operandos booleanos y devuelven un resultado del mismo tipo. Contamos con las operaciones algebraicas booleanas estandar: AND, OR, NOT y XOR.

El primero de ellos, AND, devolverá el valor true cuando ambos operandos son ciertos. OR devolverá dicho valor cuando cualquiera de los operandos sea `true`. Java dispone de dos versiones de ambos operadores.

- La primera versidn (& para AND; | para OR) obliga a que se efectúe una valoracidn de ambos operadores.
- La segunda versidn (&& para AND y || para OR) no valorará el segundo operador si puede determinar su valor después de evaluar el primero.

Veamos las diferencias existentes entre ambas versiones con un ejemplo. El funcionamiento de los dos operadores AND es similar.

```
boolean b;
// para calcular lo siguiente, la VM valorará ambas expresiones
// y efectuará ambas multiplicaciones
b = ( 100 > ( 5 * 6 ) ) | ( 100 > (8 * 8));  // b será true
// para calcular lo siguiente, la VM sólo valorará la expresión
// de la izquierda, por lo que únicamente efectuará la multiplicación
// (7 * 9)
b = ( 100 > ( 7 * 9 ) ) || ( 100 > (4 * 5));  // b será true
```

Nota

En ciertas ocasiones querremos que la Maquina Virtual valore los dos lados de la ecuacion, independientemente de que el resultado de la operación AND/OR se pueda deducir a partir del primero. En otras, no querremos que se continue con las comparaciones o con las excepciones, porque se habrán cumplido las condiciones de las hipotesis.

El operador XOR devolverá el valor `true` si los operandos no dan el mismo resultado, es decir, si uno es true y el otro es false (true ^ false == true). NOT es un operador unitario que devuelve el operando opuesto (!false = true).

La tabla 4.11 recopila los operadores booleanos.

Tabla 4.11. Operadores booleanos

Operador	Función	Prioridad	Asociatividad
!	NOT	2	Derecha
&	AND booleano	9	Izquierda

Operador	Función	Prioridad	Asociatividad
^	XOR	10	lzquierda
I	OR booleano	I1	lzquierda
&&	AND lógico	12	lzquierda
I I	OR lógico	13	lzquierda

Operadores bitwise

Los tipos integrales (byte, short, int y long) se representan dentro de la memoria del ordenador como una secuencia de bits (digitos binarios). Al igual que los números decimales, los digitos más importantes de los binarios también aparecerán a la izquierda. En la numeración decimal, el dígito más importante del numero 325 es el 3 porque representa a 300 y el menos importante sera el 5 porque representa a la unidad. Cuando se escribe esta cifra en notación binaria, obtendremos el 101000101. El dígito más representativo es el 100000000 (256 en decimal).

Los enteros en Java son números con signo, de tal forma que Java utilizara el que se encuentre más a la izquierda de todo el espacio de almacenamiento para representar el signo del número entero.

En los tipos de datos para los enteros de Java, el bit más alto se utiliza para representar el signo del número, tal y como se puede apreciar en la figura 4.2. Si éste es un 1, el número será negativo.

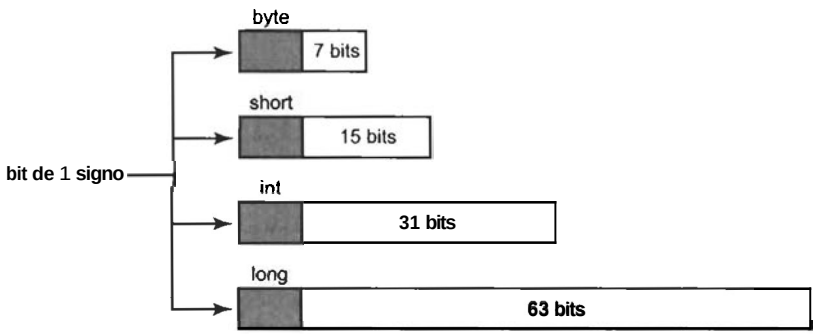


Figura 4.2. Los tipos de datos de Java utilizan el número mas alto para indicar el signo. Un 1 indicara que este es negativo

Para manipular los bits de enteros, Java utiliza dos conjuntos de operaciones: bitwise y de desplazamiento de bits.

- **Operaciones bitwise:** Los operadores bitwise son &, |, ~ y ^. Realizan los mismos tipos de funciones que los operadores booleanos. El operador bitwise AND aplica la operación AND a los bit correspondientes de cada operando. Los

operadores bitwise OR, XOR y NOT trabajan de forma similar. La lógica de los operadores bitwise aparece en la figura 4.3.

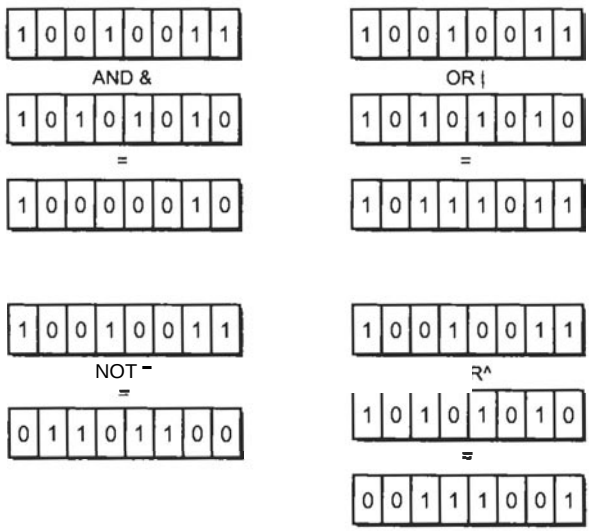


Figura 4.3. Operadores booleanos bitwise

- **Operaciones de desplazamiento de bits:** Los operadores de desplazamiento de bits son <<, >> y >>>. Mueven todos los bits de un tipo integral hacia la izquierda o la derecha, como muestra la figura 4.4. Estos operadores de desplazamiento son operadores binarios. El segundo operando es un entero que determina el numero de bits que se deben desplazar.

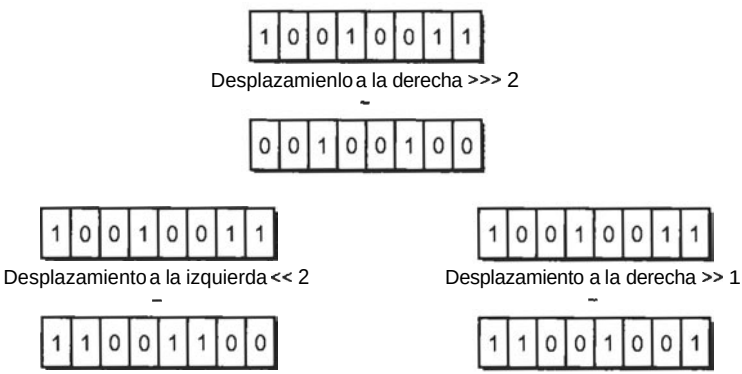


Figura 4.4. Ejemplos de operadores de desplazamiento de Java

- Observe que los operadores de desplazamiento estándar de C, >> y << actúan de forma ligeramente diferente en Java. En Java, todos los números tienen signo y el signo de los bits se conserva a través de los desplaza-

mientos. Aunque el bit con signo se desplace, también se copia, de modo que el número resultante tendrá el mismo signo que el original.

- Java agrega el operador >>>, que desplaza todos los bits hacia la derecha si el numero entero no tenia signo.

Nota

Para hacerlo mas sencillo, la figura 4.4 usa bytes con signo de 8 bits. No obstante, tenga en cuenta que los operadores de desplazamiento de Java funcionan solamente con tipos de datos int y long. Si desplaza un short negativo hacia la derecha usando el operador >>, puede terminar con un numero mayor que con el que comenzo. Esto es debido a que el operador izquierdo es `cast` (convertido de forma explicita) en un int antes de ser desplazado.

La tabla 4.12 recoge los operadores de manipulación de bits.

Tabla 4.12. Operadores de manipulación de bits

Operador	Función	Prioridad	Asociatividad
~	NOT (complemento bitwise)	2	Derecha
<<, >>	Desplazamiento a la derecha, desplazamiento a la izquierda	6	Izquierda
>>>	Desplazamiento a la derecha como si no tuviera signo	6	Izquierda
&	AND bitwise	9	Izquierda
^	XOR bitwise	10	Izquierda
|	OR bitwise	11	Izquierda

Operadores de cadena

Un operador de unión (+) es el único que se puede aplicar a cadenas. Al igual que el operador de suma numérica, el operador de concatenación de cadena tiene una prioridad 5 y asociatividad izquierda. Une dos cadenas para formar una tercera.

```
String s;
s = "Hello" + " " + "World"; // "Hello World" se le asigna a s
```

Si sólo uno de los operadores es una cadena, el otro se convertirá al tipo string automáticamente:

```
String s;
s = "5 + 6 =" + (5 + 6); // "5 + 6 = 30" se le asigna a s
```

Si el operador que no es una cadena es un objeto, Java utilizará el método `toString()` para obtener una cadena equivalente. El método `toString()` pertenece a todas las clases porque el encargado de su implementación es `Object`. El comportamiento predeterminado de `toString()` es devolver el nombre de la clase del objeto junto con un simbolo @ y el código hash del objeto. El código hash es un numero que se calcula a través del método `hashCode()`. Todos los objetos pueden invocar a este método ya que se ha definido dentro de la clase `Object`.

El siguiente código nos muestra el uso de `toString()`.

```
String s;
SomeObject m;
s = "m is " + m; // "m is a SomeObject@1393870" se asigna a s
```

Nota

Si se encuentra agregando constantemente mas contenido a una `String`, quizás no sea la mejor de las clases que puede emplear. Dado que la clase `String` es inmutable, cuando concatena mas contenido a una `String`, está creando un nuevo objeto y eliminando el anterior tras copiar el contenido antiguo. Hay otra clase adicional para manejar cadenas en la librería llamada `StringBuffer` que funciona mejor para esto. Puede imaginar `StringBuffer` como un objeto `string` modificable. Por ejemplo, cuando agrega texto a un `StringBuffer`, solamente está agregando el contenido de forma interna, sin crear ningún objeto nuevo.

Operadores de asignación

Tal y como hemos mencionado con anterioridad, el operador de asignación se encarga de asignar el segundo operando al primero y lo devuelve como resultado.

Otro de los operadores de asignación se utiliza con las operaciones de combinación y asignacion. Si queremos sumar 5 a un número, tendremos que escribir:

```
int i;
i = i + 5;
```

También se puede utilizar el operador de asignación +=:

```
int i;
i += 5; // es el mismo i = i + 5
```

Hay operadores de asignacion para la mayoría de los operadores mencionados (vea la tabla 4.13).

Tabla 4.13. Operadores de asignacion

Operador	Función	Prioridad	Asociatividad
=	Asignación	15	Derecha
*=	Asignación con operación	15	Derecha
/=	Asignación con operación	15	Derecha
=	Asignacion con operación	15	Derecha
+=	Asignacion con operación	15	Derecha
-=	Asignacion con operación	15	Derecha
>>=	Asignacion con operación	15	Derecha
<<=	Asignacion con operación	15	Derecha
>>>=	Asignacion con operación	15	Derecha
^=	Asignacion con operación	15	Derecha
|=	Asignacion con operación	15	Derecha
&=	Asignacion con operación	15	Derecha

Operadores especiales

Hay algunos otros operadores que merecen ser mencionados aquí. Éstos son para convertir, crear y comprobar.

Conversión de datos

El operador **cast** convierte un tipo de datos en otro. Estos convertidores se escribirán entre paréntesis, como el nombre del tipo al que se van a convertir los datos. Aqui tiene un ejemplo:

```
int i;
long l;
l = 1 << 40; // l es un número muy grande
l--;
i = (int) l; // i se va a convertir en un integer
```

Si a una variable de una precisidn determinada se le asigna un valor con una precisión menor, no hará falta que se efectue ningún tipo de conversión. Por ejemplo, no se necesita realizar ninguna conversión para asignar una `int` a un dato `long` o un **`float`** a uno `double`.

Los operadores de conversión tienen una preferencia de 1 y su tipo de asociacion es por la derecha.

Advertencia

Los programadores de C/C++ deben de tener mucho cuidado con esto. En C, la conversion de tipos es automática para los tipos de datos relacionados. Java, sin embargo, necesita que especifique explicitamente cualquier conversion que pueda alterar el contenido (o precision) de una variable.

El operador instanceof

El operador `instanceof` se utiliza para probar la clase de un objeto:

```
boolean b;
Person p;
Student s; //Student es una subclase de Person
p = new Person();
s = new Student();
b = p instanceof Person;  // b será true
b = p instanceof String;  // b será false
b = s instanceof Person;  // b será true porque s es una
                          // subclase de Person
```

El operador `instanceof` tiene una prioridad de 7 y su asociación es a la izquierda.

El operador new

La asignación de objetos incluye el operador new, usado para invocar al constructor para una clase que quiera instanciar. Aquí tiene un ejemplo de new:

```
int length = 300;
new StringBuffer(length);
```

El operador new tiene una prioridad de 3 y su asociatividad es a la derecha.

Modificadores

Los modificadores en el lenguaje de programación Java se usan para modificar la durabilidad, accesibilidad o comportamiento de una clase, variable o método. La tabla 4.14 muestra la aplicabilidad de cada modificador a clases, métodos y variables.

Almacenamiento y duración de los modificadores

En las siguientes secciones describiremos el almacenamiento y la duración de los siguientes modificadores: `abstract`, `static`, `synchronized`, `native`, `volatile`, `transient` y `final`.

Tabla 4.14. Aplicacion de modificadores a clases, metodos y variables

Modificador	Clases	Clases de rniernbro	Método de miernbro	Variables de de rniembro	Variables locales
`abstract`	✓	✓	✓	-	-
`final`	J	✓	✓	✓	✓
`native`		-	✓	-	-
`private`		✓	✓	✓	-
`protected`	J	✓	✓	✓	-
`public`	✓	✓	✓	✓	-
`static`	✓	✓	✓	✓	-
`strictfp`	J	✓	✓	-	-
`synchronized`	-	-	✓	-	-
`transient`		-	-	✓	-
`volatile`		-	-	✓	-

El modificaclor abstract

Cuando se aplica a una ~lasee,l modificador **`abstract`** indica que la clase no se ha implementado por completo y no se debe iniciar. Si se aplica a la declaracidn de método de un miembro, indicard que ésta se implementard dentro de una subclase. Como la función no tiene ninguna implementación, no se podrá iniciar la clase, por lo que se tendrd que declarar como abstracta. Por defecto, las interfaces son **`abstract`**.

El moclificador static

Generalmente, cada caso de una clase tiene su propia copia de cualquier variable de los miembros. De todas formas, se puede designar una de estas variables a la propia clase, independientemente de los objetos que posea. A este tipo de variables se las conoce como miembros estdticos y su declaracidn se efectuará con el modificador `static`. Este tipo de variables se suelen utilizar para registrar información sobre los distintos casos de una clase. La clase que le mostramos a continuación registra el número de casos que se utiliza de una misma clase a traves de una variable estática llamada `instanceCount`:

```
public class MyClass {
    public static int instanceCount;
    public MyClass() {
        // cada vez que se invoca al constructor,
        // se aumenta una unidad el contador
```

```
        instanceCount++;
    }
    static {
        instanceCount = 0;
    }
}
```

Obskrvese que para iniciar la variable estática se ha utilizado un iniciador estático. Los mktodos tambikn se pueden declarar como estáticos. Por ejemplo, `resetCounter()` es un método estdtico que puede reiniciar la variable `instanceCounter` de `MyClass`:

```
public class MyClass {
    public static int instanceCount;
    public MyClass() {
        // cada vez que se invoca al constructor,
        // se aumenta una unidad el contador
        instanceCount++;
    }
    public static void resetCounter() {
        instanceCount = 0;
    }
    static {
       instanceCount = 0;
```

Al método `resetCounter()` se le puede invocar a travks de la clase `MyClass` o de uno de sus casos:

```
m = new MyClass(); // instanceCount serd igual a 1 después de la llamada
                   // del constructor
n = new MyClass(); // instanceCount serd igual a 2 después de la llamada
del constructor
System.out.println(MyClass.instanceCount + " instances have been created");
m.resetCounter();       // reinicia el contador
MyClass.resetCounter(); // otra forma de reiniciar el contador
```

La clase system, que se encuentra dentro del paquete `java.lang` de la interfaz API, define sus propios mktodos publicos y variables como static. El acceso a las variables y funciones se efectua a travks de la propia clase Systern. De hecho, su constructor es `private` por lo que no se podrá crear ningún objeto System con el código usual:

```
System MySystem = new System(); // ilegal
```

En vez de eso, para acceder a las variables y funciones de la clase Systern, se utiliza la propia clase.

Recuerde uno de los ejemplos anteriores donde imprimiamos informacidn en la consola utilizando el siguiente código:

```
System.out.println["Hello World");
```

La variable out es estiitica del tipo PrintStream. Se define en la propia interfaz de la siguiente manera:

```
public final static PrintStream out;
```

Como la clase System define sus propias funciones como static, podemos llamar a la función currentTimeMillis() para obtener el tiempo actual a traves de la siguiente expresión:

```
long timeNow = System.currentTimeMillis();
```

El método currentTimeMillis() se define dentro de la clase System de esta forma:

```
public static native long currentTimeMillis()
```

Utilizaremos todas las clases estáticas (las clases que se encuentran dentro de los miembros son estiiticas) siempre que queramos modelar una entidad unica. Es decir, cuando queramos crear una unica copia de una variable, por ejemplo en los casos en que se desea guardar un registro del numero de casos que se han creado.

Si utilizamos la palabra clave static cuando se declara una interfaz o una clase interna, nos encontraremos con que la estamos relacionando con una clase de nivel superior a través de un nombre basado en una jerarquia, suponiendo que sus modificadores de acceso (public o protected) así lo permitan.

Por ejemplo, la siguiente clase define la interfaz anidada StudentLounge dentro de PrivateSchool:

```
public class PrivateSchool extends School {
    static public interface StudentLounge {
       public void useVendingMachine(String drink);
    }
    Lounge lounge = new Lounge();
    PrivateSchool(String name) {
       super(name);

    class Lounge implements StudentLounge {
       public void useVendingMachine(String drink) {
           System.out.println("Please pay $1 for a " + drink);
       }
    }
    public StudentLounge getLounge() {
        return lounge;
    }
```

Ahora, una clase externa como PrivateSchool podrá acceder a una interfaz anidada como PrivateSchool.StudentLounge:

```
PrivateSchool.StudentLounge lounge;
PrivateSchool ps = new PrivateSchool("George");
lounge = ps.getLounge();
lounge.orderDrink("Coke");
```

El modificador synchronized

Una función de un miembro sincronizado sólo permite que se utilice un thread para ejecutar la función. De esta forma se evita que dos thread puedan molestarse mutuamente.

Por ejemplo, supongamos que tenemos dos threads, A y B, que se utilizan para actualizar el balance de un banco. Supongamos que una cuenta tiene 100$. Ahora, de forma simultánea, A trata de depositar 50$ en la cuenta, mientras que B depositar6 75$. Ambos thread tratarán de obtener un extracto inicial de la cuenta. Ambos ven que el saldo es de 100$. Cada uno de ellos, por separado, tratará de depositar su dinero y de hacer unbalance de la nueva cantidad. Si A termina el ultimo, el saldo de la cuenta será de 150$. Si lo hace B, será de 175$. Obviamente, ninguno de estos saldos es el correcto. La cuenta deberia tener realmente 225$ (100$ + 50$ + 75$). El problema es que ambos utilizan el mismo código a la vez (solicitar y actualizar el balance). Este problema es el que se evita con la sincronización.

Por defecto, los métodos de sincronización no son estáticos, pero se pueden declarar como tales.

El modificador `synchronized` no se aplica a las clases o a las variables de los miembros, pero se puede aplica a bloques de código.

El modificador native

Los mktodos nativos se implementan en otros lenguajes de programación como C, de modo que carecen de bloque de codigo. La mayoría de las clases que se encuentran dentro de la API Core son nativas porque han de acceder al sistema operativo (a unas rutinas especificas, como aquellas que permiten dibujar algo en la pantalla). El modificador `native` se aplica solamente a mktodos.

Aqui tenemos un extracto de la clase `Math` de la API:

```
public static native double sin(double a);
```

Esta declaración llama a una función que se encuentra en la biblioteca de código nativo para que calcule el seno del ángulo `a`. En una plataforma Internet x86, el código nativo deberia llamar a la función seno del procesador o del coprocesador. En otras plataformas, la función nativa puede encargarse de hacer los cálculos. Esta función también se podrá declarar con los modificadores `static` y `public`.

El modificador volatile

Una variable volátil es aquella cuyo valor puede cambiar de forma independiente al programa Java. Generalmente, este tipo de variables representa una entrada procedente del mundo exterior, como por ejemplo la hora. Tambikn se utilizan para mostrar variables a las que pueden modificar los thread de ejecución. La palabra clave `volatile` evitará que el compilador registre sus cambios. Se supondrá que su valor cambia frecuentemente. El uso de este modificador es bastante extraño. Puede parecer necesario, pero carece de compatibilidad en la implementación subyacente. En otras palabras, no se preocupe por él.

El modificador transient

El modificador `transient` se utiliza junto con la serialización para suministrar objetos persistentes. Estos objetos se pueden guardar en el disco y recuperar en cualquier máquina. Encontrará más información sobre la serialización en el capitulo 19. El modificador `transient` indica que no se guarde la variable.

El modificador final

La mayoría de los lenguajes de programación tienen una forma de declarar una variable como constantes (es decir, que no cambia), y es algo que también se cumple en Java. La palabra clave `final` indica que la variable local o del miembro no se puede modificar. Se suele utilizar con las constantes simbólicas. Puede hacer referencia a una constante por su nombre y definir ese nombre en una unica localización de su código. Si posteriormente necesita modificar el valor en su código, sólo debe realizar el cambio en el punto donde definió la variable final.

Si se declara una variable como final, se tendrá que reiniciar al mismo tiempo:

```
final int MAX_PAGES = 23;
```

o inicializarla con una asignación que nunca cambiará:

```
final int MAX_PAGES;
MAX_PAGES = 23;
```

Por ejemplo, con variables de instancia, puede inicializar variables de instancia finales en cada constructor. Encontrará más información en el capitulo 5.

Las funciones y clases de un miembro también se pueden declarar como finales. Un método de miembro final no se podrá anular y una clase final no podrá ser una subclase.

Truco

Cuando se declara algo como final, el compilador/optimizador puede efectuar una serie de suposiciones que mejoraran considerablemente el rendimiento del programa.

El modificador Strictfp

El modificador `strictfp` se usa con clases y con métodos. Si especificamos una clase como `Strictfp`, indicará que todos los métodos son `Strictfp`, donde "fp" significa punto flotante. Si especificamos que un método es `Strictfp`, todas las expresiones de ese método deben seguir la especificación de punto flotante IEEE 754. Cuando un método no es estricto, el sistema de tiempo de ejecución puede ser más agresivo optimizando las operaciones de punto flotante, buscando velocidad por encima de la predicibilidad.

Modificadores de accesibilidad

Por defecto, las unicas que tienen conocimiento de la existencia de una clase, las funciones y sus variables serán el resto de clases del paquete. Para los applet más sencillos, tenemos que sólo se puede acceder a una clase a través de las clases que se encuentren en el mismo directorio. Java tiene otros modificadores que se utilizan para diferenciar la accesibilidad de las clases y la de sus miembros a otras clases.

- private: El modificador `private` restringe el acceso a los miembros de una clase de tal forma que ninguna otra pueda llamar a sus funciones o acceder directamente a sus variables. Este modificador protege a nivel de clase, pero no a nivel de instancia. Todas las instancias de una clase pueden acceder a todas las variables privadas en todas las instancias, no sólo en su propia instancia.
- Acceso por defecto: Cuando no se encuentra presente ningún modificador, se dice que es un acceso por defecto. Un miembro con acceso por defecto proporciona a otras clases dentro del mismo paquete, acceso a miembros de clase.

Nota

No confunda la accesibilidad por defecto con la palabra `clavedefault` que se emplea en las instrucciones `switch` (ver capítulo 6); la palabra `clavedefault` no es un modificador. Algunas personas llaman al comportamiento por defecto "amigo", pero eso tampoco es un modificador (aunque hay un modificador `friend` en C++).

- protected: Un miembro con el modificador `protected` se parece al acceso predeterminado, pero otorga cierto acceso privilegiado a las subclases de otros paquetes. A menudo, las funciones de los miembros se crean para utilizarse dentro de la propia clase (y no para el consumo general). Si se declaran como protected las podremos utilizar dentro de la clase y todas sus subclases.
- public: El modificador `public` no restringe el acceso. Cualquier parte del codigo puede acceder a clases, métodos y variables de miembros.

Los efectos de los diferentes modificadores de accesibilidad aparecen en la figura 4.5. En la figura, hay dos barras para cada tipo de modificador. La barra superior representa la capacidad de una subclase para heredar una variable o método de miembro. La segunda barra muestra la accesibilidad de un miembro.

Por ejemplo, la figura muestra que los métodos o variables declaradas como protected tienen las siguientes caracteristicas.

- Las subclases las heredarán, independientemente de que se encuentren en el mismo paquete o no. Esto permite a una subclase acceder a estos miembros solamente en instancias de la subclase.
- Cualquier clase del mismo paquete podrá acceder a ellas, pero en caso contrario no serán visibles.

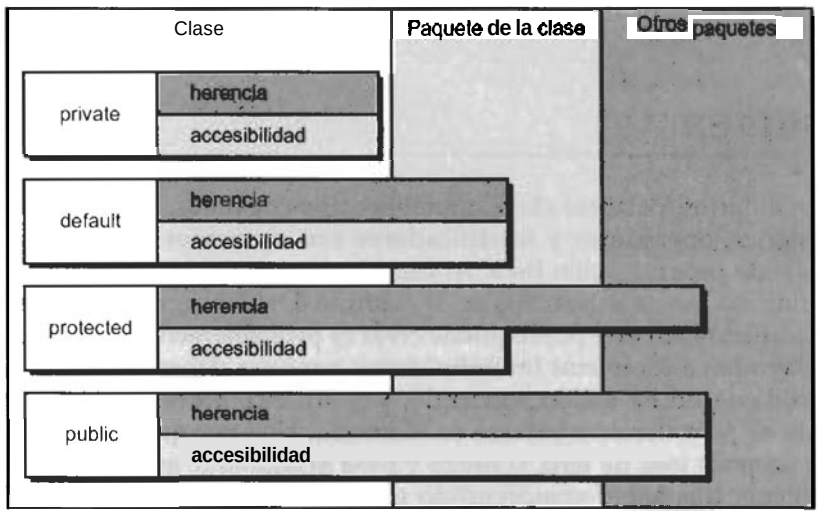


Figura 4.5. La accesibilidad de clases y sus metodos y variables de miembro depende del modificador empleado al declarar la clase o el miembro

Cualquier clase puede acceder a una publica. Cuando se le pide a una Máquina Virtual que ejecute un nuevo applet, su clase tendrá que ser publica. De todas formas, las clases con las que tenga que trabajar el applet no podrán ser públicas durante el tiempo que las utilice.

Se podrá acceder desde cualquier clase a los métodos y variables de las clases públicas que se declaren como tal. Los controles ActiveX, JavaScript y Visual VBScript podrán acceder a los miembros publicos cuando trabajan fuera de la Máquina Virtual. Si declaramos como public los miembros de una clase no publica, se podrá acceder a ellos desde cualquiera de las clases pertenecientes a su mismo paquete. De todas formas no conviene utilizar el modificador `public` de no ser completamente necesario.

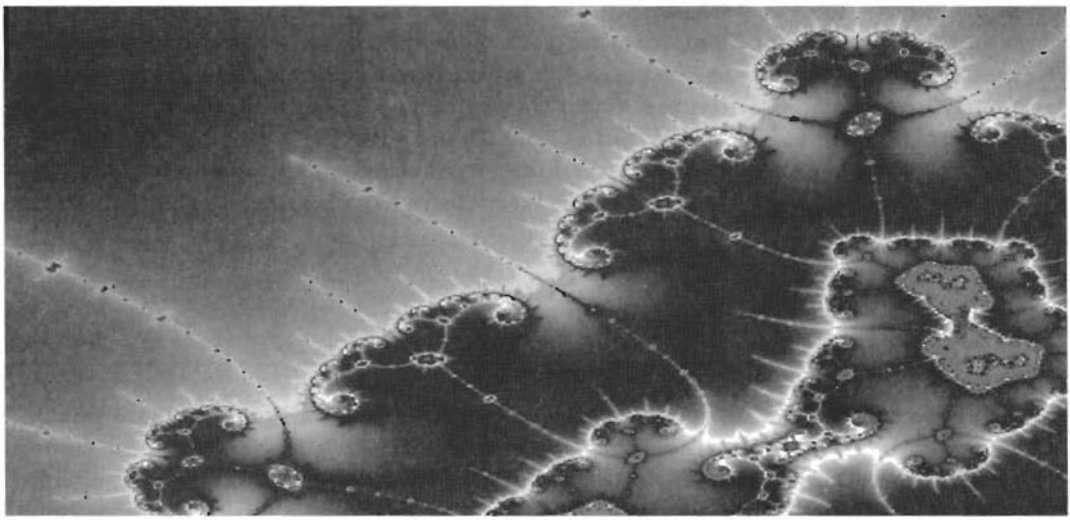
Si no se especifica ningun modificador, se utilizará el predeterminado para la clase, variable o método con el que se trabaje. Supongamos que creamos una clase llamada Chicken, que sabe como cruzar la carretera. El método de miembro público `crossRoad()` tiene dos pasos: (1) comprobar si se acercan vehículos y (2) atravesar la carretera. Estos pasos pueden escribirse como dos métodos de miembro llamados `checkTraffic()` y `walkAcrossRoad()`. Los métodos de utilidad no deben ser publicos porque no queremos que un programador que sólo usa la clase para invocar `walkAcrossRoad()`, sin antes invocar a `checkTraffic()`; estopuede

resultar perjudicial para `Chicken`. Sin embargo, si hace que los métodos de la utilidad sean privados, no sera posible mejorar estos metodos cuando escriba la subclase `SmarterChicken`. Lo ideal seria que la clase `SmarterChicken` pudiera invalidar el método `crossRoad()` para buscar un semaforo y luego invocar `checkTraffic()` y `walkAcrossRoad()` si es necesario. Al declarar los métodos de utilidad como protegidos, estarán disponibles para los programadores que crearán subclases de `Chicken` y que, por tanto, comprenderán los peligros que conlleva cruzar la carretera.

Resumen

Comentarios, palabras clave, variables, tipos de datos, literales, declaraciones y expresiones, operadores y modificadores son elementos clave de la sintaxis del lenguaje de programación Java. Al combinar estos del lenguaje con los elementos de definición de clase tratados en el capitulo 3, sus programas empezarán a estar más completos. En este punto puede crear (y probablemente comprender) métodos que le ayudan a completar las definiciones para sus clases y objetos.

Si todavia no ha salido corriendo, pronto encontrará más pruebas de que la sintaxis de Java tiende a basarse en sí misma. Una vez que comprende una parte, arroja un poco más de leña al fuego y pasa al siguiente aspecto. Pronto tendrá un gran fuego, tras haber comprendido todas las diferentes partes de la sintaxis de Java.



5
Clases de paquetes y acceso a interfaces

Ahora que ya controla los fundamentos del diseiio de clases y la sintaxis necesaria, examinemos ciertas complejidades del lenguaje para enseiiarle más sobre las clases y ayudarle a crear mejores programas de Java. En este capitulo presentaremos algunos de los elementos importantes de la programación en Java. En concreto, aprenderemos qué es el casting o la conversión del tipo de dato de un valor en otro. Veremos cómo se utiliza `this` y `super` en las referencias a datos, métodos ocultos y constructores de una clase y de su superclase. También veremos el modelo de la memoria de objetos (cómo se controla la memoria asignada a un objeto) y sus interfaces, las soluciones de Java para las herencias multiples y las funciones de retroinvocacion.

A continuación, presentaremos una serie de paquetes para agrupar clases e interfaces con lo que se mejorara la organización y se reduciran los conflictos entre los nombres. Por ultimo, concluiremos con algunas convenciones para los nombres para ayudar a los responsables del mantenimiento de los programas a comprenderlos mejor.

Casting o conversion de tipos de datos

El casting es la conversión explicita de un valor desde un tipo de datos a otro. La forma general de una operación de casting es la siguiente:

```
(tipodato) expresión
```

Donde `tipodato` puede ser un tipo de referencia o uno primitivo. En cuanto a la preferencia, el casting tiene lugar después de los operadores unarios (corno ++ y --) y antes de los operadores binarios (corno *, + y -). El compilador exige que realice conversiones entre tipos de datos primitivos cuando existe la posibilidad de perder datos, como cuando se pasa de un numero de punto flotante a uno entero. Básicamente, mediante el casting está indicando al compilador que puede eliminar la parte decimal del numero. Por ejemplo, el siguiente código convierte un valor `double` en un valor `int` mediante casting. Dado que se elimina la parte decimal del numero, el valor que aparece es *2.*

```
double d = 2.5;
int i = (int)d;
System.out.println(i);
```

En otras ocasiones, la conversión es necesaria para mejorar un tipo de datos. Cuando realiza una división entera, el resultado es un numero entero. Por ejemplo, la ecuación 5/2 da como resultado *2.* Como *5* y *2* son número `integer`, el resultado de la expresión también será un numero `integer`. De modo que si quiere que el resultado sea *2.5,* debe convertir el *5* o el *2* a `float` (o `double`). Esta conversión también se realiza mediante casting.

```
float f = (float)5 / 2;
System.out.println(f);
```

Como la prioridad del operador `(float)` es mayor que la de la división (/), el entero *5* se convertirá en un numero con coma flotante antes de utilizarlo en la operación matemática. Cuando uno de los operandos de una división es un numero con coma flotante, el segundo operando se convertirá automáticamente en otro y la división tendrá lugar con numeros con coma flotante. Si se modifica la expresión a:

```
(float) 3 + 5 / 2
```

nos encontraremos con una división entre números con coma flotante, seguida de una suma con otro número del mismo tipo. Esto es debido a que el operador de división tiene una mayor prioridad de operador que el operador de suma.

Conversion entre tipos primitivos

Los tipos de datos primitivos se pueden dividir en dos: booleanos y numéricos. Los primeros no pueden intervenir en procesos de conversión.

En cambio, sí que se pueden efectuar conversiones con el resto de tipos de datos numéricos (`byte`, `char`, `short`, `int`, `long`, `float` y `double`). Pero conviene tener en cuenta que con estos procesos se pueden llegar a perder datos. Si se pasa de un tipo con muchos decimales (corno `int`) a otro que trabaje con menos (corno `byte`), estaremos descartando información.

Para un numero con signo, el signo del número puede cambiar tras la conversión. Por ejemplo, el resultado de `(byte)` *256* es 0 y el de `(byte)` *255* es -1, porque

este tipo de datos unicamente trabaja con el rango de numeros comprendido entre -128 y 127.

Si se efectua una conversión de tal forma que no se pierda información, como de `byte a int` o de éste a `float`, el casting sera automático. Las conversiones sólo se harán manualmente cuando exista el riesgo de perder información.

La asignación de un valor primitivo a una variable de este tipo sólo se podrá efectuar cuando la asignación no suponga la pérdida de información. En cualquier otro caso, se tendrá que efectuar cierta conversión. La misma regla se puede aplicar a los argumentos de las llamadas de los métodos. Si el ancho del tipo de dato del argumento es menor que el que establece el método, se ensanchara automáticamente. Por ejemplo, en la segunda declaración del siguiente fragmento de un programa, se entrega a `sqrt()` una variable del tipo `float` y dicha función necesita un argumento del tipo `double`. El valor de la variable `f` se convertirá en `double` antes de invocar al método:

```
float ff = 4;
double dd = Math.sqrt(ff);
int  ii = 1;
byte bb = ii;
```

Además, la ultima declaración no es legitima porque se puede perder información al convertir un tipo `int` en un `byte`. El compilador generará un mensaje de error como el siguiente.

```
CastTest.java:13: possible loss of precision
found   : int
required: byte
       byte bb = ii;
```

Para que el compilador efectue una primera comprobación, la declaración se tendrá que modificar de la siguiente manera:

```
byte b = (byte) ii;
```

Conversión entre referencias

La primera regla de la conversión entre referencias es que uno de los tipos que intervenga en el proceso ha de ser igual a la clase del otro tipo (o en su defecto, una de sus subclases).

Sólo se permitirá la asignación de distintas clases si se asigna uno de los valores de la clase a una de las variables de la superclase. La asignación de una variable perteneciente a una subclase, necesitará una conversión mas concreta. Por ejemplo, la segunda declaración del siguiente fragmento de código no es legal porque la clase `String` es una subclase de `Object`:

```
Object o = new Object();
String s = o;
```

El compilador mostrará un mensaje de error parecido a éste:

```
CastTest.java:16: incompatible types
found   : java.lang.Object
required: java.lang.String
       String s = o;
                  ^
```

Habria que efectuar una conversión explicita:

```
String s = (String)o;
```

Si durante el proceso de ejecución aparece una conversión ilegal, se iniciará una excepción `ClassCastException`. Su aparición se puede deber a que la conversión explicita confunda al compilador a la hora de permitir que un objeto acceda a los datos o a los miembros de una subclase. Al fallar el intento aparecerá la excepción.

Por ejemplo, agregar una de las siguientes instrucciones al anterior fragmento de código producirá una excepción, porque `s` y `o` hacen referencia a un objeto de tipo `Object` y el método `length()` está definido en la clase `string`:

```
int iii = s.length();
int jjj = ((String)o).length();
```

Si no se controla la excepción, el programa dejara de ejecutarse y aparecera un mensaje como el siguiente:

```
Exception in thread "main" java.lang.ClassCastException: java.lang.Object
        at CastTest.main(CastTest.java:16)
```

Uso de this y super con las referencias al constructor y los miembros

Ya vio el uso de las palabras clave `this` y `super`. Ahora examinaremos con más atencion el empleo de estas palabras clave. Se pueden utilizar para acceder a otros métodos del constructor o para acceder a los métodos y datos de una clase o su superclase directa.

Para referencias al constructor

Los constructores de clase tienen una primera declaración implicita. A menos que la primera instrucción de un constructor sea `this()` o `super()`, con otros posibles argumentos, la primera instrucción es una llamada implicita al constructor de la clase sin argumentos. Es como decir "Antes de construir los detalles especificos de la subclase, prepara los elementos de la superclase". Sólo cuando haya terminado con el constructor de la clase, continuará y configurara el resto de la subclase.

Sin embargo, si la primera instruccidn del constructor de la clase es `super()` o `this()`, se invocará a ese constructor, dependiendo de los argumentos pasados a `super()` o `this()`.

En el caso de `this()`, usaría este comportamiento para proporcionar argumentos predeterminados al constructor. Por ejemplo, la clase `java.awt.TextField` tiene cuatro constructores:

```
public TextField()
public TextField(int columns)
public TextField(String text)
public TextField(String text,int columns)
```

Estos constructores le permiten inicializar cero, uno o dos valores diferentes para la clase; los valores son la anchura y el texto inicial. Sin `this()`, estos constructores habrian duplicado cualquier código de inicializacidn adicional de la clase. Pero al usar `this()`, todo el código de inicialización debe encontrarse en un único constructor y los otros tres simplemente pasan los valores por defecto apropiados.

Éste es el aspecto de los constructores `TextField`:

```
public TextField() throws HeadlessException {
    this("", 0);
}
public TextField(int columns) throws HeadlessException {
    this("", columns);
}
public TextField(String text) throws HeadlessException {
    this(text, (test != null) ? text.length() : 0);
}
public TextField(String text, int c) throws HeadlessException {
    super(text);
    columns = (c >= 0) ? c : 0;
}
```

El constructor final es el unico que invoca a un constructor para la superclase en lugar de invocar a `this()`. En este caso concreto, en lugar de llamar a la versión predeterminada sin argumentos, el constructor `TextField` pasa el valor del texto inicial, que puede ser la cadena vacia (o null).

La clase `TextComponent` es la superclase de `TextField`. Cuando se invoca `super(text)` en el cuarto constructor de `TextField`, se invoca al constructor `TextComponent` que acepta un argumento `String`.

```
TextComponent(String text) throws HeadlessException {
    . . .
```

Como el unico constructor de `TextComponent` requiere un argumento `String`, cualquier subclase que cree debe llamar a `super(string)` en su constructor. Si omitió esa llamada y confid en la llamada implicita `super()` sin argumentos, se producirá un error.

Para referencias de los miembros

Al ir más all6 de los constructores (o incluso dentro de los constructores) nos vemos obligados a usar `this` y `super` con referencias de miembro. Las variables locales que se encuentran dentro de un miembro pueden compartir los mismos nombres como casos de una variable o como las variables de una clase. Este método puede producir errores y debe evitarse, pero se puede emplear la palabra clave this para mostrar las clases y variables de instancia que de otra forma quedarian ocultas. Del mismo modo, la palabra clave `super` muestra metodos y variables en una superclase que, de otro modo, quedarian anuladas o permanecerian ocultas:

- `this`: Se utiliza para hacer una referencia al objeto bajo el cual se ha llamado al método.
- `super`: Se utiliza para acceder a los métodos o datos de los miembros definidos dentro de la superclase.

Por ejemplo, revise el constructor `TextField` que vimos en la anterior sección. La variable de instancia `columns` se inicializó con el parámetro `c`. Sin embargo, si el parámetro `c` se llamase `columns`, esto podría "ocultar" la variable de instancia, porque todas las referencias a `columns` serian a la variable local. Aqui es donde la palabra clave `this` resulta útil. Simplemente anteponga al acceso a la variable de instancia la palabra clave `this` (y un punto), como cualquier otra referencia de clase y ahora tiene acceso a la variable de instancia:

```
public TextField(String text, int columns) {
     super(text);
     this.columns = (columns >= 0) ? columns : 0;
```

En este caso, `this.columns` en-la parte izquierda de la instrucción de asignación hace referencia a la variable de instancia `columns`, mientras que el `columns` del lado hace referencia a la variable de argumento.

Se puede hacer lo mismo con la palabra clave `super`, pero está mostrando las variables y métodos de la superclase. Por ejemplo, en la siguiente subclase de `TextField` se introduce otra variable `columns`. En el constructor `MyTextField`, para acceder al parámetro `columns`, simplemente tiene que usar `columns`. Para acceder a la variable de instancia de `MyTextField`, use `this.columns`. Y para acceder a la variable de instancia de la superclase, use `super.columns`.

```
import java.awt.TextField;
public class MyTextField extends TextField {
    int columns;
    public MyTextField(String text, int columns: {
       super(text, columns);
       this.columns = super.columns + 1;
    }
```

Nota

La definición MyTextField y nuestra descripción del uso de super y la variable columns, suponga que la variable columns de TextField es protected o public. La variable columns de variable de TextField está declarada corno paquete privado (arnistoso) de modo que no es visible en subclases. Asi pues, la clase MyTextField no Se puede compilar.

El modelo de memoria de objetos de Java

Acceder a las variables de sus programas mediante this, super ,y directamente, le ayudará a comprender dónde se almacenan todos estos datos. El cambio dinámico de la memoria asignada a un programa se puede dividir en dos partes:

- **Memoria** stack: Esta zona de memoria de programa se utilizaria para albergar las variables locales declaradas en los métodos o en los bloques de datos. Su tamaño creceria en una única dirección y se reduciria en la opuesta. Dicho crecimiento estaria ligado a la declaración de las variables locales (incluyendo los argumentos de las variables a las que invoquen los métodos). En el momento en que se cierren estos métodos o bloques, las variables se colocarán fuera de la memoria.
- **Memoria** heap: Esta zona de memoria de programa se usa para almacenar memoria para los objetos. Las referencias a los objetos se podrán colocar dentro de la memoria anterior, pero el espacio que ocupasen los datos de los objetos se encontraria dentro de la memoria heap. En realidad esta memoria no es otra cosa que una gran tabla de contenidos donde cada celdilla se utiliza para guardar los datos de los objetos. Cuando desaparezcan las referencias que apuntan a uno de los objetos que se encuentra en esta memoria, se libera el espacio que ocupa.

Para demostrar el uso de la memoria, supongamos que tenemos dos métodos definidos como veremos a continuación, donde la clase TextField se define para contener las variables columns y text del anterior ejemplo con this y super. (Olvide que text en realidad forma parte de la superclase):

```
void m1() {
    int v1 = 1;
   TextField t1;                  // punto de control n°1
    t1 = new TextField("Hi", 3); // punto de control n°%
    m2(v1, t1);
    v1 = 8;                       // punto de control n°5
}
void m2(int v2, TextField t2) {
    int v3 = 4;
```

```
TextField t3;                    // punto de control nº3
 t3 = new TextField("Ho", 6);
 v2   7;                         // punto de control nº4
```

Cuando se invoca el método `m1()` y se ejecuta la declaración de la variable local (es decir, cuando se llega al punto de control nº1) las dos variables locales `v1` y `t1` se colocariin en la memoria stack, con los valores `1` y `null` respectivamente. La figura 5.1 nos muestra el aspecto que presentaria en este punto el modelo de la memoria.

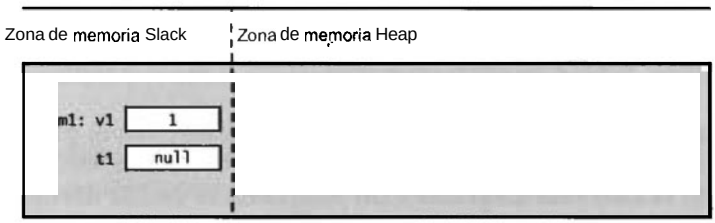


Figura 5.1. Aspecto del rnodelo de memoria en el punto de control nol

En el siguiente punto de control nº2, el objeto recien creado se coloca en la memoria heap y se guarda una referencia a él en la variable local `t1` que se encuentra en la memoria stack. El aspecto que presentará el modelo de memoria será el que muestra la figura 5.2.

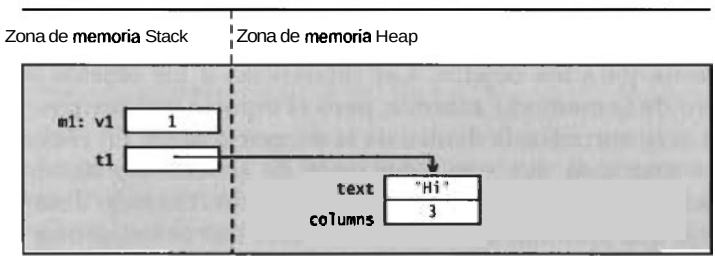


Figura 5.2. Aspecto del rnodelo de rnernoria en el punto de control nº2

A continuación, se llama al método `m2()` con dos argumentos. En la llamada a un método, se traspasan los valores de los argumentos como si su tipo de dato fuese primitivo. Si los tipos son de referencia, se pasa el valor de la referencia. Por lo tanto el argumento de la variable `t2` se referirii al mismo objeto que al que hace referencia la variable local `t1` del método `t1()`. En el punto de control nº3, donde se declaran las variables `v3` y `t3`, el aspecto del modelo de memoria seria el que representa la figura 5.3.

A continuación, se utiliza la declaración `new` para crear otro objeto, al que se le asigna la variable lógica `t3`. Una vez más, el objeto que acabamos de crear se coloca dentro de la memoria heap. Luego, se le asigna el argumento de la variable `v2`. Como la llamada se establece a través del valor, el valor de la variable local `v1` del

método `m1()` no se ve afectada. Llegados a este punto nos encontramos con el modelo de memoria que aparece en la figura 5.4.

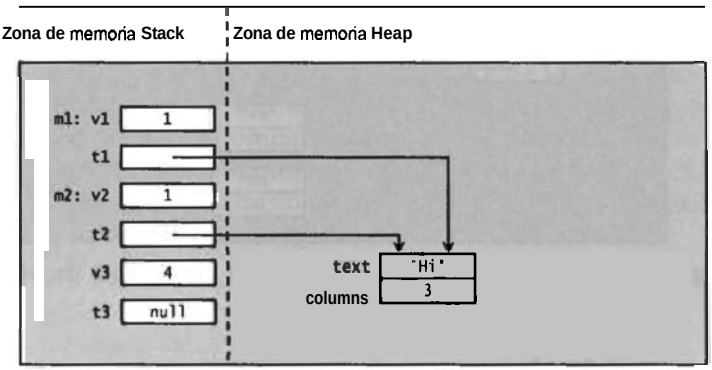


Figura 5.3. Aspecto del modelo de mernoria en el punto de control nº3

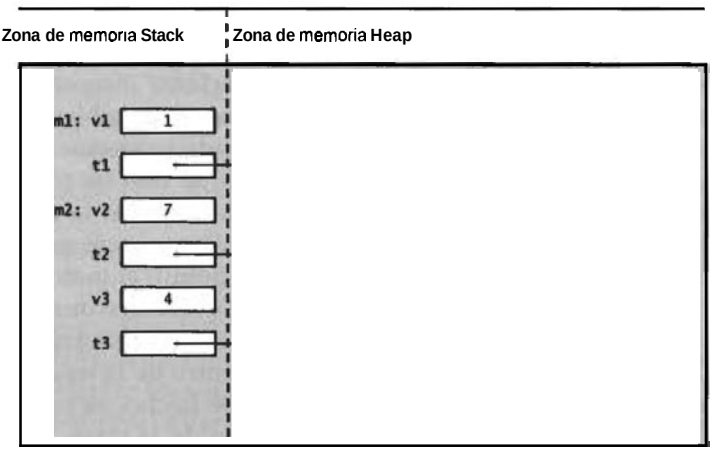


Figura 5.4. Aspecto del modelo de rnemoria en el punto de control nº4

Por último, se sale del método `m2()` y se libera la memoria ocupada por las variables locales. De esta forma se deja la variable `t3` sin ninguna referencia. Llegado el momento, el recolector de basura se encargará de buscar los objetos que se hayan dejado de utilizar para eliminarlos. En el punto de control nº5 se ejecuta una declaración de asignación. La figura 5.5 muestra el aspecto del modelo de memoria.

Nota

Para la variable `text` de tipo `string` de `TextField`, el contenido de la instancia `TextField` sólo es una variable de referencia. Los autenticos contenidos de cadena se almacenan fuera de la instancia, en algun lugar de la memoria heap.

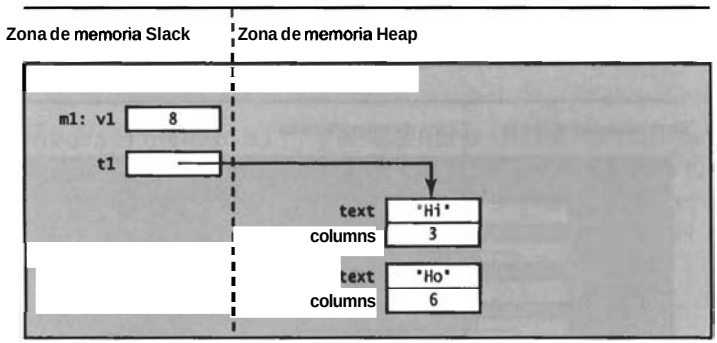


Figura 5.5. Aspecto del modelo de memoria en el punto de control nº4

Interfaces de Java

En Java, una clase sólo tiene una superclase. Las herencias multiples, donde una clase tiene una o más superclases (como ocurre con C++), no se utilizan en este lenguaje de programación. Cuando varias superclases ofrecen varias implementaciones del mismo método nos podemos encontrar con problemas, además de que las subclases retrasan la decisión del método cuando se ejecute.

En Java, las clases donde se definan métodos, se han de presentar durante la compilación de tal forma que el compilador pueda verificar la validez de cada uno de los métodos. Todas las clases a las que pueda llamar este método, tendrán que compartir una superclase de tal forma que pueda definir el método en la superclase y las clases lo anulen siempre que sea necesario. Si queremos que todas las subclases tengan su propia implementación del método, éste se tendrá que definir como abstracto. Cuanto más alto se defina el método dentro de la jerarquia de herencias, tantas clases tendrán la capacidad de anularlo. De hecho, es posible que llegue el momento en que queramos crear métodos dentro de la clase `Object`, la raiz de todas las clases, de tal forma que puedan utilizarse en cualquiera de ellas.

Aqui es donde entran en escena las interfaces de Java. Una interfaz es una colección de constantes y de métodos abstractos. Una clase puede implementar una interfaz. Para ello, todo lo que tiene que hacer es utilizar la cláusula `implements` de la clase y anular los métodos abstractos definidos dentro de la interfaz. Se puede declarar una variable como un tipo de interfaz, de forma que desde ella se pueda acceder a todas las constantes y métodos que se declaren dentro de la interfaz. A esta variable se le podrán asignar todos los objetos que se implementen dentro de la interfaz. Por lo tanto, para solucionar el problema de cómo decidir el método de implementación que se utilizará durante la ejecución, podremos definir una interfaz con un método que comparta todas las clases. Todas estas clases se declararán para implementar la interfaz y crear la propia implementación de su método. Podremos asignar casos de estas clases a las variables de la interfaz; y, durante la ejecución, se resolverán todas las referencias al método compartido.

Definir una interfaz

La definicion de una interfaz es exactamente igual que la definición de una clase, con la unica salvedad de que se sustituye la palabra clave `class` por `interface` y unicamente se permitira el uso de las constantes y de los métodos abstractos. Por defecto, todas las interfaces son abstractas. Todos los métodos que se declaran como una interfaz seran (por omisión) abstractos y publicos, por lo que no tendremos que precederlos con los modificadores `final`, `static` o `public`. Por ejemplo, el paquete `java.lang` tiene una interfaz `Runnable` con la siguiente definición:

```
interface defined as follows:
public interface Runnable {
    void run();
}
```

Aunque `run()` no tiene modificadores publicos o abstractos, el método es público y abstracto. No puede tener métodos que no sean publicos en una interfaz ni puede introducir en ella ningun comportamiento, de modo que todas las interfaces deben ser abstractas. Para ver esto por usted mismo, ejecute el comando `javap` sobre la interfaz `Runnable`.

```
>javap java.lang.Runnable
Compiled from Runnable.java
public Interface java.lang.Runnable
       /* ACC_SUPER bit NOT set */

       public abstract void run();
}
```

Las interfaces pueden formar jerarquias y clases. Una interfaz utiliza la cláusula `extends` para heredar los métodos y constantes definidos en la superinterfaz. Sin embargo, una interfaz nunca podrá tener una clase normal.

Además, la principal diferencia entre las clases y las interfaces extendidas es que con las ultimas se puede trabajar con herencias multiples. Para ello, todo lo que tenemos que hacer es colocar una lista de interfaces separadas por comas dentro de una cliusula `exteds`. Por ejemplo, el siguiente fragmento de código nos muestra la declaración de una interfaz, `ReadWritable`, que hereda otras dos interfaces, `Readable` y `Writeable`:

```
interface ReadWritable extends Readable, Writeable {
    . . .
```

Clausula implements

Una clase declara todas las interfaces que implementa en la cláusula `implements` de su declaración de clase. Ésta se compone de la palabra clave `implements`,

seguida de una lista de las interfaces separadas por comas. Y ha de aparecer detrás de la clfiusula `extends` (si hubiese alguna).

Una misma clase puede implementar más de una interfaz. Aquella que implemente una interfaz tendrd que anular todos los métodos que se declaren en ella y en todas las superinterfaces. Si no, habrá que declarar la clase como abstracta. Por ejemplo, en el siguiente fragmento de código, implementamos la interfaz `Runnable` que hemos definido anteriormente:

```
class MyClass extends MySuperClass implements Runnable {
    . . .
    public void run() {
          . .

\
```

Como todos los métodos que se han definido en la interfaz son publicos por herencia, los métodos que se utilicen para anularlos, también tendrán que ser públicos. En Java no se pueden anular métodos para obtener otros más privados. Si intentamos sobrescribir un método sin utilizar el modificador public nos encontraremos con un mensaje de error similar a éste:

```
InterfaceTest.java:5: InterfaceTest should be declared abstract;
    it does not define methods() in InterfaceTest
public class InterfaceTest implements MyInterface {
```

Si nos olvidamos de anular alguno de los métodos de una interfaz y no declaramos la clase como abstracta, nos encontraremos con un mensaje de error similar a éste:

```
OneError.java:1: class OneError must be declared abstract;
     it does not define run() in OneError.
public class OneError implements Runnable {
            ^
```

El programa del listado 5.1, en el que se proporcionan dos implementaciones diferentes de la misma interfaz, ofrece una demostración de las interfaces. Observe en el método `tryout()`, que el método no se preocupa de qué clase se le pasa, solamente se ocupa de que la clase que se pase al método implemente la interfaz `Act`. Entonces el método `tryout()` invoca al método `act()` de esta interfaz, lo que permite que cada clase ejecute su propia implementación de `Act`.

Listado 5.1. Ejemplo del uso de una interfaz

```
interface Act {
   void act();
}
class Actorl implements Act {
   public void act() {
       System.out.println("To be, or not to be");
   }
```

```
}

class Actor2 implements Act {
   public void act() {
       System.out.println("Wherefore art thou Romeo?");
   }
}

public class TryOut {
    public static void main(String args[]) {
       Actor1 hamlet = new Actor1();
       Actor2 juliet = new Actor2();
       tryout(hamlet);
       tryout(juliet);
    }
    private static void tryout(Act actor) {
       actor.act();
    }
}
```

Al ejecutar este programa se producen los siguientes resultados:

```
To be, or not to be
Wherefore art thou Romeo?
```

Conexión UML

¿Cómo se representan las interfaces en UML? Una flecha hueca al final de la línea entrecortada indica una relación de implementación de interfaz. En la figura 5.6, las clases **Actor1** y **Actor2** implementan la interfaz `Act`. Las interfaces tendrán sobre su nombre el estereotipo <<Interface>> sobre su nombre y apareceran en cursiva porque son abstractas.

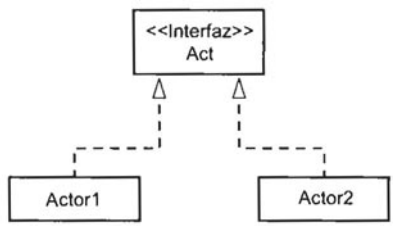


Figura 5.6. Las clases Actor1 y Actor2 implementan la interfaz Act

Trabajo con clases internas e interfaces

El concepto de trabajar con "clases dentro de clases" se introdujo por primera vez en la versión 1.1 de Java. (Anteriormente, todas las clases e interfaces debian

definirse en el nivel externo de un archivo fuente.) Las clases dentro de clases reciben el nombre de clases internas y hay cuatro tipos diferentes: clases superiores, clases de miembro, clases locales y clases anónimas.

Nota

La clase externa es la clase en la que se define una clase interna. El nombre "clase externa" solamente es figurativo y no se usa con mucha frecuencia.

Al definir clases dentro de clases, sus programas pueden organizarse de forma más lógica. Además, le permite restringir el acceso a estas clases internas porque puede usar modificadores de acceso para estas clases internas porque puede usar modificadores de acceso cuando lo considere necesario.

Clases superiores e interfaces

Las clases superiores y las interfaces son como las demás, definiciones de clase externas. La única diferencia es que los elementos superiores se declaran en otra clase o interfaz y se aiiade la palabra clave `static` antes de la palabra clave `class` o `interface`.

```
public class OuterClass {
    static class InnerClass {
```

Cuando compile el archivo fuente `OuterClass.java`, se generarán dos archivos .class, uno para cada una de las `OuterClass` e `InnerClass`.

¿Por qué haría eso? Al colocar una clase superior (o interfaz), le está diciendo a los usuarios de esa clase interior que está relacionada con otra clase externa. Por ejemplo, la interfaz Map se encuentra en el paquete `java.util`. Proporciona la definición de una estructura de datos para pares de valor de clave. La interfaz Map tiene una interfaz de `Entry` interna, que indica cada entrada de par de valor de clave en el mapa. Al estar definidas en la interfaz Map, la interfaz `Entry` está conectada de forma lógica con la interfaz Map. Esto clarifica el papel de la clase `Entry` y no intentará trabajar con una `Entry` para alguna otra estructura de datos.

¿Cómo se hace referencia a clases internas? Al estar definidas en el interior de otras clases, no puede simplemente usar su el nombre de una clase interna, a menos que esté dentro de la clase externa asociada. En vez de eso, para usar una clase superior interna (e interfaces) debe anteponer a la clase interna el nombre de la clase externa. Por ejemplo, los usuarios de la interfaz `Entry` conocen la interfaz `Entry` de la interfaz Map como `Map.Entry`. Es sencillo comprender cómo dar nombres; debe colocar un punto entre el nombre de la clase externa y el nombre de la clase interna.

Las definiciones para las clases superiores y las interfaces pueden tener modificadores de acceso si desea restringir su disponibilidad. Puede usar todos los modi-

ficadores de acceso: private, protected, public y el acceso amistoso predeterminado. Los modificadores funcionan igual para las variables de clase, pero el usuario de la clase interna también debe conseguir el modificador de acceso de la clase externa. Asi, una clase interna, superior y pública dentro de una clase externa amistosa es inaccesible desde fuera del paquete en el que se definió la clase interna.

Nombres de archivo de clase interna

Cuando se compilan clases con clases internas, el compilador debe determinar que nombre dar al archivo .class para la clase interna. Este comportamiento no está definido en la Java Language Specification sino que está controlado por los proveedores del compilador. Por suerte, todos parecen haber alcanzado la misma solución: El nombre de archivo .class para la clase `InnerClass` de la clase `OuterClass` será `OuterClass$InnerClass.class`. Solo tiene que separar el nombre de la clase externa y el nombre de la clase interna mediante el signo del dolar ($). Para el usuario de estas clases, esta descripcion no debería ser importante. Sin embargo, si está mirando a un listado de directorio, sabra de donde procede el archivo de la clase con el nombre raro.

Clases de miembro

Si no se incluye la palabra clave static de la definicidn de la clase interna, la clase será una clase de miembro y no una clase superior. La diferencia principal entre clases de miembro y clases superiores es que en las primeras tiene acceso a los campos de miembro y los métodos de la clase contenedora. Esto significa que cada instancia de la clase externa tiene su propia instancia de la clase interna.

Desde la clase interna, si necesita acceder a la instancia actual de la clase contenedora, especifique el nombre de la clase seguido por la palabra clave this, como en `OuterClass.this`. Por ejemplo, en el listado 5.2, tanto la clase externa (`MemberClass`) como la clase interna (`Counter`) tienen una variable `counter`. Para acceder a la variable `counter` de la clase externa, debe usar `MemberClass.this.counter`.

Listado 5.2. Acceso a las variables de clase **externas**

```
public class MemberClass {
    int counter = 0;
    public class Counter {
        int counter = 10;
        public void increaseCount() {
            counter++;
            MemberClass.this.counter++;
        }
        public void displayCounts() {
            System.out.println("Inner: " + counter);
```

```
            System.out.println("Outer: " + MemberClass.this.counter);
        }
    }
    public void go() {
        Counter ct = new Counter();
        ct.increaseCount();
        ct.increaseCount();
        ct.increaseCount();
        ct.displayCounts();
    }
    public static void main(String args[]) {
        MemberClass mc = new MemberClass();
        mc.go();
    }
```

Tras ejecutar este programa, obtendrá los siguientes resultados:

```
Inner: 13
Outer: 3
```

Clases locales

Las clases locales estdn definidas como clases de miembro, con una diferencia. En lugar de definirlas a nivel de clase, fuera del dmbito de los métodos, puede definirlas en el dmbito de un método o un constructor. La ventaja de usar clases locales es que asi puede definir clases en el lugar en el que las emplea y limitar mejor su accesibilidad. Si la clase está definida de forma local dentro de un método, entonces no se puede acceder a ella desde fuera del método.

¿Le parece poco intuitivo? Quizds se este preguntando, si no se puede acceder a la clase desde fuera del método ¿Por qué debemos definirla en primer lugar? Lo que suele suceder con las clases internas locales es que implementan interfaces. Desde fuera de la definición de área de clase local, los unicos elementos accesibles desde la clase son para la interfaz implementada por esa clase.

Si la clase interna local necesita acceder a una variable con dmbito local dentro del método definido, esa variable debe ser declarada final. Esto significa que la clase interna local no puede cambiar el valor de una variable primitiva, ni la referencia de una variable de objeto.

Clases anónimas

Las clases internas e interfaces anónimas llevan las clases internas locales al siguiente nivel. Con las clases internas locales, debe dar nombre a las clases internas, aunque ese nombre sea inaccesible para cualquier otra persona. Las clases internas anónimas le permiten trabajar con clases locales sin preocuparse de darles nombre. Dado que el nombre no se va a usar fuera de la clase en la que estd definida, no es necesario.

Para demostrarlo, examinemos la interfaz `ActionListener` del paquete `java.awt.event`. Se encontrard con él muy a menudo cuando trabaje con el control de eventos de componentes grdficos.

```
public interface ActionListener extends EventListener {
    public void actionPerformed(ActionEvent)
}
```

Básicamente, la interfaz contiene un método, `actionPerformed()`. Cuando implementa la interfaz, define la implementación del método. Aqui tiene un modo de implementar la interfaz con una clase anónima:

```
ActionListener listener = new ActionListener() {
     public void actionPerformed(ActionEvent actionEvent) {
         System.out.println("Anonymous class listener");
     }
};
```

Esto declara la variable `listener` de tipo `ActionListener` y proporciona su definición tras la asignación. ¿Cómo se dice nuevo `InterfaceName()`? Puede crear una implementación de la interfaz implementando completamente la interfaz, inmediatamente. Debe proporcionar definiciones de todos los métodos de interfaz.

Los componentes usan las implementaciones `ActionListener`, que se pasan al método `addActionListener()`. Básicamente, pasará la nueva instancia de `listener` al método `addActionListener()` del componente, como se indica a continuación:

```
component.addActionListener(listener);
```

Sin embargo, puede combinar la definición de la clase interna anónima y su uso y asi no necesitara la variable local. A medida que las definiciones de clase se vayan haciendo más grandes, esto será más dificil de leer y comprender, de modo que deberia usarlas con precaución desde el punto de vista del mantenimiento. No obstante, cuando trabaje con IDE como JBuilder, la herramienta generará código como este.

Cuando cree el código de forma manual, asegúrese de cerrar todos los paréntesis y corchetes. Aqui tiene una instrucción que hace esto:

```
component.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent actionEvent) {
        System.out.println("Anonymous class listener");
    }
});
```

Truco

De los cuatro tipos de clases internas, probablemente terminara usando con mas frecuencia las clases anonimas internas. Las clases de miembros y las clases superiores tambien se usan a menudo, aunque no tanto como las clases anonimas internas. Las clases locales internas apenas se usan, aunque son una opción mas.

Los paquetes de Java

Un paquete es una colección de clases e interfaces relacionadas. Como primera declaracidn no comentada de un archivo, utilizaremos `package` para especificar el paquete al que pertenecen las clases e interfaces definidos en el programa. Si no especifica un paquete, esas clases e interfaces se definirán en el paquete por defecto, sin nombre. Cómo se menciono en el capitulo 4, los modificadores de acceso le permite restringir el acceso a nivel de paquete (y clase). Asi, una de las principales ventajas de los paquetes es que sirven para controlar la visibilidad de las clases, interfaces, métodos y elementos de datos. Los paquetes también le ayudan a reducir la posibilidad de conflicto de nombres con otro programador. Por ejemplo, puede llamar a una clase `List`, aunque ya exista una clase `List` en las bibliotecas estándar de Java. (De hecho, hay dos.) El modo de diferenciar las tres clases List es mediante la inclusidn los nombres de sus paquetes con sus referencias. Tambidn puede usar la instrucción `import` para especificar una clase `List` a partir de un paquete particular que desee usar en su programa.

La declaración import

La declaración `import` nos permite hacer referencias a clases e interfaces definidas en otros paquetes. Estas declaraciones únicamente tendrán que utilizar el nombre de las clases en vez de todo el nombre completo. Pueden tener dos formas:

```
import thePackage.TheClass;
import thePackage.*;
```

donde `thePackage` es el nombre de un paquete o de una lista de paquetes anidados, donde los distintos nombres aparecerán separados por puntos, como en `java.awt.event`. En la primera forma, sólo se especifica la clase del paquete que se ha importado. En la segunda, se utilizará el nombre simple para acceder a cualquiera de las clases e interfaces que se encuentren en dicho paquete.

Nota

Es importante recordar que la declaración `import` solamente le indica al compilador donde debe buscar una definicion de clase. El compilador no es como los compiladores de C/C++ y no incorpora el resto de los contenidos de clase en la definicion de clase que se está empleando. Esto garantiza que las últimas definiciones de clase de las clases importadas se usan en tiempo de ejecucion

Las declaraciones `import` han de aparecer al principio del archivo del programa. La única declaracidn que puede aparecer antes de dsta es `package` u otra

declaración `import`. En el siguiente fragmento de programa, importaremos la clase `Random` del paquete `java.util`, asi como todas las clases e interfaces de `java.awt`, y excluiremos las que se definan dentro del subpaquete `java.awt.image`.

Por lo tanto, podremos hacer una referencia directa de `Random` a través de un nombre sencillo, como las clases definidas en el paquete `java.awt` (como `Label` y `Button`). Pero ni el paquete `java.io` ni la clase `PrintStream` se han importado, por lo que para acceder a `Printstream` tendremos que utilizar el nombre completo:

```
import java.util.Random;
import java.awt.*;
class ImportTest {
    Random r;                    // en el paquete java.util
    Label  label;                // en el paquete java.awt
    Button button;               // en el paquete java.awt
    java.io.PrintStream out;     // se ha de utilizar el nombre completo
                                 // de la clase
}
```

Resolución de los nombres de las clases

Las clases sólo pueden compartir el mismo nombre si pertenecen a paquetes distintos. Cuando desde un programa se hace referencia a un paquete, el compilador comprobará todos los paquetes que se han importado para localizar donde se define la clase.

Si la definición de la clase se ha efectuado en un unico paquete, no se producirá ningun conflicto y se utilizari. Sin embargo, si aparece en más de uno y su programa importa ambos paquetes por completo(`import thePackage.*`) tendremos que utilizar un prefijo para identificar el paquete que será la fuente de la clase. Por ejemplo, el siguiente código provocará un error porque hay clases `List` en los paquetes `java.util` y `java.awt`:

```
import java.util.*;
import java.awt.*;
...
    List list = ...;
```

Para corregir esta situación, debe usar siempre nombres con prefijo; o si sólo necesita usar una de las clases que se encuentran el varios paquetes, importe la clase especifica, además de los elementos importados. Como la clase `java.util.List` se importa de forma explicita, el compilador sabe que la variable `list` es del tipo `java.util.List` y no `java.awt.List`.

```
import java.util.*;
import java.awt.*;
import java.util.List;
...
    List list = ...;
```

Paquetes y directorios

Cuando se trabaja con paquetes, es importante comprender la descripción de los nombres de paquete para la estructura de subdirectorio~Todos los paquetes se han de convertir a un subdirectorio que tenga el mismo nombre que el sistema de archivos. Los paquetes anidados aparecerán como una jerarquia de directorios dentro de dicho sistema. Por ejemplo, si define una clase llamada `TheClass` en un paquete llamado `ThePackage`, el archivo TheClass.class estará en el subdirectorio `ThePackage`. Si hay varios niveles de subpaquetes, como en `thePackage.subPackage.TheClass`, el archivo `TheClass.class` estará en el directorio `subPackage` en el directorio `thePackage`.

Debe especificar el directorio superior de sus clases en la variable de entorno CLASSPATH. Aqui es donde el sistema debe buscar las clases de apoyo que no se encuentran entre las clases de sistema. Por defecto, el entorno de ejecución Java buscará clases en el directorio en el que se encuentre. De este modo las clases en los subpaquetes deben encontrarse en los subdirectorio~del directorio en el que esté trabajando. Sólo tiene que cambiar el valor de CLASS PATH si quiere que el entorno de ejecución busque en un lugar concreto además de en el directorio en el que se encuentre.

Los elementos de la lista de directorios están separados por puntos y coma en los sistemas Windows, y por comas en Unix. Por ejemplo, dentro de Windows, nos podriamos encontrar una linea de comandos similar a ésta dentro del archivo autoexec.bat:

```
SET CLASSPATH-.;c:\jaz\support
```

Y en Unix podriamos utilizar la siguiente linea:

```
setenv CLASSPATH .:/users/jaz/support
```

En ambos casos, el punto al principio de la definición indica que la aplicación debe buscar en primer lugar en el directorio en el que se encuentre en ese momento, antes de buscar más archivos de clase en el directorio `C:\jaz\support` o `/users/jaz/support`. También puede especificar archivos JAR en la ruta de la clase.

Todas las clases integradas de Java (es decir, las API Core de Java)se encuentran en el paquete `java`. Se pueden encontrar m6s clases estándar en el paquete `javax`. Evite colocar más paquetes en estas jerarquias.

Nota

Además de cambiar la ruta de la clase estableciendo la variable de entorno CLASSPATH, puede especificar el valor de la ruta de clase para una ejecucion especifica del comando `java` usando las opciones de linea de comando `-classpath` o `-cp`.

Declaración package

¿Cómo se coloca una clase en un paquete? La forma general de la declaración `package` es la siguiente:

```
package nombre;
```

donde `nombre` puede ser el nombre de un paquete o una lista de nombres separados por puntos. Por ejemplo, las dos declaraciones siguientes son completamente legales:

```
package myPackage;
package java.awt.image;
```

En un archivo de programa unicamente puede aparecer una declaración `package` y ésta ha de estar en la primera linea no comentada del archivo. Si se omite, las clases que se generen se guardartin dentro de un paquete predeterminado que no tendrti nombre. Aunque este paquete nunca tendrti que importarse, es aconsejable evitar usarlo en programas. Es como instalar todos los programas de Windows directamente en el directorio `C:\`, en lugar de en directorios separados para cada programa. Deberti llamar a los paquetes con el nombre de dominio al revés. Por ejemplo, uno de mis dominios es `zukowski.net`. Deberia llamar a mis paquetes con `net.zukowski` y luego agregar subpaquetes para agrupar los elementos en consecuencia.

Compilar y ejecutar codigo empaquetado

Aunque la declaración `import` le permite usar clases en otros paquetes, ejecutar código empaquetado requiere un poco de trabajo extra. Como se dijo anteriormente, las clases en los paquetes deben colocarse en una estructura de directorio especial. Esto puede ser un trabajo muy pesado cuando tiene muchas clases y muchos paquetes diferentes. Por suerte, la opción `-d` del compilador `javac` le ayuda.

Cuando compila código fuente Java, puede especificar un directorio en el que desea generar los archivos `.class`.

Asi, si tiene `package thePackage.subPackage; statement` en la definición de `TheClass`, si compila `TheClass.java` con la opción `-d`, colocará `TheClass.class` directamente en el directorio `subPackage`, situado en el directorio `thePackage`.

El nivel superior de ese directorio se especifica tras la opción `-d`. Si ese directorio es el directorio "punto", los archivos de clase estartin en el directorio en el que se encuentre trabajando, como se muestra aqui:

```
javac -d . TheClass.java
```

Cuando tenga la clase empaquetada en el directorio correcto y tenga el directorio superior adecuado en la ruta de la clase, puede ejecutar la clase con el método

`main()`. Si pasa la clase completa y su ruta de acceso al comando `java`, se ejecutará:

```
java thePackage.subPackage.TheClass
```

Pruebe esto con el siguiente código fuente:

```
package thePackage.subPackage;

public class TheClass {
    public static void main(String args[]) {
    System.out.println("Running from a package");

}
```

Convenciones recomendadas

Estructurar de forma lógica sus aplicaciones en clases, interfaces y paquetes le ayudará en el mantenimiento a largo plazo de las aplicaciones. Aqui tiene algunas reglas para dar nombre a sus clases, interfaces, paquetes, métodos y variables.

- Clases e interfaces:
  - Los nombres deben ser descriptivos, normalmente en forma de nombre para las clases y de nombre o adjetivo para las interfaces.
  - Los nombres deben comenzar con una letra mayúscula.
  - Los nombres formados por varias palabras deben contener varias letras mayúsculas, como `OneTwoThree`.
- Paquetes:
  - Los nombres deben tener una sola palabra.
  - Los nombres deben estar formados por letras minusculas.
  - Es aconsejable usar siempre paquetes.

Nota

Sí, tiene razon, la ortografia de `thePackage` y `subPackage` no sigue estas normas. Estos nombres se escogieron por su legibilidad y como meros ejemplos.

- Métodos:
  - Los nombres se deben basar en lo que hacen.
  - Los nombres deben comenzar con letra minúscula.

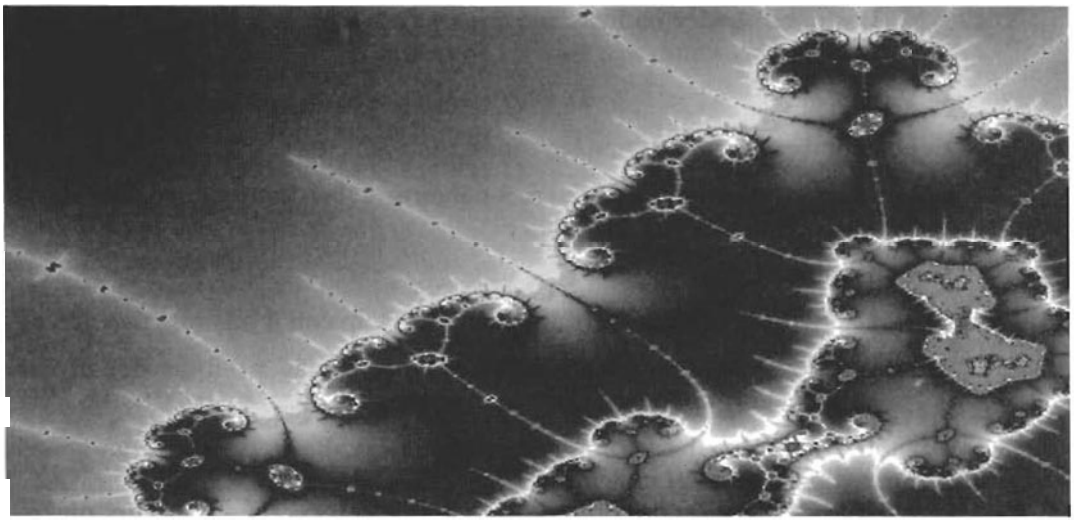
  - Los nombres formados por varias palabras deben contener varias letras mayusculas, como `actionPerformed`.
- Variables:
- Los nombres deben comenzar con letra minúscula.
- Los nombres formados por varias palabras deben contener varias letras mayúsculas, como `returnStatus`.
- Constantes:
  - Los nombres deben escribirse por completo con letras mayusculas.
  - Los nombres de variable formados por varias palabras deben usar el guión bajo (_) para separar palabras, como en `EXIT_ON_CLOSE`.

Cuando comience a desarrollar componentes JavaBean, deberri tener en cuenta otras normas para la programación.

Para temas de creación de código, como dónde colocar llaves, no dire qué modo es mejor (al final de la linea o al comienzo de la siguiente linea). Elija el que más le agrade, pero para poder realizar mejor el mantenimiento del código, sea coherente. Si quiere conocer más convenciones sobre código, puede ir a

- Convenciones de código de Sun para la programación con Java en http://java.sun.com/docs/codeconv/html/CodeConvTOC.doc.html
- Cómo escribir código imposible de mantener. Articulo de Roedy Green en http://www.mindprod.com/unmain.html

Además, el uso de herramientas como JIndent le ayudará a mejorar las convenciones de codificación y elementos de espaciado más allá de simples elecciones de nombre. Encontrarri una versión de prueba de JIndent en el CD que acompaiia a este libro.



6
Arrays y declaraciones de control de flujo

De momento hemos visto ejemplos de programas que únicamente utilizaban variables simples y se ejecutaban de forma secuencial. Todas las variables eran tipos de datos primitivos como `int` o `float` u objetos llenos de variables primitivas. Sin embargo, si sólo usa estos constructores sus programas sólo podrán realizar operaciones muy simples.

Java le permite trabajar con grupos de elementos similares en un constructor de programas llamado matriz (array). A medida que estudia y trabaja con estos objetos en este capitulo, tambikn conocerá las diferentes instrucciones de control de flujo. (Otro modo de trabajar con grupos de objetos son las colecciones, que se estudiarán en el capitulo 17.)

Arrays en Java

En los programas de Java se suelen emplear arrays. En esta sección le mostraremos cómo trabajan mientras le enseñaremos a crearlos y a introducir y sacar elementos de ellos. Aunque, primero debemos saber qué es un array.

Como en otros lenguajes, un array es un modo de manipular un grupo de variables del mismo tipo que tienen un nombre en común. El tipo puede ser primitivo como `int`, o de otro tipo como `String`. Ya ha visto una matriz de objetos `String` en la declaración del mktodo `main()` para una aplicación:

```
public static void main(String args[])
```

En este caso, `args` es una matriz de elementos `String`. Estos elementos proceden de la linea de comandos y los `[]` indican que es un array.

Al igual que muchos otros elementos de Java, los arrays son objetos. Tienen una variable de instancia: `length`. De modo que para descubrir cuantas líneas de comando de argumentos se han pasado a un programa, deberá acceder a la variable `length` del parametro declarado para `main()`. Para la declaración `main()` anterior, ésta se realizaría con `args.length`, como muestra el siguiente ejemplo.

```
public class ArgLength {
    public static void main(String args[]) {
        System.out.println("Received " + args.length + " arguments");
    }
}
```

Si invocó al programa `ArgLength` con la siguiente linea de comandos:

```
java ArgLength one two three
```

al ejecutar el programa se producirán los siguientes resultados:

```
Received 3 arguments
```

Acceder a los elementos de un array

Los corchetes (`[]`) se incluyen en una declaración de matriz (`String args[]`) porque debe colocar algo en su interior cuando accede a un elemento de matriz. Ese algo es un número entero que indicará el índice de los elementos a los que se debe acceder, como en

```
System.out.println(args[2]);
```

Advertencia

El índice de un array no puede ser una variable `long`. Debe ser `int` o inferior. En caso contrario, se producira un error al compilarlo.

Java verifica todos los subscripts en tiempo de ejecución, para asegurarse de que están dentro del rango legal. Las matrices de Java comienzan sus subscripts por cero, hasta la longitud de la matriz menos uno. Asi, si usa los indices de 0 a 4, accederá a todos los elementos de una matriz de cinco elementos.

Nota

Si especifica los indices de una matriz mas allá de los límites validos, se arrojará una excepción `ArrayIndexOutOfBoundsException` en tiempo de ejecucion.

Declarar y crear arrays

La creacidn de sus propias arrays es algo mds complejo que acceder a argumentos de linea de comandos, porque en la declaracidn y creacidn de una matriz intervienen dos pasos separados.

Declaración

Para la declaracidn, aiiada corchetes (`[]`) a la declaracidn de una variable normal. Tiene varias opciones de formato; estas tres formas de colocar los corchetes es correcta:

- tipo[] variable;
- tipo []variable;
- tipo variable[];

Los primeros dos formatos indican que la variable es de tipo `tipo[]` y los ultimos indican que la variable es un array y que la array es de tipo `tipo`.

La diferencia entre estos formatos es importante cuando declara varias variables con una linea de declaracidn. En estos ejemplos:

```
int[] visual, basic[];
int []visual, basic[];
```

la variable `visual` estd declarada para ser una matriz unidimensional de tipo `int` y la variable `basic` está declarada para tener dos dimensiones. Pero cuidado, la segunda forma puede crear confusión porque parece que la variable `basic` sólo tiene una dimensión. Descubrird más sobre las matrices con varias dimensiones a lo largo del capitulo.

Cruacion e inicialización

Declarar una variable de array es simplemente eso: Le indica al compilador que existe una variable del tipo adecuado en su programa. Tras declarar una matriz, hay dos modos de crearla. Puede crear la matriz con sus valores, mediante un inicializador de arrays o puede crearla usando la palabra clave `new`, indicando el tamaiio de la matriz.

Con el inicializador de matriz, proporciona una lista de valores separados por comas entre llaves `{}`. Por ejemplo, la siguiente matriz contiene los nombres de los meses del aiio:

```
String monthNames[] = {"January", "February", "March", "April", "May",
  "June", "July", "Auqust", "September", "October", "November",
"December");
```

Como hay 12 cadenas en la lista, la matriz `monthNames` tiene un tamaiio capaz de contener 12 elementos.

Si no quiere inicializar la matriz cuando la crea, puede crearla usando la palabra clave new y especificar el tamaño y tipo de array:

```
int monthDays[] = new int[12];
```

Cuando no inicializa una matriz de forma explícita con un inicializador, la matriz se rellena con valores por defecto. El valor por defecto suele ser el equivalente a cero para el tipo de datos apropiado, como muestra la tabla 6.1.

Tabla 6.1. Valores de relleno de matriz por defecto

Tipo de datos	Valor de relleno por defecto
byte, short, int, long	0
float, double	0.0
char	\u0000
boolean	false
Object type	null

Tras crear la matriz, puede inicializar la matriz de forma manual asignando valor a cada elemento, como en `monthDays[0]=31;`. O puede usar una segunda forma del inicializador de arrays, que se encuentra en el lado derecho de una declaración de asignación:

```
int monthDays[];
monthDays = new int[] {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
```

Observe que en este caso, no tenemos que crear la matriz en primer lugar porque se crea una matriz de enteros del tamaño apropiado con el inicializador.

Uso de la memoria

Tras declarar una matriz, se reserva espacio en la memoria stack para la referencia de la matriz. La figure 6.1 muestra el modelo de memoria tras la declaración de array: `int monthDays[]`. Tras usar la palabra clave new para colocar espacio, se reserva memoria heap para el contenido de la matriz. Con un array de primitivos, este espacio se usa para los elementos reales de la matriz. La figura 6.2 muestra el modelo de memoria tras colocar el espacio `monthDays = new int[12]`.

Por otra parte, para una matriz de objetos, el espacio colocado es para contener una referencia al objeto, no el objeto en sí. Para comprender este concepto un poco mejor, vamos a definir una clase `Calendar` que contenga el nombre del mes y un número de días.

```
public class Calendar {
    String name;
```

```
int days;
Calendar (String name, int days) {
    this.name = name;
    this.days = days;
}
```

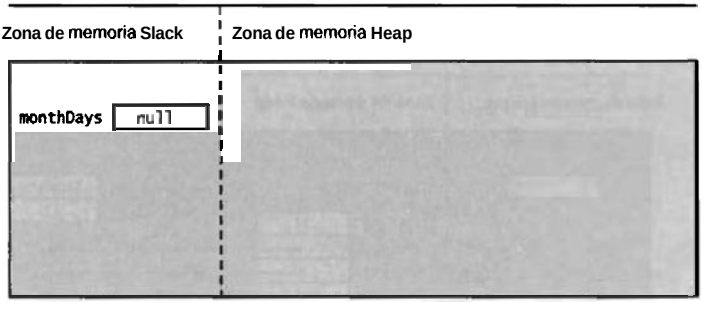


Figura 6.1. Tras declarar un array

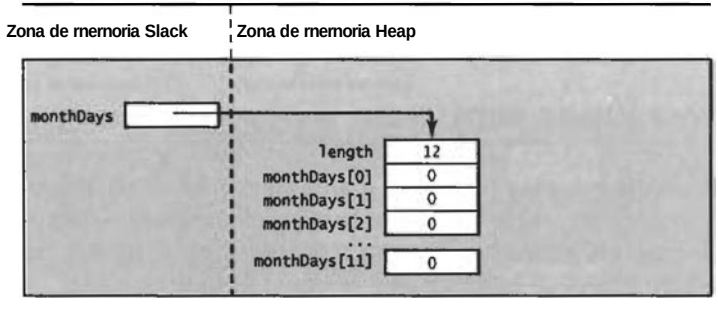


Figura 6.2. Tras colocar espacio para elernentos de rnatriz que son primitivos

Tras definir la clase, puede crear una rnatriz para esos elementos: **`new Calendar[12]`**. El modelo de memoria tendrá el aspecto de la figura 6.3 tras colocar espacio **`Calendar cal[] = new Calendar[12].`**

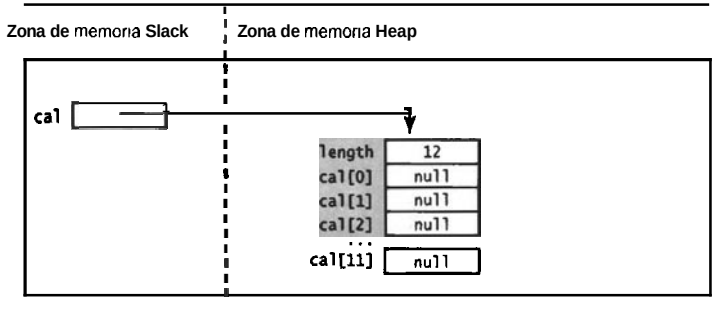


Figura 6.3. Tras colocar espacio para elernentos de rnatriz que son objetos

Observará que adn no hay espacio reservado para objetos `Calendar`. Todo lo que hay en el array es una referencia null. Hasta que no reserve espacio mediante new `Calendar(...)` y agregue el objeto a la matriz mediante `cal[x] = ...` el modelo de memoria no se verá afectado. La figure 6.4 muestra el modelo tras ejecutar el siguiente código:

```
cal[0] = new Calendar("January", 31);
cal[1] = new Calendar("February", 28);
```

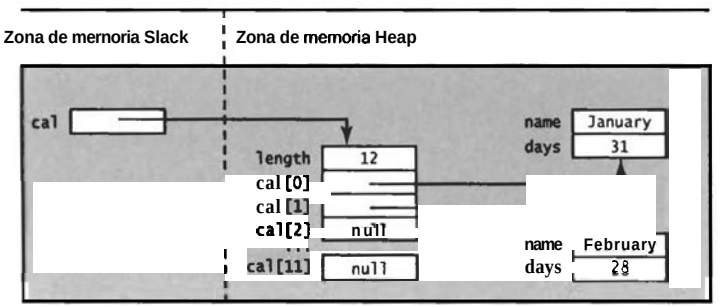


Figura 6.4. Tras agregar objetos Calendar

Copiar y clonar arrays

A veces necesita copiar un array, manteniendo el original y modificar la copia. Aunque la asignación entre variables funciona correctamente cuando necesita copiar el valor de un primitivo de una variable a otra, no funciona con arrays (o cualquier objeto) porque lo dnico que se copia es la referencia:

```
Calendar cal2 = cal;
```

Esto deja al modelo de memoria con dos variables haciendo referencia al mismo espacio en memoria, como muestra la figura 6.5. Asi, si cambia los contenidos de una matriz, los cambios aparecerán reflejados en la otra.

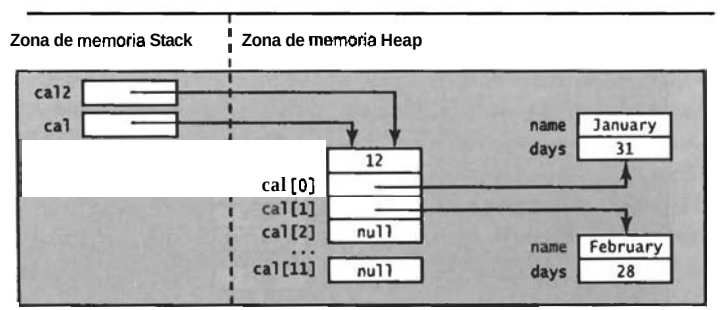


Figura 6.5. Tras la asignación

Para copiar los valores que se encuentran dentro de la matriz (o una parte de ellos) tiene dos opciones. Puede crear de forma manual la segunda matriz y colocar los elementos originales en la matriz, o puede utiliza el método `arraycopy()` de la clase System. Cuando use `arraycopy()`, aun tiene que crear la segunda matriz, pero el método realiza la copia. Esta es la forma del método:

```
public static void arraycopy(Object source,
                             int sourceStart,
                             Object destination,
                             int destinationStart,
                             int length)
```

Si queria copiar toda la matriz `Calendar` en una nueva matriz, crearia la nueva matriz y luego invocaria `System.arraycopy()`:

```
Calendar cal2 = new Calendar[cal.length];
System.arraycopy(cal, 0, cal2, 0, cal.length);
```

La figura 6.6 muestra la zona de memoria tras copiar la matriz. Para una array de primitivos, los contenidos de la matriz se copian. Sin embargo, como muestra la figura 6.6, con una matriz de objetos, sólo se copian las referencias a los objetos.

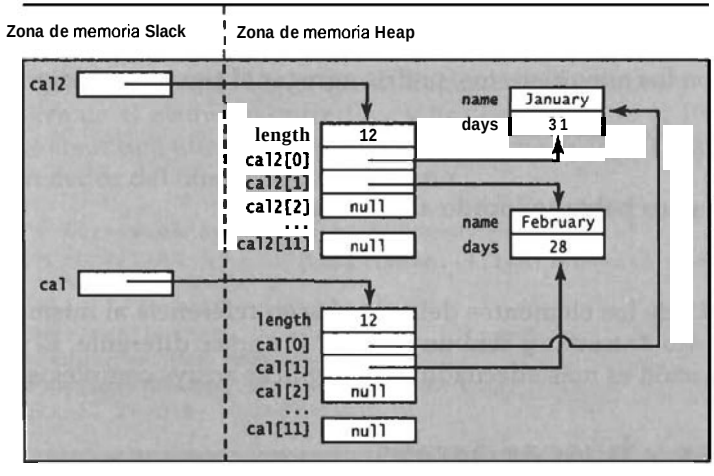


Figura 6.6. Tras copiar una matriz de objetos

Otro modo de copiar una matriz completa es invocar al método `clone()` de la matriz. El método `clone()` se pone en funcionamiento cuando quiere copiar una matriz completa (al contrario que `arraycopy()`, que puede trabajar con matrices parciales).

```
Calendar cal2[] = (Calendar[])cal.clone();
```

El método procede de la interfaz `Cloneable`, que implementan muchas de las clases de sistema. De hecho, el método `clone()` está definido en la clase `object`

pero está protegido. Como muestra el siguiente ejemplo, si quiere que su clase implemente la interfaz, agregue la clausula `implements` adecuada y defina el método. El método sólo tendrá que llamar al método invalidado.

```
public class Calendar implements Cloneable {
...
    public Object clone() throws CloneNotSupportedException {
        return super.clone();

    ...
}
```

El capitulo 7 explica el control de excepciones (la clausula `throws CloneNotSupportedException`) con más detalle. Para clonar calendarios, necesitará agregar el codigo en negrita de la siguiente linea al final de la declaración del método `main()` para la clase `Calendar`:

```
public static void main (String args[]) throw:  Exception {
```

El método `arraycopy()` y el mecanismo de clonación realizan una copia superficial de la matriz, en la que sólo se copian los elementos de la matriz o sus referencias. Si desea hacer una copia mas profunda, en la que también se copien todos los elementos de la matriz, tendrá que usar `arraycopy()` y recorrer la matriz creando de nuevo cada elemento. Por ejemplo, si quiere que la variable `cal2` funcione con los años bisiestos, podría agregar el siguiente código:

```
cal2[1] = (Calendar)cal[1].clone();
cal2[1].days = 29;
```

O podriamos haber invocado al constructor:

```
cal2[1] = new Calendar(cal[1].name, 29);
```

Ahora, 11 de los elementos del array hacen referencia al mismo elemento, pero cada elemento `February` será un objeto `Calendar` diferente. El uso del mecanismo de clonacion es más adecuado para duplicar arrays complejos.

Ordenar y buscar arrays

La clase `Arrays` del paquete `java.util` incluye elementos para ordenar matrices y realizar una busqueda binaria de elementos en las matrices ya clasificadas.

Para ordenar una matriz completa, invoque al método `sort()` de la variable de matriz. Por ejemplo, para ordenar la matriz de nombres de mes, deberia hacer lo siguiente:

```
Arrays.sort(monthNames);
```

La matriz se ordenará de modo que `monthNames[0]` será `April` y `monthNames[11]` sera `September`. Este tipo de ordenación funciona para todas

las matrices de primitivos y algunos objetos, como `String`. Si el objeto implementa la interfaz `Comparable`, se puede ordenar. Puede usar el método `binarySearch()` para comprobar la existencia de un elemento en una matriz ordenada. La matriz debe estar ordenada porque el algoritmo de bdsqueda divide constantemente la matriz en dos partes, desechando una mitad cada vez, hasta que encuentra el elemento. Esto es más rfipido que recorrer de forma manual una matriz de principio a fin hasta que se encuentra el elemento, pero exige que la matriz esté ordenada. Por ejemplo, aqui tiene cómo encontrar la posición en la que aparece el primer mes del afio en una matriz de nombre de mes ordenada:

```
int position = Arrays.binarySearch(monthNames, "January");
```

Al buscar `January` de esta forma, aparecerá en la quinta celda de la matriz o en el índice 4 (tras `April`, August, March y May). Cómo se mencionó anteriormente, la busqueda sólo funciona en matrices de primitivos o en matrices de objetos en los que el objeto implementa `Comparable`. Simplemente, pase la matriz y la clave de búsqueda. El método `binarySearch()` devuelve un número negativo si no se encuentra el elemento en la matriz. De hecho, puede usar este numero negativo como punto de inserción para agregar el nuevo elemento en la matriz. Dado que no se puede cambiar el tamafio de una matriz, tendrfi que insertar el nuevo elemento en una nueva matriz, copiando los elementos apropiados del original.

Por ejemplo, si intenta insertar `"New Month"` en la matriz `monthNames`, `binarySearch()` devolverá -10, lo que indica que no se encontró `New Month` y que se encuentra en el elemento entre May y November como el 10º elemento (9º posición). Tras crear una nueva matriz más grande, puede copiar la matriz original por partes, alrededor del nuevo elemento, corno:

```
int position = Arrays.binarySearch(monthNames, "New Month");
int newIndex = -position - 1;
int length = monthNames.length;
String newMonths[] = new String[length+1];
System.arraycopy(monthNames, 0, newMonths, 0, newIndex);
newMonths[newIndex] = "New Month";
System.arraycopy(monthNames, newIndex, newMonths,
       newIndex+1, length-newIndex);
```

Nota

Existen otras dos rutinas de aplicacion general en la clase `Arrays`. Puede usar el método `equals()` para comprobar la igualdad entre dos matrices o el método `fill()` para completar una matriz con un elemento repetido.

Uso de matrices multidimensionales

En Java no existen las matrices multidimensionales. Si necesita trabajar con matrices en varias dimensiones, cree una matriz de arrays, simplemente agregando

otro conjunto de corchetes a la declaración. Por ejemplo, el siguiente fragmento de código define una matriz que combina `monthNames` y `monthDays`, tratando todo como cadenas:

```
String months[][] = {
  {"January", "31"},
  {"February", "28"},
  {"March", "31"},
  {"April", "30"},
  {"May", "31"},
  {"June", "30"},
  {"July", "31"},
  {"August", "31"},
  {"September", "30"},
  {"October", "31"},
  {"November", "30"},
  {"December", "31"}};
```

En este caso, `months.length` sera igual a 12 porque hay 12 subarrays de cadenas dentro del array más externo. Cada array interno tendrá una longitud de 2 porque hay dos elementos en cada uno de estos arrays. Se puede acceder a January comomonths[0][0], donde June será `months[5][O]`.

No es obligatorio que las matrices multidimensionales sean rectangulares. Cada matriz interna tiene su propia longitud. Por ejemplo, aqui tenemos una matriz multidimensional con forma de diamante declarada e inicializada. El modelo de memoria aparece en la figura 6.7.

```
int multi[][] = {{1}, {2, 3}, {4, 5, 6}, {7, 8}, {9}};
```

En este array tiene diferentes longitudes:

`multi.length` es 5

- `multi[0].length` y `multi[4].length` son 1

  `multi[1].length` y `multi[3].length` son 2

  `multi[2].length` es 3

Si observa la figura 6.7 vera que la memoria colocada para los arrays no tiene que estar contigua, como ocurre en otros lenguajes de programación.

Uso de declaraciones para el control de flujo

Agreguemos otra dimensión a nuestro aprendizaje del lenguaje de programación Java. Java contiene una serie de declaraciones de control de flujo que le permiten ejecutar algunas líneas de código de forma selectiva o repetitiva. Estas declaraciones se basan en las declaraciones de control de otros lenguajes, de modo que si ya conoce lenguajes como C o C++, comprenderá facilmente estos constructores:

Las declaraciones `if` y `switch` se utilizan para efectuar una ejecución selectiva de los segmentos de código.

Las declaraciones `for`, `while` y `do` se utilizan para ejecutar repeticiones de distintos segmentos de código.

Las declaraciones `break`, `continue` y `return` se utilizan para interrumpir el flujo de datos.

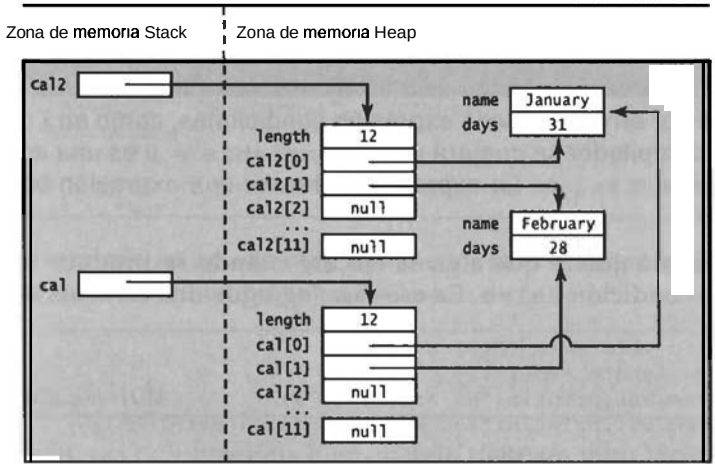


Figura 6.7. Modelo de rnernoria para una rnatriz multidimensional con forrna de diarnante

Declaraciones condicionales

Las declaraciones condicionales nos permiten efectuar una ejecución selectiva con `if` y `switch`. También hay un operador terciario (?:) para ayudar a simplificar los bloques de código `if-else`.

Declaración *if*

La declaración condicional m6s sencilla es el bloque `if` que permite la ejecución si la expresidn condicional es `true`. Por ejemplo, para comprobar los argumentos que han pasado desde la linea de comandos, compruebe si la longitud de la matriz que se pas6 a `main()` es 0:

```
public static void main (String args[]) {
    if (args.length == 0)
       System.out.println("No arguments");
```

Para todas las expresiones condicionales de `if`, la expresidn parentética debe evaluar un valor booleano `true` o `false`. Si la expresión resulta ser `true`, se

ejecuta la declaración que sigue a la cláusula if. Si quiere que se ejecuten declaraciones multiples, debe poner las declaraciones entre llaves, como se muestra aqui:

```
public static void main (String args[]) {
    if (args.length == 0) {
       System.out.println("No arguments");
       System.out.println("You should enter something");
    }
}
```

Truco

Si coloca por error un = en la expresion condiciones, como en `if (variable = 0)`, el compilador se quejara porque `variable = 0` es una asignacion y no evallja `true` o `false`. La expresion debe ser una expresión booleana.

Digamos que quiere que algo se ejecute cuando se produce una condición de `true` y una condición `false`. En ese caso, agregue una clausula `else`:

```
public static void main (String args[]) {
    if (args.length == 0) {
       System.out.println("No arguments");
       System.out.println("You should enter something");
    } else { // args.length != 0
       System.out.println("The first argument is " + args[0]);
    }
}
```

La parte `else` también puede ser otra condición `if`, si quiere responder a varias condiciones:

```
public static void main (String args[]) {
    if (args.length == 0) {
       System.out.println("No arguments");
       System.out.println("You should enter something");
    } else if (args.length == 1) {
       System.out.println("The first argument is " + args[0]);
    } else { // args.length >= 2
       System.out.println("The last argument is " + args[args.length-1]);
    }
}
```

El operador terciario

El operador terciario (`? :`) recibe este nombre porque toma tres operandos. Funciona como una declaracion `if-else`, comprimido en una línea de codigo. La expresion que evallja un booleano se coloca antes del signo de interrogacion (`?`). Si la expresion resulta ser `true`, la declaracion antes de los dos

puntos (:) se ejecuta. En caso contrario, se ejecuta la declaración tras los dos puntos.

```
(expresión) ? (declaración true) : (declaración false);
```

Las declaraciones `true` y `false` que se van a ejecutar pueden devolver cualquier tipo, siempre que sean equivalentes. Por ejemplo, para ver si el primer argumento que se pasó desde la línea de comandos es el nombre de un mes, puede usar el siguiente operador terciario y que devuelva Yes o No:

```
String monthNames[] = {"January", "February", "March", "April", "May",
"June",
     "July", "August", "September", "October", "November", "December"};
Arrays.sort(monthNames);
String msg =
   (Arrays.binarySearch(monthNames, args[0]) >= 0) ?
         "Yes" : "No";
```

El operador terciario no se usa muy a menudo, pero a veces es justo lo que se debe usar en lugar de un bloque `if-else`.

Declaraciones switch

La declaración `switch` funciona bien cuando tiene un gran bloque de declaraciones `if-else` que estdn buscando un valor entero especifico (o algo que puede representarse mediante una variable `int`, como `char`, `byte` o `short`, pero no `long`, `float`, `double` o `boolean`). Examine este bloque `if-else`:

```
int days = …;
if (days == 28) {
   doMonth28();
} else if (days == 30) {
   doMonth30();
} else if (days == 31) {
   doMonth31();
} else {
   doLeapYear();
}
```

Algunas personas consideran la declaración `switch` un modo mds limpio de hacer las cosas. La condición que estd cotejando es la variable de cada una de las comprobaciones `if`:

```
switch (variable)
```

donde cada valor que estd comprobando es el valor especifico (`case valor:`) o la cldusula final (`default:`). Entonces, por cada vez que el valor de la variable concuerde con el que buscamos, la declaración `switch` comienza a ejecutar el código de la etiqueta.

Observernos el punto en el que nos encontramos ahora antes de seguir con las explicaciones. Aqui tiene el bloque `if` anterior como un bloque `switch` incompleto:

```
switch (days) {
    case 28:
      doMonth28();
    case 30:
      doMonth30();
    case 31:
      doMonth31();
    default:
      doLeapYear();
}
```

Este bloque está incompleto porque hay un problema que debe ser corregido. La declaración `switch` comienza a ejecutarse en el `case` adecuado. En realidad sigue ejecutándose hasta que algo la detiene, de modo que aunque haya 30 dias en un mes, los métodos `doMonth30()`, `doMonth31()` y `doLeapYear()` se ejecutarian. Para detener la ejecución, debe agregar la palabra clave `break` después de cada bloque de código que quiera ejecutar. Aqui tiene el mismo bloque `switch` pero con el problema corregido (aunque en algunos casos puede querer que no se produzca la ejecución, como explicaremos en breve):

```
switch (days) {
    case 28:
       doMonth28();
       break;
    case 30:
       doMonth30();
       break;
    case 31:
       doMonth31();
       break;
    default:
       doLeapYear();
       break;
```

Truco

No es obligatorio que el caso `default` este en ultimo lugar, pero en beneficio de la legibilidad, suele encontrarse en esa posición. Aunque puede colocarlo en primer lugar y se ejecutara correctamente, es un poco extraño ver el caso general antes de las elecciones. La ultima clausula `final` es opcional porque ya no queda mas por ejecutar.

El listado 6.1 muestra la capacidad de escapar de la declaración `switch`. Podemos crear un bloque `switch` para que nos indique el número de dias, dependiendo

del mes (y el aiio, para controlar los aiios bisiestos). Aqui, month puede ser un numero entre 1 y 12 y vamos a establecer la variable days dependiendo del mes que se introduzca. Observe que los casos para April, June, September y November (30 dias) comparten el mismo bloque de código para ejecutar.

Listado 6.1. Uso de una declaración switch para calcular los dias del rnes

```
int getDays(int month, int year) {
   int days;
   switch (month) {
         case 4:  // April
         case 6:  // June
         case 9:  // September
         case 11: // November
            days = 30;
            break;
         case 2:  // February
             if (((year% 4 == 0) &&
                    (year % 100 != 0)) || (year % 400 == 0)) {
               days = 29;
             } else {
               days = 28;
             }
             break;
         default: // rest
             days = 31;
             break;
     }
      return days;
}
```

Declaraciones para bucles

Java proporciona varios constructores para bucles para cuando quiera ejecutar bloques de código repetidamente. Hay tres diferentes constructores disponibles, for, while y do-while, aunque, hablando técnicamente, sólo una es necesaria. Si sólo existe una de estas tres estructuras, puede simular el comportamiento de las otras dos empleando de forma selectiva bloques if.

Declaraciones for

La declaración for se usa cuando se necesita ejecutar una declaración (o un bloque de declaraciones si usa llaves) varias veces, donde hay un simple paso de inicialización, una condición de finalización y un paso de actualización. Durante el paso de actualización se cambia algún valor, acercando ese valor a la condición de finalización.

```
for (initializacibn; condicibn; actualización)
       cuerpo;
```

Cuando el bucle `for` se ejecuta, realiza el paso de inicializacion. A continuación comprueba la condición. Cuando el cuerpo termina de ejecutarse, se ejecuta el paso de actualización. Después de eso, se comprueba de nuevo la condición. Este bucle continua hasta que la condición resulte `false`. La figura 6.8 le ayudará a visualizar este proceso.

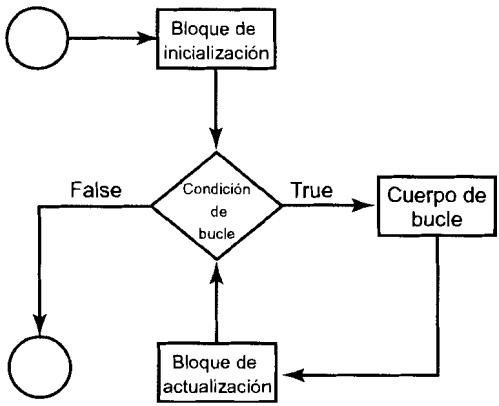


Figura 6.8. El flujo de bucle for

Para mostrar el bucle `for`, crearemos un bucle para ver todos los argumentos de linea de comandos. Aqui, vamos a crear una variable `i` e inicializarla con cero para el paso de inicializacion. Esta variable será local para el bucle `for` e inaccesible fuera del bucle. Nuestra condición para el bucle se asegurará de que `i` no ha crecido demasiado y ha pasado el limite de la matriz. El paso de actualización incrementa la variable y el cuerpo escribe el valor en el índice de la matriz especifica:

```
for (int i=0; i < args.length; i++) {
   System.out.println(args[i]);
}
```

Cómo la longitud de la matriz `args` no va a cambiar mientras ejecuta el bucle, podemos atraparlo en el bloque de inicialización y guardarlo en una variable local. Si quiere tener varias declaraciones en el bloque de inicializacion o de actualizacion, puede separarlas por comas, como se muestra aqui:

```
for (int i=0, n=args.length; i < n; i++) {
   System.out.println(args[i]);
}
```

o incluso esto:

```
for (int i=0, n=args.length; i < n; System.out.println(args[i]), i++);
```

También puede mover el autoincremento de `i` en la declaración `println`:

```
for (int i=0, n=args.length; i < n; System.out.println(args[i++]));
```

Advertencia

Aunque estos dos últimos bucles `for`, con `System.out.println()` en la parte de actualización, funcionan correctamente, no recorniendo usar ninguno de los dos formatos. Tan pronto como necesite agregar otra declaración al bucle, sera necesario carnbiar demasiadas cosas y es facil cometer errores.

Declaraciones while

El bucle **`while`** es más sencillo que el bucle `for` porque sólo tiene una condición y un cuerpo. La condición se comprueba en primer lugar y si se cumple, el cuerpo se ejecuta antes de comprobar la condición de nuevo:

```
while (condición)
     cuerpo;
```

Por lo general, el cuerpo termina cambiando el estado de lo que se comprueba en la condición, aunque si la condición siempre resulta ser cierta, nunca dejará de ejecutarse. Al igual que el bucle `for` el cuerpo del bucle **`while`** nunca se ejecuta si la condición no se cumple inicialmente. La figura 6.9 muestra el flujo general del bucle `while`.

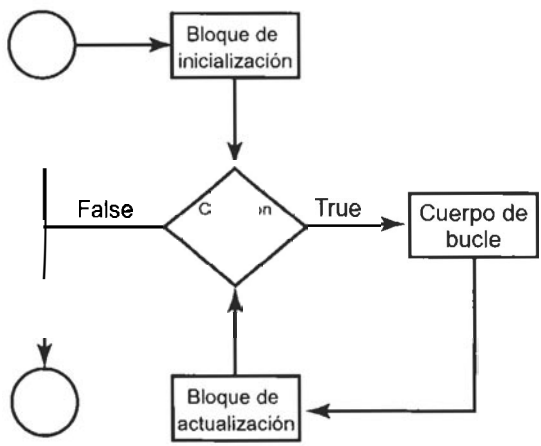


Figura 6.9. El flujo de bucle while

Para mostrar el bucle **`while`**, resumiremos el número de dias del aiio, usando el método `getDays()` definido en el listado 6.1:

```
int sum = 0;
int i = 1;
while (i <= 12) {
     sum += getDays(i++, 2002);
```

Este cddigo funciona de forma muy parecida al bucle **for.** Tenemos que recordar mantener la variable sum visible fuera del bucle; en caso contrario, el valor calculado desapareceria cuando el bucle finalizara.

```
int sum = 0;
for (int i=1; i<=12; i++) {
   sum += getDays(i, 2002);
}
```

Bucles for contra bucles while

La elección entre un bucle **for** y un bucle **while** depende de las preferencias personales y basandose en la condición final. El bucle **for** es seguramente la rnejor elección cuando se pasa un numero fijo de valores, corno cuando se realizan bucles a lo largo de elernentos de una matriz u otro tipo de estructura de datos con forma de colección. El bucle **while** suele ser la mejor eleccion cuando la condición final depende de que ocurra algo, como que el usuario introduzca algo desde la línea de cornandos o la recepción de un valor procedente de una invocación de rnetodo.

Declaraciones do-while

El menos usado de los constructores de bucle es el bucle **do-while** (el menos usado porque siempre ejecuta el cuerpo menos en una ocasión, antes de comprobar su condicidn. Vea la figura 6.10). No obstante, a menudo necesitará comprobar una condicidn en primer lugar. Aqui tiene la estructura básica del bucle **do-while:**

```
do
    cuerpo;
while (condición);
```

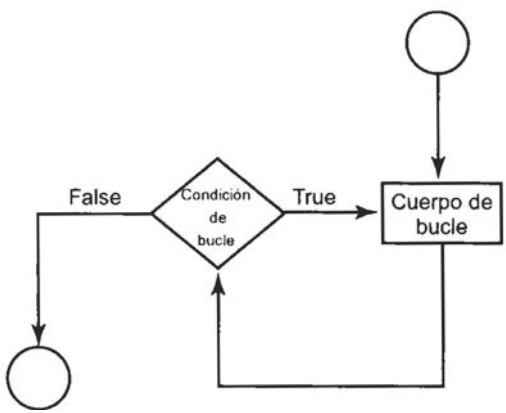


Figura 6.10. El flujo de bucle do-while

Para mostrar el bucle `do-while`, el listado 6.2 le solicita los nombres de mes que debe buscar en la matriz de nombres de mes ordenados. Es lo contrario a leer cadenas de búsqueda de los argumentos de la linea de comandos. El control de la introducción de datos se estudiará posteriormente. Por ahora, todo lo que debe saber es que `System.in` es la introducción por defecto, que se realiza a través del teclado. Para finalizar el bucle, escriba `exit`. En esta situación, debe introducir al menos un dato antes de comprobar si el bucle se ejecutará, para que el bucle `do-while` tenga más sentido.

Listado 6.2. Ernpleo de do-while para buscar rneses

```
import java.io.*;
import java.util.Array:;;
public class MonthPrompt {
    public static vnid main (String args[]) throws Exception (
        String monthNames[] = {"January", "February", "March", "April",
           "May", "June", "July", "August", "September", "October",
           "November", "December"};
        Ar ays.sort(monthNames);
        for (int i=0; i<monthNames.length; itt) {
           System.out.println(i + ": " + monthNames[i]);
        }
        String line;
        BufferedReader in = new BufferedReader(
             new InputStreamReader(System.in));
        do {
             System.out.println("Enter month [exit to end]");
             line = in.readLine();
             if (!line.equals("exit")) {
               System.out.println(Arrays.binarySearch(monthNames, line));
             }
        } while (!line.equals("exit"));
    }
}
```

Declaraciones para romper el flujo

Existen diferentes declaraciones para alterar el flujo general de un programa. De hecho, ya ha visto uno de ellos en funcionamiento: la declaración `break` usada en un bloque `switch`. Examinaremos `break` con más atención, junto con la declaración `continue`, interrupciones con etiqueta y la declaración `return`.

Nota

En muchos otros lenguajes, `goto` es una declaracion de interrupcion de flujo. Sin embargo, en Java, `goto` sólo es una palabra reservada que no puede emplear en sus programas. No funciona como declaracion de interrupcion de flujo ni de ninguna otra forma.

Declaraciones break

Puede usar la declaracion `break` para detener la ejecucidn de un bloque de codigo para un bucle o un `switch`. Cuando el sistema encuentra la declaracidn `break`, el sistema deja de ejecutar inmediatamente el bloque de código y transfiere el control del programa a la declaracion que sigue inmediatamente al bloque.

Por ejemplo, supongamos que queremos iterar 10 veces pero queremos salir del bucle si se produce alguna situación especial. Puede agregar una declaración `break` para ir a la declaracidn que se encuentra tras el bucle (que, en este caso, es la linea `System.out.println("Three");`).

```
for (int i=0; i<5; i++) {
   System.out.println("One");
   if (i == 3) {
      break;
   }
   System.out.println("Two");
}
System.out.println("Three");
```

Al ejecutar este código se produce el siguiente resultado, que escribe `One` y `Two` tres veces cada uno, antes de escribir `One` y salir del bucle para escribir `Three`.

```
One
Two
One
Two
One
Two
One
Three
```

Declaraciones continue

La declaracion `continue` funciona como la declaracidn `break`, pero en lugar de detener la ejecución del bucle, regresa para comprobar la condición del bucle. En el anterior bloque de código, si cambia la declaracion `break` a `continue`, obtendrá el siguiente resultado:

```
One
Two
One
Two
One
Two
One
One
Two
Three
```

Observe que `One` aparece dos veces seguidas, evitando Two cuando `i` es 3, pero sigue escribiendo `One` y `Two` cuando `i` es 4.

Declaraciones de etiqueta:

No existe la palabra clave "label" en Java. Sin embargo, puede usar etiquetas cuando usa declaraciones `break` y `continue`, cuando necesite salir de varios niveles. Una etiqueta puede ser cualquier identificador valido de Java y debe colocarse al principio del bucle de control. La etiqueta indica el bloque desde el que debe salir un `break` o el bloque a cuya parte superior debe volver un `continue` y examinarlo. Veamos cómo usar una etiqueta con la declaración `break`. Imagine un bucle de dos dimensiones y quiere salir de ambas, empezando por la interior. Al usar `break` de forma independiente sólo saldrá del bucle interno, de modo que debe etiquetar el bucle externo para poder salir de ambos. Aqui, esa etiqueta es la linea `outer:`

```
outer:
for (int i=0; i<10; i++) {
   for (int j=0; j<10; j++) {
      if (condición)
          break outer;
   }
}
```

Para una declaración `continue`, la etiqueta también se coloca al principio. Sin embargo, la continuación tiene lugar en la condición de prueba para el bucle y no en la declaración que sigue al bucle.

```
outer:
for (int i=0; i<10; i++) {
   for (int j=0; j<10; j++) {
      if (condición)
          continue outer;

}
```

Declaraciones return

La declaración `return` se emplea para transferir el control al invocador del método. Si declaró el método para que devolviera `void`, use `return` sin ninguna expresión después de 61:

```
return;
```

Sin embargo, si declaró el método para devolver un valor, debe asegurarse de que el método devuelve el tipo de valor adecuado. Hicimos esto en el método `getDays()` en el listado 6.1, donde lo declaramos para que devolviese un `int`. Aqui tiene de nuevo el código, con la declaración `return` en negrita.

```
int getDays(int month, int year) {

   int days;
   switch (month) {
       case 4:  // April
       case 6:  // June
       case 9:  // September
```

```
        case 11: // November
           days = 30;
           break;
        case 2:  // February
            if (((year % 4 == 0) &&
                 (year % 100 != 0)) || (year % 400 == 0)) {
              days = 29;
            } else {
              days = 28;
            }
            break;
        default: // rest
           days = 31;
           break;
      }
      return days;
}
```

Asi, el invocador invocará al método y usará el valor devuelto en consecuencia:

```
int value = getDays(2, 2000);
```

Cuidado: el compilador de Java es bastante exigente al requerir una declaración `return` en cada ruta de un método, aunque no todas las rutas se puedan ejecutar. Por ejemplo, si eliminase la declaración `return` final (`return 0`) del siguiente método, el compilador produciria un error y no crearia el archivo .class:

```
int wild() {
   boolean foo = true;
   while (foo) {
        return 1;
   }
   return 0;
```

Truco

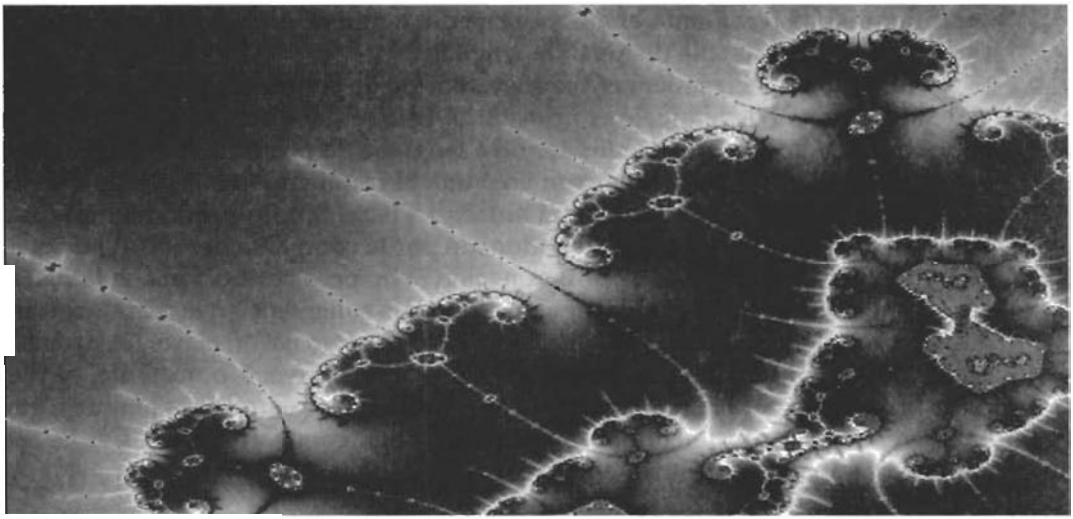
El compilador no considera una invocación a `system.exit()` como equivalente a la declaracion `return`. Invocar al método `system.exit()` hace que el entorno de ejecucion se detenga y salga. El sistema considera esto como otra llamada de método y no sabe que el sistema no volvera de la llamada de método. Si tiene una llamada a `System.exit()` en un método que debe devolver algo, debe hacer que el método devuelva algo para que satisfaga los requisitos de compilacion, aunque nunca se alcance la declaracion `return`.

Resumen

Llegar a ser un buen programador de Java significa que debe aprender el lenguaje de programación y todo lo que pueda sobre las bibliotecas. Con el final de este

capitulo, casi ha terminado el aprendizaje del lenguaje y sólo queda el control de excepciones como detalle de nivel de lenguaje.

En términos de herramientas de programación de Java procedentes de Microsoft, el detalle de nivel de lenguaje es prioritario. Esto se debe a que estas herramientas ahora tratan a Java como a cualquier otro lenguaje de programación. Tras pagar a Sun y cerrar el juicio Java, Microsoft creó una herramienta llamada Visual J# .NET ("J-almohadilla") que funciona en el entorno Visual Studio y que reemplaza a la herramienta Visual J++ que fue abandonada por la compañía debido al juicio. Si, J-almohadilla usa el lenguaje de programación Java, pero no funciona en la plataforma Java. En vez de eso, compila algo que funciona solamente en la plataforma .NET de Microsoft. Esto puede ser precisamente lo que quiere hacer; claro, los programas de Java también funcionarán solamente en el marco Java. No obstante, si piensa crear programas para la plataforma de Microsoft, mejor emplee un lenguaje que se haya creado especificamente para esa plataforma, como VB.NET o C#. Es mejor mantenerse apartado de una versión corrupta de Java.



7

Control de excepciones y aserciones

El control de condiciones de error en *C* y Visual Basic y otros lenguajes se realiza mediante el uso de declaraciones `if`, ramificaciones `goto` para los controladores de error y códigos de devolución cripticos para devolver las condiciones anormales a las rutinas que las invocaron. Este estilo de programación oculta el flujo de operaciones habitual de la fuente, lo que hace que el código sea enrevesado y dificil de mantener. El código será más preciso y claro si controla los errores y las condiciones inusuales mediante el mecanismo de control de Java. El control de excepciones de Java funciona de la misma forma que el de *C++* y apareció por primera vez en Modula 3 en los años 70.

Este capitulo le muestra como realizar esta importante tarea de administración. Aprenderá la sintaxis de programación especifica necesaria para tratar con las excepciones y cómo recuperarse correctamente. Desde arrojar y capturar métodos hasta leer volcados stack y usar la cláusula `finally`, recogerá toda la basura. También estudiará una extensión del control de excepciones, llamada aseveración, con los que puede especificar comprobaciones para la configuración de variables adecuada.

Constructores de control de excepciones de Java

Una excepción es una condición que interrumpe el flujo normal de operaciones dentro de un programa. Imagine una excepción como una condición de error (un

error de programación inesperado, como una división entre cero o un posible error que ocurra en tiempo de ejecución, como un "disco completo" mientras escribe un archivo. Cuando ocurre una excepción, el flujo normal de instrucciones se rompe. En concreto, se arroja la excepción y el flujo continua en el punto en el que se capturó la excepción. Al arrojarse la excepcion se permite la eliminación del código controlador de excepciones del flujo regular de operaciones. Si la condición anormal no se controla adecuadamente, pueden aparecer resultados incorrectos u otras condiciones anormales.

El constructor de lenguaje para controlar excepciones en un programa de Java es el bloque `try-catch`. Basicamente, las declaraciones que podrian arrojar una excepción se encuentran dentro del bloque `try`. Si se produce una excepción, se controlard mediante cláusulas `catch` ejecutadas con éxito. Si hay varias cláusulas `catch`, se ejecutará el bloque de código para el primer tipo de excepción que concuerde. Este es el aspecto de una estructura `try-catch` básica:

```
try {
   body
} catch (ExceptionClass1 exceptionVariable1) {
   exceptionHandlerBody1
} catch (ExceptionClass2 exceptionVariable2) {
   exceptionHandlerBody2

}
```

`ExceptionClass1` y `ExceptionClass2` del anterior ejemplo serían reemplazadas por los nombres de clase de la excepción. Las variables `exceptionvariable1` y `exceptionVariable2` pueden reemplazarse por cualquier nombre de variable válido, pero no importa el nombre que emplee, cada variable es local para su respectivo bloque `catch` de control de excepciones.

Jerarquiade excepciones

Para comprender lo que pueden ser los elementos `ExceptionClass1` y `ExceptionClass2`, usaremos la ilustración de la jerarquia de clases de la figura 7.1.

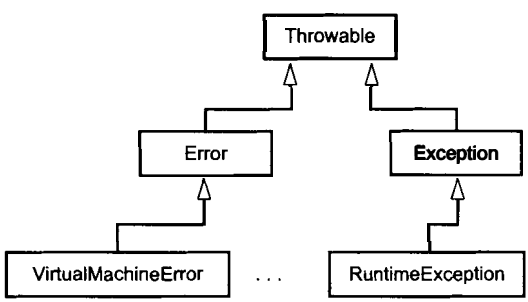


Figura 7.1. Jerarquia de excepcion de clases

Cualquier subclase de la clase **java.lang.Throwable** se puede arrojar; esto significa que cualquier clase también se puede atrapar. Las dos subclases mas conocidas de **Throwable** son **Error** y **Exception.** De estas dos clases proceden muchas otras subclases. Decidir qué clases atrapar puede ser un poco delicado, porque la cláusula **catch** hace que se atrape la clase especifica y todas sus subclases. En casi todas las circunstancias, nunca deberia atrapar un **Error** o una de sus subclases. Éstas suelen representar condiciones de error irrecuperables, como "out of memory".

Al igual que **Throwable,** la subclase **Exception** tiene su propio tipo de subclases: Está `RuntimeException`, y el resto (las excepciones que no son de ejecución). Los tipos de excepciones especificas que son subclases de la clase `RuntimeException` no necesitan ser atrapadas, pero pueden serlo. Todas las otras excepciones deben ser atrapadas o el compilador producirá un error.

Por lo general, si ocurre una excepción en tiempo de ejecución, significa que no se ha detectado una condición durante la comprobación. Esto se suele considerar un error del programador, un error que requiere que se corrija el programa. Las típicas condiciones de error del programador son:

- Division entera entre cero (debe comprobar el divisor antes de realizar la operación).
- Intentar leer mds allá del final de una matriz (ya conoce el tamaño y deberia detenerse antes de intentar leer mds a116 del final).

Las excepciones que no son de tiempo de ejecución tienden a ser más complicadas. Aunque algunas condiciones se pueden probar antes de intentar una operación, en general las excepciones de tiempo de ejecución pueden ocurrir en cualquier momento y deben ser controladas después de que se produzcan. Aquí tiene algunos tipos de problemas que son el resultado de una excepción que no es de tiempo de ejecución:

- La URL que está intentando usar contiene un protocolo no vdlido. [**java.net.**`MalformedURLException`]

Nota

La diferencia entre una excepcion de tiempo de ejecucion y una excepcion que es de tiempo de ejecucion suele ser muy sutil. Los programadores de Java inexpertos suelen usar mal los bloques **try-catch.** Intentan compilar su programa y el compilador avisa de un error, entonces se dan cuenta de que algo del código que estan usando ha arrojado una excepcion que no es tiempo de ejecucion y de que deben agregar el bloque **try-catch.** Con el tiempo, los programadores aprenden a reconocer que codigo necesita el bloque **try-catch** y puede pasar al siguiente paso.

- El archivo que quiere abrir ha sido borrado. [**java.io.**`FileNotFoundException`]

- El archivo desde el que estd leyendo no dispone de mds datos. [java.io.EOFException]
- El archivo de clase que quiere abrir no se encuentra o estd en un formato equivocado. [java.lang.ClassNotFoundException]
- El servidor al que estd intentando conectarse no existe. [java.net.UnknownHostException]

Un ejemplo de control de excepciones

Ahora que ya conoce el control de excepciones y lo que puede arrojarse y atraparse, haremos una demostracibn del control de excepciones. En este ejemplo, se pasan numeros desde la linea de comandos y se suman. Dado que los argumentos de la linea de comandos se pasan como cadenas, es necesario convertirlas en números. Esto se hace con el método `parseInt()` de la clase `Integer`. Dada una cadena, la convertimos en `int`:

```
int number = Integer.parseInt("1234");
```

En circunstancias normales, los usuarios simplemente pasardn numeros y todo funcionará correctamente. Pero si se pasa un valor de cadena no numérica se arrojará `NumberFormatException`. Al no incluir la llamada `parseInt()` enunbloque `try-catch`, el programa se compilard y se ejecutard porque `NumberFormatException` es una excepcidn de tiempo de ejecucibn. Sin el bloque `try-catch`, el programa se detiene si ocurre la excepcibn, dado que no existe nada que atrape la excepcibn. No obstante, `NumberFormatException` es una excepcidn de tiempo de ejecucibn y no es imprescindible que esté en el bloque `try-catch`. Suponiendo que no desea que el programa se detenga, necesita que el bloque `try-catch` vigile las entradas de cadenas no numéricas.

El listado 7.1 incluye el cbdigo completo para este bloque `try-catch`.

Listado 7.1. Surna de argurnentos de linea de comando

```
public class Sum {
    public static void main(String args[]) {
        long answer = 0;
        int number;
        for (int i=0; i < args.length; i++) {
           try {
              number = Integer.parseInt(args[i]);
              answer += number;
           } catch (NumberFormatException nfe) {
              System.err.println("Unable to convert " + args[i]);
           }
        }
        System.out.println("The sum is " + answer);
    }
```

Si se pasa una cadena no numérica al método `parseInt()`, se arroja `NumberFormatException`. La línea `answer += number;` nunca se alcanza debido al paso del bucle `for`; en vez de eso, el programa salta hasta la cláusula `catch`. Dado que el tipo de la excepción concuerda con el que se encuentra en la cláusula `catch`, se ejecuta el código del bloque de la cláusula `catch`, que muestra un mensaje de error.

Una vez compilado, puede ejecutar el programa y pasarle numeros para conseguir su suma, o numeros y cadenas para obtener la suma de los numeros, obviando las cadenas no numéricas. Como `NumberFormatException` es una excepción de tiempo de ejecución, ¿cómo sabe el nuevo programador atraparla? Hay dos modos: La documentación de la clase (los archivos `javadoc`) contiene una lista de excepciones que pueden arrojarse normalmente. La figura 7.2 lo demuestra. Al revisar cada método que emplee, puede descubrir qué excepciones se pueden arrojar y tratarlas adecuadamente.

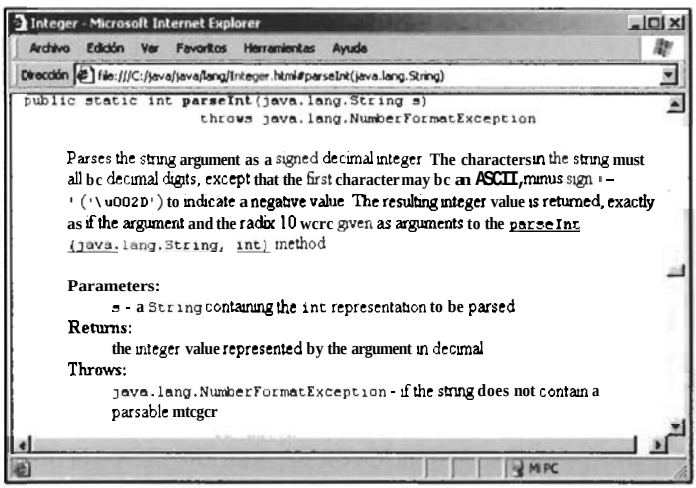


Figura 7.2. El javadoc parseInt() para la clase Integer

Casi siempre, los programadores escriben el programa sin el código de control de excepciones y lo agregan a posteriori. Por lo general, esto sucede cuando el compilador indica que el código es necesario, en el caso de excepciones que no son de tiempo de ejecución, o después de que ocurra la excepción en el caso de las excepciones de tiempo de ejecución. De nuevo, con el tiempo aprenderá a reconocer cuando son necesarios los bloques `try-catch`.

Métodos Throwable comunes

No hay restricciones a lo que la cláusula `catch` de un bloque `try-catch` puede hacer. Una vez en la cláusula `catch`, ésta usará probablemente uno de los siguientes métodos:

- `getMessage()`: Todos los objetos `Throwable` pueden tener un mensaje de error asociado. Si se invoca este método, devolverd el mensaje si encuentra alguno.
- `getLocalizedMessage()`: Si el mensaje de error ha sido traducido al idioma del usuario, este método obtiene la versión traducida del mensaje. En caso contrario, devuelve lo mismo que `getMessage()`.
- `printStackTrace()`: Este método envía el seguimiento stack a la consola del sistema o a otro punto. Un seguimiento stack es una lista de llamadas de método que conduce a la condición de excepción. También incluye numeros de linea y nombres de archivo, si se encuentran disponibles. El seguimiento stack imprimird el comportamiento predefinido de una excepción de tiempo de ejecución cuando no es atrapada de forma explicita. Por ejemplo, si se hubiera agregado una llamada a `printStackTrace()` al anterior programa Sum (Listado 7.1), habria visto el siguiente resultado al introducir un valor no vdlido:

```
>java Sum 1 2 3 a
Unable to convert a
java.lang.NumberFormatException: a
            at java.lang.Integer.parseInt(Integer.java:414)
            at java.lang.Integer.parseInt(Integer.java:463)
            at Sum.main(Sum.java:7)
The sum is 6
```

Cómo controlar métodos que arrojan excepciones

Puede ocurrir que esté trabajando con un método y decida no controlar la excepción que se encuentra en su interior, sino que prefiere pasar la excepción al método que la invocó. Debe indicarle esto al compilador mediante la cldusula `throws` en la declaración de método:

```
modifiers returnType methodName(args) throws ExceptionType
```

Esto indica al compilador que el método debe arrojar una excepción de tipo `ExceptionType`, lo que exige que el método que realiza la llamada coloque la llamada de método dentro de un bloque `try-catch` o que proporcione su propia cldusula `throws`. Como sucede con las cldusulas `try-catch`, los métodos necesitan tener una cldusula `throws` para las excepciones de tiempo de ejecución.

Para mostrar el uso de la cldusula `throws`, vamos a usar el método estdtico `sleep()` de la clase `Thread`. El método `sleep()` permite a su programa detenerse durante un numero indicado de milisegundos. El método `sleep()` puede arrojar una `InterruptedException` por lo que debe colocarse dentro de un bloque `try-catch`.

Comenzaremos definiendo una clase ayudante con un método `fiveSeconds()`. Este método hiberna durante cinco segundos y luego devuelve la `InterruptedException` que arrojó la llamada `sleep()` a `Thread`. Aqui tiene la clase ayudante:

```
public class SleepUtil {
    public static void fiveSeconds()
            throws InterruptedException {
        Thread.sleep(5000);
    }
}
```

Aunque el método `fiveSeconds()` funciona perfectamente, no oculta la posiblemente arrojada excepción `InterruptedException`, de modo que el invocador debe seguir colocando la llamada a `fiveSeconds()` en un bloque `try-catch`.

```
try {
   // pausa durante cinco segundos
   SleepUtil.fiveSeconds();
} catch (InterruptedException exception) {
   // controla la excepción
}
```

Si el método `fiveSeconds()` no tiene una cldusula `throws`, verá el siguiente error de compilacion, porque `sleep()` se declaró para arrojar una `InterruptedException`:

```
SleepUtil.java:3: unreported exception java.lang.InterruptedException;
          must he caught or declared to be thrown
      Thread.sleep(5000);

1 error
```

Truco

Si diseña un método para que arroje varias excepciones, necesitara proporcionar una lista delirnitada por comas; por ejernplo: `throws IOException, InterruptedException`.

Cuando se declara un método para arrojar una excepción, esencialmente se expande la definición de la firma del método. El conjunto de excepciones arrojadas es parte de la firma del método. Si decide invalidar un método en una subclase, la subclase no podrd arrojar ninguna excepción que no pudiera arrojar el origen. Por ejemplo, si va hacer subclase `SleepUtil`, puede tener un nuevo método `fiveSeconds()` que no arroja excepciones.

Sin embargo, no puede tener un método que arroja una `IOException`, porque `IOException` no es un tipo de `InterruptedException`. No puede arrojar nuevos tipos de excepción, pero puede arrojar subclases de tipos de excepción vdlidas.

Por ejemplo, supongamos que tiene un mbtodo que arroja una `IOException`. Una subclase puede arrojar una `FileNotFoundException` (una subclase de `IOException`). La declaración no puede arrojar un tipo de excepción menos especifico. Siempre es aceptable no arrojar excepciones, mientras el método no arroje una excepción.

Cómo arrojar excepciones por uno mismo

Pasar excepciones arrojadas por los mbtodo no es muy complicado, pero suele llegar un momento en el que quiere pasar excepciones cuando suceden condiciones de error. Esto nos trae el papel de la palabra clave **`throw`**. Por ejemplo, para arrojar una `InterruptedException`, crearia una instancia de `InterruptedException` y luego la arrojaria:

```
throw new InterruptedException();
```

O, si quiere incluir un mensaje con la excepción, haría esto:

```
throw new InterruptedException("Message");
```

En realidad eso es todo. Sólo tiene que encontrar la excepción apropiada para arrojar. Es mejor arrojar las excepciones más especificas posibles, con lo que el controlador de excepciones podra responder de la forma más especifica posible. En caso contrario, podría diseñar el controlador para que tuviera una cláusula **`catch (Exception e)`**.

Esto significa que tiene que atrapar todas las excepciones, incluyendo las excepciones de tiempo de ejecución, y responder de alguna forma adecuada. No obstante, el modo más eficiente es arrojar una excepción especifica que pueda atrapar el receptor y no tener que preocuparse por los otros casos.

Aqui tiene algunas excepciones que se suelen arrojar, pero hay muchas mas disponibles. (Y recuerde, éstas son todas excepciones en tiempo de ejecución.)

- **IllegalArgumentException:** Esta excepción se arroja cuando un argumento que se pasa aún no está entre el conjunto válido de valores.
- **SecurityException:** Esta excepción se arroja cuando un programa intenta realizar una operación que no tiene los suficientes permisos.
- **NullPointerException:** Esta excepción se produce cuando se usa una referencia null pero se necesita una que no sea null.
- **ArithmeticException:** Esta excepción se usa cuando ocurren condiciones aritmbticas ilegales que produciran procesos numbricos ilegales.
- **IndexOutOfBoundsException:** Esta excepción se arroja cuando un índice de una colección o conjunto, como una matriz, se encuentra fuera de los limites.

Cómo crear clases de excepción personalizadas

A menudo necesitamos arrojar más que las excepciones existentes; no siempre son la mejor elección. Observemos cómo crear sus propios tipos de excepciones. Se hace subclasificando una de las subclases `Exception` existentes. (Se puede subclasificar `RuntimeException`, pero no deberia, porque entonces la excepción no tendrá que ser atrapada.)

Esto no es muy complicado, simplemente cree la clase. En la mayoría de los casos, proporcionará dos constructores, uno que acepta el mensaje y otro que no. Aqui tiene un ejemplo:

```
public class MyException extends Exception {
    public MyException() {
    }
    public MyException(String message) {
        super(message);

}
```

Una vez creada la clase, puede invocar a `throw` en una nueva instancia de la clase cuando ocurra el problema. Por ejemplo, el siguiente fragmento puede agregarse a la definición de `MyException`:

```
public static void main(String args[]) {
    try {
       throw new MyException("Arg Length: " t args.length);
    } catch (MyException e) {
       e.printStackTrace();
    }
```

Cuando se ejecuta este código, verá el mensaje "Arg Length" en el seguimiento stack:

```
>java MyException
MyException: Arg Length: 0
                at MyException.main(MyException.java:9)
```

Encadenar excepciones

Una de las novedades de Java 1.4 es la capacidad de encadenar excepciones. El encadenado de excepciones le permite registrar el hecho de que una excepción causa otra. Por ejemplo, si una `InterruptedException` fuera arrojada por `sleep()`, podría querer arrojar una `MyException`. Antes de Java 1.4, tendría que crear su propio mecanismo para mantener la causalidad. Ahora, esto estfi automfiticamente disponible. La clase raíz de la excepción, `Throwable`, proporciona las posibilidades de encadenado de una forma compatible con las versiones

anteriores de tal forma que esta funcidn estd disponible para todas las excepciones definidas con anterioridad, no sólo con las excepciones de sistema.

Hay dos formas de encadenar excepciones en Java. Para las nuevas clases de excepcidn, deberia proporcionar dos constructores adicionales: uno que acepta `Throwable` y otro que acepta una `String` y `Throwable`. Por ejemplo, aquí tiene la definicidn de clase `MyException` actualizada para admitir excepciones encadenadas:

```
public class MyException extends Exception {
    public MyException() {

    public MyException(String message) {
        super(message);

    public MyException(Throwable throwable) {
        super(throwable);
    }
    public MyException(String rnessaqe, Throwable throwable) {
       super(message, throwable);
    }
```

Cuando tiene una excepcidn de clase que no se puede cambiar y necesita encadenarla, resulta útil emplear el método `initCause(Throwable)`. Este método, heredado de `Throwable`, permite a cualquier subclase inicializar su causa.

En cualquier caso (una nueva excepcidn o una ya existente) cuando llega el momento de controlar la excepcidn, puede preguntar por la causa mediante `getCause()`. El método `printStackTrace()` también mostrará la causa. Aqui tiene un ejemplo con un método `main()` actualizado:

```
public static void main(String args[]) throws MyException {
    try {
       throw new InterruptedException("Interrupted");
    } catch (InterruptedException e) {
       throw new MyException("Arg Length: " + args.length, e);
    }
}
```

Al ejecutar el programa ahora se generará el siguiente resultado:

```
>java MyException
Exception in thread "main" MyException: Arg Length: 0
              at MyException.main(MyException.java:17)
Caused by: java.lang.InterruptedException: Interrupted
              at MyException.main(MyException.java:15)
```

Lectura de un volcado Stack

Otra nueva funcidn de control de excepciones agregada en la version **1.4** de Java es la capacidad para examinar los elementos stack mds atentamente. No sólo puede

invocar a printStackTrace(), sino que puede invocar a getStackTrace() para conseguir información y mostrarla en cualquier formato que quiera. El método getStackTrace() devuelve una matriz de elementos StackTraceElement. Como muestra la tabla 7.1, por cada elemento stack se pueden encontrar varios elementos de información sobre cada punto de ejecucidn del stack.

Tabla 7.1. Composición de la clase StackTraceElement

Variables/Métodos	Descripción
equals()	Comprueba la igualdad con otro objeto
getClassName()	Devuelve el nornbre completo de clase del punto de ejecucion
getFileName()	Devuelve el nombre del archivo fuente para el punto de ejecucion
getLineNumber()	Devuelve el nurnero de linea del archivo fuente para el punto de ejecucion
getMethodName()	Devuelve el rnetodo de clase para el punto de ejecucion
hashCode()	Devuelve el codigo hash calculado
isNativeMethod()	Indica si el rnetodo para el punto de ejecucion es nativa
toString()	Convierte los elementos contenidos en cadena

Para comprobar el uso de StackTraceElement, observe el siguiente bucle for. Mostrará el nombre de archivo y el número de linea de cada elemento:

```
StackTraceElement elements[] = e.getStackTrace();
for (int i=0, n=elements.length; i<n; i++) {
System.err.println(elements[i].getFileName() + ":" +
    elements[i].getLineNumber());
}
```

Trabajar con la cláusula finally

Ademds de las cláusulas try y catch, los bloques de control de excepciones tienen una cldusula opcional finally. En realidad, catch y finally son opcionales, pero debe aparecer una de las dos y suele ser finally la desechada. La función del bloque finally es incorporar cddigo de limpieza u otro código que deba ejecutarse independientemente de si en el medio del bloque try – catch hay una excepción o una cldusula return.

Aqui tiene un buen ejemplo de por qué se emplea una cláusula finally para asegurarse de que un archivo estd cerrado. Observe que en el listado 7.2 no hay una

instrucción `return` en el centro del bloque `try`. Sin embargo, el bloque `finally` se ejecuta a pesar de esta declaracidn `return`, lo que asegura que `FileInputStream` está perfectamente cerrada.

Listado 7.2. Uso de finally para asegurarse de que una archivo esta cerrado

```
FileInputStream fis = null;
try {
   fis = new FileInputStream(filename);
   // lee desde fis
   ...
   if (somecondition) {
      return -1;
   }
} catch (IOException exception) {
   System.err.println("Error accessing file: " + filename);
} finally {
   if (fis != null) {
      try {
         fis.close();
      } catch (IOException ignored) {
      }
   }
}
return 0;
```

Trueo

Considere el usar un bloque `finally` sin un bloque `catch` cuando necesite asegurarse de que algún codigo se ejecute incluso si no se arrojan excepciones que no son de tiempo de ejecucion.

El listado 7.3 es un ejemplo más completo que muestra el uso de `finally` con cláusulas `continue` y `break`. Dentro de los dos bucles `for` anidados hay declaraciones `continue` y `break`, etiquetados y sin etiqueta.

Listado 7.3. Ejemplo de finally ampliado

```
public class FinallyTest {
    public static void main(String args[]) {
outer:
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
               try {
                  System.out.println("try before if: i=" + i +
                     ", j=" + j);
                  if ((i == 0) && (j == 1)) {
                     continue;
                  } else if ((i == 0) && (j == 2)) {
                     continue outer;
```

```
                } else if ((i == 1) && (j == 0)) {
                    break;
                } else if ((i == 2) && (j == 1)) {
                    break outer;
                }
                System.out.println("try after  if: i=" + i +
                    ", j=" + j);
            } finally {
                System.out.println("finally:       i=" + i +
                    ", j=" + j);
                System.out.println();
            }
        }
    }
}
```

Cuando ejecute `FinallyTest`, observe cómo la cláusula `finally` siempre se ejecuta, no importa cómo los bucles internos detienen cada ruta de ejecución. Aqui tiene el resultado de ejecutar el programa:

```
>java FinallyTest
try before if: i=0, j=O
try after  if: i=0, j=O
finally:       i=0, j=O

try before if: i=0, j=1
finally:       i=0, j=1

try before if: i=0, j=2
finally:       i=0, j=2

try before if: i=1, j=0
finally:       i=1, j=0

try before if: i=2, j=O
try after  if: i=2, j=0
finally:       i=2, j=O

try before if: i=2, j=1
finally:       i=2, j=1
```

Comprobar aseveraciones

Otra nueva función agregada en la versión 1.4 de Java es una utilidad para aseveraciones. Puede imaginar esta utilidad como una forma avanzada de control de excepciones. Las aseveraciones representan expresiones booleanas que el programador cree que se cumplen en puntos especificos del programa. La comprobación de aseveraciones se puede activar o desactivar en cualquier punto, de modo que puede realizar pruebas con ellas activadas y realizar la implementación con

ellas desactivadas. En ese caso, un usuario final podrd mds tarde reactivar las aseveraciones si surge un problema con el programa.

Nota

Para conocer la historia de como se llegó a la creación de esta utilidad, vea http://www.elj.com/eiffel/feature/dbc/java/ge/, que incluye información sobre Java y el diseño por contrato (DBC).

La comprobación de aseveraciones es como el control de excepciones, pero no se arrojan excepciones si falla una aseveración. En su lugar, se arroja una `AssertionError`. Como todos los errores, es dificil, si no imposible, recuperar de una aseveración fallida porque el estado del programa se ha desviado de su operación normal.

Uso de la palabra clave assert

Hay dos constructores básicos para usar aseveraciones:

```
assert booleanExpression;
assert booleanExpression : message;
```

Puede agregar declaraciones de aseveraciones como condiciones anteriores o posteriores de un método. La expresión booleana también puede llamar a un método. Otros lugares obvios para las aseveraciones son las condiciones `else` en bloques `if`, donde las condiciones se presuponen o en casos predefinidos de una declaración `switch`, que nunca deberia alcanzarse. El unico limite a la palabra clave `assert` es que debe encontrarse en un bloque ejecutable. Por ejemplo, no puede estar con las declaraciones de variable de clase, pero puede estar dentro de cualquier método.

Cuando use aseveraciones, debe incorporar algunas opciones de linea de comandos especiales para compilar y ejecutar sus programas. Para compilar, debe pasar al compilador de Java una `-source 1.4`. Por defecto, el compilador se ejecuta en modo de compatibilidad **1.3.** Debe solicitar de forma explicita el modo 1.4. Entonces, en tiempo de ejecución, debe activar el control de aseveraciones mediante la opción `-enableassertions` o la opción mds corta `-ea` (por defecto, el control de aseveraciones estd desactivado).

Aqui tiene un simple ejemplo que le lleva a través de los pasos necesarios para usar aseveraciones. Comprueba el número de argumentos de linea de comandos e informa de un problema si el recuento no es 0.

```
public class AssertTest {
    public static void main(String args[]) {
        assert args.length == 0 : args.length + " !  0";
        System.out.println(args.length);
```

```
    }
}
```

Para compilar el programa, asegurese de usar la opción `-source`:

```
javac -source 1.4 AssertTest.java
```

En primer lugar, para comprobar el programa, ejecútelo y pdsele argumentos de linea de comandos:

```
java AssertTest 1 2 3 4
```

Esto dard como resultado 4 porque el control de aseveraciones estd desactivado por defecto. A continuación, ejecute el programa con el control de aseveraciones activado:

```
java -ea AssertTest a
```

y verd un mensaje de error que informa del siguiente problema:

```
Exception in thread "main" java.lang.AssertionError: 1 != 0
            at AssertTest.main(AssertTest.java:3)
```

Bdsicamente, esta es toda la utilidad `assertion`. La parte mds complicada es descubrir donde colocar las palabras clave `assert`.

Activar el control de aseveraciones

La opción `-ea` es un modo sencillo de activar las aseveraciones para un programa entero, pero hay otras opciones disponibles. Casualmente, `-ea` no funciona para las clases de sistema; para éstas deberd usar `-enablesystemassertions` o el mds sencillo `-esa`.

Para activar las aseveraciones para todo un drbol de paquetes (el paquete y todos los subpaquetes), puede utilizar la opción `-ea` seguida de dos puntos, el nombre del paquete y tres puntos (`...`). Por ejemplo, el siguiente fragmento activará la comprobación de aseveraciones para el paquete `net.zukowski.util` y todos sus subpaquetes:

```
java -ea:net.zukowski.util... MyClass
```

Para una clase determinada, especifique el nombre completo de la clase, en lugar de finalizar el nombre del paquete con tres puntos.

Si quiere activar el control de aseveraciones para todo un conjunto de paquetes, pero no para una clase o un paquete (árbol), debe usar la etiqueta `-da`. Enumere las opciones para activar y desactivar y ya estard listo. El siguiente ejemplo activa la comprobación de aseveraciones para el paquete `net.zukowski.util` pero no para la clase `net.zukowski.util.Math`:

```
java -ea:net.zukowski.util... -da:net.zukowski.util.Math MyClass
```

Activar el control de aseveraciones en tiempo de ejecución

Ademiis de activar la comprobación de aseveraciones desde la linea de comandos, también puede activarla (o desactivarla) mediante programacion. Hacer esto sólo afecta a las clases cargadas después de haber activado la comprobación de aseveraciones. No vuelve hacia atras y permite la comprobación de las clases cargadas con anterioridad. Esto no es algo que se pueda activar y desactivar a voluntad.

`ClassLoader` de la clase controla la activacion y desactivacion de las aseveraciones mediante los siguientes cuatro métodos:

- **setDefaultAssertionStatus():** Este método establece el estatus por defecto para los paquetes y clases cargados por el cargador de clases. El valor puede sobrescribirse designando las opciones de paquete especifico y clase desde la linea de comandos.
- **setPackageAssertionStatus():** Para anular el estatus para un paquete y todos sus subpaquetes, pase el nombre del paquete y el estatus de activación.

> **Nota**
>
> Para establecer el estatus de la aseveracion para el paquete sin nombre, proporcione un nombre de paquete null.

- **setClassAssertionStatus():** Para anular el estatus para una clase especifica, pase el nombre completo de la clase y su ruta de acceso y el estatus de activación.
- **clearAssertionStatus():** Este método reinicia todas las configuraciones de paquete y clase a `false` y luego reinicia el cargador de clase con el valor `false`.

Para conseguir el cargador de clase para una clase, invoque al método `getClassLoader()` del objeto `class`. En un método no estiitico, esto se puede conseguir mediante la siguiente declaración:

```
ClassLoader loader = getClass().getClassLoader();
```

Para un método estiitico, no puede invocar `getClass()` porque no tiene instancias, de modo que agregue `.class` al final del nombre de la clase en cuestión:

```
ClassLoader loader = MyClass.class.getClassLoader();
```

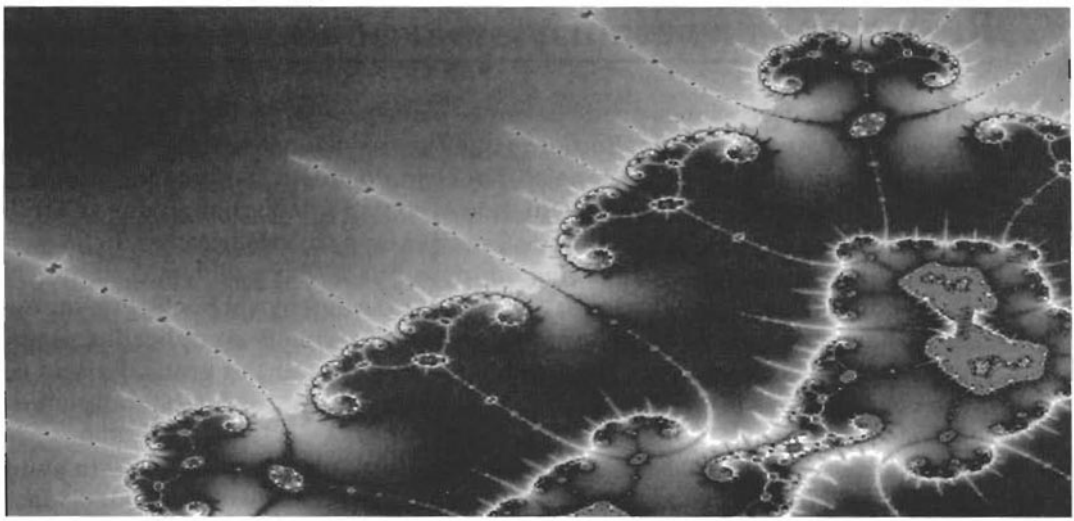
Recuerde que esto sólo afecta a las clases cargadas tras el cambio. En la mayoría de los casos, permitiendo aseveraciones desde la linea de comandos.

Resumen

Trabajar con excepciones puede llevarnos a una programacion perezosa, de modo que no se vuelva descuidado. Recuerde que las excepciones se deben emplear solamente en circunstancias excepcionales. Por ejemplo, mientras suma los numeros de una matriz, ¿debería seguir incrementando el contador y confiar en que se arroje la excepción `ArrayIndexOutOfBoundsException` para avisarle cuando ha pasado el limite? Por supuesto, esto funcionará, pero puede dar problemas de mantenimiento porque necesitara averiguar qué hace que se detenga el bucle. Por supuesto, esto se hace más dificil de determinar en las situaciones complicadas. Aunque protegerse de las excepciones (con un bloque `try-catch`) es gratis, hay un coste indirecto asociado a la creación y envio de la excepción, porque debe completarse el seguimiento del stack.

Recuerde ser lo más especifico posible cuando arroje excepciones. Esto ayuda a mantener las cosas ordenadas en el extremo receptor de la excepción arrojada.

Por lo que respecta a las aseveraciones, el constructor general es relativamente sencillo, pero requiere práctica llegar a saber cuándo usar este constructor; o mas especificamente, dónde colocarlo. Y las aseveraciones aumentan el volumen de los archivos .class. Definitivamente, debe evitar usar aseveraciones que tienen efectos secundarios (por ejemplo, aumentar una variable en 1). Cuando las aseveraciones están desactivadas, el efecto secundario no tendrii lugar y el comportamiento de su programa se verá alterado.



8

Subprocesos y multitarea

Uno de los conceptos más importantes de la programación de Java es la multitarea. Hasta ahora, todos los programas de ejemplo que hemos visto han tenido un sólo proceso. Se ejecutan desde el principio hasta el final, una tarea cada vez. Por ejemplo, cuando solicita al usuario que introduzca datos, el programa no hace nada hasta que el usuario introduce el dato. Para crear programas que respondan, debe entender lo que es la multitarea, ya que Java se ejecuta en varios subprocesos. Por ejemplo, el proceso de petición de memoria conocido como recolección de elementos no utilizados tiene lugar en un subproceso secundario del sistema. Este capitulo explica lo que son los subprocesos y cómo puede trabajar con ellos en el entorno de ejecución de Java. Aprenderii a controlar subprocesos y a hacer que ejecuten periódicamente sus tareas. Usar varios subprocesos en sus programas también requiere que realice trabajo extra para garantizar la seguridad de la memoria compartida. Aprenderii a comunicar thread y a sincronizar sus acciones para proteger esa memoria. Al establecer prioridades para los thread, puede afectar a la programación de tareas en el JRE. Al final de este capitulo, estarii cerca de convertirse en un maestro del threading.

Bases de la multitarea

La multitarea es el modo que tienen los sistemas modernos, como Linux, Unix y Windows 2000, de controlar la ejecución de varios programas. Aunque parezca que

ejecutan varios elementos de forma simultanea, la CPU en realidad esta alternando los contextos, dando a cada aplicación un poco de tiempo de la CPU antes de cambiar a otra aplicaci6.n y continuar con las operaciones de otra. La multitarea sucede cuando hay que realizar varias tareas en cualquiera de estos programas multitarea. Por ejemplo, en un procesador de texto, como Microsoft Word, podría estar imprimiendo la ultima copia de un documento mientras escribe un resumen, mientras el sistema comprueba la ortografia de lo que ha escrito. Cada una de estas tareas, imprimir, escribir y realizar la corrección ortográfica, es controlada por un thread diferente del programa.

Imagine cada thread como un flujo de control en un programa. A veces también se le llama contexto de ejecucidn porque cada thread cuenta con sus propios recursos, como un contador de programa o un stack de ejecucidn. De todas formas, los thread de un programa pueden compartir recursos como el espacio de memoria o los archivos que se hayan llegado a abrir. Se parece a un proceso de ejecucidn de un programa, pero la verdad es que su creación y destrucción es más sencilla porque no hace falta tanta administracidn.

A continuación verá una introducción de los fundamentos básicos del trabajo con thread, incluyendo cómo crearlas y ejecutarlas, cdmo controlar su ejecucidn, a través de qué estados pasan, cdmo se organizan y se asignan prioridades a los thread y las descripciones de la informacidn que puede obtener sobre ellos.

Crear y ejecutar un thread

Tenemos dos formas de ejecutar una tarea en paralelo con otras:

- Crear una subclase de `java.lang.Thread` y anular su método `run()`

  Implementar la interfaz `java.lang.Runnable` y pasarla a `Thread` para ejecutarla.

La elección entre estos dos mecanismos depende de lo que haya que hacer. Como el lenguaje de programación Java sólo acepta herencia sencilla, asegurese de que usa esta herencia subclasificando `Thread`. Si lo que quiere es usar `Thread` como `Thread`, hará mejor implementando la interfaz y creando un sencillo subproceso. Aunque, en la mayoría de los casos, lo único que tiene que hacer es implementar la interfaz `Runnable`.

Usar una subclase de la clase Thread

Cuando crea una tarea definiendo una subclase de la clase `Thread`, la subclase debe anular el método `run()` vacio de la clase `Thread`. Este método `run()` es donde se define la tarea a realizar:

```
public class MyThread extends Thread {
    public void run() {
        // tarea
    }
}
```

Para que el thread comience la ejecución, invoque el método `start()`, que activa una llamada a `run()`. El método `run()` no se invoca directamente; si lo hace, la invocación se ejecutarti en el thread principal en lugar de en un nuevo thread. Para mostrar el principio de un thread, aqui tiene un ejemplo de la creación de una subclase `Thread` y la necesaria invocación a `start()`:

```
// En otra clase
Thread t = new MyThread();
t.start();
```

Implementar la interfaz Runnable

La clase `Thread` implementa realmente la interfaz `Runnable`. Es la interfaz `Runnable` la que define el método `run()`. Una alternativa al empleo de la subclase `Thread` y proporcionar su propio método `run()` es implementar la interfaz `Runnable` y proporcionar el método `run()`. Necesitarti crear un `Thread`, pero el constructor `Thread` aceptará un argumento `Runnable` para indicarle lo que debe hacer. Aqui tiene un ejemplo:

```
class MyRunnable implements Runnable {
   public void run() {
       // tarea

}
```

En el siguiente ejemplo, la ejecución requiere la creación del `Runnable` y el `Thread` antes de invocar a `start()`:

```
// En otra clase
Runnable r = new MyRunnable();
Thread t = new MyThread(r);
t.start();
```

Invocar al método `start()` de `Thread` en este caso hace que la ejecución del método `run()` del objeto `Runnable` pasado al constructor. En lugar de crear una clase que implemente `Runnable`, otro modo comun de crear una implementación `Runnable` es usar una clase interna anónima, como la que aparece a continuación. Esto evita la necesidad de crear un nombre de clase sin sentido.

```
Runnable r = new Runnable() {
   public void run() {
       // tarea
   }
};
Thread t = new MyThread(r);
t.start();
```

Métodos de control de thread

La clase `Thread` proporciona muchos métodos utiles para su control. Como breve introducción, aqui tiene algunos de los métodos más empleados:

`start()`: Se utiliza para iniciar la ejecución del cuerpo del thread que se ha definido dentro del método `run()`. El control del programa se devolverd al método que efectua la llamada para que se pueda iniciar la ejecución de otro thread.

- `interrupt()`: El método `interrupt()` establece el estatus de interrupción del thread. También hace que un thread deje de estar bloqueado si estd esperando en un bloqueo de supervisión, esperando una E/S en un `java.nio.channels.channel` o bloqueado en un `java.nio.channels.Selector`. Si un thread estd esperando en una operación de E/S que no estd orientada a un canal, `interrupt()` no desbloquea la operación.

  `join()`: Utiliza la función que invoca al thread para esperar a que éste complete su ejecución. Este método se invoca para otro thread, no para el thread en el que se estd ejecutando el código.

Nota

La clase `Thread` tambien contiene los metodos `suspend()`, `resume()` y `stop()`. Su uso esta desaconsejado y deberia evitar el utilizarlos. Esto es debido a que pueden hacer que el sistema se vuelva inestable. Para mas información sobre por qué no deberia usar estos métodos, vea http://java.sun.com/tutorial/post1.0/preview/threads.html.

Algunas operaciones de subproceso, indicadas a continuación, sólo funcionan en el subproceso que se esté ejecutando en ese momento. En otras palabras, los métodos son estdticos.

- `currentThread()`: Este método devuelve una referencia al thread que se está ejecutando en ese momento.
- `dumpStack()`: Este es un método depurador, útil para vaciar, a petición del usuario, la pila del subproceso que se estd ejecutando en ese momento. La pila es una invocacidn de rastreo de métodos, que lleva al subproceso a su punto de ejecución actual.

  `sleep(long TiempoMedidoEnMilisegundos)` y `sleep(long TiempoMedidoEnMilisegundos, long TiempoMedidoEnNanosegundos)`: Estos dos métodos hacen que el entorno de ejecución ponga en estado de hibernación el subproceso que se esté ejecutando en ese momento durante el tiempo especificado. Esta invocación debe colocarse en un bloque `try-catch` porque se puede arrojar una `InterruptedException` si el thread se ve interrumpido con `interrupt()`. El subproceso no se inicia automdticamente después del periodo de hibernación especificado, pero se puede ejecutar. El uso de la versión en nanosegundos de este método es excesivo, pero es una opción disponible.

- `yield()`: Este método hace que el thread que se está ejecutando se detenga de forma temporal y lo coloca al final de una cola, en espera a que le llegue su turno de ejecución. Si no hay más elementos para ejecutar, el thread sigue ejecutándose sin intercambiar el contexto.

El próximo ejemplo, el listado 8.1, nos muestra cómo se utilizan los metodos anteriores. El thread principal creará otros dos y esperará a que el primero termine su ejecución. Para ello utilizará el método `join()`. El primer thread llama al método `sleep()` para que espere durante 10 segundos. Mientras tanto, el segundo thread llama a su propio método `wait()` para pasar a modo de espera hasta que el thread principal llame al método `notify()` del método suspendido. Cuando el primer thread finaliza su ejecución, el segundo entrari en funcionamiento invocando a su método `notify()` y esperará a que termine la ejecución del segundo thread invocando el método `join()` del segundo thread.

Listado 8.1. Ejemplo de comunicacion de subprocesos

```
public class MethodTest {
    static class FirstThread extends Thread {
      public void run() {
          try {
            System.out.println("  First thread starts running.");
            sleep(10000);
            System.out.println("  First thread finishes running.");
          } catch (InterruptedException e) {
            System.err.println("Error in thread: " + e);
          }
      }
  }
  static class SecondRunnable implements Runnable {
     public synchronized void run() {
         try {
           System.out.println("  Second thread starts running.");
           System.out.println("  Second thread suspends itself.");
           wait();
           System.out.println("  Second thread runs again and finishes.");
         } catch (InterruptedException e) {
           System.err.println("Error in runnable: " + e);
         }
      }
  }

  public static void main(String args[]) {
      Thread first = new FirstThread();
      Runnable secondRunnable = new SecondRunnable();
      Thread second = new Thread(secondRunnable);
      first.start();
      second.start();
      try {
        System.out.println("Waiting for first thread to finish...");
        first.join();
        System.out.println("It's a long wait!");
```

```
            System.out.println("Waking up second thread...");
            synchronized(secondRunnable) {
               secondRunnable.notify();
            }
            System.out.println("Waiting for second thread to finish...");
            second.join();
         } catch (InterruptedException e) {
            System.err.println("Error in main: " + e);

         System.out.println("I'm ready to finish too.");

}
```

El resultado de este programa es:

```
>java MethodTest
Waiting for first thread to finish...
     First thread starts running.
     Second thread starts running.
     Second thread suspends itself.
     First thread finishes running.
It's a long wait!
Waking up second thread...
Waiting for second thread to finish...
     Second thread runs again and finishes.
I'm ready to finish too.
```

El ciclo vital de un subproceso

Use la clase **Thread** para hacer que sus programas sean multitarea. Internamente, el entorno de ejecución también confía en los thread para realizar tareas críticas como la carga de clases, el control de eventos y la interacción con el sistema operativo. Todos los thread, después de su creación y antes de proceder a su eliminación, pasarán por uno de estos cuatro estados: recién creado, ejecutable, bloqueado o eliminado. Los podemos ver en la figura 8.1.

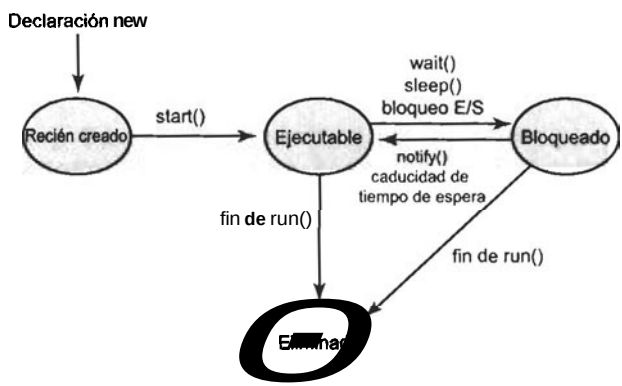


Figura 8.1. Ciclo de vida de un thread

Thread recien creado

Un thread tendrá el estado recién creado inmediatamente después de su creación. Es decir, entra en este estado justo después de la ejecucion de la declaración new. En este estado se procede con la distribución e iniciacion de los miembros locales. De todas formas, no se ejecutard el método `run()` hasta que se invoque el método `start()`. Hecho esto, el thread pasa al estado ejecutable.

Thread ejecutahle

Una vez iniciado, el entorno de ejecución Java puede escoger los thread ejecutables para ejecutarlos realmente.

Imagine el estado "ejecutable" como si tuviera dos subtipos llamados preparado (o en cola) y ejecución. Cuando el thread se encuentra en el estado de ejecucion, se le asigna a los ciclos de la CPU. Cuando se encuentra en estado de preparado y en cola, sólo está esperando a que se le asignen los ciclos de la CPU. La transición entre estos dos subestados la controla el programador de la Máquina Virtual. De todas formas, el thread podrá invocar voluntariamente al método `yield()` para salir del estado de espera y entrar en el de ejecutable. El método `yield()` es un método estático y actua sobre el thread que se está ejecutando en ese momento. No puede usar `yield()` sobre otro thread para detenerlo.

Thread bloqueado

El thread entrará en el estado bloqueado cuando ocurra una de las siguientes situaciones:

- El thread llama al método `wait()` del objeto.
- El thread llama al método `sleep()`.
- El thread espera a que se complete alguna operación E/S.
- El thread utilizara `join()` para unirse a otro thread.

Un thread bloqueado no se tendrá en cuenta a la hora de programar los thread que se ejecutaran. Regresara al estado ejecutable, luchando por los ciclos de la CPU, cuando tiene efecto el contra-evento del evento bloqueador, como se indica a continuación:

- Si el thread se bloquea invocando al método `wait()` de un objeto, el contra-evento aparece cuando se invoca el método `notify()` o el `notifyAll()` del objeto.
- Si se hace hibernar al thread, el contra-evento aparece cuando ha pasado el tiempo especificado de hibernación.
- Si el thread está bloqueado en E/S, el contra-evento aparece cuando se completa la operacion de E/S especificada.

- Si el thread está esperando a que se le unan, el contra-evento aparece cuando termina el otro thread.
- Excepto los thread bloqueados por E/S, los thread bloqueados tambikn pueden ser interrumpidos invocando a `interrupt()`.

Thread eliminados

Cuando un thread llega al final de su método `run()`, entra en el estado de eliminado. Los thread eliminados no se pueden recuperar. Para evitar su eliminación, puede incluir un bucle `while` en el método `run()` de un thread que mantendrá el thread funcionando. Por supuesto, si el thread sólo tiene que realizar una tarea cada vez, no es necesario un bucle `while`.

Por ejemplo, para hacer que un thread se detenga en el siguiente paso del bucle, lo normal es mantener una referencia al thread que se está ejecutando y comprobar si es null, donde algún factor externo podría cambiar la variable que señala el thread que se va a detener. Tiene este aspecto:

```
Thread currentThread = Thread.currentThread();
while (currentThread != null) {
     // .. trabaja
}
```

Cuando se elimina un thread, se celebra un funeral y se arroja un error `java.lang.ThreadDeath`. La clase `ThreadDeath` es una subclase de `java.lang.Error` y se puede atrapar en un bloque `try-catch`. Si atrapa un `Error` genérico o un `ThreadDeath` especifico en la clhusula `catch` del bloque, debe asegurarse de que se vuelva a arrojar el error `ThreadDeath`. Si el error no alcanza el tiempo de ejecución, el sistema no liberarh los recursos del thread. Aqui tiene una demostración del error `ThreadDeath` arrojándose de nuevo:

```
try [
    // thread elirninado
} catch (ThreadDeath death) {
    // posiblemente hace algo
    throw death;
```

Crupos de thread

Todos los thread son parte de exactamente un `ThreadGroup`. Cada miembro de un grupo de thread puede contener thread sueltos o agrupados. Los thread se distribuyen en forma de árbol, con los thread del sistema en la raiz del mismo.

Los grupos de thread se usan para la organización y le permiten manipular y controlar grupos como un todo. Por ejemplo, puede facilitar la seguridad organizando los thread en un grupo y no modificará el comportamiento de los thread en otro. Ademds, se pueden usar los grupos para limitar las prioridades de los thread.

Cuando se inicia una aplicacibn de Java, la Maquina Virtual crea un ThreadGroup principal de thread como un miembro de los thread del sistema. Dentro de este grupo principal, se crea un thread que se encargue de la ejecución del método main() de la aplicacibn.

Por defecto, todos los thread que crea el usuario seran miembros de ThreadGroup, a no ser que se especifique lo contrario. Para crear thread en su propio grupo de pase el grupo como el primer argumento del constructor de thread:

```
ThreadGroup group = new ThreadGroup(groupName);
Thread t1 = new Thread(group, aRunnable);
Thread t2 = new Thread(group, aRunnable, threadName);
Thread t3 = new Thread(group, aRunnable, threadName, stackSize);
Thread t4 = new Thread(group, threadName);
```

Advertencia

El comportamiento asociado a proporcionar al thread un tamaño de pila depende mucho de la plataforma. Algunas plataformas pueden no tener en cuenta este parametro.

Observe que, en la clase ThreadGroup, los métodos `stop()`, `suspend()` y `resume()` están despreciados, como en la clase Thread.

Truco

Los grupos de thread tambien sirven para atrapar las excepciones arrojadas en tiempo de ejecucion por los thread del grupo y registrar las que no se pueden atrapar. Por defecto, estas excepciones en tiempo de ejecucion vacian su pila mediante `printStackTrace()`. Si quiere atrapar estas excepciones, haga una subclase de `ThreadGroup` e invalide `public void uncaughtException(Thread source, Throwable throwable)`.

Prioriclades y planificación

La planificacibn de los thread define el modo en el que el entorno de Java intercambia tareas y escoge tarea que debe ejecutarse a continuación. Para ayudar a esta planificación, el entorno de ejecucion confia en una configuración de prioridades para cada tarea.

Cuanto mayor sea la prioridad, mayor será la importancia de la tarea y será más probable que se ejecute antes (si no hay otras tareas pendientes con prioridad más alta).

Organización y espacio de tiempo

Cuando aparece una nueva tarea que quiere ejecutarse, el tiempo de ejecucion comprueba la prioridad de la tarea. Si la prioridad de la nueva tarea es superior que la que se está ejecutando en ese momento, la tarea con la prioridad más alta se ejecutará, devolviendo a la tarea que se está ejecutando en ese momento al organizador de tareas. Este comportamiento recibe el nombre de organizacidn preferente.

Además de detener los thread mediante la organización preferente, tambikn se les puede obligar a que abandonen voluntariamente la CPU invocando al mktodo `yield()` de `Thread`. Cuando esto sucede, el thread que se está ejecutando está indicando que le gustaría seguir usando la CPU pero que ahora es un buen momento para comprobar si algun otro thread quiere realizar algun trabajo. Si hay algun thread ejecutable con la misma prioridad que el thread que se está ejecutando, uno de los thread que están esperando tomara la CPU. Si el thread (que se está ejecutando) no deja la CPU a otros thread con la misma prioridad, puede ser invalidado (esto se debe a que la ejecucidn de thread se puede dividir en espacios de tiempo). La división en espacios de tiempo significa que los thread asociados a la CPU con la misma prioridad se alternarán en su ejecucidn, "dividiendo" el tiempo para que cada thread pueda funcionar.

El algoritmo exacto para organizar thread es especifico de cada plataforma y no está definido formalmente. Basicamente, el orden de ejecucidn de un thread no depende de las prioridades o la preferencia. Además, los organizadores pueden invertir las prioridades para asegurarse de que ningún thread se quede a la larga sin acceso a la CPU.

Valores de prioridad de thread

Como se mencionó, se confia en las prioridades de thread para controlar qué tareas tienen acceso a la CPU. La clase `Thread` define un conjunto de 10 valores para estas prioridades, comenzando por `MIN_PRIORITY` y terminando con `MAX_PRIORITY`. Tambikn existe una prioridad "normal", `NORM_PRIORITY`, que es la prioridad por defecto para los thread recién creados. El sistema de ejecucidn tambikn puede usar otras prioridades para las tareas de sistema. Para cambiar el valor de prioridad se emplea el mktodo `setPriority()`:

```
aThread.setPriority(Thread.NORM_PRIORITY+1);
```

Obtener información sobre thread y grupos de thread

Los thread y los grupos de thread se usan básicamente para realizar manipulaciones, como iniciar y detener thread o cambiar las prioridades. Además, las dos clases `Thread` y `ThreadGroup` pueden proporcionar muchos detalles sobre thread individuales. A continuación tiene seis elementos de informacidn que se pueden solicitar desde una instancia `Thread`:

- `getName()`: Devuelve el nombre del thread.
- `getPriority()`: Devuelve la prioridad del thread.
- `getThreadGroup()`: Devuelve el grupo de thread primario.
- `isAlive()`: Devuelve `true` si el thread se ha iniciado pero aun no esta eliminado.
- `isDaemon()`: Devuelve `true` si el thread es un demonio.
- `isInterrupted()`: Devuelve `true` si se ha interrumpido el thread.

La clase `ThreadGroup` tiene su propio conjunto de seis mktodos de informacion. Algunos se aplican al grupo como un todo y otros a un thread especifico.

- `getName()`: Devuelve el nombre del grupo de thread.
- `getParent()`: Devuelve el grupo de thread primario.
- `getMaxPriority()`: Devuelve la prioridad máxima para thread del grupo.
- `activeCount()`: Devuelve el numero de thread activos.
- `activeGroupCount()`: Devuelve la cantidad de grupos de thread activos.
- `enumerate(Thread list[])`, `enumerate(Thread list[], boolean recursive)`, `enumerate(ThreadGroup list[])`, `enumerate(ThreadGroup list[], boolean recursive)`: Agrega todos los thread o grupos de thread activos en este grupo de thread a la matriz `list`. Si `recursive` es `true` (por defecto), todos los elementos apropiados de cualquier grupo de subthread también se copiarán. Este método devuelve un numero de elementos agregados a la matriz. (Puede usar el método `activeCount()` para indicar el tamafio de la lista cuando asigne espacio pero es más seguro agregar uno o dos extra al tamafio de la lista por si cambia el tamafio entre la creación y el momento de atraparlo.)

Ejecutar tareas periodicamente

Frecuentemente, lo que sucede con las tareas de thread es que se ejecuta una tarea y luego descansa durante un momento antes de volver a ejecutar la tarea. En lugar de programar esa parte "descansa un momento", puede hacer que un par de clases de biblioteca hagan ese trabajo. Esto hace que usted sólo tenga que preocuparse de la tarea. La clase `java.util.Timer` proporciona la organización y `java.util.TimerTask` proporciona la tarea.

Con la clase `Timer`, en lugar de crear una implementación `Runnable` o subclasificar `Thread` para definir el trabajo que se debe realizar, se subclasifica la clase `TimerTask` para proporcionar una implementación `run()`. Por ejemplo, para definir una tarea que escribira un mensaje, podría usar la siguiente definición de clase. Observe lo sencilla y clara que es esta definición.

```
TimerTask task = new TimerTask() {
    public void run() {
       System.out.println("Message");
    }
};
```

Para ejecutar el `TimerTask`, antes debe crear una instancia `Timer`:

```
Timer timer = new Timer();
```

Entonces organiza una tarea repetitiva mediante los métodos `schedule()` o `ScheduleAtFixedRate()`. Hay seis variantes de los dos métodos organizadores:

- Estas dos variaciones ejecutan la tarea después del retraso especificado o en el momento indicado:

```
schedule(TimeTask task, retraso long)
schedule(TimeTask task, Fecha hora)
```

- Estas dos variaciones ejecutan la tarea repetidamente, comenzando tras esperar el retraso especificado en milisegundos o en el momento especificado y luego repite la tarea después de, haya transcurrido, al menos, el tiempo especificado en milisegundos:

```
schedule(TimeTask task, retraso long, periodo long)
schedule(TimeTask task, Fecha hora, periodo long)
```

- Estas dos variaciones ejecutan la tarea repetidamente, comenzando después de esperar el tiempo especificado en milisegundos o en el momento especificado y luego repite la tarea. Si el sistema está ocupado cuando llega el momento de ejecutar una de las tareas, las siguientes ejecuciones se agruparán.

```
scheduleAtFixedRate(TimeTask task, retraso long, periodo long)
scheduleAtFixedRate(TimeTask task, Fecha hora, periodo long)
```

La diferencia entre el método `schedule()` y el método `scheduleAtFixedRate()` es el tiempo entre las sucesivas ejecuciones de la tarea organizada. Con el método `schedule()`, el periodo es un intervalo mínimo. Por otra parte, con `ScheduleAtFixedRate()` el organizador temporal agrupará sucesivas ejecuciones si el sistema está ocupado y no puede realizar una o mas ejecuciones de la tarea.

El periodo de tiempo especificado para ambos métodos organizadores no es el retardo desde el final de la ejecución de la tarea, sino la diferencia de tiempo desde el principio de la anterior ejecución.

La tabla 8.1 le ayudarti a comprender el comportamiento organizativo y las diferencias entre `schedule()` y `scheduleAtFixedRate()`. Básicamente, `ScheduleAtFixedRate()` juega a coger para asegurarse de que se ejecuta el numero de tareas adecuado en el tiempo asignado, mientras que `schedule()` sólo se preocupa de comenzar la siguiente ejecución tras esperar, al menos, el periodo de tiempo especificado.

Tabla 8.1. Comparación de schedule() y scheduleAtFixedRate() para el comportamiento de temporizador de thread

schedule(tarea, 500)	**scheduleAtFixedRate(tarea, 500)**
1: 500 ms	1: 500 ms
2: 1000 ms	2: 1000 ms
Sisterna ocupado hasta dentro de 1700 ms	**Sisterna ocupado hasta dentro de 1700 ms**
3: 1700 ms	**3:** 1700 ms
4: 2200 ms	4: 2000 ms
Sisterna ocupado hasta dentro de 3100 ms	**Sistema ocupado hasta dentro de 3100 ms**
5: 3100 ms	5: 3100 ms
6: 3600 ms	6: 3110 ms
7: 4100 ms	7: 3500 ms

*** Presupone 10ms para ejecutar la tarea**

La clase `UtilTimerTest` proporciona un ejemplo del uso de `java.util.Timer` mejor que el anterior. Aparte de escribir un mensaje, la clase descansa durante un segundo y cuarto.

También notará que se escribe el tiempo real en varios puntos para que vez el efecto organizador. Es libre de cambiar la invocación al método `schedule()` para `scheduleAtFixedRate()` o de incrementar el tiempo de `sleep()` para ver cómo cambia el comportamiento organizador.

```
import java.util.*;

public class UtilTimerTest {
    public static void main(String args[]) {
        TimerTask task = new TimerTask() {
            public void run() {
                System.out.println("Message");
                System.out.println("Before sleep:" + new Date());
                try {
                   Thread.sleep(1250);
                } catch (InterruptedException ignored) {
                }
                System.out.println("After sleep:" + new Date());
            }
        };
        Timer timer = new Timer();
        timer.schedule(task, 500, 5000);
        System.out.println("Main: " + new Date());
    }
}
```

Nota

El paquete `javax.swing` proporciona su propia clase `Timer` que funciona de la misma manera que la version `java.util`, aunque la versidn swing proporciona algunas nuevasfunciones. En lugarde programar una`TimerTask` para ejecutarla, proporcione un `ActionListener`. La diferencia clave entre los dos es que la irnplernentacion `Timer` Swing ejecuta las tareas de temporización en el thread que envía eventos. Se percatara de la importancia de esto cuando aprenda a trabajar con interfaces graficos.

Detener la repetición de tareas periódicas

Aunque los métodos `schedule()` y `scheduleAtFixedRate()` aceptan un valor para el periodo de repetición de una tarea, se repiten automática y perpetuamente. Si necesita cancelar la ejecución de todas las tareas repetitivas, puede invocar al método `cancel()` de `Timer`.

Para cancelar la ejecución de una unica tarea, invoque el método `cancel()` del `TimerTask` especifico.

Demostración de prioridades y temporización

El programa del listado 8.2 es un ejemplo mas extenso de muchas de las funciones explicadas en este capitulo hasta el momento. La principal función de este programa es crear cuatro thread, de diferentes prioridades, cada uno con un contador hasta 10 millones.

Periódicamente, se muestran lo valores del contador que esté activo. Por defecto, el contador realiza un bucle sobre `yield()` cada 5.000 pasadas. Puede pasar cualquier argumento para hacer que el programa no abandone.

El comienzo de la definición de clase es la parte que vuelca la información de thread. Para que se muestre la información, sólo tiene que invocar al método `printAll()`. Este encontrara el grupo de thread raiz y luego escribirá la información sobre todos los thread y los grupos de thread de ese grupo.

Listado 8.2. Controlador de organizaciones, parte 1: Volcar el estado de un thread

```
import java.util.*;
public class ScheduleTester {

    private static int DEPTH-WIDTH = 2;
    private static int SAFETY-BLIFFEE = 3;

    public static void printAll() {
        ThreadGroup parent = Thread.currentThread().getThreadGroup();
        ThreadGroup root = parent;
        // Sigue hasta que encuentres root.
```

```
        while ((parent = parent.getParent()) != null) {
             root = parent;
        }
        printGroup(0, root);
    }

    private static void printGroup(int depth, ThreadGroup group) {
        String indent = makeIndentationString(depth*DEPTH_WIDTH);
        if (group != null) {
        // Escribe información de grupo
        System.out.println(indent +
            "Thread Group: " + group.getName() t
            " / Mas Priority: " t group.getMaxPriority());
        // Obtiene el conjunto de subthread
        int activeCount = group.activeCount();
        Thread threads[] = new Thread[activeCount+SAFETY_BUFFER];
        activeCount = group.enumerate(threads, false);
        // Escribe información del thread
        for (int i=0; i<activeCount; i++) {
           printThread(depth+1, threads[i]);
        }
        // Obtiene el conjunto de subgrupos
        int activeGroups = group.activeGroupCount();
        ThreadGroup groups[] = new ThreadGroup[activeGroups+SAFETY_BUFFER];
        activeGroups = group.enumerate(groups, false);
        for (int i=0; i<activeGroups; i++) {
           printGroup(depth+1, groups[i]);
        }
      }
    }

    private static void printThread(int depth, Thread thread) {
       String indent = makeIndentationString(depth*DEPTH_WIDTH);
       if [thread != null) {
            System.out.println(indent +
            "Thread: " + thread.getName() +
            " / Priority: " + thread.getPriority() +
            (thread.isDaemon() ? " [Daemon]" : "") +
            (thread.isAlive() ? " [Alive]" : " [NotAlive]") +
            (thread.isInterrupted() ? " [Interrupted]" : "") +
            ((Thread.currentThread() == thread) ? " <<== current" : ""));
       }
    }

    private static String makeIndentationString(int length) {
        StringBuffer indentation = new StringBuffer(length);
        for (int i=0; i<length; i++) {
           indentation.append(" ");
        }
        return indentation.toString();
    }
```

La segunda parte del programa (**listado 8.3**) es la clase interna `PriorityThread` que funciona como actor. Su constructor requiere una prioridad y una ID, donde ID es el índice de la matriz contadora con la que se trabaja.

Listado 8.3. Controlador de organizaciones, parte 2: Thread actor

```
private static int LIMIT = 10000000;
private static int YIELD = 5000;
private static int priorities[] = 11, 2, 4, 41;
private static int THREAD-COUNT = priorities.length;
public static boolean yield = true;
public static int counter[] = new int[THREAD_COUNT];

static class PriorityThread extends Thread {
   int id;
  PriorityThread(int priority, int id) {
     super("PriorityThread#" + id);
     this.id = id;
     setPriority(priority);
  }
  public void run() {
      for (int i=0; i<LIMIT; i++) {
         if (ScheduleTester.yield && ((i % YIELD) == 0)) {
         yield();
      }
      ScheduleTester.counter[id] = i;


}
```

La parte 3 es la tarea temporizador (listado 8.4). Esto mostrará los valores actuales en las matrices contadoras. Cuando todos los thread alcanzan su limite, el programa termina.

Listado 8.4. Controlador de organizaciones, parte 3: Tarea ternporizadora

```
static class DumpTimerTask extends TimerTask {
    int step;
    public void run() {
       boolean done = true;
       System.out.print("Step " + (step++) + ": COUNTERS:");
       for (int i=0; i<THREAD_COUNT; i++) {
          System.out.print(" " + counter[i]);
          if (counter[i] < LIMIT) {
            done = false;
       }

    System.out.println();
    if (done) {
       System.exit(0);

  }
}
```

El resto de la definición de la clase `ScheduleTester` (listado 8.5) es el bucle del programa `main()`. Lo primero que hace el programa es comprobar si se ha pasado algun argumento para establecer el marcador **`yield`**. A continuación se crean los

nuevos `thread` y se inician. La información del thread general se vuelca antes de que se inicie `Timer`. La tarea de volcar la información del thread sólo muestra los valores del contador thread actual.

Listado 8.5. Controlador de organizaciones, parte **4:** Ejecucion principal

```
    private static int DELAY = 150;

    public static void main(String args[]) {
       if (args.length > 0) {
         yield = false;
       }
       System.out.println("Yielding: " + yield);
       for (int i=0; i<THREAD_COUNT; itt) {
          new PriorityThread(priorities[i], i).start();
       }
       printAll();
       Tinier timer = new Timer();
       timer.schedule(new DumpTimerTask(), 0, DELAY);
    }
}
```

Los resultados de este programa `ScheduleTester` pueden variar considerablemente entre plataformas dependiendo de la velocidad de la CPU y el JRE particular. Puede ajustar los valores de `DELAY`, `LIMIT` y `YIELD` para que la ejecución predeterminada con abandono voluntario de la CPU (la que hay por defecto) produzca como resultado, al menos, los 10 pasos mostrados. Si el programa va más allá del paso 20, deténgalo y cambie algunos valores; cambie al menos la variable `LIMIT` a un valor inferior o la variable `YIELD` a uno superior.

El listado 8.6 es una ejecución de ejemplo de `ScheduleTester` en un equipo con Windows 2000. Observe que el sistema trata a las thread con prioridad 1 y 2 como si fuesen prácticamente iguales. Las prioridades sólo son sugerencias para el organizador de thread.

Listado 8.6. Controlador de organizaciones, resultados con abandono voluntario de la CPU en Windows 2000

```
>java ScheduleTester
Yielding: true
Thread Group: system / Max Priority: 10
    Thread: Reference Handler / Priority: 10 [Daemon] [Alive]
    Thread: Finalizer / Priority: 8 [Daemon] [Alive]
    Thread: Signal Dispatcher / Priority: 10 [Daemon] [Alive]
    Thread: CompileThread0 / Priority: 10 [Daemon] [Alive]
    Thread Group: main / Max Priority: 10
        Thread: main / Priority: 5 [Alive] <<== current
        Thread: PriorityThread#0 / Priority: 1 [Alive]
        Thread: PriorityThread#1 / Priority: 2 [Alive]
        Thread: PriorityThread#2 / Priority: 4 [Alive]
        Thread: PriorityThread#3 / Priority: 4 [Alive]
Step 0: COUNTERS: 0 0 0 0
```

```
Step 1: COUNTERS: 0 0 1539999 1539645
Step 2: COUNTERS: 0 0 3206330 3204999
Step 3: COUNTERS: 0 0 4884999 4882373
Step 4: COUNTERS: 0 0 6540906 6539999
Step 5: COUNTERS: 0 0 8237149 8234999
Step 6: COUNTERS: 0 0 9934999 9934726
Step 7: COUNTERS: 1603155 1599999 10000000 10000000
Step 8: COUNTERS: 3140714 3139999 10000000 10000000
Step 9: COUNTERS: 4834999 4830242 10000000 10000000
Step 10: COUNTERS: 6528964 6524999 10000000 10000000
Step 11: COUNTERS: 8180632 8179999 10000000 10000000
Step 12: COUNTERS: 9874999 9874403 10000000 10000000
Step 13: COUNTERS: 10000000 10000000 10000000 10000000
```

Sin abandono voluntario de la CPU, el sistema se ejecutara mucho más deprisa y necesitara menos pasos.

Observe el listado 8.7. El rendimiento es mucho mejor porque el tiempo entre las llamadas a `yield()` era muy breve. (El sistema gastaba más tiempo intercambiando tareas que realizando el trabajo.) Sin abandono voluntario de la CPU, las tareas con menor prioridad tienen la posibilidad de ejecutarse mientras las tareas con la prioridad más alta son ejecutables.

Listado 8.7. Controlador de organizaciones, resultados sin abandono voluntario de la CPU en Windows 2000

```
>java ScheduleTester xxx
Yielding: false
Thread Group: system / Max Priority: 10
    Thread: Reference Handler / Priority: 10 [Daemon] [Alive]
    Thread: Finalizer / Priority: 8 [Daemon] [Alive]
    Thread: Signal Dispatcher / Priority: 10 [Daemon] [Alive]
    Thread: CompileThread0 / Priority: 10 [Daemon] [Alive]
    Thread Group: main / Max Priority: 10
        Thread: main / Priority: 5 [Alive] <<== current
        Thread: PriorityThread#0 / Priority: 1 [Alive]
        Thread: PriorityThread#1 / Priority: 2 [Alive]
        Thread: PriorityThread#2 / Priority: 4 [Alive]
Step 0: COUNTERS: 0 0 0 10000000
Step 1: COUNTERS: 253928 0 10000000 10000000
Step 2: COUNTERS: 5514618 5304057 10000000 10000000
Step 3: COUNTERS: 10000000 10000000 10000000 10000000
```

Veamos ahora algunos ejemplos que muestran las diferencias entre las plataformas de ejecución. El listado 8.8 muestra los resultados de ejecutar el programa en una plataforma Linux con abandono voluntario de la CPU. Observara que las tareas con baja prioridad, en realidad terminan en primer lugar.

Aparentemente, sólo un superusuario de ese sistema puede ejercer completamente la prioridad de tareas, de modo que algunos entornos de ejecución Linux omiten por completo las prioridades. El desvio hacia thread con prioridad inferior sucede porque se inician en primer lugar y por tanto se ejecutan un poco antes que tareas posteriores.

Listado 8.8. Controlador de organizaciones, resultados con abandono voluntario de la CPU en Linux

```
%java ScheduleTester
Yielding: true
Thread Group: system / Max Priority: 10
    Thread: Reference Handler / Priority: 10 [Daemon] [Alive]
    Thread: Finalizer / Priority: 8 [Daemon] [Alive]
    Thread: Signal Dispatcher / Priority: 10 [Daemon] [Alive]
    Thread: CompileThread0 / Priority: 10 [Daemon] [Alive]
    Thread Group: main / Max Priority: 10
        Thread: main / Priority: 5 [Alive] <<== current
        Thread: PriorityThread#0 / Priority: 1 [Alive]
        Thread: PriorityThread#1 / Priority: 2 [Alive]
        Thread: PriorityThread#2 / Priority: 4 [Alive]
        Thread: PriorityThread#3 / Priority: 4 [Alive]
Step 0: COUNTERS: 1783110 785986 390084 92394
Step 1: COUNTERS: 3150198 1177399 767819 553434
Step 2: COUNTERS: 3965283 1664744 1314320 1276359
Step 3: COUNTERS: 3965283 2369067 1875541 2592578
Step 4: COUNTERS: 4542082 2930867 2584559 3324036
Step 5: COUNTERS: 4938041 3659054 3148398 4199557
Step 6: COUNTERS: 6048823 3985042 3467921 5003547
Step 7: COUNTERS: 7328404 4226067 3885628 5652569
Step 8: COUNTERS: 8614111 4786045 4207069 6054552
Step 9: COUNTERS: 9087468 5619897 4829406 6692180
Step 10: COUNTERS: 9320449 6333716 5299285 7872090
Step 11: COUNTERS: 10000000 6808611 5813590 8766744
Step 12: COUNTERS: 10000000 8065726 6873753 9009859
Step 13: COUNTERS: 10000000 9395831 7485969 9662046
Step 14: COUNTERS: 10000000 10000000 10000000 10000000
```

El listado 8.9 muestra una ejecución en Linux sin abandono voluntario de la CPU, que produce resultados similares con menos pasos.

Listado 8.9. Controlador de organizaciones, resultados sin abandono voluntario de la CPU en Linux

```
%java ScheduleTester xxx
Yielding: false
Thread Group: system / Max Priority: 10
    Thread: Reference Handler / Priority: 10 [Daemon] [Alive]
    Thread: Finalizer / Priority: 8 [Daemon] [Alive]
    Thread: Signal Dispatcher / Priority: 10 [Daemon] [Alive]
    Thread: CompileThread0 / Priority: 10 [Daemon] [Alive]
    Thread Group: main / Max Priority: 10
        Thread: main / Priority: 5 [Alive] <<== current
        Thread: PriorityThread#0 / Priority: 1 [Alive]
        Thread: PriorityThread#1 / Priority: 2 [Alive]
        Thread: PriorityThread#2 / Priority: 4 [Alive]
        Thread: PriorityThread#3 / Priority: 4 [Alive]
Step 0: COUNTERS: 4280416 567786 372742 234757
Step 1: COUNTERS: 6927695 806208 745330 495580
Step 2: COUNTERS: 7908208 1630064 1885543 1335447
```

```
Step 3: COUNTERS: 8237990 3047497 2887339 2223215
Step 4: COUNTERS: 9099254 3626935 4141290 2987115
Step 5: COUNTERS: 10000000 4751856 4337709 4151860
Step 6: COUNTERS: 10000000 6631228 5193104 4933626
Step 7: COUNTERS: 10000000 7164854 6637425 6397575
Step 8: COUNTERS: 10000000 7970586 8328366 7627052
Step 9: COUNTERS: 10000000 9383576 8916498 9196464
Step 10: COUNTERS: 10000000 10000000 10000000 10000000
```

Para ofrecer un tercer punto de vista, el listado 8.10 es la ejecución en Solaris. Mientras Linux simplemente no hacía caso de las prioridades, Solaris tiene menos prioridades de thread que Java. De modo que hay niveles, como que las prioridades 1 y 2 sean tratadas casi de la misma forma (aunque el primer thread obtiene algo más de tiempo de CPU antes de que los otros comiencen).

Listado 8.10. Controlador de organizaciones, resultados con abandono voluntario de la CPU en Solaris

```
%java ScheduleTester
Yielding: true
Thread Group: system / Max Priority: 10
    Thread: Reference Handler / Priority: 10 [Daemon] [Alive]
    Thread: Finalizer / Priority: 8 [Daemon] [Alive]
    Thread: Signal Dispatcher / Priority: 10 [Daemon] [Alive]
    Thread: CompileThread0 / Priority: 10 [Daemon] [Alive]
    Thread Group: main / Max Priority: 10
        Thread: main / Priority: 5 [Alive] <<== current
        Thread: PriorityThread#0 / Priority: 1 [Alive]
        Thread: PriorityThread#1 / Priority: 2 [Alive]
        Thread: PriorityThread#2 / Priority: 4 [Alive]
        Thread: PriorityThread#3 / Priority: 4 [Alive]
Step 0: COUNTERS: 868473 0 0 0
Step 1: COUNTERS: 868473 0 584999 306
Step 2: COUNTERS: 868473 0 590624 514999
Step 3: COUNTERS: 668473 0 965733 734999
Step 4: COUNTERS: 868473 0 1259743 1064999
Step 5: COUNTERS: 868473 0 1704945 1209999
Step 6: COUNTERS: 868473 0 1929999 1574172
...
Step 29: COUNTERS: 868473 0 8594999 7805304
Step 30: COUNTERS: 868473 0 8719374 8220961
Step 31: COUNTERS: 868473 0 9272490 8220961
Step 32: COUNTERS: 868473 0 9272490 8778849
Step 33: COUNTERS: 868473 0 9835342 8778849
Step 34: COUNTERS: 868473 27357 10000000 9077379
Step 35: COUNTERS: 868473 204298 10000000 9370990
Step 36: COUNTERS: 868473 587916 10000000 9555591
Step 37: COUNTERS: 868473 920727 10000000 9777680
Step 38: COUNTERS: 979620 1148708 10000000 10000000
Step 39: COUNTERS: 1404999 1379999 10000000 10000000
Step 40: COUNTERS: 1694999 1664999 10000000 10000000
Step 41: COUNTERS: 2009999 1919999 10000000 10000000
...
Step 67: COUNTERS: 9203646 9219999 10000000 10000000
```

```
Step 68: COUNTERS: 9481205 9494999 10000000 10000000
Step 69: COUNTERS: 9761063 9774999 10000000 10000000
Step 70: COUNTERS: 9994999 10000000 10000000 10000000
Step 71: COUNTERS: 10000000 10000000 10000000 10000000
```

Y el listado 8.11 es Solaris sin abandono voluntario de la CPU.

Listado 8.11. Controlador de organizaciones, resultados sin **abandono** voluntario de la CPU en Solaris

```
%java ScheduleTester xxx
Yielding: false
Thread Group: system / Max Priority: 10
    Thread: Reference Handler / Priority: 10 [Daemon] [Alive]
    Thread: Finalizer / Priority: 8 [Daemon] [Alive]
    Thread: Signal Dispatcher / Priority: 10 [Daemon] [Alive]
    Thread: CompileThread0 / Priority: 10 [Daemon] [Alive]
    Thread Group: main / Max Priority: 10
        Thread: main / Priority: 5 [Alive] <<== current
        Thread: PriorityThread#0 / Priority: 1 [Alive]
        Thread: PriorityThread#1 / Priority: 2 [Alive]
        Thread: PriorityThread#2 / Priority: 4 [Alive]
        Thread: PriorityThread#3 / Priority: 4 [Alive]
Step 0: COUNTERS: 0 0 0 0
Step 1: COUNTERS: 0 612769 1079119 421535
Step 2: COUNTERS: 0 1237022 2315653 1558685
Step 3: COUNTERS: 0 2328117 4208960 3435569
Step 4: COUNTERS: 0 4091667 4859928 5330150
Step 5: COUNTERS: 0 5367236 6757948 6513415
Step 6: COUNTERS: 0 6473000 8653705 7778949
Step 7: COUNTERS: 0 8236654 10000000 9672562
Step 8: COUNTERS: 1691084 8890876 10000000 10000000
Step 9: COUNTERS: 2845464 10000000 10000000 10000000
Step 10: COUNTERS: 5071077 10000000 10000000 10000000
Step 11: COUNTERS: 7301280 10000000 10000000 10000000
Step 12: COUNTERS: 9705576 10000000 10000000 10000000
Step 13: COUNTERS: 10000000 10000000 10000000 10000000
```

Multitarea avanzada

Tras ver los conceptos basicos de la multitarea veremos la sincronización de thread, comunicación entre thread y variables locales de thread

Sincronización de thread

La sincronización es la forma en que se puede evitar que dos datos corruptos accedan a la vez al mismo dato. Como los thread de un programa comparten el mismo espacio de memoria, se puede dar el caso en que dos thread traten de acceder a la vez al mismo objeto. Y es aqui donde pueden surgir los problemas. Los

thread competirhn unos con otros por acceder a los mismos datos. De hecho, un thread sobrescribirá la información que haya creado otro thread. Un thread puede trabajar con el resultado que acaba de crear otro, y romper asi la consistencia de los datos. Ciertos mecanismos se han de bloquear para acceder a los datos criticos. Es la unica forma de asegurarse que se trabaja correctamente.

Por ejemplo, supongamos que tenemos un programa que se hace cargo de nuestra cuenta del banco. Para que el usuario pueda hacer un depósito se tendrhn que desarrollar tres subtareas:

- Hacerse con el balance que se encuentra en un servidor remoto, tarea que puede tardar unos cinco segundos.
- Agregar la cantidad del ingreso.
- Enviar la nueva cantidad al servidor remoto, operación que puede durar varios segundos.

Si tenemos dos thread dedicados a efectuar un depósito de 1.000 € cada uno y se inician a la vez, ambos trabajarán sobre la misma cantidad, por lo que el saldo final únicamente mostrará un unico ingreso de 1.000 €. En la tabla 8.2 tenemos un ejemplo. ¡El balance almacenado en el servidor remoto sólo almacena un depósito!

Tabla **8.2.** Dos thread para efectuar un depósito que se ejecutan a la vez

Hora	1º thread	2º thread	Balance en el servidor remoto
a	Obtencion de saldo		1.000€
b	Esperando...	Obtencion de saldo	1.000€
c	Balance = 1.000€	Esperando...	1.000€
d	Calcula nuevo saldo = 2.000€	Esperando...	1.000€
e	Determina el nuevo saldo	Esperando...	1.000€
f	Esperando...	Balance = 1.000€	1000€
g	Esperando...	Calcula nuevo saldo = 2.000€	1.000€
h	Esperando...	Determina el nuevo saldo	1.000€
i	Nuevo balance efectivo	Esperando...	2.00063
j		Nuevo balance efectivo	2.000€

El siguiente ejemplo muestra un programa que simula este caso. Tenemos una clase **Account** que se ha definido con tres métodos: `getBalance()`, que se encarga de solicitar el balance del servidor remoto aplicando un retraso ficticio de 5 segundos; `setBalance()` que escribe el nuevo saldo en el servidor remoto, simulando un retraso de otros cinco segundos; y `deposit()` que utiliza otros dos méto-

dos para completar la transacción. La clase `DepositThread` se devuelve a su estado original en el momento en que se inicia la operación. El programa principal crea una cuenta e inicia los dos thread que se encargan de efectuar sendos depósitos de 1.000€.

Listado 8.12. Sirnulacion bancaria erronea

```
public class Deposit {
     static int balance = 1000; // simula el saldo que se encuentra en el
servidor remoto

     static class Account {
        void deposit(int amount, String name) {
            int balance;
            System.out.println(name + " trying to deposit " + amount);
            System.out.println(name + " getting balance...");
            balance = getBalance();
            System.out.println(name + " balance got is " + balance);
            balance += amount;
            System.out.println(name + " setting balance...");
            setBalance(balance);
            System.out.println(name + " new balance set to " +
Deposit.balance);
        }
        int getBalance() {
           try {   // simula el retraso producido en la transferencia de
datos
              Thread.sleep(5000);
            } catch (InterruptedException ignored) {
            }
            return Deposit.balance;
        }
        void setBalance(int balance) {
            try {   // simula el retraso generado al actualizar el saldo
del servidor remoto
               Thread.sleep(5000);
            } catch (InterruptedException ignored) {

          Deposit.balance = balance;
        }
     }

     static class DepositThread extends Thread {
        Account account;
        int depositAmount;
        String message;
        DepositThread(Account account, int amount, String message) {
             this.message  = message;
             this.account  = account;
             this.depositAmount = amount;
        }
        public void run() {
            account.deposit(depositAmount, message);
        }
```

```
    }

    public static void main(String args[]) {
        Account account = new Account();
        DepositThread first, second;
        first  = new DepositThread(account, 1000, "#1");
        second = new DepositThread(account, 1000, "\t\t\t\t#2");
        // inicia las transaccioncs
       first.start();
       second.start();
       // espera a que terminen las dos transacciones
       try {
          first.join();
          second.join();
       } catch (InterruptedException e) {}
       // escribe el balance final
       System.out.println("*** Final balance is " + balance);
    }
}
```

Aqui tiene un ejemplo del resultado del programa del listado 8.12:

```
>java Deposit
#1 trying to deposit 1000
#1 getting balance...
                                        #2 trying to deposit 1000
                                        #2 getting balance...
#1 balance got is 1000
#1 setting balance...
                                        #2 balance got is 1000
                                        #2 setting balance...
#1 new balance set to 2000
                                        #2 new balance set to 2300
*** Final balance is 2300
```

Modelo del monitor de Java para la sincronización

Java utiliza la idea de los monitores para sincronizar el acceso a los datos. Un monitor es algo asi como una zona reservada donde se guardan todos los procesos protegidos que tiene el mismo bloqueo.

Sólo hay una llave que abra estos monitores y que, por lo tanto, permita el acceso a estos recursos protegidos. Si varios thread tratan de acceder a la vez a un monitor, sólo uno de ellos tendrd la llave que le franquearri la entrada. Los otros tendrrin que esperar a que éste termine de utilizar los recursos y devuelva la llave a la Máquina Virtual.

Una vez que un thread consigue la llave del monitor, tendrd acceso a todos sus recursos durante todo el tiempo que desee. Pero si quiere acceder a los recursos que controla otro monitor, tendrd que esperar a hacerse con la llave correspondiente. En cualquier momento, un thread puede tener llaves para varios monitores y varios thread pueden tener llaves para diferentes monitores al mismo tiempo.

Si varios thread estdn esperando a que actúe otra llave, se puede llegar a un punto muerto.

Nota

Para evitar los puntos muertos y supervisar el acceso a los monitores thread, pruebe la copia de la herramienta `OptimizeIt Thread Debugger` de `VMGear` que se encuentra en el **CD** que acompaña al libro.

En Java, los recursos que se encuentran bajo la protección de un monitor son fragmentos de código en forma de métodos o bloques de declaraciones que se encuentran entre parentesis. Si sólo se puede acceder a los datos a través de métodos o bloques protegidos por el mismo monitor, se sincronizari el acceso indirecto a los datos. Para indicar que el siguiente método o bloque de declaraciones se ha de sincronizar, se utilizari la palabra clave `synchronized` delante del método o bloque de declaraciones que debe sincronizar un monitor. Cuando se va a sincronizar un bloque de elementos, habrá que encerrar entre paréntesis un caso del objeto, y dicho bloque apareceri inmediatamente después de la palabra clave `syncronized` de tal forma que la Máquina Virtual de Java sepa qué monitor ha de comprobar.

Un monitor es algo así como un aparcamiento, donde todos los métodos sincronizados o bloques son los coches que se pueden conducir. Todos los coches comparten la misma llave. Todo lo que hay que hacer para entrar en el parking es disponer de la llave. Con ella podremos abrir y arrancar los coches que queramos. Cuando terminemos, devolveremos la llave para que otro conductor conduzca los coches. La figura 8.2 muestra la idea.

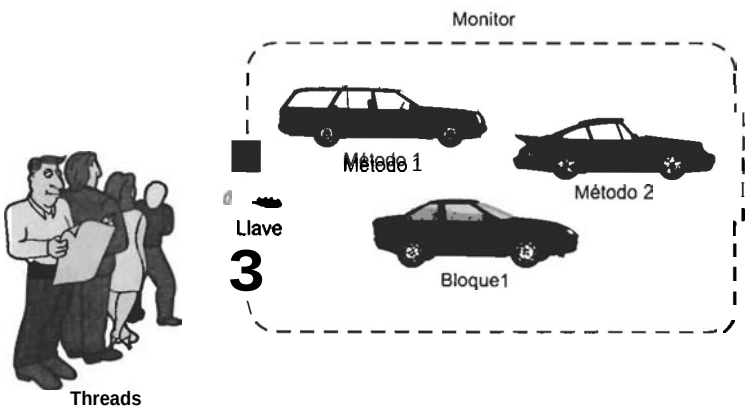


Figura 8.2. Los thread necesitan una misrna llave para acceder a los recursos que protege un monitor

Por ejemplo, el método `deposit()` del ejemplo anterior se puede sincronizar de tal forma que únicamente permita la ejecución de un thread a la vez. El único cambio que tenemos que hacer es la clase `syncronized`, que apareceri antes de la definición del método:

```
synchronized void deposit(int amount, String name) {
```

A continuación tiene un ejemplo de la nueva salida del programa:

```
#1 trying to deposit 1000
#1 qetting balance...
#1 balance got is 1000
#1 setting balance...
#1 new balance set to 2000
                                        #2 trying to deposit 1000
                                        #2 getting balance...
                                        #2 balance got is 2000
                                        #2 setting balance...
                                        #2 new balance set to 3000
*** Final balance is 3000
```

Como alternativa, se puede sincronizar un bloque de declaraciones dentro del método deposit()de la llamada al objeto:

```
void deposit(int amount, String name) {
    int balance;
    System.out.println(name t " trying to deposit " + amount);
    synchronized(this) {
        System.out.println(name + " getting balance...");
        balance = getBalance();
        System.out.println(name + " gets balance = " t balance);
        balance += amount;
        System.out.println(name + " setting balance...");
        setBalance (balance);
    }
    System.out.println (name + " set new balance = " + balance);
 }
```

La salida que genera este programa es muy parecida a la anterior, con la excepción de que el mensaje del segundo thread depende de los mensajes que genera el primer thread. Esto se debe a que el método `println()` no se encuentra dentro del bloque sincronizado. Esta es la salida que obtenemos ahora:

```
#1 trying to deposit 1000
#1 getting balance...
                                        #2 trying to deposit 1000
#1 gets balance = 1000
#1 setting balance...
                                        #2 getting balance..
#1 set new balance = 2000
                                        #2 gets balance = 2000
                                        #2 setting balance...
                                        #2 set new balance = 3000
*** Final balance is 3000
```

Todos los objetos que contengan un método sincronizado o a los que se haga una referencia desde un bloque sincronizado, tendrán una unica llave. Para los métodos de las clases sincronizadas la encargada de la distribución de la llave será la propia clase, porque se podrá invocar al método antes de que salga de la clase en cuestidn. Es decir, todos los objetos de las clases pueden tener un monitor si contienen algun

método sincronizado o bloques de declaraciones asociadas con él. Por lo tanto, la clave del monitor de una clase será diferente de las llaves de los monitores de los ejemplos de dicha clase

Diferencias en las tecnicas de sincronizacion

El siguiente ejemplo muestra la diferencia entre un método y un bloque sincronizado, asi como la diferencia que hay entre una sincronizacidn basada en una clase y una basada en un objeto. La clase `SyncToken` contiene tres métodos con distinta sincronizacidn pero todas llaman al mismo método `tricker()` para imprimir los tres trucos a intervalos irregulares. La clase `SyncTestRunner` es un thread que seleccionará entre los distintos métodos de la clase `SyncToken` para adaptar la ejecucidn a la ID que se le suministra. El método `main()` de la clase `SyncTest` generará 10 thread que se ejecutan con sincronizaciones distintas para que se puedan efectuar las comparaciones pertinentes. El listado del programa es el siguiente:

Listado 8.13. Comparacion entre bloqueos basados en clases y basados en objetos

```
import java.util.*;
public class SyncTest {

    static class SyncTestRunner extends Thread {
       SyncToken token;
       int id;
       SyncTestRunner(SyncToken token, int id) {
           this.token = token;
           this.id = id;
       }
       public void run() {
           switch (id % 3) {
               case 0:
                  SyncToken.classTicker("\t\t\tClass #" + id, token);
                  break;
               case 1:
                  token.methodTicker("Method #" t id);
                  break;
               case 2:
                  token.blockTicker("Block  #" t id);
                  break;

     }
   }

   static class SyncToken {
      Random random = new Random();
      // el método de las señales: da tres señales en un intervalo
      // aleatorio
      void ticker (String message) {
          for (int i = 0; i < 3; i++) {
             try {
               Thread.sleep (random.nextInt(800));
```

```
            } catch (InterruptedException ignored) {
            }
           System.out.println(message t ", tick #" + i);

      }

      // sincronizacihn basada en la clase
      static synchronized void classTicker(String message, SyncToken token)
{
         token.ticker(message);


      // sincronización basada en objeto: método sincronizado
      synchronized void methodTicker(String message) {
          ticker(message);


      // sincronizacihn basada en objeto: bloque sincronizado
      void blockTicker (String message) {
      synchronized(this) {
          ticker(message);
      I

   }

   public static void main(String args[]) {
      SyncToken token = new SyncToken();
      SyncTestRunner runners[] = new SyncTestRunner[10];
      for (int i = 0; i < 10; i++) {
          runners[i] = new SyncTestRunner(token, i);
          runners[i].start();

    }
 }
```

La salida del programa es la siguiente:

```
                                                  Class #0, tick #0
Method #1, tick #0
Method #1, tick #1
                                                  Class #0, tick #1
Method #1, tick #2
Block  #2, tick #0
                                                  Class #0, tick #2
                                                  Class #3, tick #0
Block  #2, tick #1
Block  #2, tick #2
                                                  Class #3, tick #1
Method #4, tick #0
Method #4, tick #1
                                                  Class #3, tick #2
                                                  Class #6, tick #0
Method #4, tick #2
                                                  Class #6, tick #1
                                                  Class #6, tick #2
Block  #5, tick #0
```

```
Block  #5, tick #1
                                        Class #9, tisk #0
Block  #5, tick #2
                                        Class #9, tick #1
Method #7, tick #0
Method #7, tick #1
Method #7, tick #2
                                        Class #9, tick #2
Block  #R, tick #0
Block  #0, tick #1
Block  #8, tick #2
```

Como se puede ver, los métodos sincronizados basados en objetos y en bloques comparten la misma llave del monitor cuando trabajan para un mismo objeto.

La sincronización es una operación cara, por lo que hemos de reducir su uso al minimo, sobre todo con los métodos que se ejecutan con cierta frecuencia o con los bloques de declaraciones. De todas formas, una sincronización ayudará a reducir la interferencia que se pueda producir entre los diferentes thread. Convenientemente utilizado, mejorara considerablemente la estabilidad y robustez del programa.

Comunicaciones entre thread

Este tipo de comunicación es la que permite que los thread de un programa hablen entre si. Los diferentes thread pueden comunicarse compartiendo datos o haciendo seiiales cuando se ha completado una actividad especifica.

La compartición de datos se estudió en una sección anterior con el ejemplo Deposit. Como todos los thread dentro de un programa comparten el mismo espacio de memoria, debe sincronizar el acceso a los datos más importantes para forzar el acceso exclusivo. Este acceso sincronizado se adquiere mediante el uso cuidadoso de la palabra clave `synchronized`.

Además de comunicarse mediante datos compartidos, los thread pueden comunicarse esperándose entre sí y enviándose seiiales cuando se completan ciertas operaciones. Estas seiiales se realizan mediante la invocación de los métodos de control de thread de las clases `Thread` y `Object`.

- El modo más sencillo de usar el método `join()` de `Thread`, que pide que se espere hasta que se complete otro thread antes de continuar.
- La clase `Object` ofrece tres métodos para obtener un mecanismo de comunicación entre thread más sofisticado: `wait()`, `notify()` y `notifyAll()`.

Uso de wait(), notify() y notifyAll()

Estos tres métodos de la clase `Object` deben usarse en un bloque sincronizado; en caso contrario, se arrojará `IllegalMonitorStateException`. El objeto usado como monitor sincronizado actua como guarda para las esperas y la notificación. El método `wait()` hace que el thread que invoca espere hasta que pase el tiempo

establecido o que otro thread llame a los métodos `notify()` o `notifyAll()` del monitor. La sinopsis del método `wait()` es:

```
wait()
wait(long esperaEnMilisegundos)
```

El primero espera hasta que se produzca la notificación del thread. El segundo esperará hasta que se complete el tiempo de espera o hasta que la notificación tenga efecto (lo que antes ocurra).

Cuando un thread llama al método `wait()`, la llave se le entregará al otro thread para que pueda acceder a los recursos que se encuentran dentro del monitor. El método `notify()` activará el thread que está esperando. En cambio, `notifyAll()` activará todos los thread que se encuentren en la cola de espera y que hayan solicitado acceso al monitor.

Estos métodos unicamente se pueden utilizar dentro del monitor o de un bloque sincronizado. El thread que invoca un método `notify()` o `notifyAll()` de un objeto habrá de tener la llave del monitor. Si no es asi, se lanzará una excepción `IllegalMonitorStateException` (un tipo de `RuntimeException`).

El siguiente ejemplo nos muestra el uso de los métodos `wait()` y `notify()` para resolver un problema de consumo y producción. En este problema, el productor generará los datos que consumirá. De todas formas, si la velocidad con que éste genera datos es mayor a la que tiene el consumidor para trabajar con ellos, los nuevos datos anularán a los antiguos. Por otro lado, si el consumidor absorbe más datos que los que genera el productor, llegará un momento en que consumirá datos que ya han pasado por sus manos. La sincronizacion por si sola no puede solucionar este problema porque sólo garantiza el acceso exclusivo a los datos, no su disponibilidad.

La primera implementación utiliza un monitor, un ejemplo de la clase `NoWaintMonitor`, para controlar el acceso a los datos, `token`. El productor y el consumidor trabajarán con `set()` y `get()` respectivamente, para configurar y hacerse con el valor del objeto `token` a intervalos aleatorios. El periodo máximo entre dichas acciones vendrá determinada por el argumento `speed` (velocidad) que se entrega a los constructores. El programa principal acepta hasta dos argumentos, que se pasarán a través de la linea de comandos, para determinar la velocidad de consumo y de producción. Crea un ejemplo del monitor, un productor y un consumidor, y los vigila durante 10 segundos

Listado 8.14. Producción erronea y comportamiento de consumo

```
import java.util.*;
public class NoWaitPandC {
    private static int PRODUCE_SPEED = 200;
    private static int CONSUME_SPEED = 200;
    private static Random random = new Random();

    static class NoWaitMonitor {
       int token = -1;
```

```
    synchronized int get() {
        System.out.println("Got: " + token);
        return token;
    }
    synchronized void set(int value) {
        token = value;
        System.out.println("Set: " + token);
    }
}

static class NoWaitProducer implements Runnable {
    NoWaitMonitor monitor;
    int speed;
    NoWaitProducer(NoWaitMonitor monitor, int speed) {
        this.monitor = monitor;
        this.speed = speed;
        new Thread(this, "Producer").start();
    }
    public void run() {
        int i = 0;
        while (true) {
            monitor.set(i++);
            try {
                Thread.sleep(random.nextInt(speed));
            } catch (InterruptedException ignored) {
            }
        }
    }
}

static class NoWaitConsumer implements Runnable {
    NoWaitMonitor monitor;
    int speed;
    NoWaitConsumer(NoWaitMonitor monitor, int speed) {
        this.monitor = monitor;
        this.speed = speed;
        new Thread(this, "Consumer").start();
    }
    public void run() {
        while (true) {
            monitor.get();
            try {
                Thread.sleep(random.nextInt(speed));
            } catch (InterruptedException ignored) {
            }
        }
    }
}
public static void main(String args[]) {
    if (args.length > 0)
        PRODUCE_SPEED = Integer.parseInt(args[0]);
    if (args.length > 1)
        CONSUME_SPEED = Integer.parseInt(args[1]);
    NoWaitMonitor monitor = new NoWaitMonitor();
    new NoWaitProducer(monitor, PRODUCE_SPEED);
    new NoWaitConsumer(monitor, CONSUME_SPEED);
```

```
      try {
        Thread.sleep(1000);
      } catch (InterruptedException ignored) {
      }
      System.exit(0);
    }
}
```

Aqui tiene un ejemplo de la salida del listado 8.14 donde el productor sobrepasa las posibilidades del consumidor. Podra apreciar que se generan muchos datos (por `Set`) pero se sobrescriben antes de ser procesados (por `Got`):

```
>java NoWaitPandC 100 400
Set: 0
Got: 0
Set: 1
Got: 1
Set: 2
Set: 3
Set: 4
Got: 4
set: 5
Set: 6
Set: 7
Set: 3
Set: 4
Set: 10
Got: 10
Set: 11
Set: 12
Got: 12
Set: 13
Set: 14
Set: 15
Set: 16
Set: 17
Set: 18
Got: 18
```

Ahora veremos un ejemplo de la salida que generará si la velocidad de consumo es mayor que la de producción: En esta ocasión, la mayor parte de los datos se procesa varias veces.

```
>java NoWaitPandC 400 100
Set: 0
Got: 0
Set: 1
Got: 1
Got: 1
Got: 1
Gut: 1
Set: 2
Got: 2
Got: 2
Got: 2
```

```
Got: 2
Got: 2
Got: 2
Got: 2
Got: 2
Got: 2
Set: 3
Got: 3
Set: 4
Got: 4
Got: 4
Got: 4
Got: 1
Got: 1
Got: 4
Set: 5
Got: 5
```

La segunda implementación de nuestro programa ejemplo (el próximo listado 8.15) utiliza los métodos `wait()` y `notify()` para asegurarse que se han creado los datos y que únicamente se utilizan una vez. El programa es igual que el del ejemplo anterior, con la excepción de la implementación del monitor. Se ha aiiadido una variable booleana, `valueSet`, para indicar si los datos están listos para su consumo. Lo primero que se hará será comprobar el método `get()` para ver si los datos están listos para entregarlos al consumidor. Si no es asi, el thread que efectua la llamada esperará a que otro thread prepare los datos y notifique su acción. A continuación, se configurará la variable booleana para indicar que se han consumido los datos. Para iniciar la producción, se notificará cualquier thread que vaya a generar nuevos datos. Si no hay ningun thread esperando para proceder con la producción, se ignorara el método `notify()`. Este es el método `get()`:

```
synchronized int get() {
    while (!valueSet) {
        try {
            wait();
        } catch (InterruptedException e) {
            System.err.println("Interrupted: " + e);
            System.exit(-1);

    }
    valueSet = false;
    System.out.println("Got: " + token);
    notify();
    return token;
}
```

Del mismo modo, el método `set()` comenzara por comprobar si se están utilizando los datos. Si no es así, el thread que efectúa la llamada tendrá que esperar a que otro thread los genere y proceda con su notificación. A continuación se configura la variable booleana para indicar que los datos están listos para proceder a su consumo. Luego se procedera a efectuar una notificación a todos los thread que estén esperando datos para que inicien su proceso de consumo. Si no hay ningun

thread esperando, se ignorará el método `notify()`. El método `set()` es el siguiente:

```
synchronized void set(int value) {
    while (valueSet) {
        try {
          wait();
        } catch (InterruptedException e) {
          System.err.println("Interrupted: " t e);
          System.exit(-1);


    valueSet = true;
    token = value;
    System.out.println("Set: " + token);
    notify();
\
```

El listado 8.15 muestra todo el programa.

Listado 8.15. Producción corregida y comportamiento de consumo

```
import java.util.*;
public class PandC {
    private static int PRODUCE_SPEED = 200;
    private static int CONSUME_SPEED = 200;
    private static Random random = new Random();

    static class Monitor {
       int token;
       boolean valueSet = false;
       synchronized int get() {
          while (!valueSet) {
               try {
                  wait();
               } catch (InterruptedException e) {
                  System.err.println("Interrupted: " t e);
                  System.exit(-1);
               }
          }
          valueSet = false;
          System.out.println("Got: " t token);
          notify();
          return token;
    I
    synchronized void set(int value) {
        while (valueSet) {
            try {
              wait();
            } catch (InterruptedException e) {
               System.err.println("Interrupted: " + e);
               System.exit(-1);
            }
        }
        valueSet = true;
```

```
      token = value;
      System.out.println("Set: " + token);
      notify();
    }
}

static class Producer implements Runnable {
   Monitor monitor;
   int speed;
   Producer(Monitor monitor, int speed) {
      this.monitor = monitor;
      this.speed = speed;
      new Thread(this, "Producer").start();
   }
   public void run() {
       int i = 0;
       while (true) {
            monitor.set(i++);
            try {
               Thread.sleep(random.nextInt(speed));
            } catch (InterruptedException ignored) {
            }
       }
   }


static class Consumer implements Runnable {
   Monitor monitor;
   int speed;
   Consumer(Monitor monitor, int speed) {
        this.monitor = monitor;
        this.speed = speed;
        new Thread(this, "Consumer").start();
   }
   public void run() {
       while (true) {
            monitor.get();
            try {
               Thread.sleep(random.nextInt(speed));
            } catch (InterruptedException ignored) {
            }
       }
  }


public static void main(String args[]) {
    if (args.length > 0)
      PRODUCE-SPEED = Integer.parseInt(args[0]);
    if (args.length > 1)
      CONSUME-SPEED = Integer.parseInt(args[1]);
    Monitor monitor = new Monitor();
    new Producer(monitor, PRODUCE-SPEED);
    new Consumer(monitor, CONSUME-SPEED);
    try {
      Thread.sleep(1000);
    } catch (InterruptedException ignored) {
```

```
        System.exit(0);
    }
```

La siguiente es una de las salidas que puede generar el listado 8.15, donde el productor es mas rapido que el consumidos. En esta ocasión, todos los datos generados se han consumido una sola vez.

```
>java PandC 100 400
Set: 0
Got: 0
Set: 1
Got: 1
Set: 2
Got: 2
Set: 3
Got: 3
Set: 4
```

Uso de notify() y notifyAll()

El metodonotify() advierte de un thread que esta esperando en un monitor, mientras que notifyAll() advierte de todos ellos. En la mayor parte de los casos, es mejor usar el método notifyAll() que notify(). Quizás piense que en un solo codigo las llamadas wait() y notify() son iguales y que funcionan de la misma forma. No obstante, si algo que no esta bajo su control esta esperando en el mismo monitor, podría ocurrir que nunca se advirtiera del thread original. Para asegurarse de que el comportamiento es el correcto cuando se despierta el thread y no tiene nada que hacer, coloque la llamada wait() en un bucle while. Este es el motivo por el que los metodos get() y set() de la clase Monitor del ejemplo PandC tienen bucles while; no se limitan a comprobar si valueSet es true o false.

Thread variables locales

Los thread variables locales (ThreadLocal) son especiales. Las variables de clase proporcionan una copia de una variable para todas las instancias de la clase y las variables de instancia proporcionan copias separadas de variables para cada instancia de la clase, los thread variables locales permiten que instancias de thread individuales tengan copias independientes de variables.

Lo normal es que las variables ThreadLocal no sean necesarias, aunque la probabilidad con InheritableThreadLocal sea muy distinta. Se convierten en una pieza muy importante cuando se precisan una serie de variables estáticas donde se pueda guardar un identificador, posiblemente para acceder a las bases de datos o al identificador de una sesión. A continuación, se compartirá la variable con

los thread que se estén ejecutando (`ThreadLocal`) o con un thread y todos los que desciendan de él (`InheritableThreadLocal`).

Nota

Una variable `ThreadLocal` **se inicializa mediante su método protegido** `initialValue()`. **El valor inicial por defecto de una variable** `ThreadLocal` **es null. Asi, para proporcionar un valor inicial diferente, debe subclasificar** `ThreadLocal` **y reemplazar el método. Tambien puede cambiar el valor posteriormente con el método** `set()`.

El **listado 8.16** muestra las diferencias que hay entre las variables de clase y las `ThreadLocal`. Con la multitarea, el programa crea varios ejemplos de una clase y utiliza la variable `counter` para llevar la cuenta del numero creado. Todos los thread tendrdn a su disposición la variable estdtica `ThreadLocal`. Su valor se utilizard para determinar el tiempo de espera. La clase interna, `MyThreadLocal`, permite que los thread descansen hasta **1.000** milisegundos. Para ello utiliza su método `initialValue()`.

Listado 8.16. Dernostracion de los thread variables locales

```
import java.util.Random;
public class ThreadLocalTest {

    // Crea el thread clase local
    // El valor inicial es un número aleatorio entre 0 y 999
    static private class MyThreadLocal extends ThreadLocal {
        Random random = new Random();
        protected Object initialValue() {
            return new Integer(random.nextInt(1000));
        }
    }

    // Define/crea el thread variable local
    static ThreadLocal threadLocal = new MyThreadLocal();
    // Crea la variable de clase
    static int counter = 0;

    // Incrementa la variable counter
    private ThreadLocalTest() {
        counter++;
    }

    // Muestra el thread variable local, counter y el nombre del thread
    private void displayValues() {
        System.out.println (threadLocal.get() + "\t" + counter +
            "\t" + Thread.currentThread().getName());
```

```
    public static void main (String args[]) {
        // Cada thread crea una nueva instancia de clase
        // Muestra información de variable
        // Y descansa durante una cantidad de tiempo aleatoria
        Runnable runner = new Runnable() {
            public void run() {
                ThreadLocalTest tlt = new ThreadLocalTest();
                tlt.displayValues();
                try {
                    Thread.currentThread().sleep (
                        ((Integer)threadLocal.get()).intValue());
                    tlt.displayValues();
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }

        };

        // Se crea otra instancia de clase y se muestran los valores
        ThreadLocalTest tlt = new ThreadLocalTest();
        tlt.displayValues();

        // Aquí es donde se crean realmente los thread
        for (int i=0; i<5; i++) {
            Thread t = new Thread(runner);
            t.start();
        }

}
```

Nota

Los numeros aleatorios se pueden generar mediante el método `Math.random()`, **que genera un double entre 0 (inclusive) y 1 (exclusive) o pueden generarse mediante la clase** `Random`. **El método** `nextInt()` **de** `Random` **devuelve un numero entre cero y el valor pasado.**

Observe que ambas variables son estdticas, es decir, se espera que haya una copia de todas las variables de la clase. Tras ejecutar el programa `ThreadLocalTest`, pensará de otro modo. Aqui tiene un ejemplo del resultado. (Como el programa utiliza numeros aleatorios, la salida que consigamos en nuestro ordenador puede variar de la que mostramos a continuación.)

```
4231  main
9712  Thread-0
8373  Thread-1
9804  Thread-2
9725  Thread-3
1776  Thread-4
1776  Thread-4
8376  Thread-1
```

```
9716  Thread-0
9726  Thread-3
9806  Thread-2
```

Observe que todos los thread (a excepción de `ThreadLocal`) comparten la variable `counter`. Si se compartiese `ThreadLocal`, veriamos el mismo valor repetido una y otra vez en vez de tener varias clases dentro del mismo thread.

Apreciará más la clase `ThreadLocal` cuando realice programación de red. Por ahora, sólo debe saber que existe y tenerla presente para cuando la necesita.

Thread demonio

El entorno de ejecución Java se beneficia de un tipo de thread especial para las tareas en segundo plano. Reciben el nombre de thread demonio. Estas thread de apoyo gestionan las tareas en segundo plano, como la recolección de elementos no deseados. Lo que hace a los thread demonio especiales es que si los unicos thread que se estin ejecutando en la máquina virtual son thread demonio, el entorno de ejecución se apagará y se cerrará.

> **Advertencia**
>
> Cuando trabaje con thread demonio, asegurese especialmente de recordar que la tarea del thread demonio puede detenerse en la mitad de una ejecucion cuando el entorno de ejecucion se cierra.

Señalar un thread como demonio es tan sencillo como invocar al método `setDaemon()`. Si toma el valor `true` el thread es un demonio; `false` significa que no lo es. Por defecto, todos los thread se crean con el valor inicial `false`.

```
setDaemon(boolean isDaemon)
```

El listado 8.17 muestra el comportamiento de los thread demonio. El thread principal crea un thread demonio que muestra un mensaje cada medio segundo. A continuación, el thread principal descansa durante cinco segundos. Aunque el thread demonio siga ejecutándose, el programa terminard porque los únicos thread que se estdn ejecutando son thread demonio.

Listado 8.17. Trabajar con thread dernonio

```
public class DaemonTest {
    public static void main(String args[]) {
        // Crea una acción ejecutable para un demonio
        Runnable daemonRunner = new Runnable() {
            public void run() {
                // Se repite por siempre
                while (true) {
```

```
                    System.out.println("I'm a daemon");
                    // Descansa medio segundo
                    try {
                       Thread.sleep(500);
                    } catch (InterruptedException ignored) {
                    }
                }
            }
        };
        // Crea e inicia el thread demonio
        Thread daemonThread = new Thread(daemonRunner);
        daemonThread.setDaemon(true);
        daemonThread.start();
        // Descansa cinco segundos
        try {
           Thread.sleep(5000);
        } catch (InterruptedException ignored) {
        }
        System.out.println("Done");
    }
```

Si se ejecuta el programa `DaemonTest` se produce el siguiente resultado:

```
I'm a daemon
I'm a daemon
I'm a daemon
I'm a daemon
I'm a daemon
I'm a daemon
I'm a daemon
I'm a daemon
I'm a daemon
I'm a daemon
Done
```

Es posible que aparezca otro mensaje "I'm a daemon" después de "Done" si el reparto de tiempos es correcto (o equivocado, dependiendo de su punto de vista). Si el programa no estuviera seiialado como demonio, el programa continuaria indefinidamente hasta que lo detenga manualmente, seguramente pulsando Control-C.

Nota

Si quiere un `java.util.Timer` demonio, pase `true` al constructor de clase. A partir de aquí, todas las tareas programadas seran demonios.

Agrupar thread

Con frecuencia se crean thread para muchas tareas a corto plazo. El thread se crea para realizar una tarea y permitir que recojan los elementos no utilizados

cuando se realiza una tarea. Aunque la creación de thread no es muy complicada, requiere algunos gastos generales. Asi, en las circunstancias adecuadas, la agrupación de thread es bastante util. Cuando debe realizar una tarea, en lugar de crear un `Thread`, solicite uno al grupo. Si hay una disponible, el thread se activarfi y realizará la tarea. Si no hay un thread disponible, deberá esperar hasta que se devuelva uno al grupo.

Las siguientes clases proporcionan un sencillo ejemplo de agrupación de thread. El ejemplo usa la clase `LinkedList` para mantener una cola de tareas que se deben ejecutar.

Si no conoce `LinkedList`, es una estructura de datos que mantiene referencias a los elementos anteriores y subsiguientes para insertarlos y eliminarlos más fácilmente. (Piense en una matriz que use punteros para ir a cada elemento en lugar de usar un índice de numeros enteros.)

La primera clase, `TaskThread`, es el thread que será parte del grupo de `thread`. Su mktodo `run()` es un bucle `while` que sólo espera hasta que aparece una tarea que realizar. Cuando el thread consigue una tarea, la ejecuta y espera hasta el siguiente trabajo. No obstante, aunque el thread esté esperando, no usa tiempo de CPU.

```
public class TaskThread extends Thread {
    private ThreadPool pool;

    public TaskThread(ThreadPool thePool) {
       pool = thePool;


  public void run() {
       while (true) {
            // bloqueado hasta que aparece el trabajo
            Runnable job = pool.getNext();
            try {
               job.run();
            } catch (Exception e) {
               // No hace caso de las excepciones arrojadas por los trabajos
               System.err.println("Job exception: " + e);
            }
       }
    }
}
```

La clase `ThreadPool` definida en el listado 8.18 es algo más complicada. El constructor crea e inicia todos los thread del grupo. Cuando se necesita que el grupo ejecute una tarea, el usuario invoca al mktodo `run()` del grupo. Cuando se agrega una tarea, se avisa al thread que estfi esperando. El grupo de thread usa el mktodo `getNext()`, por lo que no es declarado publico. Si no hay tareas que ejecutar, el thread esperará.

En caso contrario, la tarea que se agregó antes al grupo se pasará al thread de tarea para realizar el trabajo. Eso es todo lo que hay que saber.

Listado 8.18. La clase main para agrupar thread

```
import  java.util.*;

public class ThreadPool {
    private LinkedList tasks = new LinkedList();

    public ThreadPool(int size) {
        for (int i=0; i<size; i++) {
           Thread thread = new TaskThread(this);
           thread.setDaemon(true);
           thread.start();
        }
    }

    public void run(Runnable task) {
        synchronized (tasks) {
           tasks.addLast(task);
           tasks.notify();
        }


    Runnable getNext() {
        Runnable returnVal = null;
        synchronized (tasks) {
           while (tasks.isEmpty()) {
                try {
                   tasks.wait();
                } catch (InterruptedException e) {
                   System.err.println("Interrupted: " + e);
                }
           }
           returnVal = (Runnable)tasks.removeFirst();
        }
        return returnVal;
    }

}
```

A continuación tiene el código de prueba que es parte de la definición de clase del listado 8.18. La tarea de prueba que se realizará es simplemente mostrar un mensaje diez veces. El tamaiio del grupo es la mitad del numero de cadenas de mensaje que aparecen en la variable **message** (o **2,** porque la división entera elimina el resto).

```
    public static void main (String args[]) {
        final String message[] = ["One", "Two", "Three", "Four", "Five"];
        ThreadPool pool = new ThreadPool(message.length/2);
        for (int i=0, n=message.length; i<n; i++) {
           final int innerI = i;
           Runnable runner = new Runnable() {
               public void run() {
                   for (int j=0; j<10; j++) {
                    System.out.println("j: " + j t ": " + message[innerI]);
```

```
                    t
                }
            };
            pool.run(runner);
            }
            try {
               Thread.sleep(1000);
            } catch (InterruptedException ignored) {
            }
    }
```

Aqui tiene el aspecto **que** tendría **una** ejecución **del programa de prueba.** Probablemente **sus resultados** sean **distintos.**

```
j: 0: One
j: 1: One
j: 2: One
j: 3: One
j: 4: One
j: 5: One
j: 6: One
j: 7: One
j: 0: Two
j: 1: Two
j: 2: Two
j: 3: Two
j: 4: Two
j: 5: Two
j: 6: Two
j: 7: Two
j: 8: Two
j: 9: Two
j: 0: Three
j: 1: Three
j: 2: Three
j: 3: Three
j: 4: Three
j: 5: Three
j: 6: Three
j: 7: Three
j: 8: Three
j: 9: Three
j: 0: Four
j: 1: Four
j: 2: Four
j: 3: Four
j: 4: Four
j: 5: Four
j: 8: One
j: 9: One
j: 0: Five
j: 1: Five
j: 2: Five
j: 3: Five
j: 4: Five
```

```
j: 5: Five
j: 6: Five
j: 7: Five
j: 8: Five
j: 9: Five
j: 6: Four
j: 7: Four
j: 8: Four
j: 9: Four
```

Nota

Se pueden realizar muchas mejoras al grupo creado en el listado 8.18. Aunque funciona perfectamente, considere la posibilidad de agregar opciones para liberar los recursosdel grupo; crear con poco trabajo los thread en el momento en que se necesitan, en lugar de crearlos todos a la vez y agregar o eliminar thread del grupo.

Resumen

Hay muchos obstdculos en la programación de thread. Comprender y ser capaz de crear aplicaciones multitarea requiere tiempo y prtictica. Mientras crea programas multitarea, frecuentemente se encontrarti con los mismos patrones de desarrollo. Aqui tiene algunos indicadores para los patrones comunes con los que se encontrará.

- **Organización:** Use la clase `Timer` descrita en "Ejecutar tareas periódicamente" cuando necesite ejecutar varias tareas en un sólo thread con diferentes retrasos. No merece la pena intentar calcular el tiempo de descanso necesario entre el final de la tarea 1 y el comienzo de la tarea 2. Ademds, podría incluir un método `sleep()` al final del método `run()`, para probar y ejecutar una tarea cada n milisegundos. Pero a menos que pueda calcular la duración de la ejecución de la tarea, sertin n milisegundos desde el final de la ejecución y no desde el principio.
- **Bloqueo comprobado dos veces:** El bloqueo comprobado dos veces se produce cuando intenta evitar el acceso sincronizado a una variable sincronizando el acceso solamente a esa variable si aún no se ha inicializado. Si quiere realizar una inicialización de variables de instancia sin mucho esfuerzo, no use el bloqueo comprobado dos veces. No funcionarán con algunos compiladores optimizadores ni con algunos sistemas multiprocesador.

```
// Ejemplo de bloqueo comprobado dos veces roto
class MyClass {
   private Object obj = null;
```

```
   public Object getObject() {
       if (obj == null) {
          synchronized {
              if (obj == null)
                obj = new Object();
              }
          }
                return obj;

          // El acceso a obj siempre se realiza mediante getObject()
}
```

Si desea mas información sobre los entresijos de este tema, visite un de estos sitios en Internet:

```
http://www.cs.umd.edu/~pugh/java/memoryModel/DoubleCheckedLocking.html
http://www.javaworld.com/javaworld/jw-02-2001/jw-0209-double.html
```

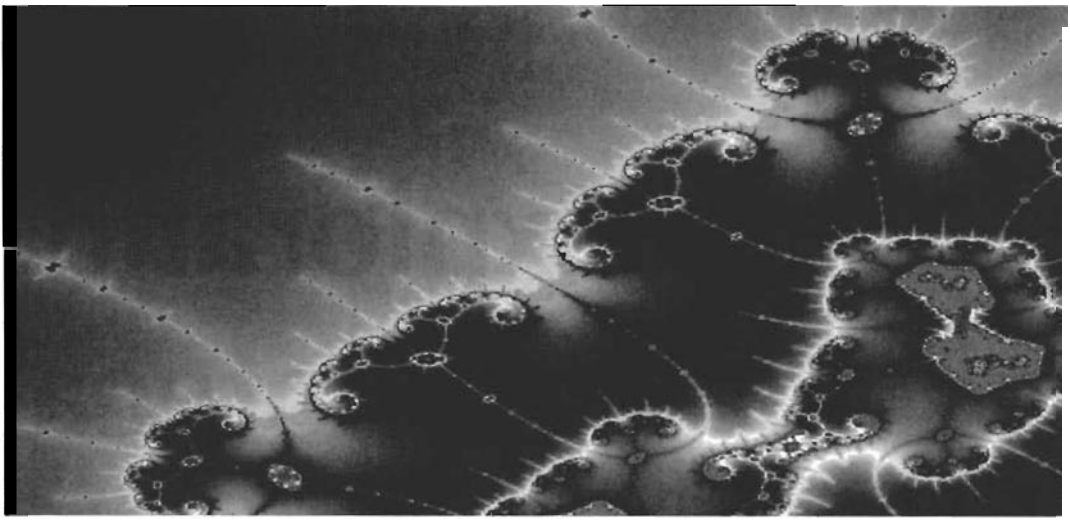
- **Acceso a datos:** Hay muchas maneras de optimizar el acceso concurrente a estructuras de datos. Puede examinar otras colecciones que optimizan las estructuras de datos para el acceso concurrente. Doug Lea mantiene una de estas colecciones en

```
http://gee.cs.oswego.edu/dl/classes/EDU/oswego/cs/dl/util/concurrent/
```

Además del diseño y el desarrollo de los patrones aqui mencionados, otros patrones tienen lugar al crear programas gráficos, especialmente cuando se trata con Java Foundation Classes y el conjunto de componentes Swing component set.

Parte II

Explorando las bibliotecas Java



9

Paquetes estándar de Java

En la primera parte de este libro, aprendió sobre las partes fundamentales del lenguaje de programación Java. En esta segunda parte, cambiaremos los modos y estudiaremos las diferentes bibliotecas que ayudan a formar Java Platform. Como Java ha evolucionado a lo largo de los años a través de sus diferentes versiones, los cambios más importantes entre las diferentes versiones se han concentrado en el conjunto de bibliotecas estándar. Por supuesto, la sintaxis del lenguaje de programación también ha cambiado algunas (revise el tema de las aserciones, que se agregaron en la versión 1.4, en el capitulo 7). No obstante, hablando de forma comparativa, los cambios a nivel de lenguaje han sido relativamente pocos. Lo que realmente define cada nueva versión de Java es el conjunto de bibliotecas. Y ésta sigue creciendo con cada nueva versión, como muestra la tabla 9.1

Tabla 9.1. Tamaños de las bibliotecas estandar de Java

Versión	Paquetes	Clases e interfaces
1.0	8	212
1.1	23	504
1.2	60	1781
1.3	77	2130
1.4	135	2738

Nota

Las estadisticas para las bibliotecas de Java 1.O-1.3 proceden de Java Developer's Almanac 2000 de Patrick Chan (Addison Wesley, 2000).

En este capitulo, le presentaremos las bibliotecas que son estdndar para Java 2 Standard Edition (J2SE) versión 1.4, con indicadores sobre dónde encontrar información adicional en capitulos posteriores. Aunque el lenguaje es relativamente sencillo, deberá estar familiarizado con lo que se encuentra disponible con el conjunto de bibliotecas estdndar. En caso contrario, pasará mas tiempo intentando adivinar lo que es cada cosa que usando realmente las clases de biblioteca adecuadas. Aparte de las bibliotecas estdndar presentadas aquí, el capitulo 23 explora muchas otras bibliotecas estandar de Java que no están incluidas en la versión J2SE. Ese capitulo examina bibliotecas que aparecen en Java 2 Enterprise Edition (J2EE), además de bibliotecas no empaquetadas con ninguna edición, conocidas como extensiones estandar. Las extensiones estdndar tambikn son conjuntos de biblioteca de Java, pero Sun no las incluye en las ediciones estdndar de Java. Si quiere usar las extensiones, debe descargarlas por separado.

Muchas bibliotecas se inician como extensiones estdndar antes incorporarse al ciclo de versiones estándar. Un sencillo indicador de si una biblioteca se inició como extensión estándar es comprobar el nombre del paquete. Si ese nombre comienza `javax.*`, probablemente fue una extensión. Como minimo, estos paquetes se pueden utilizar con varias versiones de Java.

Bibliotecas de utilidades

Algunos paquetes estdn tan arraigados en la tecnologia de desarrollo Java que todo el mundo debe conocerlos. Incluyen clases para compatibilidad con lenguajes principales, utilidades, corriente de E/S, formato de texto, componentes reutilizables y seguridad.

Paquete java.lang

Las clases del paquete `java.lang` están tan profundamente integradas en el desarrollo de Java que ni siquiera debe indicarle al compilador que las importe. Desde luego, ya se ha encontrado con varias de las clases que forman esta biblioteca: `String`, `Thread` y `Exception`, por nombrar algunos. En este paquete encontrard varias categorias de clases:

- Clases envolventes: Son clases que le permiten tratar tipos primitivos como objetos reales, por ejemplo, los objetos `int` y `float` reciben el nombre de `Integer` y `Float`. Use las clases envolventes para realizar conversiones

entre cadenas o tipos de datos especificos y las otras utilidades de conversión numérica.

- Compatibilidad de **cadenas:** Las clases `String` y `StringBuffer` le permiten trabajar con texto Unicode. La clase `string` se usa cuando necesita texto permanente de sólo lectura; y la clase `StringBuffer` se usa cuando necesita texto que pueda ser modificado. Estas clases son compatibles con emparejamiento de patrones, conversión y recorte, entre otras tareas.
- Compatibilidad con multitarea: Como se vio en el capitulo 8, las clases de apoyo se encuentran en `java.lang: Thread, Runnable, ThreadGroup, ThreadLocal` y `InheritableThreadLocal`. Con estas clases puede crear varios thread de ejecución en sus programas de Java.
- Bibliotecas Math: Las clases `Math` y `StrictMath` proporcionan una colección de funciones matemdticas para generar numeros aleatorios, funciones trigonométricas (seno, coseno, tangente, etc.), funciones logaritmicas y exponenciales y cdlculos como encontrar el valor mdximo y el minimo. La clase `Math` estd optimizada para la velocidad. Se garantiza que la clase `StrictMath` producird los mismos resultados en todas las plataformas, como ciertos algoritmos publicados.
- Acceso al sistema: Algunas clases de este grupo (`System, Runtime` y `Process`) proporcionan acceso a información y capacidades especificas de sistema. Otras ayudan a la carga de programas de Java y a un alto nivel de seguridad: `ClassLoader, SecurityManager, Class` y `Package`. La clase `System` es la que encontró con anterioridad con `system.out` para escribir mensajes. También puede encontrar información como la memoria disponible y la posición del entorno de ejecución a lo largo de estas clases. Las clases de acceso al sistema también se usan para iniciar programas especificos de ciertas plataformas.
- Errores y excepciones: Las clases base para el control de excepciones se encuentran en el paquete `java.lang: Throwable, Exception, Error` y muchas de sus subclases. Muchas de las subclases de excepción se encuentran en paquetes mds apropiados, como `IOException` en `java.io`.

Otros dos subpaquetes de Java usados con menos frecuencia de `java.lang` son `java.lang.ref` y `java.lang.reflect`. El paquete `java.lang.ref` ofrece compatibilidad con objetos de referencia, un modo de supervisar el uso de objetos para ayudar al recolector de elementos no utilizados. El paquete `java.lang. reflect` se usa para la reflexión (la capacidad de preguntar a una clase sobre si misma).

Paquete java.util y java.math

Los paquetes `java.util` y `java.math` contienen clases con utilidades generales, la mayoría de las cuales aparecen descritas en el capitulo 17. En el paquete

java.util, encontrará ayudas para la compatibilidad con tipos de datos abstractos (ADT) llamada Collections Framework, para fechas y calendarios y algunas clases para ayudar a la internacionalización de sus programas. Con el paquete java.math, hay ayudas para las operaciones aritméticas con precision arbitraria.

Las únicas clases de propósito general que no se describen en el capítulo 16 son las usadas para la internacionalización. Éstas incluyen las clases ResourceBundle, ListResourceBundle y PropertyResourceBundle, para personalizar los mensajes de texto basándose en la localización y el lenguaje elegidos.

Además, hay varios subpaquetes de java.util, todos ellos descritos en el capítulo 19:

- java.util.jar y java.util.zip, que ofrecen compatibilidad con archivos JAR (Java Archive), GZIP y ZIP.
- java.util.logging, la utilidad de registro básica.
- java.util.regex, la biblioteca de expresiones regulares para comparar patrones.
- java.util.prefs, que proporciona almacenamiento de las preferencias del usuario y la aplicación.

Truco

En plataformas basadas en Microsoft Windows, el paquete java.util.prefs proporciona acceso al registro de Windows.

Paquete java.io y java.nio

Los servicios de E/S de archivos y corrientes se encuentran el los paquetes java.io y java.nio. Curiosamente, la "n" de "nio" significa "nuevo". La E/S y los nuevos servicios de E/S aparecen descritos en el capítulo 19. Bisicamente, estas clases proporcionan la posibilidad de leer y escribir a nivel de byte, de cardcter, de primitivo y de objeto, incluyendo el acceso directo a la memoria intermedia de E/S nativa. Los filtros de E/S proporcionan nuevas posibilidades a las operaciones de lectura y escritura básicas. Por ejemplo, las corrientes de E/S básicas no se almacenan temporalmente, pero si no le gustaría realizar una operación de lectura y escritura más optimizada, puede agregar un filtro de memoria intermedia.

Paquete java.text

El paquete java.text proporciona clases de apoyo adicionales para la internacionalización de sus aplicaciones, además de las ya mencionadas. Estas tres grandes clases son DateFormat, NumberFormat y MessageFormat. Estas clases,

trabajando junto a la clase `java.util.Locale`, le permiten escribir fechas, números y mensajes en formatos específicos de cada país (por ejemplo, con los nombres de los meses en el idioma apropiado). De las tres clases de formato, `DateFormat` y `NumberFormat` aparecen descritas en el capítulo 17. `MessageFormat` funciona de la misma forma.

Otra clase importante de `java.text` es `AttributedString`. Use esta clase como `String` o `StringBuffer`, pero con el añadido de atributos como negrita, cursiva o tamaño potencialmente asociados con cada cardcter de la cadena. El modo de empleo de esta clase aparece en el capítulo 11.

Paquete java.beans

El paquete `java.beans` es la base del entorno de componentes reutilizables para Java, llamado tecnología JavaBeans. Todo el entorno estd tratado de forma más extensa en el capítulo 21.

Básicamente, los beans permiten un entorno Plug and Play en el que se crean pequeños componentes reutilizables que pueden reciclarse por todo el sitio. Los bean se pueden usar en aplicaciones GUI de cliente o con aplicaciones de servidor que usen Java Servlets y JavaServer Pages (JSP). Los componentes reutilizables no son el paquete `java.beans`, sino el marco que le ayuda a crear sus propios componentes reutilizables.

Paquetes de seguridad

Java Platform se construyó desde el principio pensando en la seguridad. SecurityManager en `java.lang` forma la base de esa arquitectura de seguridad. Con cada edición de Java, se han agregado nuevas posibilidades a la arquitectura de seguridad estdndar de Java. Java 1.1 agregó `java.security` al conjunto estdndar de paquetes Java, ofreciendo listas de control de acceso, generación de claves públicas/privadas y compendios de mensajes. Java 1.2 agregó normas de seguridad y un PolicyManager, que ofrecía un control de acceso más depurado. Java 1.3 sólo tenía cambios menores, pero 1.4 introdujo en el núcleo varias extensiones API que ya estaban disponibles anteriormente:

- `javax.security`: Java Authentication and Authorization Service (JAAS) agrega un marco de autentificación y autorización.
- `javax.crypto`: Java Cryptographic Extension (JCE) agrega compatibilidad con encriptación.
- `javax.net`: Java Secure Socket Extension (JSSE) agrega compatibilidad con SSL (Secure Socket Layer).
- `org.ietf.jgss`: Java GSS-API agrega compatibilidad con Kerberos V5. (GSS significa Servicio genérico de seguridad.)

Bibliotecas multimedia

Las bibliotecas relacionadas con multimedia de Java incluyen las bibliotecas necesarias para crear aplicaciones grdficas, dibujar, sonido, manipulación de imágenes y accesibilidad.

Paquete java.awt

AWT significa Abstract Window Toolkit. El paquete AWT se divide en dos partes. La primera es la ayuda para dibujar. Aqui encontrará clases y subpaquetes para dibujar formas, establecer patrones de relleno, manipular imágenes y trabajar con fuentes. Estos temas se estudiardn en los capitulos 10 y 11. La otra parte de AWT es un conjunto de componentes para crear aplicaciones gráficas y sus clases de apoyo.

Nota

En la mayoría de los casos, no usara estos componentes AWT; en su lugar usará componentes de `using javax.swing`, que se explicaran a continuacion, además de las clases de apoyo. Los capitulos 12, 14 y 16 describen las clases de apoyo.

Directamente relacionadas, pero fuera de la jerarquia de paquetes **`java.awt`** se encuentran dos bibliotecas, `java.applet` y **`javax.imageio`**. Ya ha visto al paquete **`java.applet`** en funcionamiento; es la base de la compatibilidad con applets. También hay un marco que se puede insertar para leer y escribir imbgenes en el paquete **`javax.imageio`**. El capitulo 11 estudia la E/S de imdgenes.

Paquete javax.swing

Forma parte de las Java Foundation Classes (JFC).También los es AWT, pero el término JFC suele escucharse más frecuentemente asociado a Swing que a AWT. Swing es el componente de segunda generación integrado en Java. Al usar componentes Swing con las clases de apoyo AWT (y las utilidades de dibujo) puede crear interfaces de usuario muy elaboradas. Los componentes disponibles para Swing comienzan con los botones y etiquetas mbs basicos y continuan con elementos más elaborados como menús, barras de herramientas, barras de progreso, tablas, árboles e incluso un completo editor compatible con HTML. Incluso es compatible con ventanas emergentes especializadas para escoger colores y archivos y mostrar mensajes. Todos estos componentes se describen en los capitulos 13 y 15.

Para mostrar los componentes Swing, la figura 9.1 muestra los resultados de un sencillo programa que muestra un sencillo árbol de control (listado 9.1).

Listado 9.1. Ejemplo del arbol Swing

```
import javax.swing.*;

public class TreeExample {
    public static void main(String args[]) {
        JFrame frame = new JFrame("The Tree");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JTree tree = new JTree();
        JScrollPane pane = new JScrollPane(tree);
        frame.getContentPane().add(pane);
        frame.setSize(250, 250);
        frame.show();
    }
}
```

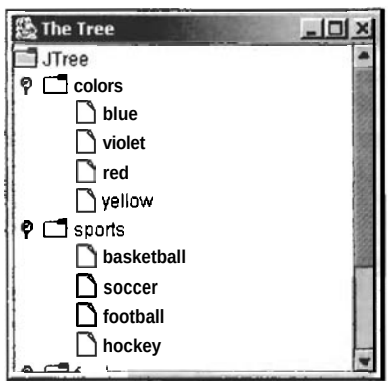


Figura 9.1. Ejemplo del componente Jtree de Swing

La API Java Accessibility (JAAPI) se encuentra en el paquete `javax.accessibility`. Las posibilidades de accesibilidad se integran en el conjunto de componentes Swing, incluyendo el apoyo para tecnologias de ayuda como lectores de pantalla, sistemas de reconocimiento de voz y pantallas Braille. Las posibilidades para trabajar sin ratón están integradas, pero se puede proporcionar información adicional para permitir que los componentes trabajen mejor con mecanismos de presentación alternativos. El capitulo 13 repasa brevemente algunas de estas opciones.

Paquetes java.awt.print y javax.print

Hay muchas clases de apoyo para que aparezcan cosas en pantalla con Java, la mayoría de ellas en `java.awt.print` y `javax.print`. Prácticamente, cada nueva versión de Java intenta mejorar los anteriormente pobres servicios gráficos. El capitulo 20 muestra el modo en que Java 1.4 muestra objetos en pantalla. Afortunadamente, los últimos intentos mejoran los métodos errdneos del pasado.

Paquete javax.sound

El siguiente es el paquete `javax.sound`. Naturalmente, esto agrega sonido para Java. Encontrará la posibilidad de reproducir archivos con formato AU, WAV, AIFF, RMF (Rich Music Format) y MIDI. Incluso hay un completo software mezclador para mezclar 64 canales de música. La reproducción de archivos de audio se estudia en el capitulo 12.

Para más información sobre cómo grabar sonidos y el mezclador, acuda a http://java.sun.com/products/java-media/sound/.

Bibliotecas para programas distribuidos

Las bibliotecas Java están llenas de clases y paquetes para programas distribuidos. Hay clases para la comunicación entre clases por Internet, establecer clientes y servidores, trabajar con XML, hablar con bases de datos relacionales, generar llamadas de procedimiento remoto (RPC) con invocación de método remoto (RMI) y CORBA (Common Object Request Broker Architecture), además de tratar con transacciones distribuidas y servicios de directorio.

Paquete java.net

El paquete `java.net` es el paquete para el trabajo en red en general, incluyendo compatibilidad con IPv6, la siguiente generación de protocolos de Internet. Las clases esenciales de `java.net` proporcionan compatibilidad con muchas tareas relacionadas con el trabajo en red:

- `InetAddress`, `URL` y `URI`, para direccionamiento en Internet.
- `Socket` y `serversocket`, para comunicaciones basadas en TCP/IP
- `DatagramPacket`, `DatagramSocket` y `MulticastSocket`, para comunicaciones basadas en UDP/IP.
- `Authenticator`, `PasswordAuthentication`, `URLDecoder`, `URLEncoder`, `URLConnection` y `JarURLConnection`, para acceso y manipulación de URL.

Muchas de estas capacidades están resaltadas en el capitulo 22.

Paquetes javax.xml, org.w3c.dom y org.xml.sax

XML (Lenguaje de marcado extensible) no es una tecnologia informática distribuida. Sin embargo, se suele usar cuando establece una comunicación entre aplicaciones porque proporciona un formato de proveedor neutral para transportar datos.

El capitulo 23 introduce Java API para XML Processing (JAXP).Encontrará compatibilidad con Document Object Model (DOM)Level 2, Simple API para XML Parsing (SAX)2.0 y XSL Transformations (XSLT)1.0 en Java 1.4.

Paquete java.sql y javax.sql

La API Java Database Connectivity (JDBC)proporciona acceso a bases de datos relacionales de Java. La API ha atravesado varias versiones propias y actualmente se encuentra en la 3.0.

Las anteriores versiones proporcionaban algunas capacidades estandar (`java.sql`) y algunas otras opcionales (`javax.sql`). Con Java 1.4, las opcionales ahora son estándar. La API JDBC se estudia en el capitulo 23.

Parte de la API de Java Transaction (JTA)es necesaria para algunas de las clases `javax.sql`, de modo que Java 1.4 ahora incluye el paquete `javax.transaction` como material estándar. No es el conjunto completo de clases para la API, sólo aquellas necesarias para mantener las capacidades JDBC.

Paquetes java.rmi, javax.rmi y org.omg

RMI y CORBA son tecnologias similares para, esencialmente, la misma función: ejecutar llamadas a métodos de forma remota. RMI se usa para activar llamadas de método en tiempos de ejecución Java diferentes. Por otro lado, CORBA se usa para activar métodos en tiempos de ejecucion separados que no tienen por qué ser Java. Para esta comunicacion, RMI usaba originalmente un protocolo llamado Java Remote Method Protocol (JRMP)para realizar comunicaciones entre tiempos de ejecución; JRMP se encuentra en el paquete `java.rmi`. CORBA, por otra parte, aun hoy usa un protocolo llamado Internet Inter-ORB Protocol (IIOP), que se encuentra en `org.omg`.

Ahora, RMI puede comunicar usando el protocolo IIOP y puede obtener el paquete `javax.rmi package`. RMI y CORBA se mencionan en el capitulo 23, pero necesitará mirar fuera de este libro para aprender cosas importantes sobre ellos. Constan de mucho más que el protocolo para comunicaciones.

Paquete javax.naming

El ultimo paquete a reseñar aqui es `javax.naming`, para la API Java Naming and Directory Interface (JNDI).Use JNDI para buscar servicios como conexiones a bases de datos, información de usuario en un directorio LDAP o componentes en un entorno distribuido.

Al igual que con RMI y CORBA, el capitulo 23 proporciona un poco de informacion sobre JNDI, pero necesitará buscar en otro sitio si quiere examinarlo a fondo.

Resumen

Aprender sobre las bibliotecas estdndar de Java requiere tiempo. Para convertirse en un buen programador de Java, deberá aprender todo lo que hay disponible antes de continuar y reinventar la misma cosa. Algunas tareas, como la creación de tipos de datos abstractos (ADT), son tareas estándar en el curriculum de un informático pero, ¿por qué preocuparse en crear y depurar una clase de lista vinculada si ya hay una disponible en el conjunto de bibliotecas estandar?

Certificación Java

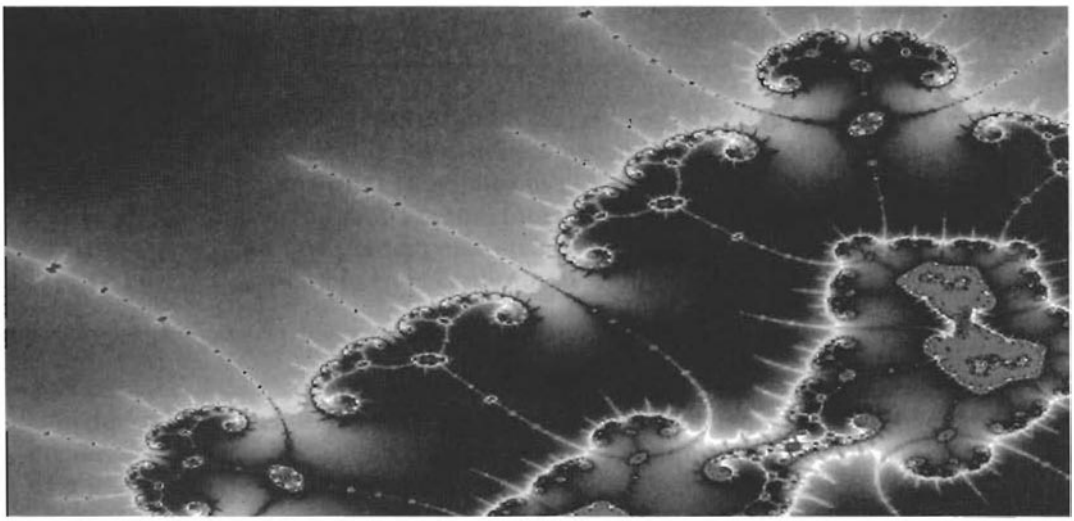
Uno de los objetivos de los programadores de Java es llegar a ser certificado por Java y para ello necesitara conocer los paquetes descritos en este capítulo. El primer nivel de certificado es:

```
Sun Certified Programmer for Java 2 Platform (conocido como SCJP)
http://suned.sun.com/US/certification/java/
```

Conocer casi todas las bibliotecas practicas es obligatorio para obtener el certificado, aunque no pasa lo mismo con las JavaBeans y la seguridad (y `nio` es demasiados nuevo, por el momento). Además, deberá conocer la mayor parte de las bibliotecas multimedia para obtener el certificado. No obstante, el dibujo y la accesibilidad no son demasiado importantes y no se requiere un conocimiento exhaustivo. Por lo que respecta a la programacion distribuida, no es imprescindible conocer las bibliotecas para el nivel inicial de certificacion. Otros niveles de certificacion Java son los siguientes:

```
Sun Certified Developer for Java 2 Platform
Sun Certified Web Component Developer for Java 2 Platform, Enterprise
Edition (J2EE)
Sun Certified Enterprise Architect for J2EE Technology
```

A medida que se adentra en los niveles de certificación, deberá conocer muchas mas API de Java, algunas de las cuales apenas se mencionan en este libro. Para ver mas detalles sobre cada uno de estos niveles, visite jCert Initiative en http:llwww.jcert.orgl.



10
Animación e imágenes

Las bibliotecas básicas proporcionan compatibilidad para muchas operaciones de dibujo diferentes. En realidad hay dos niveles de apoyo, dependiendo de la complejidad de las tareas que esté realizando. Los procesos de dibujo más simples se estudian en este capitulo, que le proporciona una sólida base para las operaciones más complejas que se estudiarán en el capitulo 11. Asi, antes de pasar a las interesantes cosas que puede conseguir con las avanzadas prestaciones gráficas, asegúrese de leer y dominar lo que aparece en este capitulo.

Aprenderá a dibujar todos los objetos gráficos básicos, además de a gestionar el sistema de coordenadas, la zona de recorte, los colores de la imagen y las fuentes del texto. Aparte del dibujo básico, este capitulo también trata los objetos de imágenes y la animación. Las imágenes son gráficos de mapa de bits que normalmente están guardados en el disco duro, impresas o colocadas en Internet. Este concepto es diferente de los rectángulos, óvalos, texto y otros objetos dibujados a mano en tiempo de ejecución (aunque puede crear una imagen a partir de algo dibujado a mano en tiempo de ejecución). Aprenderá a abrir, dibujar, crear y manipular objetos de imagen en sus programas. Ademds, aprenderd a abrir y reproducir archivos de sonido en programas Java.

Graficos básicos

La base de todas las operaciones de dibujo es la clase `Graphics`, definida en el paquete `java.awt`. La clase define una serie de métodos para operaciones bdsicas

de dibujo como dibujar una linea, un cuadrado o un circulo. La tabla 10.1 enumera todo lo que puede hacer la clase **Graphics.**

Tabla **10.1.** Composición de la clase Graphics

Variables/Métodos	Descripción
Graphics()	Crea un objeto **Graphics**
clearRect()	Limpia una zona rectangular
clipRect()	Contrae la zona de dibujo
copyArea()	Copia la zona de dibujo
create()	Crea una referencia secundaria en este objeto **Graphics**
dispose() finalize()	Libera recursos del sistema
draw3DRect() fill3DRect()	Dibuja un rectangulo tridimensional
drawArc() fillArc()	Dibuja un arco eliptico
drawBytes() drawChars() **drawstring()**	Dibuja texto
drawImage()	Dibuja una imagen
drawLine()	Dibuja una linea
drawOval() fillOval()	Dibuja un óvalo
drawPolygon() fillPolygon()	Dibuja un poligono cerrado
drawPolyline()	Dibuja un poligono abierto
drawRect() fillRect()	Dibuja un rectangulo
drawRoundRect() fillRoundRect()	Dibuja un rectangulo con los bordes redondeados
getClip() getClipBounds()	Obtiene la zona de recorte actual
getColor()	Obtiene el color con el que se esta dibujando
getFont()	Obtiene la fuente actual
getFontMetrics()	Obtiene la medida de la fuente
hitClip()	Comprueba si la zona cruza la zona de recorte
setClip()	Cambia la zona de recorte
setColor()	Cambia el color con el que se esta dibujando
setFont()	Cambia la fuente actual
setPaintMode()	Cambia el modo de dibujo a normal

Variables/Métodos	Descripción
`setXORMode()`	Cambia el modo de dibujo a exclusivo o pintando
`toString()`	Devuelve la representación de cadena del estado del objeto
`translate()`	Mueve el origen de las operaciones de dibujo

Cuando se trabaja con **`Graphics`**, nunca se crea un objeto gráfico directamente. En su lugar, se pasa el contexto gráfico a uno de los métodos de dibujo: **`paint()`**, **`update()`** y `paintComponent()`. Se le proporciona un controlador para la subclase especifica de la plataforma. Recuerde el primer programa del capitulo 2:

```
public void paint(Graphics g) {
     g.drawString(msg, 20, 30);
```

Las siguientes secciones describen cómo usar la mayoría de las funciones indicadas en la tabla 10.1.

Creación de un marco básico

Antes de pasar a los detalles del empleo de un objeto **`Graphics`**, tomemos un desvio y veamos la clase AWT **`Frame`** y algunas subclases. La clase **`Frame`** proporciona la ventana superior creada para la aplicación. Le concede una zona de dibujo y una barra de titulo, incluyendo decoraciones de ventana especificas para cada plataforma. Por ejemplo, la figura 10.1 le muestra el icono de marco que hace que aparezca el menu especifico de cada plataforma, junto con los iconos para minimizar, maximizar y cerrar el marco.

Figura 10.1. Ejemplo de marco

Una parte de la creación de un fotograma es especificar el titulo, pero para que el marco sea completamente útil, debe especificar el tamaño del marco y mostrarlo. El modo de empleo básico de la clase **`Frame`** es el siguiente:

```
Frame f = new Frame();
f.setTitle("Title Bar");
f.setSize(200, 100); // anchura, altura
f.show();
```

Cuando se trabaja con componentes **`swing`** en lugar de AWT, la ventana superior no es una instancia **`java.awt.Frame`** sino que se usa la clase `JFrame` del paquete

javax.swing.JFrame es una subclase de **Frame.** Aunque sus fundamentos son distintos, ambas ofrecen las mismas funciones básicas: una ventana superior con decorados de ventana especificos de cada plataforma. Las diferencias internas se tratarán en capitulos posteriores. Con **Frame y JFrame,** si hace clic en el icono para cerrar la ventana, la aplicación no finaliza. En el caso de la clase **Frame,** no sucede nada. En el caso de la clase **JFrame,** se oculta el marco. Este comportamiento puede parecer poco útil, pero piense que le permite definir lo que se puede realizar antes de cerrar la ventana y salir de la aplicación (por ejemplo, el importante paso de guardar los datos de la pantalla).

La clase ExitableJFrame

Para simplificar todas las aplicaciones de ejemplo de este libro, en lugar de trabajar directamente con **Frame** o **JFrame,** vamos a Crear una subclase una vez que se cierre la aplicacion. A continuación, para todos los ejemplos, usaremos esta nueva clase como base para nuestras ventanas superiores. La clase `JFrame` proporciona un método `setDefaultCloseOperation()` que permite especificar el comportamiento que desea para el icono Cerrar. Si desea salir de la aplicacion, solo tiene que pasar la constante **JFrame** de `EXIT_ON_CLOSE`. Esto realiza una llamada a **System.exit**`(0)` cuando el usuario selecciona el icono Cerrar.

```
import javax.swing.*;
public class ExitableJFrame extends JFrame {
    public ExitableJFrame() {
    }
    public ExitableJFrame(String title) {
        super(title);
    }
    protected void frameInit() {
       super.frameInit();
       setDefaultCloseOperation(EXIT_ON_CLOSE);
    }
}
```

Cuando tenga el cierre del marco definido (vea el cuadro adjunto sobre `ExitableJFrame`), sólo queda arreglar un par de cosas respecto a dibujar en una ventana superior. En primer lugar, dado que **JFrame** tiene un comportamiento por defecto en **paint** `()`, normalmente preferirá realizar ese comportamiento. Cuando se trabaja con una subclase, recuerde invocar **super.paint** `()`, a menos que esté seguro de no desear el comportamiento. Aqui tiene la llamada a **super.paint** `()`:

```
public void paint(Graphics g) {
    super.paint(g);
    // Realiza el trabajo...
}
```

En segundo lugar, necesitará colocar el sistema de coordenadas para dibujar. El punto (0, 0) representa la esquina superior izquierda de la zona de dibujo y (anchura-1, altura-1) es la esquina inferior derecha. Cuando dibuje en un marco superior, tenga en cuenta que la barra de titulo del marco tambien aparece. Debe recordar mover su dibujo hacia abajo y posiblemente, un poco a la derecha, si la ventana está envuelta por alguna de sus decoraciones. No necesita calcular este espaciado, puede preguntar al marco por su configuración con `getInsets()` y puede obtener las distancias necesarias. Sin embargo, debe esperar hasta que aparezca el marco antes de obtener su configuración. Hasta que parezca el marco, los valores son 0. Aqui tiene el marco básico para obtener esta configuración. Cómo los valores no cambian, sólo necesita obtenerlos una vez.

```
Insets insets;

public void paint(Graphics g) {
    super.paint(g);
    if (insets == null) {
       insets = getInsets();
    }
    // Realiza el trabajo...
}
```

La clase `Insets` es muy simple. Como indica la tabla 10.2, cuando tiene un objeto `Insets`, accede a los cuatro valores, `bottom`, `left`, `right` y `top`, con una de las variables públicas de instancia. Puede obviar el resto de los métodos.

Tabla 10.2. Cornposicion de la clase Insets

Variables/Métodos	Descripción
`bottomleftrighttop`	Valor de un lado concreto
`Insets()`	Crea un objeto **`Insets`**
`clone()`	Crea una copia de los valores
`equals()`	Comprueba la igualdad
`hashCode()`	Genera codigo hash para **`Insets`**
`toString()`	Devuelve la representación de cadena del estado del objeto

Puede convertir el sistema de coordenadas arriba y abajo con el siguiente código:

```
g.translate(insets.left, insets.top);
```

Con esta disposición, el origen de todas las operaciones de dibujo será la parte superior izquierda de la parte visible del marco, en lugar de estar oculto bajo la

barra de titulo. Tambibn puede usar **insets.left** e **insets.top** como puntos de partida para sus operaciones de dibujo:

```
g.drawSomething(x+insets.left, y+insets.top, ...);
```

Si prepara el marco base le parece complicado, no se preocupe (en realidad no lo es). Y trabajar con puntos de partida sólo es necesario cuando dibuja directamente en el marco superior. En el capítulo 13 le enseñaremos a dibujar en un componente, que no requiere esta conversidn de valores. Aún tiene que invocar al método de dibujo fuente; serd `paintComponent()` en lugar de **paint**().

Dibujar líneas y rectángulos

Los siguientes mbtodos permiten las operaciones de dibujo bdsicas para líneas y rectángulos:

```
public void drawLine(int xl, int yl, int x2, int y2)
public void clearRect(int x, int y, int width, int height)
public void drawRect(int x, int y, int width, int height)
public void fillRect(int x, int y, int width, int height)
```

Estos mbtodos aparecen en el listado 10.1, cuyo resultado aparece en la figura 10.2.

Listado 10.1. Demostracion del dibujo de líneas y rectangulos

```
import java.awt.*;

public class Picasso extends ExitableJFrame {

    Insets insets;

    public void paint(Graphics g) {
        super.paint(g);
        if (insets == null) {
            insets = getInsets();

        g.translate(insets.left, insets.top);
        g.fillRect(30, 10, 200, 100);
        g.clearRect(50, 30, 70, 50);
        g.drawRect(60, 50, 40, 20);
        g.drawLine(10, 55, 250, 55);
    }

    public static void main(String args[]) {
        Frame f = new Picasso();
        f.setTitle("Picasso");
        f.setSize(300, 200);
        f.show();
    }
1
```

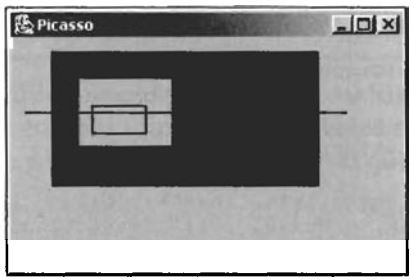


Figura 10.2. Dibujo con líneas y rectangulos

Dibujar polígonos

También encontrará ayuda para dibujar contornos de poligonos o poligonos rellenos. Un poligono se define como un conjunto de puntos, en el que el ultimo punto, si no está contenido en el conjunto, implicitamente es el primer punto. Al dibujar líneas entre un punto y el siguiente se conectan todas y se dibuja el polígono.

En el caso de los poligonos rellenos, el contexto gráfico se ocupará de las intersecciones para asegurarse de que se rellenan correctamente.

Si no quiere que se conecten los dos últimos puntos, debe usar el método de líneas múltiples, que no hace de forma implicita que el último punto sea el primero.

Aqui tiene los métodos para dibujar poligonos y líneas multiples:

```
public void drawPolygon(int xPoints[], int yPoints[], int nPoints)
public void drawPolygon(Polygon p)
public void fillPolygon(int xPoints[], int yPoints[], int nPoints)
public void fillPolygon(Polygon p)
public void drawPolyline(int xPoints[], irit yPoints[], int nPoints)
```

Los argumentos para estos métodos pueden ser cualquier conjunto de matrices que contengan las coordenadas individuales x e y de cada punto, o una instancia de la clase `Polygon`. Cuando trabaja con `polygon`, los puntos del poligono se definen agregándolos con `addPoint(x, y)`.

El programa del listado 10.2 muestra cómo se dibujan los poligonos, Observe la figura **10.3** para ver el resultado. (Esta figura usa puntos aleatorios por lo que su resultado puede ser algo diferente.)

Listado 10.2. Ejemplo del dibujo de poligonos

```
import java.awt.*;
import java.util.*;

public class Polys extends ExitableJFrame {

    Insets insets:
```

```
    Random random = new Random();

    public void paint(Graphics g) {
        super.paint(g);
        if (insets == null) {
            insets = getInsets();
        }
        g.translate(insets.left, insets.top);
        Polygon convex, concave, selfintersecting;
        convex = new Polygon();
        convex.addPoint(20, 20);
        convex.addPoint(60, 24);
        convex.addPoint(50, 50);
        convex.addPoint(21, 75);
        convex.addPoint(10, 30);

        concave = new Polygon();
        concave.addPoint(100+20, 20);
        concave.addPoint(100+60, 24);
        concave.addPoint(100+25, 50);
        concave.addPoint(100+21, 75);
        concave.addPoint(100+10, 30);
        concave.addPoint(100+20, 20);

        selfintersecting = new Polygon();

        for(int i=0; i< 8; i++) {
            selfintersecting.addPoint(200 + random.nextInt(80),
                                      10  + random.nextInt(80));
        }
        g.fillPolygon(convex);
        g.drawPolygon(concave);
        g.fillPolygon(selfintersecting);
    }
    public static void main(String args[]) {
        Frame f = new Polys();
        f.setTitle("Polygons");
        f.setSize(300, 200);
        f.show();
    }
}
```

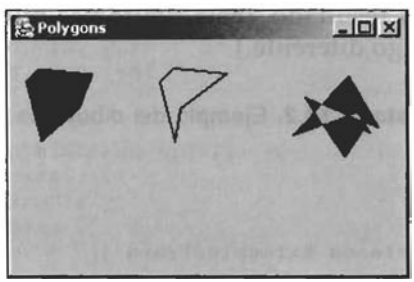


Figura 10.3. Ejemplo de dibujo de poligonos

Dibujar óvalos

No se pueden dibujar directamente circulos; en su lugar se dibuja un óvalo tan alto como ancho. La parte complicada de dibujar óvalos es que en lugar de trabajar con el centro y el radio, se proporciona un rectángulo que lo limita.

```
public void drawOval(int x, int y, int width, int height)
public void fillOval(int x, int y, int width, int height)
```

La figura 10.4 muestra el dibujo de óvalos concéntricos con relleno. Muestra el resultado del listado 10.3. En la figura dibujada no se puede apreciar, pero en la pantalla se produce un efecto de color.

Listado 10.3. Ejemplo del dibujo de ovalos

```
import java.awt.*;

public class Ovals extends ExitableJFrame {

    Insets insets;
    float hue = 0.0f;
    flcat saturation = 1.0f;
    float brightness = 1.0f;

    public void paint(Graphics g) {
        super.paint(g);
        if (insets == null) {
           insets = getInsets();
        }
        g.translate(insets.left, insets.top);

        int w = getWidth() - insets.left - insets.right;
        int h = getHeight() - insets.top - insets.bottom

        float zeroToOne;
        int squareSide = Math.min(w, h);
        for (int i=0, half=squareSide/2; i < half; i++) {
           zeroToOne = ((float) i) / (squareSide/2.0f);
           hue= zeroToOne;
           brightness = zeroToOne;
           g.setColor(Color.getHSBColor(hue, saturation, brightness));
           g.fillOval(i, i, w - 2*i, h - 2*i);
        }
    }

    public static void main(String args[]) {
        Frame f = new Ovals();
        f.setTitle("Ovals");
        f.setSize(300, 200);
        f.show();
    }
}
```

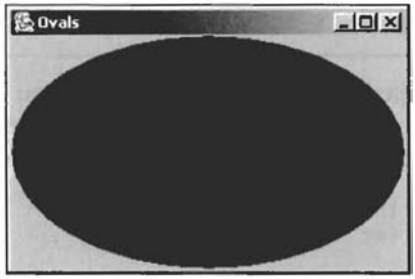


Figura 10.4. Ejemplo de dibujo de **ovalos**

Dibujar arcos

Cuando dibuja un arco, está dibujando un fragmento de un circulo. Al igual que pasaba con los dvalos, especifique el cuadro que delimita el arco. La diferencia entre un óvalo y un arco es que, en el arco, debe especificar el ingulo inicial y su longitud (no un ángulo final). Si nos fijamos en un reloj, un ángulo inicial de cero serian las 3 en punto y este ángulo aumentará en el sentido contrario al de las agujas del reloj. Un ángulo de 90º serían las 12 en punto, 180" serian las 9 en punto y 270" serian las 6 en punto. Un ángulo de 360 grados regresa a las 3 en punto.

Cuando se dibuja un arco relleno, se rellena como un trozo de tarta. El listado 10.4 y su resultado (figura 10.5) muestran el dibujo de un arco relleno.

Aqui tiene los métodos de dibujo de arcos:

```
public void drawArc(int x, int y, int width, int height, int startAngle,
    int arcAngle)
public void fillArc(int x, int y, int width, int height, int startAngle,
    int arcAngle)
```

Listado 10.4. Ejemplo del dibujo de arcos

```
import java.awt.*;

public class Arcs extends ExitableJFrame {

    Insets insets;

    public void paint(Graphics g) {
        super.paint(g);
        if (insets == null) {
           insets = getInsets();
        }
        g.translate(insets.left, insets.top);
        int w = getWidth() - insets.left - insets.right;
        int h = getHeight() - insets.top - insets.bottom;
        g.setColor(Color.MAGENTA);
        g.fillArc(0, 0, w, h, 45, 270);
    1
```

```
    public static void main(String args[]) {
        Frame f = new Arcs();
        f.setTitle("Arc");
        f.setSize(300, 200);
        f.show();
```

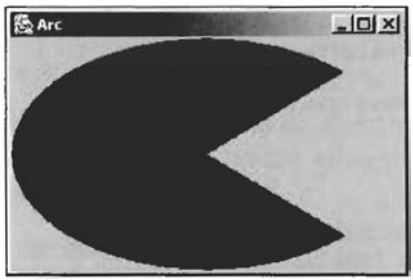


Figura 10.5. Ejemplo de dibujo de arcos

Dibujar texto

En la clase **`Graphics`** encontrará ayuda para procesar texto horizontalmente, de izquierda a derecha. Para dibujar texto girado se necesita Java 2D API descrito en el capítulo 11. Los cuatro métodos para procesar texto son los siguientes:

```
public void drawString(String str, int x, int y)
public void drawString(AttributedCharacterIterator iterator, int x, int y)
public void drawBytes(byte data[], int offset, int length, int x, int y)
public void drawChars(char data[], int offset, int length, int x, int y)
```

En tres de estos cuatro métodos, sólo tiene que pasar un una cadena al método basico **`drawstring`**`()`. El uso de `AttributedCharacterIterator` aparece en el capítulo 11, para dibujar entradas multilínea. En todos los casos, las coordenadas x e y pasadas al método especifican el principio del primer caracter.

El programa del listado 10.5 muestra la menos empleada versión `drawChars()` y el resultado aparece en la figura 10.6.

Listado 10.5. Ejemplo del dibujo de texto

```
import java.awt.*;

public class Text extends ExitableJFrame {

    Insets insets;

    public void paint(Graphics g) {
        super.paint(g);
        if (insets == null) {
```

```
            insets = getInsets();
        }
        g.translate(insets.left, insets.top);
        String subject = "ZigZagging Text";
        char text[] = subject.toCharArray();
        g.setFont(new Font("Serif", Font.PLAIN, 16));
        for (int i=0, len=text.length; i <= len-3; i+=2) {
            if (i==0) {
              g.drawChars(text, 0, len, 20, 20);
            } else if (i == len-3) {
              g.drawChars(text, 0, len, 20, 20 + (i/2)*17);
            } else {
              g.drawChars(text, (len-3-i), 4, 20+ (len-3-i)*7, 20+ (i/2)*17);
```

```
    public static void main(String args[]) {
        Frame f = new Text();
        f.setTitle("Text");
        f.setSize(200, 200);
        f.show();
```

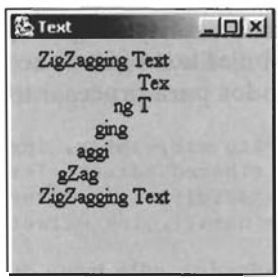


Figura 10.6. Ejemplo de dibujo de texto

Nota

El programa Text del listado 10.5 usa el método `setFont()` para cambiar la fuente que se esta usando en ese momento. El método `setFont()` toma un objeto `Font` que se puede construir al vuelo usando un nuevo `Font("fontFamilyName", fontStyle, pointSize)`. Mas adelante estudiaremos `setFont()`.

Control del estado de gráficos

El trabajo básico del contexto gráfico es dibujar líneas y formas. Además, hay que designar el modo en que se dibuja la linea o la forma. Se controlan varios

aspectos del contexto gráfico cuando se le pide que dibuje algun contenido, incluyendo:

- El origen de la región de dibujo.
- La región de dibujo delimitada por la zona de recorte.
- El color con el que se dibuja.
- El modo de dibujo.
- La fuente con la que se escribe el texto.

Cómo se demostró parcialmente con el uso de `setColor()` en el listado 10.3 y el uso de `setFont()` en el listado 10.5, todas ellas pueden ser configuradas e interrogadas por el contexto de gráficos actual.

Trasladar un sistema de coordenadas

Ya ha visto el efecto de trasladar el sistema de coordenadas en los listados de este capitulo. Para demostrarle que el efecto de translación es acumulativo, hemos realizado algunos cambios al anterior programa para dibujar poligonos. El siguiente código toma el método `paint()` empleado anteriormente y agrega algunas llamadas `translate()`, para evitar la operación de suma adicional moviendo las coordenadas de punto para cada poligono. Puede simplemente reemplazar todo el método `paint()` con el código del listado 10.6 (aunque las primeras cinco líneas no cambian).

Listado 10.6. Dernostracion de los efectos acurnulativos de la translacion

```
public void paint(Graphics g) {
      super.paint(g);
      if (insets == null) {
         insets = getInsets();


      g.translate(insets.left, insets.top);
      Polygon convex, concave, selfintersecting;
      convex = new Polygon();
      convex.addPoint(20, 20);
      convex.addPoint(60, 24);
      convex.addPoint(50, 50);
      convex.addPoint(21, 75);
      convex.addPoint(10, 30);

      concave = new Polygon();
      concave.addPoint(20, 20);
      concave.addPoint(60, 24);
      concave.addPoint(25, 50);
      concave.addPoint(21, 75);
      concave.addPoint(10, 30);
      concave.addPoint(20, 20);
```

```
        selfintersecting = new Polygon();

        for(int i=0; i< 8; i++) {
           selfintersecting.addPoint(random.nextInt(80),
                                     random.nextInt(80));
        }
        g.fillPolygon(convex);

        g.translate(100, 0);
        g.drawPolygon(concave);

        g.translate(100, 20);
        g.fillPolygon(selfintersecting);
    }
```

Si necesita reiniciar las coordenadas, simplemente introduzca un valor negativo para devolverla al punto en el que se encontraban, como se indica a continuación. ¡Pero recuerde lo lejos que las ha movido!

```
g.translate(x, y);
// dibuja
g.translate(-x, -y);
// regresa a la posición original
```

Especificar un área de recorte

Para restringir la zona de dibujo se establece el Area de recorte a una porción de la ventana. Esto le permite describir operaciones de dibujo con tareas que a gran escala son bastante sencillas, pero sólo se dibujarán pequefias partes del Area mayor. Por ejemplo, puede indicar que quiere dibujar un óvalo y, limitando el Area de recorte, sólo obtendrá medio óvalo.

Hay tres mktodos para cambiar el Area de recorte y dos para atraparla:

```
public void clipRect(int x, int y, int width, int height)
public setClip(Shape s)
public setClip(int x, int y, int width, int height)
public Shape getClip ()
public Rectangle getClipBounds()
```

La principal diferencia entre `clipRect()` y `setClip()` es que `clipRect()` puede reducir el Area de recorte solamente mediante una intersección con el Brea actual. En cambio, `setClip()` puede incrementar o reducir el tamafio a cualquier forma arbitraria.

Tambikn puede comprobar si alguna parte de una zona rectangular cruza el Brea de recorte:

```
public boolean hitClip(int x, int y, int width, int height)
```

Para mostrar estos métodos, el programa del listado 10.7 dibuja dos óvalos del tamafio de la pantalla. El segundo óvalo está relleno y se dibuja en una zona de recorte reducida (la mitad de la pantalla). Puede ver los resultados en la figura 10.7.

Listado 10.7. Demostracion del recorte de la zona de dibujo

```
import java.awt.*;
import java.util.*;

public class Clip extend:: ExitableJFrame {
    Insets insets;

    public void paint(Graphics g) {
        super.paint(g);
        if (insets == null) {
           insets = getInsets();
        }

        int x = insets.left;
        int y = insets.top;
        int width = getWidth() - insets.left - insets.right - 1;
        int height = getHeight() - insets.top - insets.bottom - 1;

        g.drawOval(x, y, width, height);

        int quarter = height/4;

        g.clipRect(x, y+quarter, width, height-quarter*2);
        g.fillOval(x, y, width, height);
    }

    public static void main(String args[]) {
        Frame f = new Clip();
        f.setTitle("Clip");
        f.setSize(300, 200);
        f.show();
    }
```

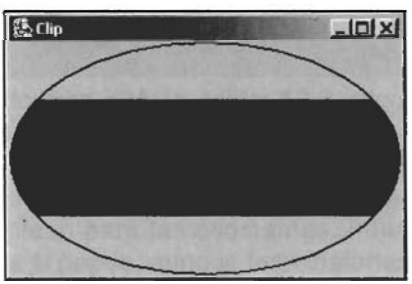


Figura 10.7. Dibujo usando la zona de recorte

Truco

Todas las propiedades del estado de los graficos se reinician con el valor por defecto para el siguiente paso por la rutina `paint()`. En el caso de la zona de recorte, ese valor por defecto seria toda la zona de dibujo.

Color del dibujo

El color con el que se dibuja en ese momento del contexto gráfico se modifica mediante el método `setColor()` y se recupera mediante `getColor()`.

```
public void setColor(Color c)
public Color getColor()
```

Aunque el color es un concepto sencillo, en la tabla **10.3** puede ver que la misma clase proporciona varias operaciones diferentes. Sin embargo, ninguno de esto métodos cambia el color; sólo permiten obtener información sobre el color actual o crear un nuevo `Color`.

Tabla 10.3. Composición de la clase Color

Variables/Métodos	Descripción
black y **BLACK**	Constantes para colores especificos. Vea la nota a continuación de la tabla
blue y **BLUE**	
cyan y **CYAN**	
darkGray y **DARK_GRAY**	
gray y **GRAY**	
green y **GREEN**	
lightGray y **LIGHT_GRAY**	
magenta y **MAGENTA**	
orange y **ORANGE**	
pink y **PINK**	
red y **RED**	
white y **WHITE**	
yellow y **YELLOW**	
Color()	Crea un objeto **color**
brighter()	
darker()	
decode()	
getHSBColor()	
`createContext()`	Crea un `PaintContext`

Variables/Métodos	Descripción
`equals()`	Conprueba la igualdad
`getAlpha()`	Obtiene una parte componente de color
`getBlue()` `getGreen()`	
`getRed()`	
`getRGB()`	
`getColor()`	Obtiene el color para la propiedad del sisterna de propiedades
`getColorComponents()`	Obtiene componentes a partir del espacio de color
`getComponents()`	
`getRGBColorComponents()`	
`getRGBComponents()`	
`getColorSpace()`	Obtiene el espacio de color
`getTransparency()`	Obtiene el rnodo de transparencia
`hashCode()`	Genera codigo hash para los insets
`HSBtoRGB()`	Para realizar conversiones entre representaciones de color
`RGBtoHSB()`	
`toString()`	Devuelve una representacion de cadena del estado de objeto

Nota

Si se esta preguntando por que la tabla 10.3 recoge las constantes en minuscula y en mayuscula, se debe a que las versiones en minuscula han estado presentes desde Java 1.0. Sun finalmente se dio cuenta de que este formato nose ajustaba a sus propias convenciones de nomenclatura en las que las mayusculas se emplean para las constantes. Intente usar solamente las versiones en mayuscula si puede, aunque las versiones en minúsculas no son incorrectas.

Si quiere cambiar el color, debe crear uno o usar una de las constantes de color. Hay varias rnaneras de crear colores:

- Puede crear un color especificando sus valores de rojo, verde y azul (RGB). Se pueden indicar como valores enteros entre 0 y 255; en un único número entero; o corno números decimales entre 0.0 y 1.0. Estos son los métodos:

```
public Color(int r, int g, int b)
public Color(int rgb)
public Color(float r, float g, float b)
```

- Puede agregar un componente alfa para describir la transparencia del color:

```
public Color(int r, int g, int b, int a)
public Color(int rgba, boolean hasAlpha)
public Color(float r, float g, float b, float a)
```

- Puede especificar los componentes del color usando los valores de matiz, saturación y brillo (HSB):

```
public static Color getHSBColor(float h, float s, float b)
```

- Puede especificar el valor del color en una cadena que contenga un valor numérico, como 0x00FF00FF para el magenta (el orden de los bites es alfa, rojo, verde, azul):

```
public static Color decode(String nm)
```

- Puede convertir un color existente aumentando o reduciendo su brillo:

```
public Color brighter()
public Color darker()
```

Nota

La clase `java.awt.SystemColor` define una serie de constantes de color que estan definidas para la tabla de busqueda de color interna del sistema de la plataforma. Estas constantes le permiten adivinar muchas cosas, como el color de fondo del escritorio (`SystemColor.desktop`), el color con el que se escribirá el texto (`SystemColor.textText`) y cual debe ser el color de fondo de una ventana (`SystemColor.window`).

Modo de dibujo

Todos los dibujos se pueden ejecutar en dos modos: paint o XOR. Para ello se utilizan estos dos métodos.

```
public void setPaintMode()
public void setXORMode(Color c)
```

- El modo paint se limita a sobrescribir los pixeles ya existentes, en el color de dibujo actual.
- El modo XOR se usa en pocas ocasiones, pero está disponible. Este modo aplica una función de intercambio de color que recuerda, aunque no es igual, a ORing exclusivo bit a bit. El mecanismo funciona de usando el color que se proporcionó como argumento a `setXORMode()`, el color de dibujo y el color

original. Por ejemplo, si el color actual es rojo (255, 0, 0), el color XOR es verde (0, 255, 0) y el color de dibujo azul (0, 0, 255), se dibujará un pixel blanco (255,255,255). Si dibuja otro pixel azul, el color del pixel volverá a ser rojo, porque (255,255,255) ^ (0, 255, 0) ^ (0, 0, 255) da como resultado el color rojo (255, 0, 0).

Fuente actual

Para controlar las fuentes tenemos dos métodos:

```
public void setFont(Font font)
public Font getFont()
```

La fuente actual se especifica usando el método `setFont()`. Toma un argumento que es un objeto `Font`, que suele crearse mediante el constructor `Font`:

```
public Font(String name, int style, int size)
```

- El argumento `name` indica la familia de la fuente. Las cinco predefinidas son Monospaced, `Dialog`, `DialogInput`, `SansSerif` o `Serif`.
- El argumento `style` puede ser `Font.PLAIN`, `Font.BOLD` o `Font.ITALIC`. Se pueden especificar en solitario o en combinación, todas ellas mediante `ORing`, como en

  ```
  Font.BOLD | Font.ITALIC
  ```

- El argumento `size` es el tamaño en pixeles.

El siguiente programa demostrativo, el listado 10.8, combina los objetos `Font` y `FontMetrics` (`FontMetrics` se explicará en la siguiente sección). Este programa determina la lista de todas las fuentes disponibles (es decir, disponibles para su tiempo de ejecución) y dibuja ejemplos de cada una. Obtiene la lista de fuentes de sistema en tiempo de ejecución de cualquier plataforma. La figura 10.8 muestra el resultado del programa.

Nota

El programa del listado 10.8 usa la clase `GraphicsEnvironment`, que se estudiara en el capítulo 11 junto con algunos trucos para usar fuentes.

Listado 10.8. Demostracion de solicitud de fuentes y su medida

```
import java.awt.*;

public class Fonts extends ExitableJFrame {

    Insets insets;
```

```
GraphicsEnvironment ge = GraphicsEnvironment.getLocalGraphicsEnvironment();
  String fontList[] = ge.getAvailableFontFamilyNames();
  Font fonts[] = ge.getAllFonts();

  public void paint(Graphics g) {
      super.paint(g);
      if (insets == null) {
          insets = getInsets();
      }
      g.translate(insets.left, insets.top);
      Font theFont;
      FontMetrics fm;
      int fontHeight = 0;
      for (int i = 0, n=fontList.length; i < n; i+=2) {
          System.out.println(fontList[i]);
          theFont = fonts[i].deriveFont(11f);
          g.setFont(theFont);
          fm = g.getFontMetrics(theFont);
          fontHeight += fm.getHeight();
          g.drawString(fontList[i] + " " t " 11 point", 10, fontHeight);

          if (i+1 != fontList.length) {
            System.out.println(fontList[i+1]);
            theFont = fonts[i+1].deriveFont(11f);
            g.setFont(theFont);
           g.drawString(fontList[i+1] + " " + "11 point", 200, fontHeight);
          }
      }


  public static void main(String args[]) {
      Frame f = new Fonts();
      f.setTitle("Fonts");
      f.setSize(435, 730);
      f.show();
  }
```

Nota

Las fuentes que aparecen en la figura 10.8 son especificas de un sisterna; su propio ejemplo dependera de las fuentes instaladas en su sisterna. Dependiendo de la cantidad de rnernoria de video disponible, quizás desee lirnitar el nurnero de fuentes que se muestra a un subconjunto.

Trahajo con tamaño de fuentes

Se proporcionan otros dos métodos relacionados con fuentes para acceder al conjunto de medidas de fuente:

```
public FontMetrics getFontMetrics()
public FontMetrics yetFontMetrics(Font f)
```

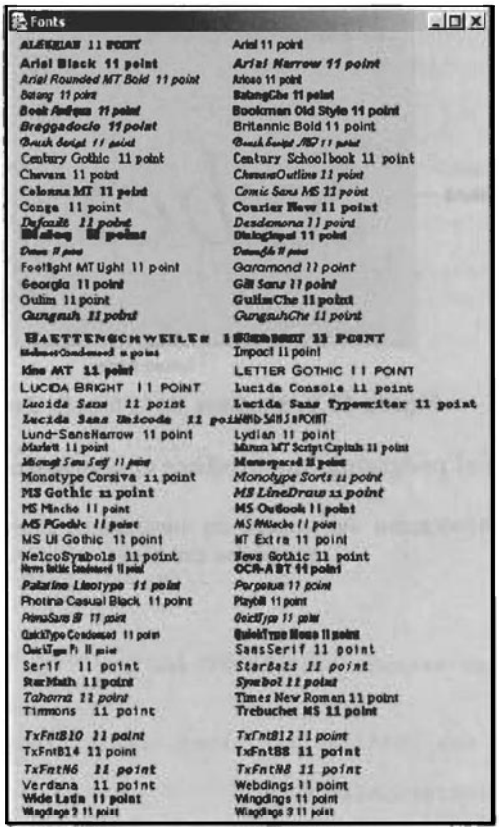


Figura 10.8. Muestra de fuentes del sistema

Cada vez que dibuje algo alrededor de texto, deberii saber las dimensiones exactas de las diferentes fuentes empleadas en el texto dibujado. Sin estas dimensiones, el resultado podría no ser simétrica o no estar centrada; peor aún, los siguientes dibujos podrian "chocar" con partes del texto. Todas las fuentes modernas traen incorporado un conjunto de medidas que definen varias alturas y anchuras de fuente. El siguiente programa muestra cómo obtener un objeto `FontMetrics` de una fuente y usarlo, en este caso, para centrar el texto y remarcar las medidas de forma gráfica, como en la figura 10.9.

Nota

Todo ese espacio sobre los caracteres, definido por la medida Ascension se usa para proporcionar espacio para dieresis, acentos y otras marcas especiales de caracter (diacriticas). La altura es para todos los caracteres de la fuente, no para cada caracter especifico. Se puede adquirir la anchura para una fuente especifica.

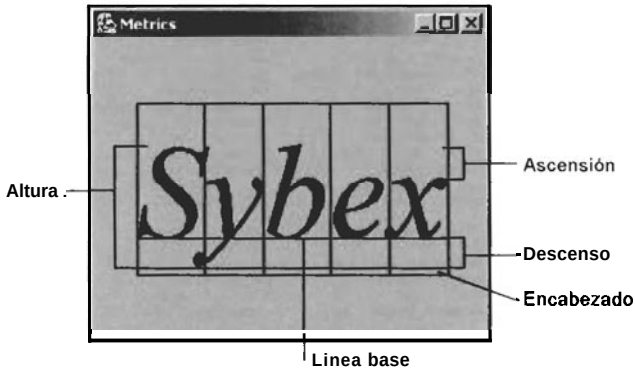


Figura 10.9. Medidas de la fuente Serif

El listado 10.9 es el programa que produce el resultado de la figura 10.9.

Listado 10.9. Demostracion del empleo de medidas de fu'ente para dibujar cuadros alrededor del texto

```
import java.awt.*;

public class Metrics extends ExitableJFrame {

    Insets inset-;
    Font theFont = new Font("Serif", Font.ITALIC, 100);

    public void paint(Graphics g) {
        super.paint(g);
        if [insets == null) {
           insets = getInsets();

        g.translate(insets.left, insets.top);
        int baseline = 150;
        String msg = "Sybex";

        g.setFont(theFont);
        FontMetrics fm = g.getFontMetrics();

        int ascent  = fm.getAscent();
        int descent = fm.getDescent();
        int height.  = fm.getHeight();
        int leading = fm.getLeading();
        int maxAdv  = fm.getMaxAdvance();
        int maxAsc  = fm.getMaxAscent();
        maxDes  = fm.getMaxDescent();
        // Centra la línea
        int width = getWidth() - insets.riyht;
        int stringWidth = fm.stringWidth(msg);
        int x = (width - stringWidth)/2;

        g.drawString(msg, x, baseline);
```

```
        drawHLine(g, x, stringWidth, baseline);
        drawHLine(g, x, stringWidth, baseline-ascent);
        drawHLine(g, x, stringWidth, baseline-maxAsc);
        drawHLine(g, x, stringWidth, baseline+descent);
        drawHLine(g, x, stringWidth, baseline+maxDes);
        drawHLine(g, x, stringWidth, baseline+maxDes+leading);

        int charX = x;
        for (int i=0; i <= msg.length(); i++) {
           drawVLine(g, charX, baseline-ascent, baseline+maxDes+leading);
           if (i != msg.length()) {
             charX += fm.charWidth(msg.charAt(i));

        }
    }

    void drawHLine(Graphics g, int x, int width, int y) {
        g.drawLine(x, y, x+width, y);


    void drawVLine(Graphics g, int x, int y, int height) {
        g.drawLine(x, y, x, height);
    }

    public static void main(String args[]) {
        Frame f = new Metrics();
        f.setTitle("Metrics");
        f.setSize(300, 250);
        f.show();
    }
}
```

Nota

Además de la clase `FontMetrics`, **tambien existe una clase** `LineMetrics` **en el paquete** `java.awt.font`. **Estas dos clases trabajan de la misma forrna. Se obtiene** `FontMetrics` **del contexto grafico y** `LineMetrics` **de la propia** Font.

Bases de la animación

La clase **`Image`** suele ser la base para realizar animaciones en Java. La clase representa una estructura interna que almacena una imagen de mapa de bits. Encontrarh ayuda para trabajar con imhgenes con formato JPEG, GIF, XBM e incluso PNG, integrada en las bibliotecas básicas. Y puede dibujar en una para crear la suya propia.

Trabajar con imágenes

La ayuda bdsica de Java para trabajar con imdgenes se encuentra en sólo dos métodos. En primer lugar, cargue la imagen mediante el método `getImage()` de `Toolkit` y luego podrd mostrarla mediante el método `drawImage()` de `Graphics`. Parece sencillo, pero hay algo mds de lo que parece a primera vista.

Carga

Veamos los dos métodos `getImage()`:

```
public Image getImage(String filename)
public Irnaqe getImage(URL url)
```

Para cargar una imagen, pase el nombre del archivo o la URL al método `getImage()`. Éste devolverd una referencia a un objeto `Image` que se cargará mds tarde. El método regresa inmediatamente, pero la imagen que se estd cargando espera (en realidad eso es todo lo necesario para cargar imdgenes en aplicaciones).

Los applets son un poco diferentes. En lugar de usar los metodos de `Toolkit`, la clase `Applet` proporciona su propio conjunto de métodos de carga de imagen:

```
Image getImage(URL url)
Image getImage(URL baseUrl, String name)
```

Debido a que las normas de seguridad de applet evitan que los applet no fiables se comuniquen con cualquier servidor aparte del que es el origen del applet, usard mds a menudo la segunda versión de `getImage()`. En ese caso, en lugar de introducir en el código de su programa la URL base para la que se ejecutard el applet, puede usar el método `getDocumentBase()` o `getCodeBase()` del applet como URL base.

Los dos método `getDocumentBase()` y `getCodeBase()` devuelven información ligeramente diferente sobre el applet. Si quiere cargar imdgenes en relación al sitio en el que el navegador cargo el archivo HTML usado para cargar el applet, use `getDocumentBase()`. Por otra parte, si quiere cargar las imdgenes en relación al sitio en el que el navegador cargo los archivos de clase del applet, escoja `getCodeBase()`.

Observe las siguientes dos líneas. Podría parecer que cargan archivos de imagen de la misma manera, pero buscan la imagen designada en sitios diferentes, dependiendo del aspecto de la etiqueta <APPLET> del cargador HTML.

```
Image image1 = getImage(getDocumentBase(), "images/dog.jpg");
Image image2 = getImage(getCodeBase(), "images/dog.jpg");
```

Si pensaba especificar el atributo `CODEBASE` para el applet, la primera línea buscará en relación al punto donde se encuentra el archivo HTML y la segunda línea buscará en el punto especificado por el atributo `CODEBASE`.

Dibujar

Para pasar de cargar imágenes a dibujarlas, observemos el método `drawImage()` o, mejor dicho, métodos. En realidad hay seis modos diferentes de dibujar una imagen. Y de estos seis, sólo deberia usar los dos primeros:

```
public boolean drawImage(Image image, int x, int y, ImageObserver observer)
public boolean drawImage(Image image, int x, int y, Color bgcolor,
    ImageObserver observer)
```

El argumento opcional `Color` del segundo método `drawImage()` es el fondo cuando los pixeles de una imagen son transparentes.

> **Nota**
>
> Las cuatro últimas versiones de `drawImage()` son para dibujar imagenes a escala. Su inconveniente es que realizan el escalado al vuelo y, posiblemente, de forma repetida cuando la pantalla debe dibujarse de nuevo. Mas adelante aprendera un modo mejor de realizar imagenes a escala.

```
public boolean drawImage(Image image, int x, int y, int width, int height,
    Color bgcolor, ImageObserver observer)
public boolean drawImage(Image image, int x, int y, int width, int height,
    ImageObserver observer)
public boolean drawImage(Image image, int dx1, int dy1, int dx2, int dy2,
    int sx1, int sy1, int sx2, int sy2, Color bgcolor, ImageObserver
observer)
public boolean drawImage(Image image, int dx1, int dyl, int dx2, int dy2,
    int sxl, int syl, int sx2, int sy2, ImageObserver observer)
```

ImageObserver

Al dibujar una imagen, se pasa el objeto Image devuelto de `getImage()` a `drawImage()`, junto con la esquina superior de la posición en la que quiere dibujar la imagen. El argumento `ImageObserver` agregará complejidad a la mezcla. La carga de imágenes no se activa hasta que algo necesita ver la imagen (ese "algo" es `ImageObserver`). A continuación, mientras se carga la imagen, el subproceso de carga informa periódicamente al observador.

Aunque no se realiza ninguna llamada de los métodos `ImageObserver`, este es el aspecto de la definición de la interfaz:

```
public interface ImageObserver {
    public static final int ABORT
    public static final int ALLBITS
    public static final int ERROR
    public static final int FRAMEBITS
    public static final int HEIGHT
    public static final int PROPERTIES
    public static final int SOMEBITS
```

```
    public static final int WIDTH
    public boolean imageUpdate(Image image, int infoflags, int x, int y,
        int width, int height.)
```

El método único de la interfaz, `imageUpdate()`, indica a qué se invoca. Pero no se preocupe, no tiene que implementar la interfaz. Esto se realiza mediante la clase `AWT Component` (y todas sus subclases). Siempre que esté dibujando la imagen en una subclase de un componente, como un `JFrame`, todo lo que tiene que hacer es pasarla como ultimo argumento:

```
public void paint(Graphics g) {
...
    g.drawImage(image, x, y, this);
...
}
```

Juntándolo todo

Revisemos los pasos de cargar, dibujar y la operación `ImageObserver`, para aclarar estas piezas revueltas del proceso de dibujar imdgenes:

1. Inicie la carga de una imagen con `getImage()`. Esta es la parte sencilla.
2. Dibuje la imagen con `drawImage()`, normalmente en el método `paint()` de un `Component`. Dado que la imagen aun no se ha cargado, la llamada a `drawImage()` comienza con la carga de la imagen.
3. Internamente, la carga se inicia y se registra `ImageObserver` con el subproceso de carga de imdgenes.
4. A medida que se cargan datos de la imagen, el subproceso de carta de imdgenes informa periódicamente a `ImageObserver`.
5. Cuando se informa de la implementación `ImageObserver` por defecto, se dibujan los datos `Image` adicional.
6. Hasta que la imagen esté completamente cargada, se repiten los dos pasos anteriores (4 y *5).*

Recuerde que el programador sólo necesita invocar a `drawImage()`. El resto del trabajo tiene lugar en segundo plano.

Para demostrarlo, el listado 10.10 es un sencillo ejemplo que carga una imagen y la dibuja. La figura 10.10 muestra el resultado.

Listado 10.10. Demostración de la carga y dibujo de una imagen

```
import java.awt.*;

public class LoadDraw extends ExitableJFrame {

    Image image;
```

```
Insets insets;

public LoadDraw(String filename) {
    super(filename);
    image = getToolkit().getImage(filename);


public void paint(Graphics g) {
    super.paint(g);
    if (insets == null) {
       insets = getInsets();
    }
    g.drawImage(image, insets.left, insets.top, this);
}

public static void main(String args[]) {
    if (args.length > 0) {
    Frame f = new LoadDraw(args[0]);
    f.setSize(300, 275);
    f.show();
} else {
    System.err.println("Please specify image to display");
}
```

Nota

En el método `paint()`, no olvide usar la configuración del marco como offset de dibujo; omítalos y comenzara a dibujar bajo la barra de título del marco.

Figura 10.10. Muestra de dibujo de imagen

Quizás no haya observado el dibujo parcial cuando está cargando y mostrando una imagen local, pero ¿qué ocurre si tiene una conexión a Internet lenta y está intentando cargar una imagen desde Internet? Al cambiar la llamada a `getImage()`

para usar una URL, puede observar que lo que aparece en pantalla llega en grupos de líneas cada vez. Asi es cómo funciona la interfaz `ImageObserver`: A medida que el subproceso de carga de imagen carga más datos, se invoca automáticamente al método `imageUpdate()` para que renueve la pantalla y dibuje los nuevos datos de la imagen.

```
import java.net.*;
...
//       image = getToolkit().getImage(filename);
       try {
          image = getToolkit().getImage(new URL(urlString));
       } catch (MalformedURLException e) {
          System.err.println("Bad URL");
          System.exit(-1);
```

Modo de empleo de MediaTracker

Si no le gusta el comportamiento de la carga retardada de imágenes, puede usar la clase `MediaTracker` (en el paquete `java.awt`) para forzar la carga de imágenes antes de que haga falta dibujarlas. Las imágenes se registran con el rastreador y luego se le indica que comience a cargar las imagenes. Puede esperar hasta que termine la carga o continuar haciendo otras cosas.

En cualquier caso, la carga puede comenzar inmediatamente. `MediaTracker` se describe en la tabla 10.4.

Tabla 10.4. Cornposicion de la clase MediaTracker

Variables/Métodos	Descripción
ABORTED	Señal para indicar el estado de la carga
COMPLETE	
ERRORED	
LOADING	
`MediaTracker()`	Crea un objeto `MediaTracker`
`addImage()`	Registra una imagen
`removeImage()`	Desregistra la irnagen
`checkAll()`	Comprueba el estado de la irnagen que se esta cargando
`checkID()`	
`getErrorsAny()`	Obtiene irnagenes descargadas erronearnente
`getErrorsID()`	

Variables/Métodos	Descripción
`isErrorAny()`	Cornprueba si alguna irnagen ha tenido errores al descargarse
`isErrorID()`	
`statusAll()`	Obtiene el estado de un conjunto de imagenes cargadas y empieza a cargarlas si es necesario
`statusID()`	
`waitForAll()`	Espera a que se cargue un conjunto de irnagenes y empieza a cargarlas si es necesario
`waitForID()`	

Al usar **`MediaTracker`** se siguen cinco pasos:

1. Crear el **`MediaTracker`**.

```
MediaTracker tracker = new MediaTracker(this);
```

2. Agregar las imágenes al rastreador. Para esto es necesaria una identificación de grupo para el seguimiento. Si se agregan múltiples imágenes con la misma identificación , no podrá esperar o comprobar el estado de una imagen especifica, sólo del grupo como un todo.

```
// 0 es la identificación del grupo
tracker.addImage(image, 0);
```

3. Iniciar la carga con uno de los métodos `statusAll()`, `statusID()`, `waitForAll()` o `waitForID()`.

```
try {
    tracker.waitForAll();
} catch (InterruptedException e) {
    System.err.println("Error waiting");
}
```

4. Comprobar el éxito de la carga.

```
if (tracker.isErrorAny()) {
    System.err.println("Error loading");
```

5. Elimina la imagen del rastreador. Si no elimina la imagen, la referencia no podrá ser recogida por el recolector de elementos no utilizados.

```
tracker.removeImage(image);
```

Para demostrarlo, el listado 10.11 ofrece un programa que usa **`MediaTracker`** para cargar una imagen y medir el marco para que ajuste a la imagen. Observe que incluso cuando atrapa las dimensiones de la imagen con `getWidth()` y

getHeight(), se le solicita que pase ImageObserver. Si la imagen adn no se ha cargado, estos métodos regresardn inmediatamente con un valor de -1 y comenzar la carga.

Listado 10.11. Dernostracion de la carga de una imagen con MediaTracker

```
import java.awt.*;

public class MediaLoadDraw extends ExitableJFrame {

    Image image;
    Insets insets;

    public MediaLoadDraw(String filename) {
        super(filename);
        image = getToolkit().getImage(filename);


    private void loadImage() {
        MediaTracker tracker = new MediaTracker(this);
        tracker.addImage(image, 0);
        try {
            tracker.waitForAll();
        } catch (InterruptedException e) {
            System.err.println("Error waiting");
        }
        if (tracker.isErrorAny()) {
            System.err.println("Error loading");
        }
     tracker.removeImage(image);
     }

     public void paint(Graphics g) {
         super.paint(g);
         g.drawImage(image, insets.left, insets.top, this);
     }

     public Dimension getScreenSize() {
         insets = getInsets();
         return new Dimension(
            image.getWidth(this) + insets.left + insets.right,
            image.getHeight(this) + insets.top + insets.bottom);
     }

     public static void main(String args[]) {
         if (args.length > 0) {
            MediaLoadDraw f = new MediaLoadDraw(args[0]);
            f.loadImage();
            f.show();
            f.setSize(f.getScreenSize());
         } else {
            System.err.println("Please specify image to display");
         }
     }
```

Si no quiere perder el tiempo esperando a que el método `loadImage()` termine, no se moleste en usar `MediaTracker` directamente. En su lugar, puede usar la clase `javax.swing.ImageIcon` para que cargue su imagen. Internamente, `ImageIcon` usa `MediaTracker` para cargar la imagen inmediatamente. El código de carga de imagen del listado 10.11 se reemplaza por las dos líneas en negrita del constructor del listado 10.12. En ambos programas, observe que el tamaño de la pantalla no se especifica hasta que ésta se muestra por primera vez. Esto se debe a que `getInsets()` no devuelve un valor válido hasta que percibe (muestra)la pantalla.

Listado 10.12. Dernostracion de la carga de una irnagen con ImageIcon

```
import java.awt.*;
import javax.swing.*;
public class IconLoadDraw extends ExitableJFrame {

    Image image;
    Insets insets;

    public IconLoadDraw(String filename) {
        super(filename);
        ImageIcon icon = new ImageIcon(filename);
        image = icon.getImage();
    }

    public void paint(Graphics g) {
        super.paint(g);
        g.drawImage(image, insets.left, insets.top, this);
    }

    public Dimension getScreenSize() {
        if (insets == null) {
           insets = getInsets();
        }
        return new Dimension(
           image.getWidth(this) + insets.left + insets.right,
           image.getHeight(this) + insets.top + insets.bottom);
        }

        public static void main(String args[]) {
           if (args.length > 0) {
              IconLoadDraw f = new IconLoadDraw(args[0]);
              f.show();
              f.setSize(f.getScreenSize());
           } else {
              System.err.println("Please specify image to display");
           }
        }
}
```

Realizar animaciones

Realizar animaciones es la tarea de agregar algo que parece tener movimiento a la pantalla. La animación puede tener dos enfoques, dependiendo del efecto que

pretenda conseguir. Uno de los enfoques es el de "imágenes alternas" (tener muchas imágenes y pasar rápidamente de una a la siguiente, En teoria, la primera imagen se muestra inmediatamente después de la última, en una secuencia repetitiva. El segundo modo de hacer animación es dibujar repetidamente la misma imagen o una ligeramente diferente, en una nueva posición. Este último enfoque es como la manecilla de los segundos en un reloj. Debe moverse constantemente, mientras borra su posición anterior. El enfoque de "imágenes alternas" es como mostrar una única imagen. Todo lo que debe hacer es agregar un bucle y guardar todas las imágenes en una matriz.

Para demostrar el segundo enfoque, (dibujar la misma imagen repetidamente), el programa del listado 10.13 muestra un gráfico circular dividido en seis porciones. Cada una de las seis porciones está rellena con un color diferente: rojo, naranja, amarillo, verde, azul y violeta. La animación se consigue modificando los colores uno a uno, haciendo que las porciones "roten". La figura 10.11 muestra el aspecto de la pantalla.

Figura 10.11. Ejemplo de anirnacion

Asé pues, ¿cómo funciona? El método `paint()` sólo es responsable de dibujar en la pantalla una vez. Aparte de `paint()`, se crea un `Thread` o `TimerTask` que es el responsable de cambiar la información del estado (en este caso, los colores) que dibuja `paint()`. Cuando los colores han cambiado, se le indica a la pantalla que se dibuje de nuevo mediante el método `repaint()`. Este los cambios de estado, la tarea deberia descansar durante poco tiempo para que el ojo pueda percibir los cambios y darle a otras tareas del sistema la oportunidad de ejecutarse.

Puede cambiar DELAY para ralentizar o acelerar el programa. También puede cambiar el tamaño de la matriz `colors` si quiere que haya más porciones en el gráfico. Sólo asegúrese de que el número de entradas sea divisible exactamente entre 360. Por ejemplo, 8, 9 y 12 porciones funcionarian perfectamente, pero 7 y 11 no.

Listado 10.13. Demostracion de animacion rotando colores

```
import java.awt.*;
import java.util.*;

public class Animate extends ExitableJFrame {
```

```
private static int DELAY = 100;
Insets insets;
Color colors[] = {
    Color.RED,
    Color.ORANGE,
    Color.YELLOW,
    Color.GREEN,
    Color.BLUE,
    Color.MAGENTA
};

public void paint(Graphics g) {
    super.paint(g);
    if (insets == null) {
       insets = getInsets();
    }
    // Calcula cada pasada por si hay que rnodificar el tamaño
    int x = insets.left;
    int y = insets.top;
    int width = getWidth() - insets.left - insets.right;
    int height = getHeight() - insets.top - insets.bottom;
    int start = 0;
    int steps = colors.length;
    int stepSize = 360 / steps;
    synchronized (colors) {
        for (int i=0; i<steps; i++) {
           g.setColor(colors[i]);
           g.fillArc(x, y, width, height, start, stepSize);
           start += stepSize;
        }
    }
}

public void go() {
    TimerTask task = new TimerTask() {
        public void run() {
            Color c = colors[0];
            synchronized (colors) {
               System.arraycopy(colors, 1, colors, 0, colors.length-1);
               colors[colors.length-11 = c;
            }
            repaint();
        }
    };
    Timer timer = new Timer();
    timer.schedule(task, 0, DELAY);
}

public static void main(String args[]) {
    Animate f = new Animate();
    f.setSize(200, 200);
    f.show();
    f.go();
}
```

Operaciones de dibujo usando el buffer

Si cree que va a tener que dibujar repetidamente las mismas formas en cada pasada por el método `paint()`, necesitará usar el doble buffer en las operaciones de dibujo. Además de evitar que el sistema tenga que dibujar repetidamente la misma escena, asi se asegura de que los usuarios no vean actualizaciones de pantalla parciales entre escenas. Usar el doble buffer significa dibujar en una zona de gráficos fuera de la pantalla en primer lugar, antes de dibujar los grdficos de la zona fuera de la pantalla en la zona principal de dibujo. Como la zona de fuera de la pantalla tiene el mismo tamaiio que la zona principal, no tiene que preocuparse por borrar algo que podia encontrarse en la pantalla principal, porque va a reemplazar toda la zona que aparece en pantalla.

Cuando hace doble buffer con un `JFrame` y no se encuentran componentes en el marco, lo primero que puede eliminar es la llamada a `super.paint()`. La implementación por defecto de `paint()` producird un parpadeo debido a que el contenedor del marco estd vacio. Sin embargo, si no está dibujando de nuevo toda la pantalla o si el marco contiene componentes, no es aconsejable eliminar `super.paint()` porque el fondo no se borrará y no se dibujaran los componentes. Si no estd seguro de lo que sucede por detrds de la animación, haga un comentario en la linea de `super.paint()` del listado 10.13 y expanda la pantalla y podrá ver como ahora no se limpia la pantalla. A medida que la pantalla dibuja el siguiente ciclo de color, observard el antiguo ciclo de color que queda atrás, para el anterior tamaiio de pantalla.

Para usar el doble buffer, en primer lugar debe crear el buffer y dibujarlo. A continuación, dibuje el buffer para la pantalla. Esto es más sencillo de lo que parece. Para demostrarlo, convertiremos el programa Animate en una versión con buffer.

1. Cree el buffer: Es importante crear de nuevo el buffer cada vez que la pantalla cambie de tamaiio, de modo que el método `paint()` sirva como un enorme lugar para inicializar el buffer. Puede crear tres tipos de buffer diferentes: `Image`, `VolatileImage` o `BufferedImage` (el más comun). Este es su aspecto:

```
import java.awt.image.*;
...
    Image buffer;
    Dimension oldSize;
...
    public void paint(Graphics g) {
        if ((oldSize == null) || (oldSize != getSize())) {
          oldSize = getSize();
          buffer = new BufferedImage(getWidth(), getHeight(),
              BufferedImage.TYPE_INT_RGB);
```

2. Obtiene el contexto grafico del buffer: Invoque al método `getGraphics()` de `Image` para obtener su objeto `Graphics`.

```
synchronized (colors) {
     Graphics bufferi; = buffer.getGraphics();
```

3. Dibuja en el contexto grdfico del buffer: Cambia todas las llamadas a `g.XXX` para dibujar en el buffer. Ademds, si no rellena el fondo en primer lugar, el fondo serd negro.

```
bufferG.setColor(Color.WHITE);
bufferG.fillRect(x, y, width, height);
for (int i=0; i<steps; itt) {
   bufferG.setColor(colors[i]);
   bufferG.fillArc(x, y, width, height, start, stepSize);
   start += stepSize;
}
}
```

4. Dibuja el buffer: Cuando ha terminado de dibujar en el buffer, lo dibuja igual que haría cualquier otro objeto **`Image`**.

```
g.drawImage(buffer, 0, 0, this);
```

Y eso es todo lo que necesita saber. A continuación (listado 10.14) se encuentra el programa Animate transformado. Ejecute las dos versiones, una junto a la otra, para ver las diferencias. Comience cada ejecución con el comando `javaw` en lugar de con `java` de modo que no tenga que abrir otras ventanas de consola.

Listado 10.14. Demostracion de animación con doble buffer

```
import java.awt.*;
import java.awt.image.*;
import java.util.*;

public class BufferedAnimate extends ExitableJFrame {

    private static int DELAY = 100;
    Image buffer;
    Dimension oldSize;
    Insets insets;
    Color colors[] = {
         Color.RED,
         Color.ORANGE,
         Color.YELLOW,
         Color.GREEN,
         Color.BLUE,
         Color.MAGENTA
    };

    public void paint(Graphics g) {
         if ((oldSize == null) || (oldSize != getSize())) {
            oldSize = getSize();
            buffer = new BufferedImage(getWidth(), getHeight(),
                BufferedImage.TYPE_INT_RGB);
            }
            if (insets == null) {
```

```
                insets = getInsets();
            }
            // Calcula en cada ocasión si debe modificar el tamaño
            int x = insets.left;
            int y = insets.top;
            int width = getWidth() - insets.left - insets.right;
            int height = getHeight() - insets.top - insets.bottom;
            int start = 0;
            int steps = colors.length;
            int stepSize = 360 / steps;
            synchronized (colors) {
               Graphics bufferG = buffer.getGraphics();
               bufferG.setColor(Color.WHITE);
               bufferG.fillRect(x, y, width, height);
               for (int i=0; i<steps; i++) {
                  bufferG.setColor(colors[i]);
                  bufferG.fillArc(x, y, width, height, start, stepSize);
                  start += stepSize;
               }
            }
            g.drawImage(buffer, 0, 0, this);
         }

         public void go() {
             TimerTask task = new TimerTask() {
                 public void run() {
                    Color c = colors[0];
                    synchronized (colors) {
                       System.arraycopy(colors, 1, colors, 0,
colors.length-1);
                       colors[colors.length-1] = c;
                    }
                    repaint();

         };
         Timer timer = new Timer();
         timer.schedule(task, 0, DELAY);
    }

    public static void main(String args[]) {
         BufferedAnimate f = new BufferedAnimate();
         f.setSize(200, 200);
         f.setTitle("Buffered");
         f.show();
         f.go();
    }
}
```

Modificar la escala de imágenes

Si necesita aumentar o disminuir la escala de una imagen, la clase `Image` proporciona un método `getScaledInstance()`:

```
public Image getScaledInstance(int width, int height, int hints)
```

Al usar una de las constantes de ayuda para escalas de la clase `Image` y enumeradas en la tabla 10.5, puede aumentar o reducir el tamaño de una imagen. Puede especificar explícitamente una nueva altura y una nueva anchura, si quiere; pero también puede especificar una de ellas y dar a la otra el valor de -1 para mantener su relación entre altura y anchura.

Tabla **10.5.** Trucos de escala de imagen

Variables/Métodos	Descripción
SCALE – DEFAULT	Usa el valor por defecto (normalmente en beneficio de la velocidad sobre la fluidez)
SCALE – SMOOTH **SCALE – AREA – AVERAGING**	Usa `AreaAveragingScaleFilter` para realizar el escalado; mezcla pixeles con los mas cercanos cuando realiza el escalado
SCALE – FAST **SCALE – REPLICATE**	Usa `ReplicateImageFilter` para el escalado; excluye las hileras cuando reduce la imagen y repite las hileras cuando aumenta la imagen

Para demostrar el escalado, el listado 10.15 toma una imagen y realiza cada una de las cuatro posibilidades: aumentar y reducir la imagen con escalado rápido y suave.

Listado 10.15. Demostracion de escalado de imagen

```
import java.awt.*;
import javax.swing.*;

public class Scale extends ExitableJFrame {

    Image image;
    Insets insets;

    public Scale(String filename) {
        super(filename);
        ImageIcon icon = new ImageIcon(filename);
        image = icon.getImage();


    public void paint(Graphics g) {
        super.paint(g);
        if (insets == null) {
           insets = getInsets();
        }
        g.drawImage(image, insets.left., insets.top, this);


    public void go() {
```

```
        // Descansa en primer lugar para ver el original
        rest();
        Image original = image;
        // Reduce de forma rápida
        image = original.getScaledInstance(200,
-1, Image.SCALE_FAST);
        repaint();
        rest();
        // Reduce de forma lenta
        image = original.getScaledInstance(200,
-1, Image.SCALE_SMOOTH);
        repaint();
        rest();
        // Aumenta de forma rápida
        image = original.getScaledInstance(400,
-1, Image.SCALE_FAST);
        repaint();
        rest();
        // Aumenta de forma lenta
        image = original.getScaledInstance(400,
-1, Image.SCALE_SMOOTH);
        repaint();
        rest();
        System.exit(0);
    }

    private void rest() {
        try {
           Thread.sleep(5000);
        } catch (InterruptedException ignored) {
        }
    }

    public static void main(String args[]) {
        if (args.length > 0) {
           Scale f = new Scale(args[0]);
           f.setSize(400, 400);
           f.show();
           f.go();
        } else {
           System.err.println("Please specify image to display");
        }
    }
}
```

Nota

Las diferencias entre escalonados se apreciaran mejor dependiendo de la imagen que se esta escalando. Por ejemplo, si la imagen es un triangulo, vera las diferencias entre escalados mas claramente. Puede ajustar el retardo de 5.000 milisegundos (5 segundos) en el programa si necesita observarlo durante mas tiempo.

Recortar imágenes

Cuando sólo quiere mostrar algunas partes de una imagen necesitara `CropImageFilter`. El constructor acepta el rectángulo que forma los límites de la zona que quiere recortar:

```
public CropImageFilter(int x, int y, int width, int height)
```

A continuación tendrá que crear la imagen pasando el filtro al constructor `FilteredImageSource`:

```
public FilteredImageSource(ImageProducer prcducer, ImageFilter filter)
```

Para obtener el `ImageProducer`, debe solicitarlo a la imagen original, usando `getsource()`. Asi es como el filtro sabe lo que debe filtrar. Y ¿qué quiere hacer con el nuevo `FilteredImageSource`? Bien, paselo al método `createImage()`, en lugar de la altura y la anchura. Si lo juntamos todo, obtenemos:

```
Image image = ...
image = createImage(new FilteredImageSource(
    image.getSource(),
    new CropImageFilter(x, y, w, h)));
```

Para mostrar el uso de `CropImageFilter`, el programa del listado 10.16 coge una zona al azar de la imagen pasada y sólo muestra esa parte. La figura 10.12 muestra los resultados.

Listado 10.16. Demostracion de recorte de imagen

```
import java.awt.*;
import java.awt.image.*;
import javax.swing.*;

public class Crop extends ExitableJFrame {

    Image image;
    Insets insets;

    public Crop(String filename) {
        super(filename);
        ImageIcon icon = new Imagelcnn(filename);
        image = icon.getImage();
        image = createImage(new FilteredImageSource(
            image.getSource(),
            new CropImageFilter(73, 63, 141, 131)));


        public void paint(Graphics g) {
            super.paint(g);
            if (insets == null) {
               insets = getInsets();
```

```
        g.drawImage(image, insets.left, insets.top, this);
    }

    public static void main(String args[]) {
        if (args.length > 0) {
            JFrame f = new Crop(args[0]);
            f.setSize(200, 200);
            f.show();
        } else {
            System.err.println("Please specify image to display");
```

Figura 10.12. Ejemplo de recorte de imagen

Nota

En estos ejemplos de escalado y recorte hay mucho mas de lo que realmente necesita saber en este momento. Si esta interesado en aprender mas sobre filtrar y recortar imágenes, busque en el paquete **`java.awt.image`** los otros procesos que participan de estos procesos. En las clases `MemoryImageSource` hay dos clases especificas interesantes para la creación de la estructura completa de la imagen en memoria y la clase `PixelGrabber`, para determinar el color de un pixel en una posición especifica de una imagen.

Reproducir audio

No esta exactamente relacionado con los gráficos, pero merece la pena mencionar aqui el proceso de reproducir archivos de audio en programas Java. Esto se realiza mediante la interfaz `java.applet.AudioClip`:

```
public interface AudioClip {
    public void loop()
    public void play()
    public void stop()
}
```

Para una aplicación, se obtiene un clip de audio con el método estdtico de `newAudioClip(URL)` de `Applet`. En ese momento puede reproducirlo una vez con `play()` o de forma continua con `loop()`. Es asi de sencillo. Use el método `stop()` cuando necesite detener la reproducción continua de un archivo de sonido o detener una reproducción de un archivo antes de que finalice.

Para un applet, la interfaz funciona de la misma forma, pero se usa uno de los dos métodos no estdticos de la clase `Applet` para obtener el clip de audio:

```
public AudioClip getAudioClip(URL completeURL)
public AudioClip getAudioClip(URL baseURL, String name)
```

Estos métodos funcionan igual que `getImage()` para el applet, en la que se puede pasar el documento o el código base como primer argumento:

```
AudioClip clip = getAudioClip(getDocumentBase(), "sound.wav");
```

También hay dos métodos `play()` en la clase `Applet` que proporcionan la posibilidad de reproducir una sola vez.

```
public void play(URL completeURL)
public void play(URL baseURL, String name)
```

¿Qué formatos de audio se pueden reproducir? En la version 1.4, se puede reproducir AIFF, AU, WAV, MIDI Tipo 0 y Tipo 1, además de Formato de música enriquecida (RMF). Lamentamos no poder reproducir archivos MP3.

El listado 10.17 es una sencilla demostración de reproducción un archivo de sonido. Encontrard archivos de audio de ejemplo con SDK o en Internet. Sound Ring en http://www.alexcia.com/ es un buen lugar para comenzar. Para encontrar efectos de sonido libres de royalties, acuda a http://www.partnersinrhyme.com/pir/PIRsfx.html.

Listado 10.17. Demostracion de reproduccion de archivos de sonido

```
import java.applet.*;
import java.io.*;
import java.net.*;

public class Play {
    public static void main(String args[]) {
        try {
           // Bucle
           URL url = new URL("http://java.sun.com/applets/other/Hangman/
                                                          audio/whoopy.au");
           AudioClip clip = Applet.newAudioClip(url);
           clip.loop();
           Thread.sleep(5000);
           //Reproduce
           File file = new File("bark.wav");
           clip = Applet.newAudioClip(file.toURL());
           clip.play();
           Thread.sleep(500);
```

```
            System.exit(0);
        } catch (InterruptedException e) {
        } catch (MalformedURLException e) {
```

Nota

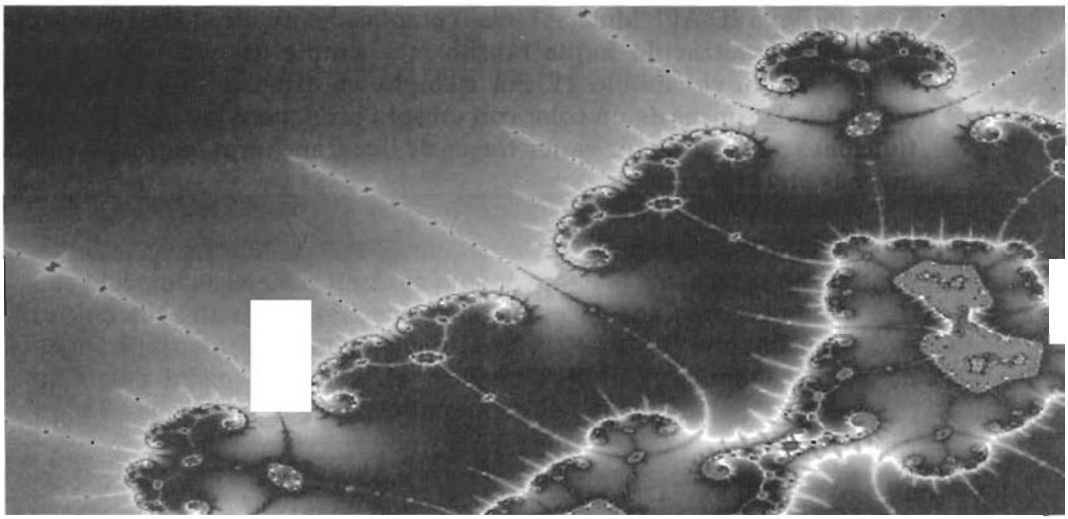
Las bibliotecas básicas de Java proporcionan la posibilidad de grabar (capturar) además de editar secuencias de audio. Puede encontrar estas funciones en los diferentes paquetes `javax.sound`. **Para mas información sobre el uso de estas bibliotecas, lea JavaSound API Programmer's Guide en ftp://** ftp.java.sun.com/docs/j2se1.4/programmer_guide.pdf.

Trucos para una animación más rápida

Cuando haya dominado las bases del dibujo y la animación, probablemente quiera trabajar en la optimización de sus operaciones de dibujo. Aqui tiene algunos apuntes para ayudarle a incrementar la tasa de sus animaciones:

- Haga doble buffer en sus operaciones de dibujo siempre que pueda.
- No vuelva a dibujarlo todo entre actualizaciones de pantalla. Por ejemplo, si esta creando una versión animada del juego de cartas Solitario, casi nunca será necesario volver a dibujar la pila que contiene cada palo. Al reducir el espacio de dibujo a lo que se puede arrastrar, la tasa de animación puede aumentar considerablemente.
- Mantenga la tasa de animación constante. Al usar `TimerTask` y `java.util.Timer`, evita que descanse durante un periodo de tiempo fijo al final de la tarea. En su lugar, se ejecuta en un momento fijado desde el comienzo de la tarea. Esto asegura una secuencia visualmente más suave.
- Calcule con antelación los valores complejos siempre que sea posible. Por ejemplo, realice los cálculos mientras espera a que el usuario introduzca datos en lugar de intentar dibujar de nuevo la pantalla. Además, las texturas y las operaciones de carnbio de color (como el que usamos en los programas `Animate` y `BufferedAnimate` de este capítulo) ocupan menos recursos que las de rotación o carnbio de tamaño.
- Para animaciones sencillas de archivos de imágenes múltiples, considere el utilizar el applet `Animator` que ofrece Sun; puede obtenerlo en http:// java.sun.com/ applets/jdk/1.1/demo/ Animator/. Éste nos ofrece la capacidad para mostrar una secuencia de imagenes, moviendo las imagenes y reproduciendo sonido junto con las imágenes.

- Examine Java 2D API. Muchas tareas complicadas explicadas en este capitulo se pueden realizar de forma mucho más simple usando técnicas que se describiriin en el capitulo 11. Por ejemplo, en este capitulo 10 se ha visto cómo dibujar líneas de un color con un sólo pixel, pero Java 2D proporciona funciones para conseguir varios trazos de linea, anchuras, patrones, puntas y muchas otras funciones.



11

Gráficos e imágenes 2D

La interfaz API 2D de Java que se present6 con la versión 1.2 de este lenguaje de programacibn, ha ampliado considerablemente las capacidades gráficas de Java. Mientras que las versiones anteriores nos permitian hacer cosas como dibujar lineas monocromas de un pixel de ancho, si queriamos algo más avanzado teniamos que hacerlo nosotros mismos. Ahora, con la API 2D, ya podemos utilizar gran parte de los gráficos avanzados y las capacidades de imágenes existentes para los mundos bidimensionales. La API 2D de Java es parte de la colección de API Java Media. Es una de las tres API multimedia que incorpora por defecto SDK; Image I/O y Sound son las otras dos. Además de estas tres, las API Java Media incluyen Java 3D, Java Advanced Imaging, Java Media Framework, Java Speech y Java Shared Data Toolkit. Éstas aún se encuentran en desarrollo o disponibles por separado como una extensión opcional. Hay cinco áreas donde la interfaz API 2D de Java mejora las capacidades gráficas de este lenguaje de programacibn.

- Procesado de imágenes, donde ahora podemos controlar elementos como el patrón de relleno o el estilo de linea.
- Proceso de texto, donde tenemos control avanzado de diseño de texto.
- Control de imigenes, donde se han mejorado las capacidades de filtrado y formato.
- Búsqueda de dispositivos gráficos, para descubrir soporte para fuentes nativas.
- Procesado a pantalla completa con aceleración gráfica por hardware.

El soporte para todas estas capacidades se puede encontrar en los siguientes paquetes: `java.awt`, `java.awt.color`, `java.awt.font`, `java.awt.geom`, `java.awt.image`, `java.awt.image.renderable` y `javax.imageio`.

Proceso grafico

La interfaz API 2D de Java puede efectuar las operaciones básicas de dibujo. Mientras que antes podiamos dibujar arcos, ovalos, rectdngulos y poligonos, con la clase `Graphics` lo único que se podia hacer con los colores era colorear y contornear elementos. No podiamos dibujar nada transparente. Ahora, con la nueva clase `Graphics2D`, se ha mejorado considerablemente el procesado de grdficos. Lo vemos en las siguientes secciones.

Un salto a Graphics2D

Para dibujar en Java hemos de seguir un proceso de tres pasos:

- Especificamos los atributos del dibujo con métodos como `setColor()`, `setFont()`, `setPaintMode()`, `setXORMode()`.
- Identificamos la forma del dibujo.
- Lo dibujamos.

En el mundo de `Graphics`, sin la API 2D de Java, la identificacion y dibujo de las distintas partes eran distintos procesos. En el proceso de dibujo tenemos la identificacion de los contornos rectangulares que se van a dibujar y las llamadas a los respectivos métodos (posiblemente con algún atributo relacionado con el dibujo como los dngulos de los arcos).

El dibujo `Graphics2D` también utiliza un proceso de tres pasos. Pero la identificación y el dibujo del contorno son dos partes independientes. En el caso generalizado, podemos especificar una forma que dibujaremos creando un objeto que implemente la interfaz `Shape`. Si queremos vivir en un mundo donde todo se especifica como `Polygon`, tendremos que utilizar `General Path` para identificar las coordenadas. `Shape` dispone de varias clases para dibujar las figuras mds comunes. Por ejemplo, la clase `Arc2D` se utilizará para dibujar un arco. Otras clases que implementa shape son `Line2D`, `Rectangle2D`, `RoundRectangle2D`, `Ellipse2D`, `QuadCurve2D` y `CubicCurve2D`. Todas ellas se encuentran en el paquete `java.awt.geom`.

La definición actual de la interfaz `Shape` es la siguiente:

```
public interface Shape {
    public boolean contains(double x, double y)
    public boolean contains(double x, double y, double w, double h)
    public boolean contains(Point2D p)
```

```
    public boolean contains(Rectangle2D r)
    public Rectangle getBounds()
    public Rectangle2D getBounds2D()
    public PathIterator getPathIterator(AffineTransform at)
    public PathIterator getPathIterator(AffineTransform at,
       double flatness)
    public boolean intersects(double x, double y, double w,
       double h)
    public boolean intersects(Rectangle2D r)
}
```

Una vez que hemos identificado la forma que vamos a dibujar, pasaremos a la acción. Para dibujarla utilizaremos uno de los tres métodos de `Graphics2D`: `clip()`, `draw()` o `fill()`; todas ellas toman el objeto `Shape` como pardmetro.

- El método `clip()` se utilizard para ajustar la zona de dibujo a la intersección de la forma dibujada con el área determinada. Este ajuste afectard a las futuras operaciones.

  El método `draw()` utiliza `Strike` para dibujar el contorno de `Shape`.

- Y el método `fill()` utiliza `Paint` para colorear el objeto `Shape`.

Gracias a estas operaciones podremos utilizar Af`fineTransform` para convertir las coordenadas 2D en otras nuevas. De esta forma podremos desarrollar operaciones como dibujar un rectángulo que gira. También podemos especificar una interfaz `Composite` con la que determinemos cómo se combinan los pixeles existentes con el color que se utiliza en el dibujo.

Nota

Además de dibujar formas geometricas, podemos tratar el texto que dibujemos como un objeto `Shape`. Para ello se utilizara `TextLayout`.

Antes de crear un programa que demuestre estas capacidades, vamos a ver la definición de la clase `Graphics2D` en la tabla 11.1.

Tabla 11.1. Composición de la clase Graphics2D

Variables/Métodos	Descripción
Graphics2D()	Crea un objeto `Graphics2D`.
`addRenderingHints()`	Agrega un mapa de preferencias para el algoritmo de dibujo.
`clip()`	Reduce la zona de dibujo actual.
`draw()`	Dibuja la forma

Variables/Métodos	Descripción
draw3DRect()	Dibuja un rectangulo 3D destacado.
drawGlyphVector()	Dibuja un conjunto de caracteres.
drawImage() drawRenderableImage() drawRenderedImage()	Dibuja una imagen.
drawstring()	Dibuja una cadena.
fill()	Dibuja una forma con relleno
fill3DRect()	Dibuja un rectangulo 3D destacado.
getBackground()	Obtiene el color de fondo actual.
getComposite()	Obtiene la composición de la configuracion actual para la mezcla.
getDeviceConfiguration()	Obtiene La configuracion actual del dispositivo grafico.
getFontRenderContext()	Obtiene el contexto actual para dibujar fuentes.
getPaint()	Obtiene el patron de relleno actual.
getRenderingHint()	Busca preferencias para el algoritmo de dibujo
getRenderingHints()	Obtiene todas las preferencias para el algoritmo de dibujo.
getStroke()	Obtiene el trazo actual
getTransform()	Obtiene la transformacidn de dibujo actual.
hit()	Consulta si se ha producido una intersección.
rotate()	Agrega rotación a la transformacion de dibujo actual.
scale()	Agrega escalado a la transformacion de dibujo actual.
setBackground()	Cambia el color de fondo actual
setComposite()	Cambia los valores de la configuracion actual para la mezcla.
setPaint()	Cambia el patron de relleno actual.
setRenderingHint() setRenderingHints()	Especifica las preferencias del algoritmo de dibujo.
setStroke()	Cambia el trazo actual.
setTransform()	Cambia la transformacion de dibujo actual.
shear()	Agrega escalado a la transformacion de dibujo actual.

Variables/Métodos	Descripción
transform()	Agrega transformacion a la transformacion de dibujo actual.
translate()	Agrega movimiento de translacion a la transformacion de dibujo actual.

Algunos de los métodos de Graphics2D pueden parecer similares a los de la clase Graphics (no es de extrañar, pues Graphics2D es subclase de Graphics) pero la mayoría de las operaciones son nuevas y diferentes.

Iniciación a Graphics2D

Ahora crearemos un programa que nos muestre un caso muy sencillo donde se utilizará Graphics2D para dibujar un rectángulo a partir de GeneralPath y un arco con Arc2D (vea la figura 11.1). Segun aprendamos más capacidades de Graphics2D, mejoraremos el aspecto del contenido del rectángulo y el arco. El programa que se encuentra detrás de la figura 11.1 es el listado 11.1

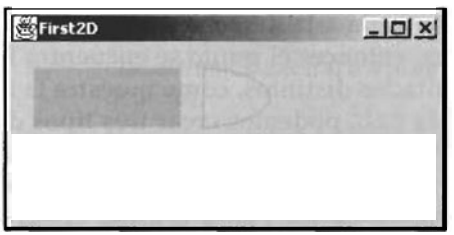


Figura 11.1. Rectangulo y arco creados con Graphics2D

Listado 11.1. Demostracion de dibujo de un rectangulo y un arco con Graphics2D

```
import java.awt.*;
import java.awt.geom.*;
public class First2D extends ExitableJFrame {

    public void paint(Graphics g) {
        Graphics2D g2d = (Graphics2D)g;
        // Configura atributos
        g2d.setColor(Color.CYAN);
        // Define cada forma
        GeneralPath rectangle = new GeneralPath(GeneralPath.WIND_EVEN_ODD);
        rectangle.moveTo(20f, 40f);
        rectangle.lineTo(120f, 40f);
        rectangle.lineTo(120f, 80f);
        rectangle.lineTo(20f, 80f);
        rectangle.closePath();
        Arc2D arc = new Arc2D.Double(90, 40, 90, 40, 270, 180, Arc2D.CHORD);
        // Dibuja la forma
```

```
        g2d.fill(rectangle);
        g2d.draw(arc);
    }

    public static void main(String args[]) {
        Frame f = new First2D();
        f.setTitle("First2D");
        f.setSize(300, 150);
        f.show();
    }
}
```

Dos cosas sobre las que se estará preguntando en este primer ejemplo son el pardmetro constructor `GeneralPath` y el ultimo parámetro para el constructor `Arc2D.Double`.

El parametro `GeneralPath` define la regla empleada para determinar cuándo un punto se encuentra dentro de la forma. Las dos opciones son `WIND_EVEN_ODD` y `WIND_NON_ZERO`. La regla par/impar dice que un punto se encuentra en una figura si hay un número impar de intersecciones entre el punto y los limites externos de la figura. Si es un número par, el punto se encuentra fuera. La regla *nonzero* dibuja una linea desde un punto que atraviesa la trayectoria de la forma, contando el numero de veces que la linea atraviesa la trayectoria. Si la linea atraviesa el mismo numero de veces desde la izquierda a la derecha que desde la derecha a la izquierda a lo largo de la trayectoria, entonces el punto se encuentra fuera de la figura. Estas dos reglas muestran resultados distintos, como muestra la figura 11.2.

En lo referente a `Arc2D`, podemos crear tres tipos de archivos: `CHORD`, `OPEN` y `PIE`. Como se puede ver, `CHORD` dibuja un ancho y conecta los puntos finales entre sí por medio de una linea recta, `OPEN` deja los puntos finales sin conectar, mientas que `PIE` los une a través de un punto central, como si se tratase de un gráfico circular.

Figura 11.2. Cornparacion de los resultados de las reglas par/impar y nonzero

Parámetros configurables

Si asociamos `Stroke` con un objeto de `Graphics2D` a través del método `setStroke()`, podremos modificar el modelo que se haya utilizado para dibujar el

borde exterior. La interfaz Stroke es muy sencilla, como nos demuestra su definición.

```
public interface Stroke {
    public Shape createStrokedShape(Shape s)
```

Y, la clase BasicStroke se encarga de implementar la interfaz por nosotros, facilitando los cambios de Stroke. A través de distintos constructores, podremos cambiar la linea de un pixel de ancho por un cuadrado que se puede dibujar en cualquier color. Como muestra la tabla 11.2, BasicStroke tiene muchos atributos. Mediante los diferentes constructores, puede cambiar la linea (de un pixel de ancho por defecto) conectada por topes cuadrados para que sea cualquier linea de un sólo color con uno de entre las diferentes tipos de tope.

Tabla **11.2.** Composición de la clase BasicStroke

Variables/Métodos	Descripción
CAP_BUTT	Los extremos del trazo no tienen adornos.
CAP_ROUND	Los extremos del trazo tienen adornos redondeados.
CAP_SQUARE	Los extrernos del trazo tienen adornos cuadrados (por defecto).
JOIN_BEVEL	Los segmentos del trazo estan conectados mediante segmentos rectos.
JOIN_MITER	Los segmentos del trazo estan conectados rnediante un borde externo, hasta el límite del angulo (por defecto).
JOIN_ROUND	Los segmentos del trazo estan conectados redondeando la esquina.
BasicStroke()	Crea un objeto BasicStroke.
createShapedStroke()	Crea una Shape para que se use como trazo.
equals()	Comprueba la igualdad.
getDashArray()	Obtiene una matriz deguiones para el trazo con guiones.
getDashPhase()	Obtiene una fase de guiones para el trazo con guiones.
getEndCap()	Obtiene la configuracion para el extremo.
getLineJoin()	Obtiene la configuracion para unir líneas.
getLineWidth()	Obtiene la configuracion para el trazo con líneas
getMiterLimit()	Obtiene el limite de angulo del trazo.
hashCode()	Genera el codigo hash del trazo.

Observemos algunas de las variables. Las tres constantes CAP_ indican cómo dibuja Stroke el final de una linea. Sus resultados aparecen en la figura 11.3. Las tres constantes JOIN_ indican cómo Stroke conecta el extremo de dos líneas. Los resultados aparecen en la figura 11.4.

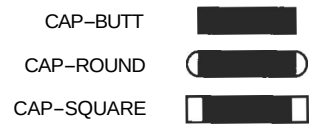


Figura 11.3. Configuracion de los extremos BasicStroke

Figura 11.4. Configuracion de la conexión entre segmentos BasicStroke

Para ver cómo funciona BasicStroke vamos a añadir una linea azul alrededor del rectángulo de la figura 11.1 y colocaremos una linea discontinua alrededor del arco. Para dibujar la linea exterior del rectángulo, la colorearemos y luego la dibujaremos, ajustando stroke entre el relleno y el dibujo. También hemos modificado el programa para que utilice la clase Rentagle2D. Para colocar la linea discontinua alrededor del arco, tendremos que especificar cómo queremos que termine para que dé la sensacion de linea cerrada, además de la separación de los trazos y los tamaños de estos. Tenemos el resultado en la figura 11.5.

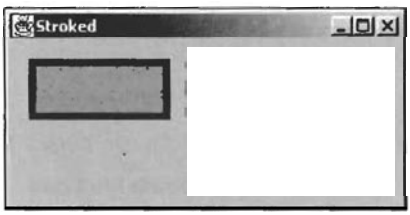


Figura 11.5. Modificación del patron Stroke

Listado 11.2. Uso de trazos para contornear formar

```
import java.awt.*;
import java.awt.geom.*;
public class Stroked extends ExitableJFrame {

    public void paint(Graphics g) {
        Graphics2D g2d = (Graphics2D)g;
        // Rectángulo
```

```
        Rectangle2D rectangle = new Rectangle2D.Double(20, 40, 100, 40);
        g2d.setColor(Color.CYAN);
        g2d.fill(rectangle);
        g2d.setColor(Color.BLUE);
        g2d.setStroke(new BasicStroke(5f));
        g2d.draw(rectangle);
        // Arco
        g2d.setColor(Color.GREEN);
        g2d.setStroke(new BasicStroke(2f, BasicStroke.CAP_ROUND,
            BasicStroke.JOIN_ROUND, 3[illegible] new [illegible] {10f}, 0f));
        Arc2D arc = new Arc2D.Double(90, 40, 90, 40, 270, 180, Arc2D.CHORD);
        g2d.draw(arc);
    }

    public static void main(String args[]) {
        Frame f = new Stroked();
        f.setTitle("Stroked");
        f.setSize(300, 150);
        f.show();
    }
```

Como aun no hemos visto cómo funciona `clip()`, vamos a colocar el arco sobre el rectdngulo, de forma que únicamente se dibuje la intersección de ambos cuerpos. Para ello, recortaremos el arco que define la zona de dibujo que se utilizard cuando dibujemos el rectdngulo. Obsérvese que no se ha utilizado ni el color ni el trazado del arco cuando se coloca sobre el rectdngulo. Estos atributos no forman parte de la definición del contorno. Como no se dibuja el arco (con `draw()` o `fill()`), se ignoran estos atributos. En la figura 11.6 tenemos los resultados obtenidos y en el listado 11.3 estd el programa que lo genera.

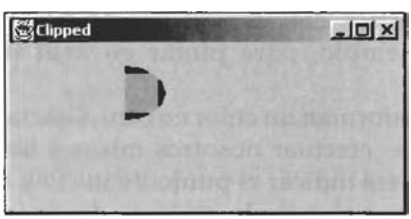


Figura 11.6. Recorte de lo que se ha dibujado

Listado 11.3. Recorte de la region dibujada

```
import java.awt.*;
import java.awt.geom.*;
public class Clipped extends ExitableJFrame {

    public void paint(Graphics g) {
        Graphics2D g2d = (Graphics2D)g;
        // Arco
        g2d.setColor(Color.GREEN);
        g2d.setStroke(new BasicStroke(2f, BasicStroke.CAP_ROUND,
```

```
            BasicStroke.JOIN_ROUND, 3f, new float[] {10f}, 0f));
        Arc2D arc = new Arc2D.Double(60, 40, 60, 40, 270, 180,
Arc2D.CHORD);
        g2d.clip(arc);
        // Rectángulo
        Rectangle2D rectangle = new Rectangle2D.Double(20, 40, 100, 40);
        g2d.setColor(Color.CYAN);
        g2d.fill(rectangle);
        g2d.setColor(Color.BLUE);
        g2d.setStroke(new BasicStroke(10f));
        g2d.draw(rectangle);


    public static void main(String args[]) {
        Frame f = new Clipped();
        f.setTitle("Clipped");
        f.setSize(300, 150);
        f.show();
    }
```

Gradientes de pintura y rellenos con textura

Ahora que ya sabemos cómo cambiar el modelo de una linea, vamos a ver cómo se puede cambiar el modelo que se utiliza para colorear una figura. La interfaz API 2D de Java puede trabajar con gradientes ciclicos y no ciclicos, además de poder utilizar colores sólidos para colorear las imágenes. Java define la interfaz `Paint` para que se haga cargo del relleno de estas superficies; y `Color`, `GradientPaint` y `TexturePaint` para que se encarguen de su implementación.

Para utilizar un color sólido, usaremos el método `setColor()` de `Graphics`. Para mantener cierta consistencia con los otros dos mecanismos (que veremos en breve) cuando trabajemos con `Graphics2D`, deberiamos utilizar el método `setPaint()`. Por ejemplo, para pintar en azul deberiamos utilizar `Paint (Color.blue)`.

Los gradientes transforman un color en otro. Gracias a la clase `GradientPaint` se terminó el tener que efectuar nosotros mismos las conversiones. Todo lo que tendremos que hacer será indicar el punto de inicio y final, asi como los colores. A continuación llamaremos al método `fill()` de `Graphics2D`. Gradualmente ira cambiando el color. Si el área que tiene que rellenar se sale del rectángulo que forman los puntos inicio y final, podremos utilizar un parámetro booleano opcional. Si es `true`, el gradiente sera circular. Si es `false`, los colores sólidos se extenderán en dos direcciones (este es el comportamiento predeterminado). La figura 11.7 muestra los tres resultados posibles: sin parámetro opcional, con el valor `true` y con `false`. En cualquiera de los tres casos podemos especificar los puntos finales utilizando dos números con coma flotante o dos objetos `Point2D`. Hemos dibujado una linea negra para que podamos ver los puntos de las distintas zonas.

El programa del listado 11.4 es el encargado de generar los tres posibles gradientes de la figura 11.7. Observe que el relleno de la primera y tercera opción es en realidad el mismo.

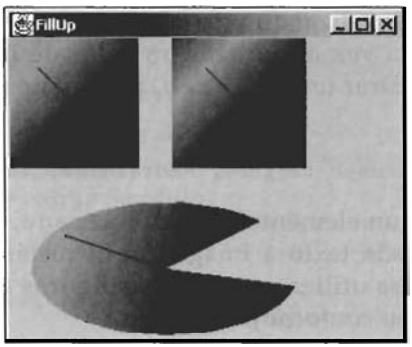


Figura 11.7. Tres opciones para el gradiente

Listado 11.4. Demostracion de dibujo con gradientes

```
import java.awt.*;
import java.awt.geom.*;
public class FillUp extends ExitableJFrame {

    public void paint(Graphics g) {
        Graphics2D g2d = (Graphics2D)g;
        GradientPaint gp = new GradientPaint
            (25, 45, Color.CYAN, 75, 95, Color.BLUE);
        g2d.setPaint(gp);
        Rectangle2D rectangle = new Rectangle2D.Double(0, 20, 100, 100);
        g2d.fill(rectangle);
        gp = new GradientPaint(150, 45, Color.CYAN, 200, 95, Color.BLUE,
true);
        g2d.setPaint(gp);
        rectangle = new Rectangle2D.Double(125, 20, 100, 100);
        g2d.fill(rectangle);
        Point2D p2d1 = new Point2D.Float(45, 170);
        Point2D p2d2 = new Point2D.Float(195, 220);
        gp = new GradientPaint(p2d1, Color.CYAN, p2d2, Color.BLUE, false);
        g2d.setPaint(gp);
        Arc2D arc = new Arc2D.Double(20, 145, 200, 100, 45, 300,
Arc2D.PIE);
        g2d.fill(arc);
        g.setColor(Color.BLACK);
        g.drawLine(25, 45, 75, 95);
        g.drawLine(150, 45, 200, 95);
        g.drawLine(45, 170, 195, 220);
    }

    public static void main(String args[]) {
        Frame f = new FillUp();
        f.setTitle("FillUp");
        f.setSize(300, 250);
        f.show();
```

Colorear una figura utilizando una textura es muy parecido a la aplicacion de los gradientes. Pero en vez de utilizar un color de inicio y otro de finalización, tendremos que suministrar un `BufferedImage` y una zona para usar como textura.

```
TexturePaint(BufferedImage texture, Rectangle2D rect)
```

Como la textura es un elemento `BufferedImage`, podremos dibujar cualquier elemento con ella, desde texto a imágenes. El rectángulo especifica la parte de `BufferedImage` que se utiliza para rellenar figuras grandes y recortar la textura (para que se adapte a su contorno).

La figura 11.8 muestra el uso de una imagen como textura para rellenar una forma. Como la textura no cambia de una llamada `paint()` a otra, lo mejor que podemos hacer es crearla fuera de `paint()` para que sólo se tenga que acceder al archivo que contiene la imagen una única vez. El programa del listado 11.5 generó la figura 11.8. El archivo de imagen usado para la textura se encuentra en el CD-ROM o puede usar su propio archivo de imagen.

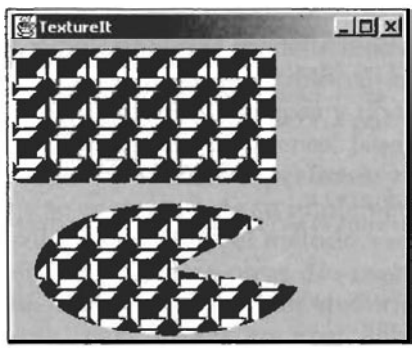


Figura 11.8. Rellenar una figura con una textura

Listado 11.5. Demostracion de dibujo con texturas

```
import java.awt.*;
import java.awt.geom.*;
import java.awt.image.*;
import javax.swing.*;

public class TextureIt extends ExitableJFrame {
    TexturePaint tp = getImageTexture("sybex.gif");

    public TexturePaint getImageTexture(String imageFile) {
        ImageIcon icon = new ImageIcon(imageFile);
        Image img = icon.getImage();
        int width = img.getWidth(this);
        int height = img.getHeight(this);
        BufferedImage buffImg =
            new BufferedImage(width, height, BufferedImage.TYPE_INT_ARGB);
```

```
        Graphics g = buffImg.getGraphics();
        g.drawImage(img, 0, 0, this);
        Rectangle2D rect = new Rectangle(0, 0, width, height);
        return new TexturePaint(buffImg, rect);
    }

    public void paint(Graphics g) {
        Graphics2D g2d = (Graphics2D)g;
        g2d.setPaint(tpj;
        Rectangle2D rectangle = new Rectangle2D.Double(0, 30, 200, 100);
        g2d.fill(rectangle);
        Arc2D arc = new Arc2D.Double(20, 145, 200, 100, 45, 300,
Arc2D.PIE);
        g2d.fill(arc);


    public static void main(String args[]) {
        Frame f = new TextureIt();
        f.setTitle("TextureIt");
        f.setSize(300, 250);
        f.show();
    }
}
```

Advertencia

Es importante observar que la textura comienza en la parte superior de la zona de dibujo, no es la parte superior de la zona que se va a rellenar. Asi, a menos que calcule una posición de inicio basandose en la altura y anchura de la imagen, es probable que la imagen no comience en la esquina superior izquierda de la textura.

Suavizar contornos

Cuando se dibuja una figura o texto sobre una cuadrícula generada a partir de pixeles, nos podemos encontrar con que sus bordes tienen saltos, generando un efecto similar a unas escaleras. Si intenta aumentar el tamaiio de una imagen dibujada de forma muy fina, observará ese efecto de escalera, en cualquier línea horizontal. Incluso sin el cambio de tamaiio, puede observar esos bordes dentados.

Cuando dibuja algo con esta técnica, se consurnen recursos. Además, el dispositivo encargado de rnostrarnos los resultados puede no ser compatible con antialiasing.

Debido a estos dos aspectos, la activación del antialiasing se hace mediante una pista de renderizado, que puede ser ignorada por el dispositivo. Al aplicar este efecto tendremos que utilizar la clase `RenderHints`, descrita en la tabla **11.3,** para especificar estas pistas de renderizado.

Tabla **11.3.** Composición de la clase RenderingHints

Variables/Métodos	Descripción
KEY – ALPHA – INTERPOLATION	Clave para la pista interpolacion alfa.
KEY – ANTIALIASING	Clave para la pista antialiasing.
KEY_COLOR_RENDERING	Clave para la pista de dibujo a color.
KEY – DITHERING	Clave para la pista de difurninado
KEY_FRACTIONALMETRICS	Clave para la pista de medidas fraccionadas.
KEY – INTERPOLATION	Clave para la pista de interpolacion.
KEY – RENDERING	Clave para dibujar.
KEY – STROKE – CONTROL	Clave para la pista de normalización de trazo.
KEY – TEXT – ANTIALIASING	Clave para la pista de antialiasing de texto.
VALUE – ALPHA_INTERPOLATION – DEFAULT	Valores posibles para la interpolacion alfa. La pista de calcular el valor alfa cuando los objetos se dibujan unos encima de otros con transparencia.
VALUE – ALPHA_INTERPOLATION – QUALITY	
VALUE – ALPHA – INTERPOLATION – SPEED	
VALUE – ANTIALIAS – DEFAULT	Valores posibles para antialiasing. La pista para especificar forrna o texto esta activada.
VALUE – ANTIALIAS – OFF	
VALUE – ANTIALIAS – ON	
VALUE – COLOR – RENDER – DEFAULT	Valores posibles para dibujo a color. La pista para calcular losvalores de color para perfiles de dispositivos especificos.
VALUE – COLOR – RENDER – QUALITY	
VALUE – COLOR – RENDER – SPEED	
VALUE – DITHER – DEFAULT	Valores posibles para la interpolación. Pista para el procedimie'nto para controlar que el dispositivo no sea compatible con el color solicitado.
VALUE – DITHER_DISABLE	
VALUE – DITHER – ENABLE	

Variables/Métodos	Descripción
VALUE_FRACTIONALMETRICS_DEFAULT	Valores posibles para rnedidas de fuente fraccionadas. Se deben usar pistas para rnedidas enteras o de coma flotante.
VALUE – FRACTIONALMETRICS – OFF	
VALUE – FRACTIONALMETRICS – ON	
VALUE_INTERPOLATION – BICUBIC	Valores posibles para transforrnaciones de irnagen. La opción especifica que algoritrno debe usarse. BICUBIC proporciona la mejor calidad, NEAREST NEIGHBOR la peor.
VALUE_INTERPOLATION_BILINEAR	
VALUE – INTERPOLATION – NEAREST – NEIGHBOR	
VALUE – RENDER – DEFAULT	Valores posibles para dibujar de forrna basics. Pista para especificarsi el dibujo debe sacrificar velocidad en beneficio de la calidad.
VALUE – RENDER – QUALITY	
VALUE – RENDER – SPEED	
VALUE – STROKE – DEFAULT	Valores posibles para la norrnalizacion del trazo. Pista para especificar si las lineas deben ajustarse para estar centradas sobre los pixeles. PURE indica que no.
VALUE – STROKE – NORMALIZE	
VALUE_STROKE – PURE	
VALUE – TEXT – ANTIALIAS – DEFAULT	Valores posibles para antialiasing. Pista para especificar si se activa el antialiasing.
VALUE – TEXT – ANTIALIAS – OFF	
VALUE – TEXT – ANTIALIAS – ON	
`RenderingHints()`	Crea un mapa `RenderingHints` vacio.
`add() putAll()`	Agrega otro rnapa al rnapa de pistas.
`clear()`	Borra el rnapa de pistas.
`clone()`	Crea una copia del mapa de pistas.
`containsKey()`	Cornprueba si existe una clave de pista en el rnapa.

Variables/Métodos	Descripción
containsValue()	Cornprueba si hay un valor de pista en el rnapa.
entrySet()	Devuelve el conjunto de entradas de pistas de dibujo del rnapa.
equals()	Cornprueba la igualdad
get()	Atrapa el valor de la clave de pista.
hashCode()	Genera codigo hash para el rnapa.
isEmpty()	Cornprueba si el rnapa esta vacio.
keySet()	Devuelve el conjunto de claves de rnapa
put()	Agrega una pista al rnapa.
remove()	Elirnina una pista del mapa.
size()	Devuelve el nurnero de pistas del rnapa
toString()	Convierte el rnapa en una cadena.
values()	Devuelve la coleccion de valores de pista del rnapa.

Se pueden ignorar todos los métodos de la clase `RenderHints`. Bastara con que le digamos a `Graphics2D` que el método `setRenderHints()` tendrá que utilizar el primer parámetro como clave y el segundo como configuración. En el caso de que queramos suavizar contornos, podremos utilizar `KEY_ANTIALIASING` como clave y `VALUE_ANTIALIAS_OFF` o `VALUE_ANTIALIAS_N` (o `VALUE_ANTIALIAS_DEFAULT`, para los casos en que queramos especificar un efecto predeterminado). Todas las pistas siguen este convenio de nombres, por lo que es muy normal ver las claves y los valores juntos. Si lo prefiere, `Graphics2D` ofrece los métodos `setRenderingHints()` y `addRenderingHints()`. Con ellos, puede reemplazar o ajustar varias pistas a la vez y podría seguir usando todos los otros métodos.

Por cierto, todos estos métodos proceden de la interfaz `Map`. La figura 11.9 muestra el uso del establecimiento de pistas de dibujo, en concreto antialiasing. El listado 11.6 es el programa que generó los dos ovalos y el texto de la figura 11.9.

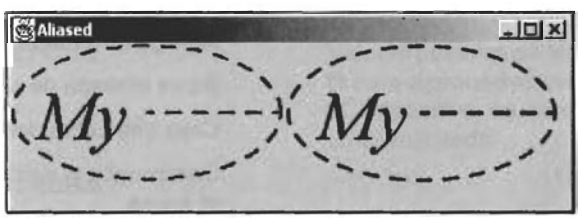


Figura 119. El antialiasing esta deshabilitado en la figura de la izquierda y activado en la otra

Observe que la forma y el texto dibujados han sido suavizados con antialiasing. Si prefiere que sólo se suavice el texto, puede cambiar la línea s e `tRenderingHint()` a lo siguiente:

```
g2d.setRenderingHint(RenderingHints.KEY_TEXT_ANTIALIASING,
    RenderingHints.VALUE_TEXT_ANTIALIAS_ON);
```

Listado 11.6. Demostracion de dibujo con y sin antialiasing

```
import java.awt.*;
import java.awt.genm.*;
public class Aliased extends ExitableJFrame {

    public void paint(Graphics g) {
       Graphics2D q2d = (Graphics2D)g;
       g2d.setColor(Color.BLACK);
       g2d.setStroke(new BasicStroke(2f, BasicStroke.CAP ROUIJD,
          BasicStroke.JOIN_ROUND, 3f, new float[] {10f}, 0f));
       Arc2D arc = new Arc2D.Double(5, 25, 200, 100, 0, 360, Arc2D.PIE);
       g2d.draw(arc);
       g2d.setFont(new Font("Serif", Font.ITALIC, 50));
       g2d.drawString("My", 25, 95);
       g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
          RenderingHints.VALUE_ANTIALIAS_ON);
       arc = new Arc2D.Double(210, 25, 200, 100, 0, 360, Arc2D.PIE);
       g2d.draw(arc);
       g2d.drawString("My", 235, 95);
    }

    public static void main(String args[]) {
       Frame f = new Aliased();
       f.setTitle("Aliased");
       f.setSize(425, 150);
       f.show();
```

Dibujar curvas

Hasta ahora hemos visto cómo se utiliza la clase `Arc2D` para dibujar arcos. Pero tenemos otras formas de dibujar curvas. Para ello utilizamos las clases `Ellipse2D`, `QuadCurve2D` y `CubicCurve2D` y los métodos `quadTo()` y `curveTo()` de la clase `GeneralPath`. Técnicamente hablando, también podemos utilizar el método `lineTo()` de `GeneralPath`, pero tendriamos que calcular todos los puntos a mano y la curva resultante podría ser algo extraña.

Clases para curvas

Son cuatro las clases especializadas en la representación de curvas. Éstas son `Arc2D`, `Ellipse2D`, `QuadCurve2D` y `CubicCurve2D`. `Arc2D` y `Ellipse2D` funcionan de forma parecida (y tienen las mismas limitaciones) que los objetos

Graphics. Podemos limitarnos a dibujar un rectángulo, pero en vez de utilizar los métodos drawArc(), fillArc(), drawOval() y fillOval() crearemos una forma Arc2D o Ellipse2D y le diremos al objeto Graphics2D que utilice el método draw() o fill() con él. Para crear un ejemplo utilizaremos dos subclases de cada uno de ellos. Para crear un objeto Arc2D utilizamos Arc2D.Float o Arc2D.Double. Ellipse2D tiene dos subclases parecidas, que son Ellipse2D.Float y Ellipse2D.Double.

Las subclases Float guardan los valores que se han definido (y que tienen una precisión de un número con coma flotante), mientras que las subclases Double utilizan una precision double.

Las clases Arc2D o Ellipse2D (asi como Rectangle2D y RoundRectangle2D) son subclases de la clase abstracta RectangularShape, definida en la tabla 11.4

Tabla **11.4.** Composición de la clase RectangularShape

Variables/Métodos	Descripción
RectangularShape()	Crea un objeto RectangularShape vacio.
clone()	Crea una copia de la forma.
contains()	Comprueba si el puntolrectangulo se encuentra por completo en la forma.
getBounds() getFrame()	Obtiene el rectangulo que delimita la forma.
getCenterX()	Obtiene el punto central horizontal de la forma.
getCenterY()	Obtiene el punto central vertical de la forma
getHeight()	Obtiene la altura de la forma.
getMaxX()	Obtiene la posicion x maxima de la forma.
getMaxY()	Obtiene la posicion y maxima de la forma.
getMinX() y getX()	Obtiene la posicion x minima de la forma.
getMinY() y getY()	Obtiene la posicion x minima de la forma.
getPathIterator()	Devuelve el iterador que dibuja el perfil de la forma.
getWidth()	Obtiene la anchura de la forma.
intersects()	Comprueba si el rectangulo se encuentra parcialmente en la forma.
isEmpty()	Comprueba si la forma esta vacia.
setFrame()	Cambia el rectángulo delimitador para la forma.
setFrameFromCenter()	
setFrameFromDiagonal()	

Aunque tenemos muchos elementos pertenecientes a todas las subclases, lo importante es que hay varios métodos s e t`FrameXXX`. Gracias a ellos, disponemos de varias maneras para cambiar la forma del rectangulo. Los métodos setFrameFromCenter() resultan especialmente interesantes porque nos permiten trabajar desde un punto central (y esquinas, en lugar de partir de los radios).

Nota

`Rectangle2D` o `RoundRectangle2D` no se han explicado en profundidad, pero su uso es casi identico al de `Graphics` con `drawRect()`, `fillRect()`, `drawRoundRect()` y `fillRoundRect()`. Con `RoundRectangle2D` tendremosque recordar que hemos de utilizar los diametros vertical y horizontal del arco de las cuatro esquinas.

Las clases `QuadCurve2D` y `CubicCurve2D` son diferentes al resto de clases especializadas en curvas ya que no trabajan con formas cerradas. `QuadCurve2D` crea curvas de segundo orden utilizando una fórmula cuadrática, mientras que la clase `CubicCurve2D` utiliza curvas de tercer nivel. Este tipo de clases utiliza las curvas de Bezier para definir los puntos que se encuentran en la linea curva. Después de crear la linea curva, utilizarán los métodos `draw()` o `fill()` para rellenarla. En el caso de `fill()`, podemos considerar que los puntos de inicio y final están conectados entre sí, con lo que determinaremos una región cerrada.

Nota

Las curvas Bezier son curvas continuas que tienen una gran cantidad de propiedades matematicas que se escapan a la finalidad de esta obra

A la hora de crear un objeto `Quadcurve2D` podremos especificar los puntos de inicio y fin, asi como uno de control. La forma que se ha creado sera una curva continua que vaya del punto inicial hasta el final, dónde se empuja la linea curva hacia el punto de control. En la figura 11.10 podemos ver como dibujar elipses y curvas cuadráticas. Para que el ejemplo sea más claro, tambien se han dibujado los puntos individuales de la curva cuadrática. El listado 11.7 es el cddigo fuente empleado para generar la figura 11.10.

Listado 11.7. Demostracion de dibujo de formas curvas

```
import java.awt.*;
import java.awt.geom.*;
public class QuadEll extends ExitableJFrame {

    public void paint(Graphics g) {
```

```
        Graphics2D g2d = (Graphics2D)g;
        g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
            RenderingHints.VALUE_ANTIALIAS_ON);
        g2d.setPaint(Color.BLUE);
        g2d.setStroke(new BasicStroke(3));
        // Circulo
        Ellipse2D ellipse = new Ellipse2D.Float(20, 30, 100, 100);
        g2d.draw(ellipse);
        GradientPaint gp = new GradientPaint(135, 30, Color.CYAN, 235, 105,
            Color.BLUE);
        g2d.setPaint(gp);
        // Óvalo
        ellipse = new Ellipse2D.Float(135, 30, 100, 75);
        g2d.fill(ellipse);
        // Curva cuadrangular
        QuadCurve2D quad = new QuadCurve2D.Float(10, 155, 150, 255, 235,
180);
        g2d.setPaint(Color.RED);
        g2d.fill(quad);
        // Puntos de la curva cuadrangular
        g.setColor(Color.BLACK);
        g.fillRect(10, 155, 5, 5);
        g.fillRect(148, 253, 5, 5);
        g.fillRect(233, 178, 5, 5);
    }

    public static void main(String args[]) {
        Frame f = new QuadEll();
        f.setTitle("QuadEll");
        f.setSize(300, 300);
        f.show();
    }
}
```

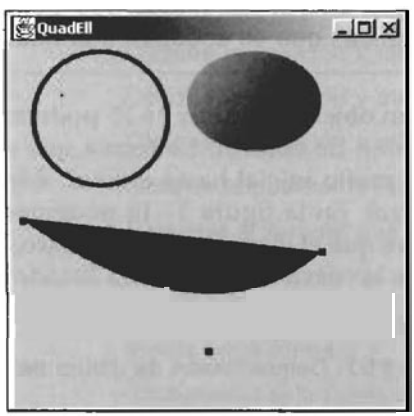


Figura 11.10. Dibujo de elipses curvas cuadraticas

Si queremos crear un elemento `CubicCurve2D`, tendremos que especificar un punto de inicio y otro de finalizacibn. Pero en vez de utilizar un punto de control se

utilizarán dos. A menos que la curva sea lineal, no pasará por el punto de control. Para calcular los puntos de la curva se interpolan los puntos finales y los de control, empujando la trayectoria curvada hacia los puntos de control entre los dos puntos finales. La figura 11.11 muestra varias curvas cúbicas, en esta ocasión con sólo los puntos de control dibujados. El codigo fuente se encuentra en el listado 11.8.

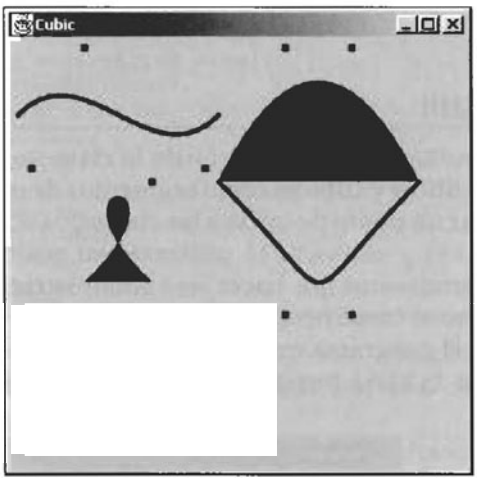


Figura 11.11. Dibujo de curvas cúbicas de Bezier

Listado 11.8. Dernostracion de dibujo de curvas cúbicas

```
import java.awt.*;
import java.awt.geom.*;
public class Cubic extends ExitableJFrame {

    public void paint(Graphics g) {
        Graphics2D g2d = (Graphics2D)g;
        g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
            RenderingHints.VALUE_ANTIALIAS_ON);
        g2d.setPaint(Color.BLUE);
        g2d.setStroke(new BasicStroke(3));
        CubicCurve2D cubic = new CubicCurve2D.Float(10, 80, 60, 30, 110, 130,
            160, 80);
        g2d.draw(cubic);
        cubic = new CubicCurve2D.Float(60, 205, 150, 120, 20, 120, 110, 205);
        g2d.fill(cubic);
        cubic = new CubicCurve2D.Float(160, 130, 210, 30, 260, 30, 310, 130);
        g2d.fill(cubic);
       cubic = new CubicCurve2D.Float(160, 130, 260, 230, 210, 230, 310, 130);
        g2d.draw(cubic);
        // Puntos de la curva
        g.setColor(Color.BLACK);
        g.fillRect(58, 28, 5, 5);   g.fillRect(108, 128, 5, 5);
        g.fillRect(148, 118, 5, 5); g.fillRect(18, 118, 5, 5);
        g.fillRect(208, 28, 5, 5);  g.fillRect(258, 28, 5, 5);
```

```
        g.fillRect(250, 220, 5, 5); g.fillRect(200, 220, 5, 5);

    public static void main(String args[]) {
        Frame f = new Cubic();
        f.setTitle("Cubic");
        f.setSize(350, 350);
        f.show();
```

Curvas GeneralPath

Los métodos `quadTo()` y `curveTo()` de la clase `GeneralPath` nos permiten utilizar curvas cuadrdticas y cúbicas como segmentos de un elemento `GeneralPath`. En vez de suministrar un punto de inicio a las clases `QuadCurve2D` y `CubicCurve2D`, los metodos `quadTo()` y `curveTo()` utilizarán su posición actual como punto de inicio. Todo lo que tendremos que hacer serd suministrarle un punto final y otro de control (o mds de uno si fuese necesario).

El listado 11.9 es el programa que produce la figura 11.12. Si lo deseamos, podemos cambiar la línea `fill()` por `draw()` para verlo mejor.

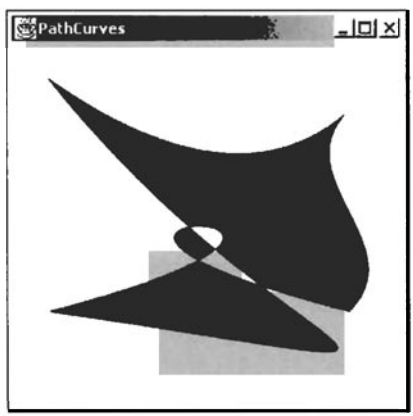


Figura 11.12. Dibujar curvas con GeneralPath

Listado 11.9. Demostración del dibujo de curvas en una trayectoria

```
import java.awt.*;
import java.awt.geom.*;
public class PathCurves extends ExitableJFrame {

    public void paint(Graphics g) {
        Graphics2D g2d = (Graphics2D)g;
        g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
            RenderingHints.VALUE_ANTIALIAS_ON);
        g2d.setPaint(Color.BLUE);
        g2d.setStroke(new BasicStroke(3));
```

```
        GeneralPath shape = new Genera: Path(GeneralPath.WIND_EVEN_ODD);
        shape.moveTo(30, 50);
        shape.quadTo(170, 150, 255, 75);
        shape.curveTo(205, 125, 305, 175, 255, 225);
        shape.curveTo(-70, 140, 355, 140, 30, 225);
        shape.curveTo(410, 280, 210, 280, 30, 50);
        g2d.fill(shape);


    public static void main(String args[]) {
        Frame f = new PathCurves();
        f.setTitle("PathCurves");
        f.setSize(300, 300);
        f.show();
    }
}
```

Transformaciones

La interfaz API 2D de Java puede transformar coordenadas. Gracias a estas transformaciones, podemos rotar, escalar, girar, cortar o cambiar la ubicacion de una operación de dibujo.

Para la gente que adora las matemiticas, la transformación de coordenadas es un proceso que utiliza una matriz de 3x3 con las coordenadas x e y originales (utiliza un 1 para las del eje z). El contenido de la matriz define la transformación real.

La clase `AffineTransform` es la responsable de las transformaciones y casi nunca necesitará buscar más allá de los métodos estáticos de la clase (que aparece aqui):

```
public static AffineTransforrn getRotateInstance(double theta)
public static AffineTransforrn getRotateInstance(double theta, double x,
    double y)
public static AffineTransform getScaleInstance(double scaleX,
    double scaleY)
public static AffineTransforrn getShearInstance(double shiftX,
    double shiftY)
public static AffineTransform getTranslateInstance(double translateX,
    double translateY)
```

Una vez que comprendemos las distintas transformaciones que podemos efectuar, creamos un objeto `AffineTransform` para dicha transformacion. Y, del mismo modo que juntábamos objetos `ImageFilter`, podemos encadenar varios casos de `AffineTransform` para que desarrollen una serie de transformaciones a través del método `concatenate()`.

- **Traslacidn:** La transformacion más sencilla es la que utiliza `getTranslateInstance()`. Su comportamiento es el mismo que el método `Translate()` de `Graphics`. Todas las operaciones de dibujo se transforman a las coordenadas que se suministran.

- **Escalado:** La siguiente transformación en dificultad es el cambio de escala. Los parámetros de `getScaleInstance()` definen el factor de la escala, junto con los ejes x e y. De esta forma se consiguen imágenes más pequeñas o más grandes que la original. El método `getScaleInstance()` es comparable a los filtros `AreaAveragingScaleFilter` y `ReplicateScaleFilter` cuando trabajamos con el modelo productor-consumidor de la imagen.
- **Rotacidn:** Para realizar una transformacion de rotación se usa uno de los metodos `getRotateInstance()`. Necesitan un ángulo en radianes para determinar la extensión de la rotación. Para convertir grados en radianes, realice la operación radianes = grados * ∏/180, aunque puede simplemente invocar a `Math.toRadians(grados)` para realizar la conversión. Rotar en posición positiva es en el sentido de las agujas del reloj; negativa seria en sentido contrario a las agujas del reloj. Cuando gira, lo hace alrededor del origen (0,0) o desde otro punto.
- **Deformacidn:** La deformación es el efecto mas complicado de explicar. Básicamente es una operación de traslación a unos ejes x e y que no son perpendiculares. En la traslación se multiplica cada posición por los factores de deformación suministrados. Este factor permite que las líneas paralelas lo sigan siendo, mientras que las perpendiculares se convierten en divergentes, como transformar un rectángulo en un paralelogramo.

La figura 11.13 nos ayudará a hacernos una idea de los distintos tipos de transformaciones tratados aqui. El listado 11.10 es el programa que se usa para crear las diferentes transformaciones que aparecen en la figura 11.13. Todas ellas son transformaciones de sencillos rectángulos.

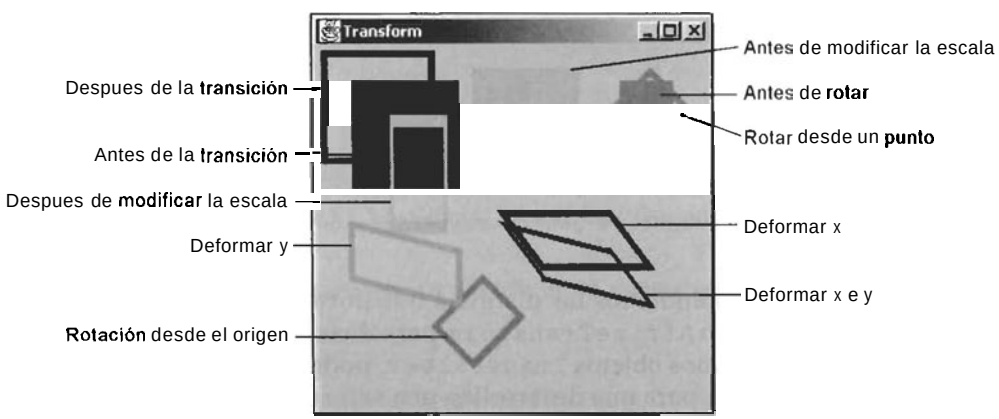


Figura 11.13. Operaciones de transformacion de dibujo

Listado 11.10. Demostracion de dibujo con transformaciones

```
import java.awt.*;
import java.awt.geom.*;
```

```
public class Transform extends ExitableJFrame {

    public void paint(Graphics g) {
       AffineTransform reset = new AffineTransform();
       Graphics2D g2d = (Graphics2D)g;
       g2d.setStroke(new BasicStroke(5f));
       // Rectángulo trasladado
       g2d.setColor(Color.RED);
       Rectangle2D rectangle = new Rectangle2D.Double(30, 50, 80, 80);
       AffineTransform at = AffineTransform.getTranslateInstance(-20, -20);
       g2d.setTransform(at);
       g2d.draw(rectangle);
       // Reinicia las transformaciones
       g2d.setTransform(reset);
       g2d.fill(rectangle);
       // Modifica la escala del rectángulo
       g2d.setColor(Color.ORANGE);
       rectangle = new Rectangle2D.Double(120, 40, 80, 40);
       at = AffineTransform.getScaleInstance(.5, 2);
       g2d.setTransform(at);
       g2d.draw(rectangle);
       // Reinicia las transformaciones
       g2d.setTransform(reset);
       g2d.fill(rectangle);
       // Rota el rectángulo
       g2d.setColor(Color.GREEN);
       rectangle = new Rectangle2D.Double(230, 50, 40, 50);
       at = AffineTransform.getRotateInstance(Math.toRadians(45),
      rectangle.getCenterX(), rectangle.getCenterY());
       g2d.setTransform(at);
       g2d.draw(rectangle);
       // Rota desde el origen
       at = AffineTransform.getRotateInstance(Math.toRadians(45));
       g2d.setTransform(at);
       g2d.draw(rectangle);
       // Reinicia las transforrnaciones
       g2d.setTransform(reset);
       g2d.fill(rectangle);
       // Deforrna el rectángulo
       g2d.setColor(Color.BLUE);
       rectangle = new Rectangle2D.Double(30, 150, 80, 40);
       at = AffineTransform.getShearInstance(.75, 0);
       g2d.setTransform(at);
       g2d.draw(rectangle);
       g2d.setColor(Color.CYAN);
       at = AffineTransform.getShearInstance(0, .25);
       g2d.setTransform(at);
       g2d.draw(rectangle);
       g2d.setColor(Color.MAGENTA);
       at = AffineTransform.getShearInstance(.75, .25);
       g2d.setTransform(at);
       g2d.draw(rectangle);
    }

    public static void main(String args[]) {
       Frame f = new Transform();
```

```
:.setTitle("Transform");
1.setSize(300, 300);
f.show();
```

Composicion

Antes de la aparición de la interfaz API 2D de Java, teniamos dos opciones a la hora de dibujar una linea de un color. Podia ser completamente opaca (ocultando cualquier objeto que se hubiese dibujado con anterioridad en dicha posicion) o se podia dibujar en modo XOR, combinando el color ya existente en dicha posicion, con el del nuevo elemento. Como resultado obteniamos un tercer color de tal forma que si repetiamos la operación, volviamos a obtener el color original. Con la ayuda de la clase **`AlphaComposite`**, ahora podemos combinar colores. De esta forma podemos mezclar un cuadro azul y un cuadro rojo y obtener un cuadro morado, un cuadro rojo, un cuadro azul e incluso no obtener ningún cuadro.

La clase `AlphaComposite` tiene doce constantes que definen las reglas de la combinación de colores. La forma en que afectan al proceso de combinación se describe en la tabla 11.5. Hemos de recordar estos efectos a la hora de dibujar sobre áreas rectangulares.

Tabla 11.5. Constantes de mezcla AlphaComposite

Modos	Efecto de mezcla
CLEAR	Borra el color y el valor alfa del objeto de destino.
DST	Dibuja solo el objeto de destino.
DST_ATOP	Dibuja el objeto fuente, pero en las partes donde el objeto de destino se solapa, dibuja objeto de destino.
DST_IN	Solo dibuja la parte del objeto de destino que se superpone al objeto fuente.
DST_OUT	Solo dibuja la parte del objeto de destino que no solapa el objeto fuente.
DST_OVER	Dibuja el objeto de destino sobre el objeto fuente.
SRC	Dibuja el color y el valor alfa del objeto fuente.
SRC_ATOP	Dibuja el objeto de destino, pero en las partes donde el objeto fuente se solapa, dibuja el objeto fuente.
SRC_IN	Solo dibuja la parte del objeto fuente que solapa el destino
SRC_OUT	Solo dibuja la parte del objeto fuente que no solapa el destino.

Modos	Efecto de mezcla
SRC_OVER	Dibuja el objeto fuente sobre el objeto de destino.
XOR	Sólo dibuja el objeto fuente y el objeto de destino en laszonas donde no se produce solapamiento.

Cuando nos hacemos con un objeto `AlphaComposite`, tendremos que especificar el valor alpha del color que se va a dibujar, además de la regla (lo vera en el siguiente apartado). El intervalo de valores en que se moverá está comprendido entre 0.0 y 0.1 y podemos verlo como el porcentaje de color de fondo que se puede ver a través del color de nuestro objeto. Para obtener un ejemplo de la clase utilizamos el método estático `getInstance()`. Si no se especifica ningún porcentaje, se utilizara el valor 1.0. También tenemos una serie de variables públicas para cada tipo de regla `AlphaComposite`, donde el valor de alpha es 1.0. Sus nombres son los siguientes: `Clear`, `Dst`, `DstAtop`, `DstIn`, `DstOut`, `DstOver`, `Src`, `SrcAtop`, `SrcIn`, `SrcOut`, `SrcOver` y Xor, respectivamente.

Reglas para combinar imágenes

Las reglas implementadas por la clase `AlphaComposite` son el conjunto de reglas Porter-Duff descritas por Thomas Porter y Tom Duff "Compositing Digital Images". Puede encontrar una versión escaneada del artículo en Internet en http://www.keithp.com/~keithp/porterduff/

Este ejemplo en concreto, es el más dificil de demostrar sobre el papel. Para comprender las reglas nos apoyaremos en el contenido de la figura 11.14 donde tenemos tres ejemplos de las doce reglas dibujadas con unos valores alpha de 0.33, 0.67 y 1.0 (de arriba abajo). La figura original es el triángulo verde que se encuentra a la izquierda del rectangulo. La figura final es el triangulo magenta que toma la parte superior del rectángulo. Este proporciona dos formas con una zona solapada y otra no. Los elementos se han dibujado por separado antes de proceder a su combinación. De esta forma, podremos especificar los atributos de combinación de cada uno por separado. El programa del listado 11.11 genera la figura 1.14.

Listado 11.11. Demostracion de dibujo de imagenes compuestas

```
import java.awt.*;
import java.awt.geom.*;
import java.awt.image.*;
import javax.swing.*;

public class CompositeIt extends ExitableJFrame {
    int rules[] = {
       AlphaComposite.CLEAR,     AlphaComposite.DST,
```

```
    AlphaComposite.DST_ATOP, AlphaComposite.DST_IN,
    AlphaComposite.DST_OUT,  AlphaComposite.DST_OVER,
    AlphaComposite.SRC,      AlphaComposite.SRC_ATOP,
    AlphaComposite.SRC_IN,   AlphaComposite.SRC_OUT,
    AlphaComposite.SRC_OVER, AlphaComposite.XOR};
float percents[] = {.33f, .67f, 1.0f};
BufferedImage source, dest;
GeneralPath sourcePath, destPath;

public CompositeIt() {
    sourcePath = new GeneralPath();
    sourcePath.moveTo(0,   0);   sourcePath.lineTo(50, 0);
    sourcePath.lineTo(50, 25);   sourcePath.closePath();
    source = new BufferedImage(80, 30, BufferedImage.TYPE_INT_ARGB);
    destPath = new GeneralPath();
    destPath.moveTo(25,  0);    destPath.lineTo(75, 0);
    destPath.lineTo(25, 25);    destPath.closePath();
    dest = new BufferedImage(80, 30, BufferedImage.TYPE_INT_ARGB);
}

public void paint(Graphics g) {
    Graphics2D g2d = (Graphics2D)g;
    Graphics2D sourceG = source.createGraphics();
    Graphics2D destG = dest.createGraphics();
    AffineTransform at = new AffineTransform();
    Composite originalComposite = g2d.getComposite();
    for(int i=0; i<3; i++) {
        for(int j=0, n=rules.length; j<n; j++) {
            at = AffineTransform.getTranslateInstance(j*80+10, i*30+30);
            if (j >= rules.length/2) {
                at.translate(-rules.length/2*80, 120);
            }
            g2d.setTransform(at);
            g.drawRect(0, 0, 80, 30);
            destG.setComposite(AlphaComposite.Clear);
            destG.fillRect(0, 0, 80, 30);
            destG.setComposite(AlphaComposite.getInstance(
                AlphaComposite.XOR, percents[i]));
            destG.setPaint(Color.MAGENTA);
            destG.fill(destPath);
            sourceG.setComposite(AlphaComposite.Clear);
            sourceG.fillRect(0, 0, 80, 30);
            sourceG.setComposite(AlphaComposite.getInstance(
                AlphaComposite.XOR, percents[i]));
            sourceG.setPaint(Color.GREEN);
            sourceG.fill(sourcePath);
            destG.setComposite(AlphaComposite.getInstance(rules[j]));
            destG.drawImage(source, 0, 0, null);
            g2d.drawImage(dest, 0, 0, this);
        }
    }
}

public static void main(String args[]) {
    Frame f = new CompositeIt();
    f.setTitle("CompositeIt");
```

```
f.setSize(525, 275);
f.show();
```

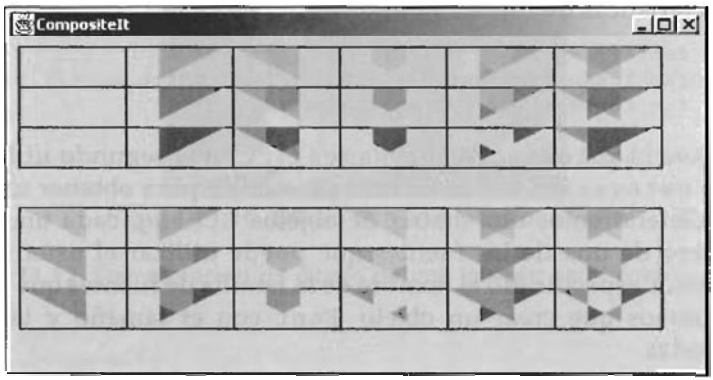


Figura 11.14. Dibujo con cornbinaciones alfa

Procesar texto

La interfaz API 2D de Java dispone de varias propiedades para procesar texto. El capitulo 10 le enseñó a acceder a todas las fuentes especificas de cada plataforma. Además, podemos utilizar el texto como patrón de corte.

Ampliación del soporte para fuentes

Cómo vimos en el capitulo 10, podemos acceder a cualquiera de las fuentes propias de la plataforma y disponibles en un entorno de Java. Aunque podemos acceder a cualquiera de las fuentes, en el codigo no se especifica ninguno de sus nombres. Si no fuese asi, podriamos especificar una fuente que no tuviese un usuario, con lo que nos encontrariamos ante un programa que no sabria cómo representar un texto en la pantalla del usuario. Lo mejor es utilizar los nombres estándar de las fuentes, como `Serif`, `SansSerif` y `Monospaced` o proporcionar una lista desplegable a los usuarios desde la cual pudiese seleccionar el tipo de fuente que fuese a utilizar. Mientras la lista de fuentes emergente se llene en tiempo de ejecución, no tendrá que preocuparse por si introduce en su código un nombre de fuente no disponible.

La creación de la lista de fuentes se efectúa de dos formas. Con ambas se utiliza la clase `GraphicsEnvironment`, de donde se obtiene un ejemplo de `GraphicsEnvironment` a través del método `getLocalGraphicsEnvironment()`.

- `getAllFonts()`: Una vez que tengamos el elemento `GraphicsEnvironment` podremos establecer una llamada a `getAllFonts()` que devolverá una

matriz de objetos Font, uno por cada una de las fuentes del sistema. Cada fuente representa una versión de un punto de la fuente. En otras palabras, no lo podemos utilizar para dibujar cadenas. Tendremos que utilizar deriveFont() para crear un objeto Font nuevo con el tamaiio apropiado y caracteristicas de estilo.

```
Font fonts[] =
    GraphicsEnvironment.getLocalGraphicsEnvironment().getAllFonts();
Font f = fonts[2].deriveFont(Font.BOLD, 24);
```

- getAvailableFontFamilyNames(): Con el segundo utilizaremos el método getAvailableFontFamilyNames() para obtener una lista de fuentes. Obtendremos una matriz de objetos String, cada uno de ellos con el nornbre de una de las fuentes que puede utilizar el usuario. Una vez que hayamos especificado el nombre de la familia de fuentes que hay que utilizar, tendremos que crear un objeto Font con el tamaiio y las caracteristicas deseadas.

```
GraphicsEnvironment ge =
GraphicsEnvironment.getLocalGraphicsEnvironment();
String fontList[] = ge.getAvailableFontFamilyNames();
Font theFont = new Font(fontList[2], Font.PLAIN, 10);
```

Truco

Se puede configurar la variable JAVA_FONTS (o la propiedad del sistema java.awt.fonts) para que cambie el directorio predeterminado de fuentes. De esta forma podremos crear un directorio con todas las fuentes del sistema, con lo que conseguiremos que la respuesta de Java sea mas rapida porque no necesitaremos generar medidas para todas las fuentes la primera vez que se accede a GraphicsEnvironment. Por supuesto, entonces sólo podra usar las fuentes de ese directorio.

Diseño avanzado de texto

En vez de dibujar una cadena con el método drawstring() de Graphics, la clase TextLayout nos permite hacer varias cosas muy interesantes. Por ejemplo, podemos pedir que se utilice una fuente determinada con una cadena de texto para cierta fuente, a continuación utilizar su contorno para recortar una región en el momento de dibujar la imagen. En la figura 11.15 tenemos un ejemplo de esta técnica, además de dibujar la irnagen no recortada. Todo lo que tenemos que hacer es crear un objeto TextLayout que se base en un objeto String, Font y FontRenderContext. A continuación, podemos utilizar el método getOutline() con Shape, con lo que podremos ajustar la transformacicin (utilizamos AffineTransform) y especificar las coordenadas del origen. El listado 11.12 es el código fuente para realizar el recorte de la figura 11.15.

Figura 11.15. Dibujo con combinaciones alfa

Listado 11.12. Dernostracion de dibujo de una irnagen en el contorno de texto

```
import java.awt.*;
import java.awt.geom.*;
import java.awt.font.*;
import javax.swing.*;

public class ClipImage extends ExitableJFrame {
    Image image = getImage("jaeger.jpg");

    public Image getImage(String imageFile) {
        ImageIcon icon = new ImageIcon(imageFile);
        return icon.getImage();
    }

    public void paint(Graphics g) {
        Graphics2D g2d = (Graphics2D)g;
        FontRenderContext frc = g2d.getFontRenderContext();
        Font f = new Font("Serif", Font.ITALIC | Font.BOLD, getWidth()/4);
        TextLayout tl = new TextLayout("Jaeger", f, frc);
        float width = (float)tl.getBounds().getWidth();
        AffineTransform transform = AffineTransform.getTranslateInstance(
            (getWidth()-width)/2, getHeight()/3);
        Shape sha = tl.getOutline(transform);
        g2d.drawImage(image, 0, getHeight()/3, this);
        g2d.clip(sha);
        g2d.drawImage(image, 0, 0, this);
    }

    public static void main(String args[]) {
        Frame f = new ClipImage();
        f.setTitle("ClipImage");
        f.setSize(300, 250);
        f.show();
    }
}
```

La clase `TextLayout` mejora las propiedades de Java. Es parecido a **`FontMetrics` y `Graphics`.** La definición de la clase de `TextLayout` es la siguiente:

Tabla **11.6.** Composición de la clase TextLayout

Variables/Métodos	Descripción
DEFAULT – CARET – POLICY	Norma por defecto para determinar la posicion del intercalado.
TextLayout()	Construye el objeto **TextLayout.**
clone()	Crea una copia del diseño.
draw()	Dibuja el diseiio.
equals()	Comprueba la igualdad.
getAdvance()	Obtiene un avance del diseño.
getAscent()	Obtiene la ascension de todo el disetio.
getBaseline()	Obtiene el tipo de linea base; puede ser Font.CENTER_BASELINE, **Font·HANGING – BASELINE** o Font.ROMAN_BASELINE
getBaselineOffsets()	Obtiene una matriz de offset de linea base, uno para cada tipo de linea base.
getBlackBoxBounds()	Obtiene un rectángulo delimitador de rango.
getBounds()	Obtiene un rectangulo delimitador.
getCaretInfo()	Obtiene información sobre la intercalacion.
getCaret**Shape()**	Obtiene la forma para que dibuje una intercalacion.
getCaretShapes()	Obtiene una matriz de dos elementos de formas para que dibuje la intercalacion.
getCharacterCount()	Obtiene el número de caracteres del diseño.
getCharacterLevel()	Obtiene el nivel de caracter.
getDescent()	Obtiene el descenso de todo el diseño.
getJustifiedLayout()	Crea un diseño con contenidos justificados.
getLeading()	Obtiene el encabezado de todo el disetio.
getLogicalHighlight**Shape()** getVisualHighlightShape()	Obtiene la forma para la seleccion de diseño.
getLogicalRangesForVisual-**Selection**()	Obtiene el rango de texto a partir de una seleccion visual.
getNextLeftHit()	Obtiene la anterior posición de la intercalación.
getNextRightHit()	Obtiene la siguiente posición para la intercalación.
getOutline()	Obtiene la forma del texto en el diseño.
getVisualAdvance()	Obtiene el avance, menos el espacio en blanco final.

Variables/Métodos	Descripción
`getVisualOtherHit()`	Obtiene la posición para el otro lado de la intercalacion.
`handleJustify()`	Realiza el justificado.
`hashCode()`	Genera codigo hash para el diseiio.
`hitTestChar()`	Deterrnina el caracter mas cercano
`isLeftToRight()`	Devuelvet`rue` si la direccion del texto es de izquierda a derecha.
`isVertical()`	Devuelve`true` si la dirección del texto esvertical.
`toString()`	Convierte el diseño en cadena.

Con la ayuda de otras clases, también puede usar `TextLayout` para dibujar texto multilinea, valiéndonos de `LineBreakMeasurer` con el nombre apropiado para descubrir dónde debe romperse cada línea. Aqui tiene el listado 11.13, que le enseña a hacer esto con el comienzo del discurso de Gettysburg. El resultado aparece en la figura 11.16.

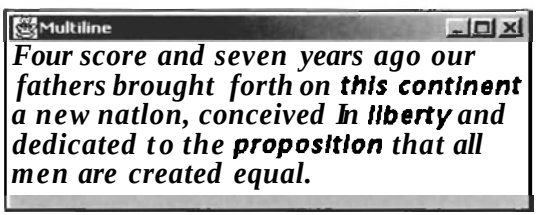


Figura 11.16. Resultado de dibujar con rnultilinea

Listado 11.13. Demostración de dibujar con texto rnultilinea

```
import java.awt.*;
import java.awt.geom.*;
import java.awt.font.*;
import java.text.*;

public class Multiline extends ExitableJFrame {

    AttributedString attributedString;

    public Multiline() {
        String fourScore = "Four score and seven years ago " +
        "our fathers brought forth on this continent a new " t
        "nation, conceived in liberty and dedicated to the " t
        "proposition that all men are created equal.";
        Font font - new Font("SanSerif", Font.ITALIC, 181;
```

```
        attributedString =
            new AttributedString(fourScore);
        attributedString.addAttribute(TextAttribute.FONT, font);
    }

    public void paint(Graphics g) {
        Graphics2D g2d = (Graphics2D)g;
        // Obtiene el iterador para la cadena
        AttributedCharacterIterator attributedCharacterIterator =
            attributedString.getIterator();
        // Obtiene el cnntexto fuente a partir de los gráficos
        FontRenderContext fontRenderContext =
            g2d.getFontRenderContext();
        // Crea un medidor
        LineBreakMeasurer lineBreakMeasurer =
            new LineBreakMeasurer(attributedCharacterIterator,
                fontRenderContext);
        // Revisa los márgenes
        insets insets = getInsets();
        int width = getSize(j.width - insets.right - insets.left;
        // Establece la posición inicial para dibujar
        int x = insets.left;
        int y = insets.top;

        while (lineBreakMeasurer.getPosition() <
               attributedCharacterIterator.getEndIndex()) {
        // Obtiene la línea
        TextLayout textLayout = lineBreakMeasurer.nextLayout(width);
        // Se mueve a la línea base
        y += textLayout.getAscent();
        // Dibuja la línea
        textLayout.draw(g2d, x, y);
        // Se mueve a la parte superior de la siguiente línea
        y += textLayout.getDescent() + textLayout.getLeading();
    }

  public static void main(String arqs[]) {
    Frame f = new Multiline();
    f.setTitle("Multiline");
    f.setSize(350, 150);
    f.show();
    }
```

Manipulador de imágenes

La interfaz API 2D de Java incluye una serie de propiedades mejoradas para manipular imdgenes. Se han incluido en el paquete `java.awt.image`. Gracias a ellas ahora contamos con la posibilidad de crear imágenes en la memoria con `BufferedImages`, así como desarrollar operaciones de filtrado. Además, la biblioteca de `Image I/O`, `javax.imageio` (fuera de la API 2D), le permite acceder directamente a los datos de la imagen para guardar los datos en primer lugar y

luego repetir la carga. Esta sección. En este apartado le ensefiaremos a hacer todo esto y más.

Formato flexible de imágenes

Siempre que queramos crear imágenes en la memoria, de tal forma que podamos desarrollar estas operaciones con un doble búfer, tendremos que utilizar las clases BufferedImage o VolatileImage. Un BufferedImage es un tipo de Image (clase que utilizamos para dibujar en la pantalla desde un segundo plano). De hecho, el doble búfer trabaja deesta forma. VolatileImage es otro tipo de Image. Funciona de la misma forma que BufferedImage, sin embargo, existe la posibilidad de que el sistema operativo la borre. Este riesgo conlleva un aumento sustancial del rendimiento.

Cuando trabaja con BufferedImage, la primera vez que se crea una zona en el búfer de la memoria, se especifica el tamaño y cómo se va a manipular dentro del bufer. Para ello se utilizan las constantes de clase TYPE_* que contiene BufferedImage, las cuales se muestran a continuación:

```
public static final int TYPE_3BYTE_BGR;
public static final int TYPE_4BYTE_ABGR;
public static final int TYPE_4BYTE_ABGR_PRE;
public static final int TYPE_BYTE_BINARY;
public static final int TYPE_BYTE_GRAY;
public static final int TYPE_BYTE_INDEXED;
public static final int TYPE_CUSTOM;
public static final int TYPE_INT_ARGB;
public: static final int TYPE_INT_ARGB_PRE;
public static final int TYPE_INT_BGR;
public static final int TYPE_INT_RGB;
public:: static final int TYFE_USHORT_555_RGB;
public static final int TYFE_USHORT_565_RGB;
public static final int TYPE_USHORT_GRAY;
```

Cada una de ellas nos permite especificar la forma en que queremos que se guarden internamente los datos. Se guardan en un DataBuffe r que se encuentra dentro de un objeto Raster, asociado a BufferedImage. Si examinamos el objeto Raster nos encontraremos que aparecen los valores de los pixeles de la imagen. En vez de utilizar el método getRaster() para hacernos con el objeto, utilizaremos una célula Raster.

Si trabajamos con células podremos asociar un elemento TileObserver al que se notificará cualquier cambio que se efectue en los datos.

Nota

No hay ninguna clase Tile. Una célula unicarnente representa una zona irnprimible de la imagen con la que estamos trabajando.

La mayoría de las formas `BufferedImage` que se crean son del tipo:

- `TYPE_INT_RGB`: Representan un objeto `DirectColorModel` que carece de valor alfa.
- `TYPE_INT_ARGB`: Representan un objeto `DirectColorModel` con valor alfa.
- `TYPE_INT_INDEXED`: Rrepresentan a un objeto `IndexColorModel`.

Los otros tipos se utilizarán en circunstancias más especiales.

La creación de un objeto `BufferedImage` es un proceso muy sencillo. Todo lo que tenemos que hacer es llamar al constructor con la anchura, altura y tipo:

```
BufferedImage bufferedImage = new BufferedImage (400, 200,
    BufferedImage.TYPE_INT_RGB);
```

Una vez que tenemos un caso de `BufferedImage`, podremos dibujar un objeto `Graphics`, procedente de `getGraphics()` o un objeto `Graphics2D`, creando un objeto con el método `createGraphics()` de `BufferedImage`.

```
Graphics g = bufferedImage.getGraphics();
Graphics2D g2d = bufferedImage.createGraphics();
```

Si queremos crear un objeto `BufferedImage` optimizado para un dispositivo gráfico especifico, tendremos que solicitar un elemento `GraphicsConfiguration`. Asi permitiremos que el dispositivo dibuje con una gran rapidez. Suponiendo que estamos trabajando con el método `paint()` y que `g` es el parámetro `Graphics` de dicho método, el siguiente extracto de código creará el elemento `BufferedImage` óptimo:

```
Graphics2D g2d = (Graphics2D)g;
GraphicsConfiguration gc = g2d.getDeviceConfiguration();
BufferedImaged bi = gc.createCompatibleImage(width, height);
```

También se puede crear un `VolatileImage`, solicitando la configuración gráfica. O se puede hacer que una subclase `Component` lo haga por nosotros:

```
VolatileImage vil = gc.createCompatibleVolatileImage(width, height);
VolatileImage vi2 = aComponent.createVolatileImage(width, height);
```

Cuando se usa una `VolatileImage`, hay que comprobar repetidas veces si se pierden los contenidos por medio de su método `contentsLost()`.

Operaciones avanradas con imágenes

Utilizar `ImageFilter` para crear un filtro (para las imágenes que examinen los pixeles que tienen a su alrededor) es un trabajo realmente tedioso. La interfaz API 2D de Java incluye un objeto `BufferedImageFilter`, que requiere la presencia de `BufferedImapeOp` para definir la operación, que permite que se filtren los objetos `BufferedImage`. Filtrar un objeto `BufferedImage` con `BufferedIma-`

`geFilter` es una operación que se puede hacer sobre el original, por lo que no podremos volver a utilizar el elemento original después de la operación. Si no queremos que se sustituya la imagen inicial, tendremos que utilizar le método `filter()` de `BufferedImageOP`, que nos permite especificar un origen y un destino. De todas formas, algunos filtros no pueden trabajar con este sistema.

Hay varias operaciones que se pueden utilizar para filtrar las imagenes que se encuentran en la memoria. La mayoría no son otra cosa que la implementación de las subclases `ImageFilter` ajustadas para que resulte más sencilla su utilización dentro del mundo `Graphics2D`. A continuación mostramos una lista de los elementos que implementan la interfaz `BufferedImageOp`, los cuales trabajan con objetos `Raster` además de `BufferedImages`:

- `AffineTransformOp`: Se utiliza para efectuar transformaciones afines, como la clase `AffineTransform`.
- `ColorConvertOp`: Se utiliza con conversiones de color, como el filtro `RGBImageFilter` o `GrayFilter`.
- `ConvolveOp`: Se utiliza para desarrollar conversiones de pixeles, basindose en los valores de los pixeles circundantes.
- `LookupOp`: Se utiliza para efectuar modificaciones no uniformes a la imagen original a través de operaciones de conversión.
- `RescaleOP`: Se utiliza para modificar la escala de las imágenes, como las clases `ReplicateScaleFilter` y `AreaAveragingScaleFilter`.

Además de las operaciones de filtrado anteriores, hay otra operación que trabaja solamente con objetos `Raster` e implementa la interfaz `RasterOp`:

- `BandCombineOp`: Combina las bandas dentro de un `Raster`.

Lo más interesante de las operaciones de filtrado es `ConvolveOp` para las operaciones de convolución. Cuando convolucionamos una imagen, calculamos la suma del valor de los pixeles circundantes. Un elemento `Kernel` especifica el coeficiente que se utilizará con `ConvolveOp`. Al crear un `Kernel` podremos especificar la ponderación que se utilizari con los pixeles circundantes. Por ejemplo, si queremos que una convolución distorsione la imagen de alguna forma, tendremos que crear un kernel de la siguiente manera:

```
public static final float[] SHARP =
      {  0.0f, -1.0f,  0.0f,
        -1.0f,  5.0f, -1.0f,
          0.0f, -1.0f,  0.0f};
Kernel kernel = new Kernel(3, 3, SHARE');
```

Obsérvese que la suma de los valores de la matriz que se utiliza para el filtro es 1. Se ha hecho a propósito. En la convolución se suma el valor de los pixeles circundantes para crear un nuevo pixel. Por eso ha de ser 1.

Una vez que tenemos un objeto **Kernel**, podemos crear un `ConvolveOp` y filtrar una imagen `BufferedImage`. Los parámetros adicionales que se le entregan al constructor, definen las condiciones extrernas y las normas de dibujo. `EDGE_NO_OP` indica que hay que hacer una copia del original sin aplicar ninguna modificación. La operacion predeterminada es `EDGE_ZERO_FILL`, que elimina los píxeles del borde.

```
ConvolveOp cop = new ConvolveOp(kernel, ConvolveOp.EDGE_NO_OP, null);
cop.filter(bufferedImageSrc, bufferedImageDst);
g.drawImage(bufferedImageDst, x, y, this);
```

Si `BufferedImageOp` puede efectuar el filtrado directamente sobre la imagen original, podrernos efectuar la operación dentro de `drawImage()`, corno se indica a continuación.

```
g2d.drawImage(bufferedImage, cop, x, y);
```

Si le echamos un vistazo a la figura 11.17 verernos que representa la irnagen de la figura 11.15 con un efecto de distorsión en los bordes. Este es el resultado que hemos obtenido al aplicar el filtro.

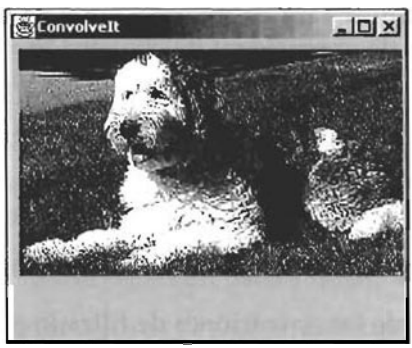


Figura 11.17. Convolución de una imagen con un filtro distorsionador

El listado 11.14 es el código encargado de crear el contenido de la figura anterior:

Listado 11.14. Dernostracion de dibujo de una imagen con una operacion de convolución

```
import java.awt.*;
import java.awt.geom.*;
import java.awt.image.*;
import javax.swing.*;

public class ConvolveIt extends ExitableJFrame {
    Image image = getImage("jaeger.jpg");

    private static final float[] SHARP =
        0.0f, -1.0f,  0.0f,
```

```
      -1.0f,  5.0f, -1.0f,
       0.0f, -1.0f,  0.0f};
  BufferedImage bufferedImage;
  ConvolveOp convolveOp;

  public Image getImage(String imageFile) {
    ImageIcon icon = new ImageIcon(imageFile);
    return icon.getImage();
  }

  public ConvolveIt() {
    int width = image.getWidth(this);
    int height = image.getHeight(this);
    bufferedImage = new BufferedImage (width, height,
        BufferedImage.TYPE_INT_RGB);
    Graphics2D big = bufferedImage.createGraphics();
    AffineTransform affineTransform = new AffineTransform();
    big.drawImage(image, affineTransform, this);
    Kernel kernel = new Kernel(3, 3, SHARP);
    convolveOp = new ConvolveOp(kernel, ConvolveOp.EDGE_NO_OP, null);
   }

  public void paint(Graphics g) {
    Graphics2D g2d = (Graphics2D)g;
    if (bufferedImage != null) {
        g2d.drawImage(bufferedImage, convolveOp, 10, 30);
    }
  }

  public static void main(String args[]) {
    Frame f = new ConvolveIt();
    f.setTitle("ConvolveIt");
    f.setSize(300, 250);
    f.show();
```

Truco

Se pueden conseguir muchas arrays de kernel interesantes. Por ejemplo, si coloca `1.0/9.0` en cada una de las células, obtendra un efecto desenfocado. Además la matriz de kernel no tiene por que ser 3 x 3. Otro tamaño de array comun es 5 x 5.

Codificación de imágenes

Como se mencionó anteriormente, la API **`Image I/O`** es uno de los tres API multimedia que acompañan a Java 1.4 SDK. Como indica el nombre de la biblioteca, se puede leer y escribir archivos de imagen con las funciones de **`Image I/O`**. (Antes

de Java 1.4, `getImage()` permitia leer imagenes, pero no habia un `writeImage()` equivalente.) Con la biblioteca `Image I/O`, no hay un soporte estándar para escribir imágenes. Y si necesita escribir o leer una imagen que no pertenece a un formato estándar como TIF, la biblioteca es compatible con arquitecturas conectables. Con sólo añadir una archivo .JAR, se admitirá el nuevo formato.

La clase clave para toda esta escritura y lectura es `javax.imageio.ImageIO`. Bdsicamente, la clase está llena de metodos estáticos. Para leer una imagen, todo lo que tiene que hacer es invocar a `ImageIO.read()`. Asi leer6 una imagen en un objeto `BufferedImage`, suponiendo que se admita el formato de la imagen. La escritura de imagenes se hace mediante llamadas a `ImageIO.write()`. Deberá especificar el formato que se va a escribir, como JPEG o PNG.

Para descubrir los formatos admitidos, se pregunta a `ImageIO` invocando otro par de métodos: `getReaderFormatNames()` y `getWriterFormatNames()`. Por defecto, los formatos del lector son GIF, JPEG y PNG y los formatos de escritura son JPEG y PNG. A riesgo de enfadar a los defensores de Unisys, le recomendamos encarecidamente que no intente escribir imágenes en formato GIF. Para demostrar el empleo de `ImageIO`, el listado 11.15 lee un archivo de imagen, realiza el filtro de desenfocado que se mostro anteriormente y guarda la imagen modificada como imagen con formato PNG. Los métodos `read()` y `write()` pueden arrojar una `IOException`, de modo que la excepción debe ser atrapada o arrojada.

Listado 11.15. Demostracion del uso de la biblioteca Image I/O Library

```
import java.awt.image.*;
import javax.imageio.*;
import java.io.*;

public class SaveIt {

    private static final float[] SHARP = {
          0.0f, -1.0f,  0.0f,
         -1.0f,  5.0f, -1.0f,
          0.0f, -1.0f,  0.0f};
    public static void main(String args[]) throws IOException {
        // Lee
        File inputFile = new File("jaeger.jpg");
        BufferedImage input = ImageIO.read(inputFile);
        // Convierte
        Kernel kernel = new Kernel(3, 3, SHARP);
        ConvolveOp convolveOp = new ConvolveOp(kernel,
ConvolveOp.EDGE_NO_OP, null);
        int width = input.getWidth();
        int height = input.getHeight();
        BufferedImage output = new BufferedImage (width, height,
            BufferedImage.TYPE_INT_ARGB);
        convolveOp.filter(input, output);
        // Guarda
        File outputFile = new File("sharp.png");
        ImageIO.write(output, "PNG", outputFile);
    }
}
```

En la biblioteca Image I/O hay mucho mfis de lo que mostramos aqui (por ejemplo, puede crear su propio plug-in de formato de imfigenes). Sin embargo, lo que hemos visto aqui es suficiente para la mayoría de las aplicaciones. Para mfis información, lea Guia de la API Java Image I/O, disponible en

http://java.sun.com/j2se/1.4/docs/guide/imageio/spec/title.fm.html

Conexiones para el dispositivo grhfico

La interfaz API 2D de Java recoge toda la configuración de los dispositivos de salida de tal forma que la podamos consultar durante la ejecución. De esta forma podremos crear programas que puedan solicitar su configuración y optimizarse a través de dicha configuración. Al modificar el modo de muestra, puede llegar a tomar el control del modo de video, escribiendo directamente en la pantalla.

Configuración de gráficos

La descripción del dispositivo de salida se encuentra dentro de la clase GraphicsDevice. Del dispositivo gráfico, podemos hacernos con el elemento GraphicsConfiguration para determinar el modelo de color (ColorModel) que se ha utilizado con el dispositivo de color. De esta forma podemos configurar las operaciones que se basan en el modelo de color. La mayoría de las veces, obtendremos el elemento GraphicsConfiguration del objeto Graphics2D que estamos utilizando para dibujar el objeto. Puede tratarse de un elemento Graphics2D que se le entrega al método paint() que se obtiene de BufferedImage o creado a partir de GraphicsEnvironment. El método printModelType() del programa que mostramos a continuación nos muestra cómo se puede determinar la técnica de optimización basada en el modelo de color. Ademfis, el modelo de color que utiliza el sistema difiere de los que utilizan los objetos BufferedImage.

Listado 11.16. Comprobación de modelos de color

```
import java.awt.*;
import java.awt.image.*;

public class GraphicsInfo extends ExitableJFrame {

    private void printModelType(ColorModel cm) {
        if (cm instanceof DirectColorModel) {
          System.out.println("DirectColorModel");
        } else if (cm instanceof IndexColorModel) {
          System.out.println("IndexColorModel");
        } else {
          System.out.println("Unknown ColorModel");
        }
    }
```

```
public void paint(Graphics g) {
    Graphics2D g2d = (Graphics2D)g;
    GraphicsConfiguration gc = g2d.getDeviceConfiguration();
    printModelType(gc.getColorModel());
    BufferedImage bi = new BufferedImage(20, 20,
    BufferedImage.TYPE_BYTE_INDEXED);
    Graphics2D g2d2 = bi.createGraphics();
    GraphicsConfiguration gc2 = g2d2.getDeviceConfiguration();
    printModelType(gc2.getColorModel());
    bi = new BufferedImage(20, 20, BufferedImage.TYPE_INT_ARGB);
    g2d2 = bi.createGraphics();
    gc2 = g2d2.getDeviceConfiguration();
    printModelType(gc2.getColorModel());
    bi = new BufferedImage(20, 20, BufferedImage.TYPE_USHORT_565_RGB);
    g2d2 = bi.createGraphics();
    gc2 = g2d2.getDeviceConfiguration();
    printModelType(gc2.getColorModel());


public static void main(String args[]) {
    Frame f = new GraphicsInfo();
    f.setTitle("GraphicsInfo");
    f.setSize(300, 250);
    f.show();
}
```

Además de contener la información del dispositivo, GraphicsEnvironment nos permite determinar las fuentes que se han instalado localmente, entre otras cosas. La tabla 11.7 enumera la composición de la clase.

Tabla 11.7. Composición de la clase GraphicsEnvironment

Variables/Métodos	Descripción
GraphicsEnvironment()	Crea un objeto `GraphicsEnvironment`.
`createGraphics()`	Crea un objeto `Graphics2D` para una `BufferedImage`.
`getAllFonts()`	Obtiene todas las fuentes del entorno.
`getAvailableFontFamilyNames()`	Obtiene los nombres de todas las familias de fuentes del entorno.
`getCenterPoint()`	Obtiene el centro del entorno de graficos; útil para centrar ventanas.
`getDefaultScreenDevice()`	Obtiene el `GraphicsDevice` por defecto.
`getLocalGraphicsEnvironment()`	Obtiene el `GraphicsEnvironment` actual.
`getMaximumWindowsBounds()`	Obtiene el máximo tamaño para una ventana, teniendo en cuenta los adornos del sistema operativo, como las barras de tareas.

Variables/Métodos	Descripción
`getScreenDevices()`	Obtiene todos los objetos `GraphicsDevice`.
`isHeadless()`	Comprueba si la pantalla/teclado/ratón son compatibles.
`isHeadlessInstance()`	Comprueba si el entorno especifico admite operaciones sin monitor.

Modos de muestra de gráficos

Aparte de obtener la `GraphicsConfiguration`, la clase `GraphicsDevice` también permite obtener o establecer el modo de muestra. Esto controla el tamaño de la pantalla (como 640 x 480), la profundidad de color y la velocidad de refresco y se define en la clase `DisplayMode`.

Hay dos maneras de obtener un `DisplayMode`. Puede crear uno directamente invocando a su constructor:

```
public DisplayMode(int width, int height, int bitDepth, int refreshRate)
```

Si no está seguro de la velocidad de refresco, use `DisplayMode.REFRESH_RATE_UNKNOWN (0)`.

También puede preguntar al `GraphicDevice` qué modos son compatibles. El programa del listado 11.17 hace precisamente eso. En primer lugar, debe preguntar a `GraphicsEnvironment` por el entorno local con `getLocalGraphicsEnvironment()`. A continuación, pregunte al entorno por el dispositivo de pantalla por defecto, con `getDefaultScreenDevice()`. Por ultimo, `getDisplayModes()` devuelve una matriz de todos los métodos.

Listado 11.17. Solicitar rmodos de rnuestra

```
import java.awt.*;

public class DisplayInfo {

    public static void main(String args[]) {
        GraphicsEnvironment graphicsEnvironment =
            GraphicsEnvironment.getLocalGraphicsEnvironment();
        GraphicsDevice graphicsDevice =
            graphicsEnvironment.getDefaultScreenDevice();
        DisplayMode displayModes[] = graphicsDevice.getDisplayModes();
        for (int i=0, n=displayModes.length; i < n; i++) {
            DisplayMode mode = displayModes[i];
            System.out.println(mode.getWidth() + "x" + mode.getHeight() +
                " \t" + mode.getRefreshRate() + " / " + mode.getBitDepth());
            }
        }
```

Antes de cambiar los modos de muestra, debe preguntar si admite el cambio. El mbtodo `isDisplayChangeSupported()` de `GraphicsDevice` devolverá `true` si lo admite. A continuación, una vez que sepa qué modo usar, invoque a `setDisplayMode()`. Cuando encuentre el modo perfecto, puede cambiar los modos con lo siguiente:

```
if (graphicsDevice.isDisplayChangeSupported()) {
   graphicsDevice.setDisplayMode(theMode);
}
```

Dibujo a pantalla completa

El concepto de dibujar pantallas completas es similar a la programación de DirectX bajo Microsoft Windows. El programador tiene control exclusivo sobre lo que se muestra. Probablemente el uso más comun sea cambiar a pantalla completa para los juegos, aunque no es el único.

Cambiar los modos de dibujo es tan simple como pasar la `Window` que representa la pantalla completa al mbtodo `setFullScreenWindow()` del actual `GraphicsDevice` (aunque la `Window` probablemente sea un `Frame`). Por supuesto, es necesario algo más. El listado 11.18 contiene código de configuración tipico para dibujar a pantalla completa:

Listado 11.18. Modificación de los modos de muestra para que admitan el dibujo a pantalla completa

```
GraphicsEnvironment graphicsEnvironment =
    GraphicsEnvironment.getLocalGraphicsEnvironment();
GraphicsDevice graphicsDevice =
    graphicsEnvironment.getDefaultScreenDevice();
DisplayMode displayModes[] = graphicsDevice.getDisplayModes();
DisplayMode originalDisplayMode = graphicsDevice.getDisplayMode();

try {
    Frame frame = new Frame();
    frame.setUndecorated(true);
    frame.setIgnoreRepaint(true);
    graphicsDevice.setFullScreenWindow(frame);
    if (graphicsDevice.isDisplayChangeSupported()) {
      graphicsDevice.setDisplayMode(displayModes[preferredMode]);

    ...
} finally {
    graphicsDevice.setDisplayMode(originalDisplayMode);
    graphicsDevice.setFullScreenWindow(null);

// Marco creado, de modo que el bucle del evento AWT siga ejecutándose
System.exit(0);
```

Una vez que estamos en pantalla completa, en lugar de llamar al mbtodo `paint()` cuando debamos actualizar la pantalla, los programadores deben crear su propio

bucle de dibujo. Aunque puede obtener el objeto `Graphics` del Window y hacer su propio doble buffer, el paquete `java.awt.image` proporciona una clase `BufferStrategy` que le permite aprovechar cualquier memoria intermedia procedente de hardware disponible en su sistema.

Para crear un `BufferStrategy`, indiquele a la ventana cuantos buffer desea, usando `createBufferStrategy()`. (Por cierto, no sólo puede crear dos buffer.) Tras crear la estrategia, la obtiene con `getBufferStrategy()` y hace un bucle hasta que termina de dibujar. En su bucle de dibujo, no alterne entre los buffer. En su lugar, lo hace la estrategia. Cuando solicite el contexto gráfico mediante `getDrawGraphics()`, la estrategia le proporciona lo siguiente que debe dibujar. A continuación, cuando haya que dibujar el buffer, invoque al método `show()` de `BufferStrategy`. Este es el aspecto del proceso:

```
frame.createBufferStrategy(2); // 2 buffer
BufferStrategy bufferStrategy = frame.getBufferStrategy();
while (!done()) {
    Graphics g = null;
    try {
       g = bufferStrategy.getDrawGraphics();
       // Dibuja en g
       bufferStrategy.show();
    } finally {
       if (g != null) {
         g.dispose();

    }
```

Hacer un doble buffer con `BufferStrategy` es algo distinto a lo habitual. El truco con `BufferStrategy` es que debe preguntarle ante si su buffer sigue siento válido o si tiene que dibujarlo todo de nuevo. Use `contentsLost()` para determinar si necesita dibujar de nuevo el buffer por completo o sólo los cambios desde el último uso. Ahora que ya ha recorrido los pasos para dibujar a pantalla completa, juntemos todas las piezas para mostrarlo en acción. A cada paso del bucle, se agrega otra linea a la pantalla para mostrar que permanece el antiguo buffer. Hay dos buffer, de modo que, esencialmente, necesitará dibujar la linea dos veces, una vez por cada buffer, pero no puede dibujar la linea una segunda vez hasta que el bucle vuelva a pasar, porque no tiene una referencia al segundo buffer. Si, en lugar de usar varios buffer, usa una `Image` para realizar el doble buffer, estará perdiendo los beneficios de una posible aceleración por hardware.

En lugar de codificar un modo de muestra especifico preferido, este programa (listado 11.19) intenta averiguar qué es lo mejor, de modo que haya la menor variación posible con el anterior código del listado 11.18. La figura 11.18 es el aspecto del programa en plena ejecución.

Listado 11.19. Demostración de dibujo a pantalla completa

```
import java.awt.*;
import java.awt.image.*;
```

```
public class FullScreen {

    private static int counter = 0;
    private static final int MAX = 50;
    private static DisplayMode MODES[] =
       new DisplayMode[] {
           new DisplayMode(640, 480, 32, 0),
           new DisplayMode(640, 480, 16, 0),
           new DisplayMode(640, 480, 8, 0)
       };

    private static DisplayMode getBestDisplayMode(GraphicsDevice device) {
       for (int x=0, xn=MODES.length; x < xn; xtt) {
           DisplayMode[] modes = device.getDisplayModes();
           for (int i=0, in=modes.length; i < in; itt) {
              if (modes[i].getWidth() == MODES[x].getWidth()
                && modes[i].getHeight() == MODES[x].getHeight()
                && modes[i].getBitDepth() == MODES[x].getBitDepth()
              ) {
                 return MODES[x];


           }
           return null;
       }

       public static void main(String args[]) {
           GraphicsEnvironment graphicsEnvironment =
              GraphicsEnvironment.getLocalGraphicsEnvironment();
           GraphicsDevice graphicsDevice =
              graphicsEnvironment.getDefaultScreenDevice();
          DisplayMode originalDisplayMode = graphicsDevice.getDisplayMode();

           try {
              Frame frame = new Frame();
           frame.setUndecorated(true);
           frame.setIgnoreRepaint(true);
           graphicsDevice.setFullScreenWindow(frame);
           if (graphicsDevice.isDisplayChangeSupported()) {
          graphicsDevice.setDisplayMode(getBestDisplayMode(graphicsDevice));
           }
           frame.createBufferStrategy(2); // 2 buffer
           Rectangle bounds = frame.getBounds();
           BufferStrategy bufferStrategy = frame.getBufferStrategy();
           while (!done()) {
              Graphics g = null;
              try {
                 g = bufferStrategy.getDrawGraphics();
                 if ((counter <= 2)) { // 2 buffers
                    g.setColor(Color.CYAN);
                    g.fillRect(0, 0, bounds.width, bounds.height);
                 }
                 g.setColor(Color.RED1;
               // dibuja de nuevo la línea anterior, también, desde 2 buffers
                 if (counter != 1) {
```

```
                    g.drawLine(counter-1, (counter-1)*5,
                       bounds.width, bounds.height);
                }
                g.drawLine(counter, counter*5, bounds.width,
bounds.height);
                bufferStrategy.show();
             } finally {
                if (g != null) {
                    g.dispose();


             try {
                Thread.sleep(250);
             } catch (InterruptedException ignored) {
          }
       }
    } finally {
       graphicsDevice.setDisplayMode(originalDisplayMode);
       graphicsDevice.setFullScreenWindow(null);
    }
    System.exit(0);
 }

 private static boolean done() {
    return (counter++ == MAX);
}
```

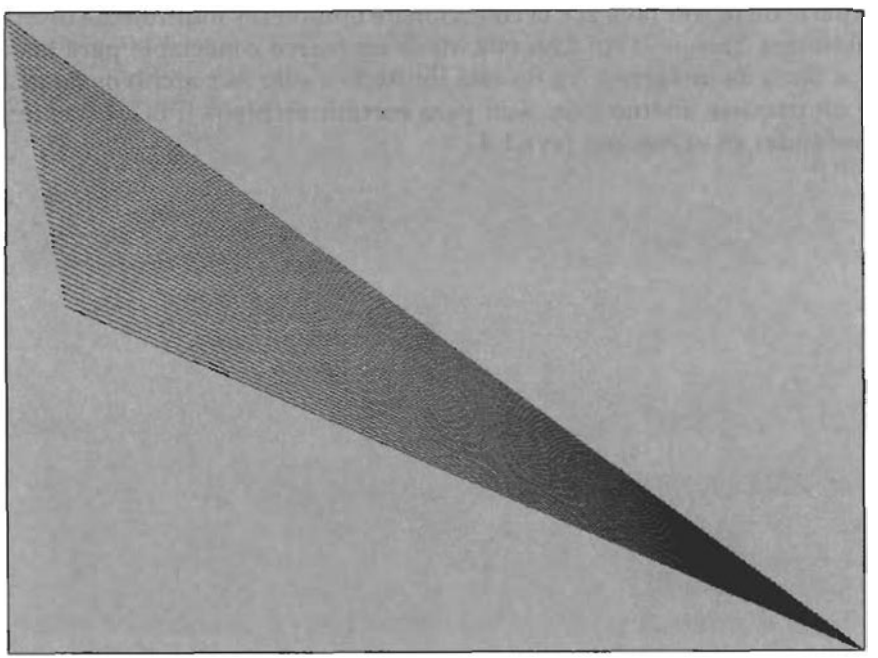


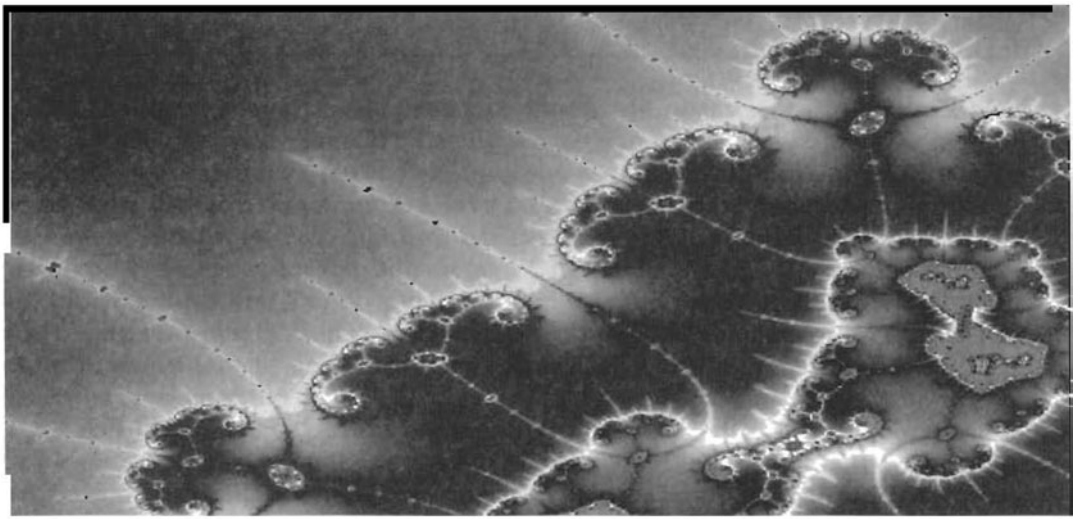
Figura 11.18. Dibujo a pantalla completa

Resumen

Esencialmente, Java proporciona dos modos de operaciones graficas. Esta el modo basico y el modo Java 2D. Las funciones que ofrece la API Java 2D superan con creces las disponibles sin el. Por supuesto, usar la API significa que los usuarios deben estar usando, como minimo, Java 1.2 o posterior para poder usar algunas de las nuevas funciones. Esto significa básicamente que los applets que usan el JRE nativo para el navegador (basado en Java 1.1.x para Netscape Communicator 4.x e Internet Explorer 4.x/5.x) no pueden usar estas funciones graficas. Si realmente quiere que sus applets usen la API Java 2D u otras de las funciones descritas en este capitulo, considere la posibilidad de emplear el Java Plug-in o Java Web Start, descritos en el capitulo 18.

La API Java 2D hace posible algunas aplicaciones graficas muy elaboradas. Incluye ayuda para crear cualquier objeto Shape para dibujar con un `stroke` personalizado o para rellenar la forma con un trabajo `Paint` personalizado. Asi pues, cuando necesita rotar, trasladar o recortar un objeto, puede crear un `AffineTransform`. Las imagenes en memoria se controlan con un `BufferedImage`. Puede mezclar estas imagenes con una de las reglas Porter-Duff y un `AlphaComposite`. O puede filtrarla con `BufferedImageOp`. A continuación, cuando llegue el momento de tomar el control de toda la zona de muestra, ajuste la pantalla a modo de pantalla completa, escoja un `DisplayMode`, use un `BufferStrategy` y use la API Java 2D para escribir, porque tiene el control absoluto.

Aparte de la API Java 2D, la coleccion de bibliotecas multimedia de SDK incluye la biblioteca `Image I/O`. Con ella, tiene un marco conectable para leer y escribir varios tipos de imagenes. Ya no está limitado a sólo leer archivos de imagenes o a usar un paquete interno `com.sun` para escribir archivos JPEG. Estas funciones ya son estándar en el entorno Java 1.4.



12
Administración de un diseño GUI

Hay tres aspectos a tener en cuenta a la hora de crear interfaces de usuario (GUI): los componentes que componen las pantallas, el control de eventos que responde a la interacción del usuario con los componentes y la posición de los componentes para elaborar la pantalla. Estos tres tópicos se estudiarán en los próximos capitulos.

Posicion de Java respecto a GUI

Para algunos programadores, la administración de disefio es un concepto extrafio. Si solamente han programado para una plataforma, no han tenido que enfrentarse al problema de conseguir pantallas que tengan buen aspecto en diferentes plataformas. No es sólo el aspecto que tendra el programa en un cuadro de Sun Solaris o de Apple Macintosh si se supone que el programa sólo se va a ejecutar en una plataforma Windows. (Sin tener en cuenta las pequefias diferencias existentes entre Windows 95, 98, NT, 2000, Me y CE.) Sin embargo, con Java, el factor clave es que los programas deben correr igualmente en todas las plataformas. Debido a este requisito, la administración de disefio se convierte en un factor importante para el programador. Los componentes simplemente no tienen el mismo aspecto en todas las plataformas; ni tienen el mismo estado real de pantalla. Controlar los principios de esta administración de diseño es el trabajo de los administradores de disefio.

Antes de pasar a los principios de la administración de disefio, volvamos a su relación con los componentes de Java. Las bibliotecas de Java proporcionan dos

conjuntos de componentes. Uno de ellos se llama AWT, por Abstract Window Toolkit y la otra se llama Swing. Estas dos tecnologías son parte del marco Java Foundation Component (JFC).

Como se mencionó en el capítulo 9, AWT es el conjunto de componentes de primera generación; Swing es la segunda generación. Usar componentes AWT o Swing es algo fácil de decidir. Para las nuevas aplicaciones, use el conjunto de componentes Swing. Swing se basa en AWT, por lo que, indirectamente, sigue usando AWT con Swing, pero los componentes Swing son mucho más elaborados que sus predecesores. Con applets, la respuesta es un poco más compleja y se tratará en el capítulo 18, junto con Java Plug-in. Los programadores no suelen usar Swing en los applets porque requiere un entorno de ejecución compatible, algo que no suele ocurrir para las plataformas de ejecución incrustadas en la mayoría de los navegadores Web.

Por cierto, todos los ejemplos de diseño de este capítulo se basan en aplicaciones y usan el conjunto de componentes Swing.

Nota

JFC (Java Foundation Component) contiene muchos más elementos aparte de AWT y Swing. El marco también incluye el API Java 2D (capítulo 10), las funciones arrastrar y colocar (capítulo 16), funciones de impresión (capítulo 20), ayuda para la internacionalización (capítulo 17) y la API de accesibilidad (capítulo 13).

Los componentes AWT y Swing se basan en los mismos principios de administración de diseño:

- Con componentes AWT, la administración de diseño es importante porque los componentes AWT son parecidos a los componentes del sistema operativo nativo. Si crea un botón AWT y el usuario está trabajando en Macintosh, el botón parecerá un botón nativo MacOS. En el mismo programa sobre un cuadro Solaris, ese botón tendrá el aspecto de un botón Motif.
- Los componentes Swing son algo diferentes. En lugar de tener siempre la apariencia del conjunto de componentes nativos del usuario, Swing admite una arquitectura de aspecto y comportamiento conectable (PLAF). Todo lo asociado a un componente se dibuja y controla desde dentro de Java, de modo que puede cambiar el aspecto y comportamiento de un componente en cualquier momento.

En ambos casos, (AWT y Swing) el tamaño del componente específico puede resultar desconocido hasta el tiempo de ejecución, de modo que no puede usar coordenadas absolutas para colocar los componentes. Y aquí es donde entra la administración de diseño.

Diseño de GUI con administradores de diseño

Para explicar cdmo funciona el administrador de diseiio, necesitaremos usar componentes reales. Los componentes de Java son los controles con los que interaccionan los usuarios, como botones, mends, barras de desplazamiento y campos de texto. Para explicar y mostrar las diferentes clases administradoras, usaremos un sencillo botón. El botón será pasivo, de modo que si lo seleccionamos no sucederá nada.

Las clases cliente Container y Component

Antes de explorar los distintos sistemas de programacidn, tendremos que comprender cdmo interactuan las clases de administración de disefio con sus clases cliente. Estas clases son `container` y `Component`.

Estudiaremos estos elementos en una sencilla aplicacidn, `FlowLayoutTest` (listado **12.1**). Introduzca este programa o cópielo desde el CD que acompafia a este libro.

Listado 12.1. Mostrar una pantalla con FlowLayout

```
import java.awt.*;
import javax.swing.*;
public class FlowLayoutTest {
    public static void main(String args[]) {
        JFrame frame = new ExitableJFrame("Flow Layout");
        Container contentPane = frame.getContentPane();
        contentPane.setLayout(new FlowLayout());
        contentPane.add(new JButton("First"));
        contentPane.add(new JButton("Second"));
        contentPane.add(new JButton("Third"));
        contentPane.add(new JButton("Fourth"));
        contentPane.add(new JButton("Fifth"));
        frame.setSize(200, 200);
        frame.show();
    }
}
```

Esta clase implementa una sencilla aplicacidn con cinco botones colocados segdn el estilo indicado por el administrador de disefio `FlowLayout`. La figura **12.1** muestra un ejemplo de este programa usando este diseiio.

Nota

El ejemplo no incluye ningun método `paint()` porque no lo necesita. Los elementos GUI se dibujan de nuevo automaticamente.

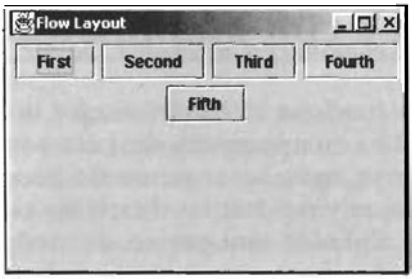


Figura 12.1. Un programa usando FlowLayout

El componente JFrame

La primera linea de `main()`:

```
JFrame frame = new ExitableJFrame("Flow Layout");
```

crea un `ExitableJFrame`. Esta clase se present6 en el capítulo 10 y proporciona un **`JFrame`** que se cerrará cuando haga clic en la decoración de la ventana. (Es la pequeiia X en la parte superior derecha de la figura 12.1, pero este elemento decorativo es especifico de la plataforma.) Si va a ser directamente una subclase de **`JFrame`**, necesitará agregar lo siguiente a todos los ejemplos de este capitulo:

```
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

El **`JFrame`** se compone de varios componentes internos. Cuando quiera hacer algo con el marco, no trabajará normalmente con el marco de forma directa, sino con los componentes internos. Sólo deberá conocer uno de ellos (el cuadro de contenido) para este ejemplo, pero pronto los veremos todos para que se haga una idea de lo que está aprendiendo.

La figura 12.2 muestra todos los entresijos de un **`JFrame`**. Lo primero que observará es la disposición.

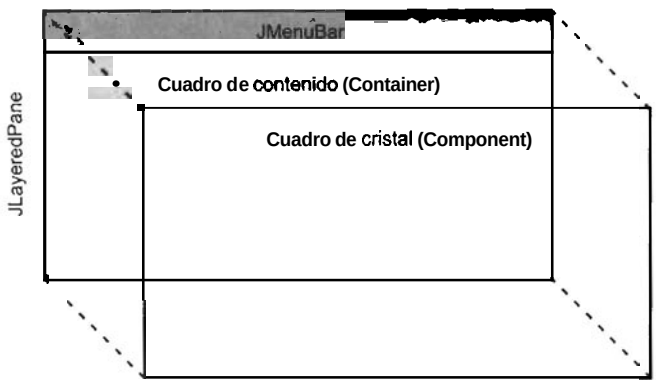


Figura 12.2. Jframe y su cuadro raiz

El cuadro de cristal proporciona una zona para que aparezcan ciertos objetos sobre los objetos normales (para operaciones de arrastrar y soltar, por ejemplo). También hay una `JMenuBar` opcional para el men6 del marco. Aunque por ahora, nos concentraremos en el cuadro de contenido. Adquirida mediante el método `getContentPane()`, esto devuelve un `Container` con el que trabajar. Aquí establecerd el administrador de diseño y agregard componentes.

Nota

La clase AWT `Frame`, que es la clase rnatriz de `JFrame`, no tiene la estructura de contenido. Trabajará directarnente con el marco y no con el cuadro de contenido.

El componente Container

Con la siguiente línea del método `main()`, la aplicación invoca al método `setLayout()` del cuadro de contenido:

```
Container contentPane = frame.getContentPane();
contentPane.setLayout(new FlowLayout());
```

Esto le permite escoger uno de los estilos de diseiio disponibles para los componentes GUI que contendrá el cuadro de contenido (en este caso, los cinco botones agregados inmediatamente después de la instrucción `setLayout()`).

Este método es uno de los muchos disponibles desde la clase `container`, enumerados en la tabla 12.1. (Por ahora, no se preocupe por los detalles de todos los métodos de la tabla. Veremos los métodos especificos a medida que aprende nuevas funciones.)

Tabla 12.1. Composición de la clase Container

Variables/Métodos	Descripción
`Container()`	Crea un objeto `Container`.
`add()`	Agrega un componente al contenedor
`addContainerListener()`	Agrega el oyente adecuado .
`addPropertyChangeListener()`	
`addImpl()`	Método de ayuda para agregar un componente.
`addNotify()`	Anexiona un contenedor a un componente nativo.
`applyComponentOrientation()`	Cambia la orientación de los componentes de un contenedor.

Variables/Métodos	Descripción
areFocusTraversalKeysSet()	Comprueba si se han establecido claves negativas de foco.
doLayout()	Distribuye los componentes de un contenedor.
findComponentAt()	Encuentra el componente en la posición especificada del contenedor.
getAlignmentX()	Obtiene el alinearniento en el eje.
getAlignmentY()	
getComponent()	Obtiene los componentes del contenedor.
getComponentAt()	
getComponents()	
getComponentCount()	Obtiene el número decornponentes del contenedor.
getContainerListeners()	Obtiene el conjunto de oyentes.
getListeners()	
getFocusTraversalKeys()	Obtiene las claves negativas de foco actuales.
getInsets()	Obtiene los desplazamientos a la derecha del contenedor.
getLayout()	Obtiene el adrninistrador de disefio.
getMaximumSize()	Obtiene el tamaño maximo para el contenedor.
getMinimumSize()	Obtiene el tamaño minimo para el contenedor.
getPreferredSize()	Obtiene el tamaño predeterminado para el contenedor.
invalidate()	Invalida la muestra del contenedor.
isAncestorOf()	Cornprueba si el cornponente es el creador del contenedor.
isFocusCycleRoot()	Cornprueba el valor de la raiz de foco de ciclo.
isFocusTraversalPolicySet()	Comprueba si se ha establecido una norrna negativa de foco.
list()	Elirnina los contenidos del contenedor.
paint()	Pinta el contenedor y sus componentes.
paintComponents()	
paramString()	Crea una cadena que representa un contenedor que puede rnostrar `tostring()`.
print()printComponents()	Escribe el contenedor y sus componentes

Variables/Métodos	Descripción
`processContainerEvent()`	Procesa los eventos de contenedor internos
`processEvent()`	
`remove()`	Elimina un componente del contenedor.
`removeAll()`	Elimina todos los componentes.
`removeContainerListener()`	Elimina el oyente del contenedor.
`removeNotify()`	Elimina la conexión con el componente nativo.
`setFocusCycleRoot()`	Cambia el estado de la raiz de ciclo de foco.
`setFocusTraversalKeys()`	Cambia las claves para el foco negativo.
`setFocusTraversalPolicy()`	Cambia la norma para el foco negativo.
`setFont()`	Cambia la fuente.
`setLayout()`	Cambia el administrador de disetio.
`transferFocusBackward()`	Mueve el foco hacia atras.
`transferFocusDownCycle()`	Se mueve al ciclo de foco para el contenedor interno.
`update()`	Dibuja de nuevo la muestra del contenedor.
`validate()`	Valida el contenedor y sus componentes.
`validateTree()`	Calcula de nuevo el disetio para los contenedores internos y los componentes contenidos.

Al identificar el origen del método `setLayout()` hemos dado nuestro primer paso en el reconocimiento de la declaración con la que vamos a trabajar. Ahora, la siguiente cuestión que tenemos que aclarar es: ¿por qué este método se define dentro de `container`? La respuesta es muy sencilla y nos lleva al asunto principal de este capitulo, los administradores de diseño. Estos elementos están asociados con todos los contenedores porque son los encargados de diseñar sus componentes. La clase `container` nos permite agrupar los elementos de la interfaz GUI que estén relacionados. Esto nos lleva a la siguiente afirmación: el segundo método de cualquier contenedor es `add()` (el mismo que se utilizó con el ejemplo del applet de los cinco botones).

Vamos a resumir lo que sabemos hasta ahora de los contenedores y los componentes; son puntos clave para comprender este y cualquier programa AWT/Swing:

- Todos los contenedores tienen su propio administrador de diseiio.
- Los componentes individuales se agrupan en contenedores.
- Todos los marcos, ventanas, cuadros de diálogo y applet son contenedores.

Cuando el código de un applet invoca a su método add(), lo hace como contenedor. En la clase container se definen cinco metodos add() diferentes.

```
public Component add(Component comp)
public Component add(Component comp, int index)
public void add(Component comp, Object constraints)
public void add(Component comp, Object constraints, int index
public void add(String constraints, Component comp)
```

Todos estos métodos aiiaden un nuevo elemento GUI al contenedor (el applet o la ventana de la aplicación). El tipo Component que se especifica en estos métodos es la clase principal de todos los elementos de los paquetes java.awt y javax.swing.

Cuando el método AWT solicita un parámetro del tipo Component, quiere decir que necesitará cualquier subclase de la clase Component: un JButton, un JPanel, un JComponent, un JTextField, un JScrollBar, etc. Se parece a todos los métodos de Java que especifican un argumento del tipo Object, cuando en realidad, quieren decir "cualquier tipo de objeto". En ambos casos, estos métodos se basan en la compatibilidad que tienen las subclases de una clase con su principal. Por ejemplo, un botón JButton es un Component. Del mismo modo, String es un Objet.

Truco

Siempre que veamos que se especifica el tipo Component **dentro de una** declaración **de un** método **AWT, tendremos que** sustituirla **mentalmente** por **cualquiera de los** JComponent **Swing. Para la mayor** parte, **la diferencia entre el nombre de la clase AWT y el nombre de la clase Swing sera que esta** última **tendra una J antes de la clase componente AWT equivalente.**

La clase Container se oculta de las subclases de Component, es decir, que a un contenedor se puede añadir (con la clase add()) mas de un subcontenedor, porque es una subclase de Component. Es muy importante comprender la capacidad de anidación porque se suele utilizar para permitir que nuestros diseños GUI sean mas modulares y, por lo tanto, mas flexibles.

Al anidar contenedores, cada uno de ellos con su propio diseiio, nos dan la potencia que nos permite organizar nuestro interfaz GUI de una forma lógica y más agradable de utilizar.

Truce

Cuando exploremos los administradores de diseño, **hemos de tener presente que no se suele utilizar un único administrador para toda la interfaz GUI. Casi siempre se** utilizan **en conjunto.**

Vamos a echar otra mirada a las cinco variantes del método `add()`. El programa utilizarh la primera, `public Component add(Component comp)` para afiadir algunos botones. El resto nos permite ubicar componentes en puntos determinados, para lo cual tendremos que suministrar la posición deseada en forma de argumento. El segundo método `add()` necesita un argumento de posición numérico, el siguiente utilizar `Object` como limitador de posición, el cuarto utiliza argumentos `Object` o numéricos y el último toma la posición como una etiqueta en forma de cadena (un limitador).

Para demostrar el método `add(Component comp, int index)`, modificaremos nuestro applet para que tenga la siguiente forma:

```
contentPane.add(new JButton("First"));
contentPane.add(new JButton("Second"));
contentPane.add(new JButton("Third"));
contentPane.add(new JButton("Fourth"));
contentPane.add(new JButton("Fifth", 2));
```

Si ejecutamos el applet, veremos que hemos cambiado el orden de los botones: **First, Second, Fifth, Third, Fourth.** El argumento extra 2 del último método `add()` es el encargado de insertar el quinto botón entre el segundo y el tercero.

Ahora que comprende el cómo y el por qué de la configuración básica del GUI del programa, podemos concentrarnos en los siete estilos de diserio de los componentes primarios proporcionados. Estos estilos, `FlowLayout`, `BorderLayout`, `CardLayout`, `GridLayout`, `BoxLayout`, `SpringLayout` y `GridBagLayout` se describirhn a continuación. Ademhs, estudiaremos los tres componentes `swing` que le facilitarán la tarea de colocar componentes: `JTabbedPane`, `JSplitPane` y Box.

Nota

`ScrollPaneLayout`, `ViewportLayout` y `OverlayLayout` se encuentran dentro del paquete `javax.swing`. De todas formas, los desarrolladores apenas las utilizan directarnente. En realidad son varios los componentes de AWT los que la usan de forma interna.

FlowLayout

El administrador `FlowLayout` es el más sencillo de todos los de AWT. Es el encargado del flujo de los elementos de la interfaz GUI. El sisterna de trabajo que desarrolla se parece a la forma en que un procesador de texto ordena las palabras de un párrafo. Cuando se alcanza el final de la linea dependiendo de la anchura del contenedor, los componentes se ajustan igual que el ajuste de texto en un procesador de palabras.

Este comportamiento puede observarse y comprenderse mejor ejecutando el prograrna del listado 12.2. El programa añade un botón nuevo cada vez y dibuja de

nuevo la distribución de estos elementos dentro de la ventana. De esta forma, podemos ver exactamente el efecto que tiene este administrador de diserio sobre la ubicación de los botones.

La figura 12.3 muestra la ventana del programa cuando ha realizado la tercera parte de sus bucles.

Listado 12.2. Agregar botones dinamicamente a FlowLayout

```
import java.awt.*;
import javax.swing.*;
public class FlowingButtons {
    public static void main(String args[]) {
        JFrame frame = new ExitableJFrame("Flowing Buttons");
        Container contentPane = frame.getContentPane();
        contentPane.setLayout(new FlowLayout());
        frame.setSize(200, 200);
        frame.show();
        for (int i = 0; i < 15; i++) {
            contentPane.add(new JButton ("" + i));
            contentPane.validate();
            try {
              Thread.sleep(500);
            } catch (Exception e) {
            }
            System.out.println(i);
        }
    }
```

Figura 12.3. Un administrador de diseño FlowLayout centrado

Normalmente, la llamada al método `validate()` no es necesaria. Sin embargo, si se agregan o eliminan componentes tras haberlos mostrado en pantalla, debe validar la muestra. Lo que indica al programa es que se asegure de que lo que aparece en pantalla es correcto.

El programa del listado 12.2 usa el constructor de objeto `FlowLayout` más sencillo posible: el que no toma argumentos. Pero `FlowLayout` se puede personalizar en cierta manera especificando un estilo de alineación opcional.

```
public FlowLayout(int align)
```

El pardmetro de alineación debe ser una de estas constantes de la clase FlowLayout: LEFT, CENTER, RIGHT, LEADING o TRAILING. El constructor usado por defecto hasta ahora es en realidad una abreviatura de

```
new FlowLayout(FlowLayout.CENTER)
```

Es el responsable de que el programa centre los botones de cada una de las líneas. Podemos familiarizarnos con los elementos de otras alineaciones sustituyendo la línea:

```
contentPane.setLayout(new FlowLayout());
```

con esta otra:

```
contentPane.setLayout(new FlowLayout(FlowLayout.RIGHT));
```

y una vez más con FlowLayout.LEFT, FlowLayout.LEADING y FlowLayout.TRAILING.

Nota

Casi nunca usara LEADING y TRAILING a menos que sus programas sean internacionales. Estos parametros estan diseñados para controlar mejor las orientaciones de los componentes de algunos idiomas aparte de la tradicional de izquierda a derecha. Si la orientación del contenedor no es ComponentOrientation.LEFT_TO_RIGHT (pordefecto), entonces las constantes LEFT y RIGHT tendrán el comportamiento opuesto, mientras LEADING y TRAILING llevaran a los componentes hacia el borde izquierdo o derecho, no importa cual sea la orientación.

El tercer constructor FlowLayout nos permite especificar el espacio entre componentes, midiéndolo en pixeles. Este espacio puede ser tanto vertical como horizontal

```
public FlowLayout(int align, int hgap, int vgap)
```

A diferencia de la mayoría de las clases que utilizamos al escribir programas en Java, las clases de los administradores de diseiio no se invocan de una forma directa (aparte de sus distintos constructores). De esta forma simplificamos considerablemente su utilización. Con el método setLayout(), todo lo que tenemos que hacer es coger uno de los administradores ya existentes.

Nota

Insets es una funcion de diseño que no es un administrador de diseño y aun así afecta a todos los estilos de diseño. Cada contenedor puede definir un borde

para que permanezca sin componentes. Este borde recibe el nombre de inset del contenedor y se representa mediante la clase `java.awt.Insets`. Sin embargo, los componentes `Swing` se basan en `javax.swing.Border` para definir los insets.

BorderLayout

El segundo administrador de diseño que vamos a ver no es tan simple como el anterior. La clase `BorderLayout` trabaja con un diseiio que se divide en cinco zonas, como se puede ver en la figura 12.4. Es el administrador de diseiio predeterminado que utilizan las clases `Frame` y `Applet` (en el caso de las subclases de `Applet`, lo utilizarán cuando trabajan con componentes Swing). Las zonas reflejan la típica distribución de componentes alrededor de una ventana. Las cinco zonas que aparecen en la figura 12.4 se rellenan con el mismo tipo de botones que los utilizados con el ejeinplo `FlowLayout`. El listado 12.3 es el programa que creó el resultado de la figura 12.4.

Listado 12.3. Disposición de una pantalla con BorderLayout

```
import java.awt.*;
import javax.swing.*;
public class BorderLayoutTest {
    public static void main(String args[]) {
        JFrame frame = new ExitableJFrame("Border Layout");
        Container contentPane = frame.getContentPane();
        contentPane.setLayout(new BorderLayout(10, 55));
        contentPane.add(new JButton("North"), BorderLayout.NORTH);
        contentPane.add(new JButton("South"), BorderLayout.SOUTH);
        contentPane.add(new JButton("East"), BorderLayout.EAST);
        contentPane.add(new JButton("West"), BorderLayout.WEST);
        contentPane.add(new JButton("Center"), BorderLayout.CENTER);
        frame.setSize(300, 200);
        frame.show();
```

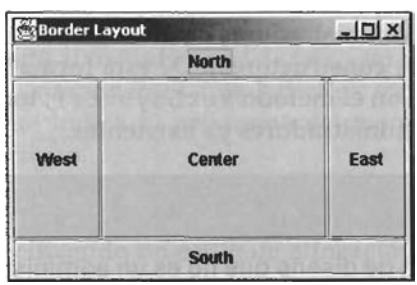


Figura **12.4.** Un BorderLayout

Como podemos ver, BorderLayout ajusta el tamaiio de sus componentes para que ocupen todo el espacio disponible en la ventana. Las reglas son las siguientes:

Los componentes en las zonas del sur y del norte ajustaran su tamaño al ancho del contenedor y a la altura preferida del componente especifico, a partir de getPreferredSize().

Los componentes en las zonas del este y del oeste ajustaran su tamaño a la altura restante del contenedor y a la anchura preferida del componente especifico, a partir de getPreferredSize().

- Los componentes del centro obtienen el espacio restante.

Todas las zonas son opcionales.

Nota

Al igual que FlowLayout, BorderLayout **tiene constantes para manejar los componentes para orientacion de idiomas. Para lenguajes con orientacion de izquierda a derecha y de arriba a abajo, el equivalente de** NORTH **es** PAGE_START **(y** BEFORE_FIRST_LINE**). Para** SOUTH **tiene** PAGE_END **y** AFTER_LAST_LINE**. Sinónimos para** EAST **son** LINE_START **y** BEFORE_LINE_BEGINS**. Las adiciones a WEST son** LINE_END **y** AFTER_LINE_ENDS**. Si el** ComponentOrientation **del contenedor es** RIGHT_TO_LEFT**, la dirección se invierte.**

Vamos a ver qué diferencias tenemos entre los ejemplos BorderLayout y FlowLayout. Por un lado tenemos la no selección del diseño BorderLayout ya que es el diseiio por defecto para el cuadro de contenido de un JFrame. En segundo lugar, el método add() que utilizamos con BorderLayout para agregar componentes al contenedor no es tan sencillo como add(Component comp), tenemosque utilizar la variante add(Component comp, Objet contraints). Como podemos ver en el código, la posición se determina a través del nombre en vez de hacerlo a través de valores numéricos. De esta forma se facilita la legibilidad del código.

BorderLayout únicamente puede trabajar con cinco componentes. Cuatro de ellos se colocarán en los bordes de la ventana y el quinto en el centro, ocupando la mayor parte de la ventana. Cuando utilizamos el administrador de diseiio BorderLayout, no tenemos que usar obligatoriamente los cinco componentes. En la figura 12.5 podemos ver un ejemplo donde no se utiliza el componente del centro.

Como podemos ver, los cuatro componentes de los bordes no han modificado su tamaiio para rellenar el tamaiio del componente que falta. Sólo lo harán para rellenar el espacio que deje alguno de los bordes. Aunque los componentes BorderLayout tienen muchas combinaciones, los ejemplos de la figura 12.6 deben

bastar para hacernos una idea de cómo `BorderLayout` manipula los componentes cuando no se especifican todos ellos.

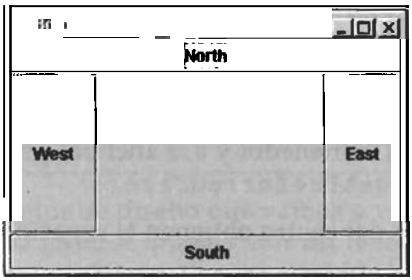
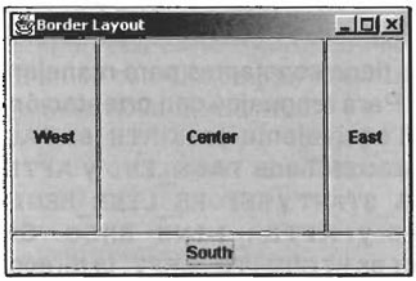
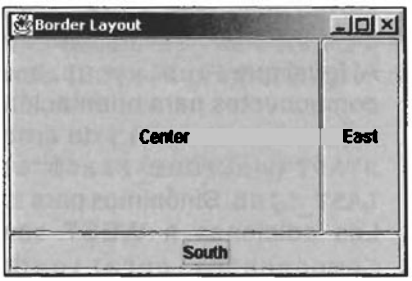


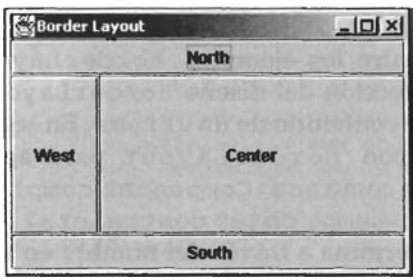
Figura 12.5. Un BorderLayout al que le falta el cornponente del centro



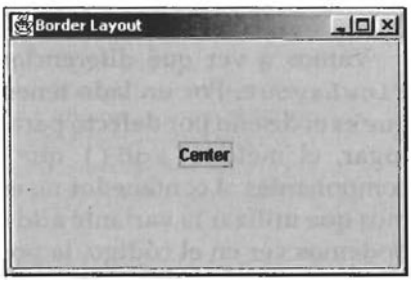
Sin el cornponente "North"



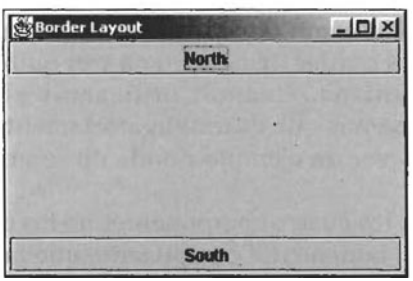
Sin los cornponentes "North" y "West"



Sin el cornponente "East"



Sólo el cornponente "Center"



Sin los cornponentes Center", "Center". "East" y "West"

Figura 12.6. Un subconjunto de combinaciones de cornponentes BorderLayout

Advertencia

Además de las limitaciones propias de la clase, podemos utilizar cadenas con `BorderLayout`. Son las siguientes (respetando las mayúsculas y minúsculas): `"North"`, `"South"`, `"East"`, `"West"`, `"center"`, `"First"`, `"Last"`, `"Before"` y `"After"`. Por ejemplo, si escribimos `"NORTH"` o `"north"` en vez de `"North"`, no obtendremos ningun resultado. Es mas, como el compilador detectara un error, prescindira de él, con lo que el resultado no sera el esperado. El errorque se genera al escribir estas palabras cambiando mayusculas por minusculas (o viceversa) es `IllegalArgumentException`. Por eso, lo mas seguro es trabajar con las constantes de la clase.

Al igual que ocurria con el administrador de diseiio `FlowLayout`, una variante de `BorderLayout` nos permitirá especificar el espacio vertical y horizontal que se tendrá que utilizar. Sustituimos la linea `setLatout()` por la siguiente:

```
contentPane.setLayout(new BorderLayout(10, 55));
```

La figura 12.7 muestra el diseño resultante de este cambio. Se ha insertado un espacio de 10 pixeles entre componentes separados horizontalmente y se ha usado un espacio de 55 pixeles para separar los componentes verticalmente.

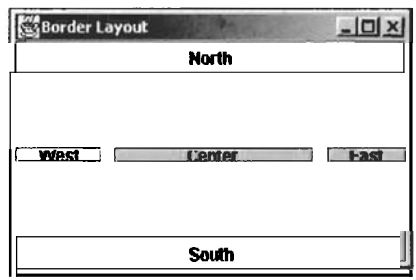


Figura 12.7. Un BorderLayout poco atractivo conseguido al especificar espacios demasiado grandes

Truco

Se pueden evitar problemas de diseño como los que aparecen en la figura 12.07. Todo lo que tenemos que hacer es evitar trabajar con distancias y dimensiones absolutas.

CardLayout

CardLayout no es exactamente un administrador de disefio. En realidad se limita a definir el numero de tarjetas que contiene (generalmente una relación lógi-

ca de componentes). En la figura 12.8 tenemos una ventana basada en tarjetas (no se ha implementado utilizando Java, pero hacerlo no tendría ninguna dificultad). CardLayout le proporciona todo lo que se encuentra bajo las "fichas" de las tarjetas. Si esta ventana hubiese empleado CardLayout, tendría que diseiiar el modo de intercambiar tarjetas.

Nota

Otros marcos de trabajo GUI podrian utilizar el termino "pagina" o "pestaña" en vez de tarjeta.

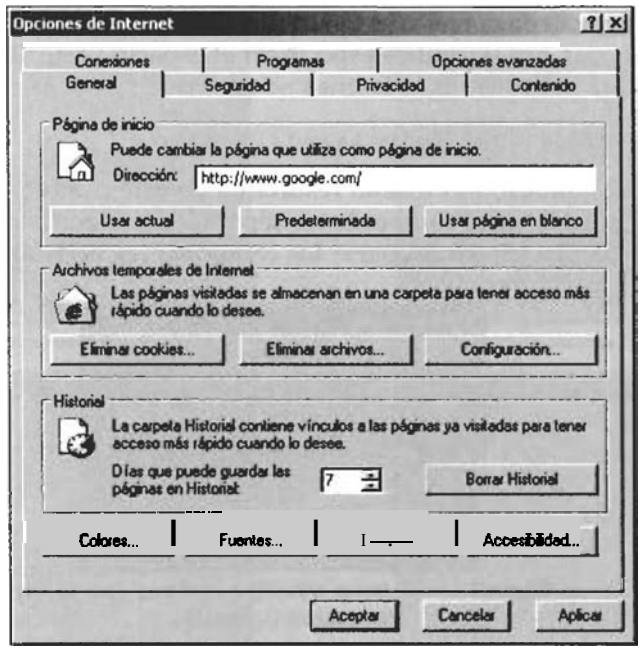


Figura 12.8. Tipico GUI basado en tarjetas

El siguiente programa muestra el uso de la CardLayout para crear una ventana con cinco tarjetas que contienen un único botón cada una:

Listado 12.4. Disposición de una pantalla con CardLayout

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class CardLayoutTest {
    public static void main(String args[]) {
        JFrame frame = new ExitableJFrame("Card Layout");
```

```
        final Container contentPane = frame.getContentPane();
        final CardLayout layout = new CardLayout();
        contentPane.setLayout(layout);
        ActionListener listener = new ActionListener() {
             public void actionPerformed(ActionEvent e) {
                 layout.next(contentPane);
             }
        };
        for (int i=0; i<5; i++) {
           String label = "Card " + i;
           JButton button = new JButton(label);
           contentPane.add(button, label);
           button.addActionListener(listener);

        frame.setSize(300, 200);
        frame.show();
    }
}
```

Nota

La aplicacion del ejemplo utiliza los metodos `addActionListener()` **y** `actionPerformed()` **para controlar los eventos.**

Observe que el método `add()` usado para agregar tarjetas a un `CardLayout`. Al igual que con los diseños de borde, debe usar las diferentes limitaciones para pasar el nombre de tarjeta (además del nombre de la tarjeta) al administrador de diseiio. Mas tarde, podremos hacer referencia a las tarjetas por su nombre en lugar de tener que indexarlas por número, por ejemplo. Observe también que se ha creado una variable `CardLayout`. En los anteriores ejemplos de administrador de diseiio, simplemente tendra un nuevo `SomeLayoutManager()` entre los corchetes de la invocation `setLayout()`. Pero en este ejemplo, necesitará hacer referencia al objeto `CardLayout` después de crearlo (por eso necesitamos que la variable le haga referencia).

Para demostrar los ciclos a lo largo de las tarjetas, el programa del listado 12.4 se creó de modo que estuviera a la espera de acciones de cualquiera de los botones. La siguiente tarjeta se mostrará cada vez que se haga clic en uno de los botones. El método `next()` indica a la aplicación que muestre la siguiente tarjeta bajo su control para el contenedor.

Cuando llega a la última tarjeta y le pide que muestre la tarjeta `next()` de nuevo, simplemente repite el ciclo desde el principio. La figura 12.9 muestra el marco creado por el programa.

Es posible que esperase encontrar un cuadro donde se viesen las cinco pestaiias (como el de la figura 12.8). Por desgracia, `CardLayout` no realiza esto. Debe implementar las fichas y los perfiles de las tarjetas o usar el contenedor `JTabbedPane`. El listado 12.5 amplía lo visto en el ejemplo anterior para mostrarnos cómo funciona `JTabbedPane`.

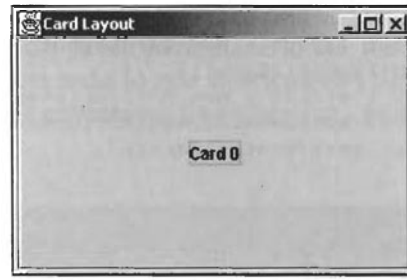


Figura 12.9. Diseño por defecto de CardLayout

Listado 12.5. Disposición de una pantalla con JTabbedPane

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class TabbedPaneTest {
    public static void main(String args[]) {
        JFrame frame = new ExitableJFrame("JTabbedPane");
        final Container contentPane = frame.getContentPane();
        JTabbedPane jtp = new JTabbedPane();
        contentPane.add(jtp, BorderLayout.CENTER);
        for (int i=0; i<5; i++) {
            JButton button = new JButton("Card " + i);
            jtp.add("Btn " + i, button);
        }
        frame.setSize(300, 200);
        frame.show();
    }
```

Cada tarjeta se agrega con el método `addTab()` en lugar de con el método habitual `add()`. Observe también que el código controlador de eventos para moverse de una carta a otra se realiza automaticamente mediante `JTabbedPane`. Por este valor que se ha agregado a `JTabbedPane` nos encontraremos con que apenas trabajaremos con `CardLayout`. En la figura 12.10 tenemos la salida que genera esta versión mejorada del programa.

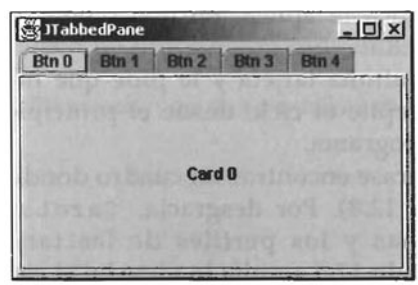


Figura 12.10. Empleamos JTabbedPane en lugar de CardLayout

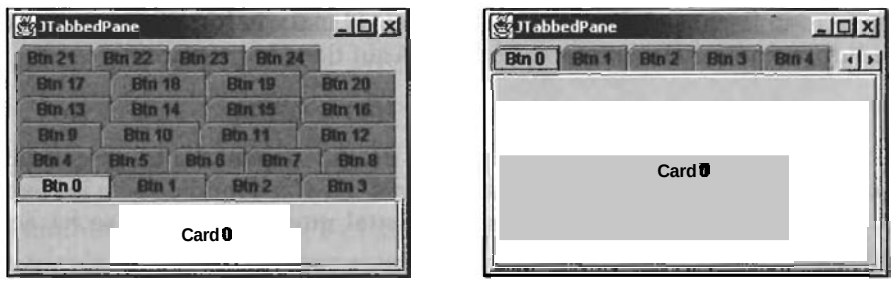
`JTabbedPane` tiene muchas opciones adicionales. Por ejemplo, puede mostrar imágenes en las pestadas, mediante

```
addTab(String title, Icon icon, Component component)
```

y mostrar las ayudas de herramienta que aparecen cuando se coloca el ratón sobre una ficha con

```
addTab(String title, Icon icon, Component component, String tip)
```

Puede mover las fichas a los lados del contenedor con `setTabbedPlacement()` usando una de las constantes de clase de TOP, BOTTOM, LEFT y RIGHT para determinar su colocación. Otra opción disponible indica lo que debe hacer con las fichas cuando sean demasiado anchas para que entren en una linea. Usando `setTabLayoutPolicy()`, puede hacer que ocupen varios niveles con la constante de clase WRAP_TAB_LAYOUT. O puede hacer que se desplacen en una línea con SCROLL_TAB_LAYOUT. La figure 12.11 muestra dos normas de diseño.



WRAP–TAB-LAYOUT SCROLL–TAB–LAYOUT

Figura **12.11.** Normas de diseño de JTabbedPane

JSplitPane

`JSplitPane` tampoco es exactamente un administrador de disefio. Pero su comportamiento es tan parecido a éstos que lo hemos incluido en esta seccion. Se parece a `JTabbedPane`, ya que es un contenedor autosuficiente. En el caso de `JTabbedPane`, la funcionalidad que se ha agregado es una lista de fichas y de páginas asociadas. Para `JSplitPane`, obtendremos un contendedor que se puede dividir vertical y horizontalmente, y el usuario o el programador podrán modificar el tamaiio de cada zona, moviendo la barra divisora.

Para utilizar `JSplitPane` creamos la orientación adecuada y los componentes que queramos desplazar aunque siempre podremos cambiarlos cuando queramos. Para un panel vertical, donde los componentes se distribuyen de arriba abajo, tendremos que utilizar `JSplitPane.VERTICAL_SPLIT` para determinar la orientacion.

```
new JSplitFane(JSplitPane.VERTICAL_SPLIT, topComponent, bottomComponent);
```

Para el caso de los paneles horizontales, será todo igual, a excepción de `JSplitPane.HORIZONTAL_SPLIT`:

```
new JSplitPane(JSplitPane.HORIZONTAL_SPLIT, leftComponent, rightComponent);
```

Hay otra variedad del constructor que aiiade un parámetro booleano que especifica si los componentes se han de dibujar de forma constante cada vez que se mueve la línea divisoria:

```
new JSplitPane(orientation, boolean isContinuousLayout, component1,
component2);
```

Las otras dos capacidades que utilizaremos con `JSplitPane` cuando haga las funciones de divisor son:

```
public void setDividerLocation(int position)
public void setDividerLocation(float percentage)
```

El otro truco es activar la opción denominada "expandibles con un toque", Esto proporciona flechas al divisor para permitir al usuario hundir una parte del panel dividido o expandirlo a su tamaiio total. Aqui tiene la instrucción:

```
public void setOneTouchExpandable(boolean expandable)
```

La figura 12.12 muestra un ejemplo del diseiio de pantalla tras mover los divisores un poco. Observe que uno de los cuadros tiene activado uno de los expandibles con un toque. (No hay una indicación visual que indique que se ha activado el diseiio continuo.)

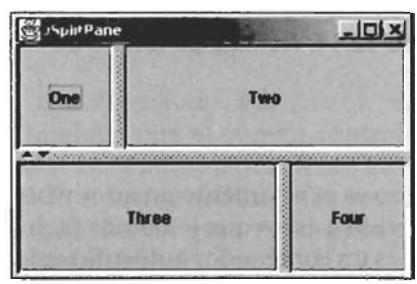


Figura 12.12. Un JSplitPane

El programa es el responsable de la creación del contenido de la figura 12.12 y nos muestra el funcionamiento de **`JSplitPane`**. Para dividir más de dos componentes, tendremos que especificar un **`JSplitPane`** como uno de los dos componentes de `JSplitPane` (como demuestra este ejemplo).

Listado 12.6. Disposición de una pantalla con JSplitPane

```
import java.awt.*;
import java.awt.event.*;
```

```
import javax.swing.*;
public class SplitPaneTest {
    public static void main(String args[]) {
        JFrame frame = new ExitableJFrame("JSplitPane");
        Container contentPane = frame.getContentPane();
        JButton b1 = new JButton("One");
        JButton b2 = new JButton("Two");
        JSplitPane jsp1 = new JSplitPane(
            JSplitPane.HORIZONTAL_SPLIT,
            true, b1, b2);
        b1 = new JButton("Three");
        b2 = new JButton("Four");
        JSplitPane jsp2 = new JSplitPane(
            JSplitPane.HORIZONTAL_SPLIT,
            false, b1, b2);
        JSplitPane jsp3 = new JSplitPane(
            JSplitPane.VERTICAL_SPLIT,
            true, jsp1, jsp2);
        jsp3.setOneTouchExpandable(true);
        contentPane.add(jsp3, BorderLayout.CENTER);
        frame.setSize(300, 200);
        frame.show();
    }
}
```

GridLayout

El administrador de diseiio `GridLayout` distribuye sus componentes en forma de cuadrícula. Todos los componentes del contenedor tienen las mismas dimensiones. En la figura 12.13 tenemos un ejemplo de este tipo de diseiio y el listado 12.7 es el programa que lo creó.

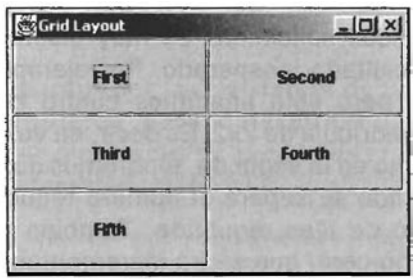


Figura **12.13.** Un GridLayout

Listado 12.7. Disposición de una pantalla con GridLayout

```
import java.awt.*;
import javax.swing.*;
public class GridLayoutTest {
    public static void main(String args[]) {
        JFrame frame = new ExitableJFrame("Grid Layout.");
```

```
Container contentPane = frame.getContentPane();
contentPane.setLayout(new GridLayout(3, 2));
contentPane.add(new JButton("First"));
contentPane.add(new JButton("Second"));
contentPane.add(new JButton("Third"));
contentPane.add(new JButton("Fourth"));
contentPane.add(new JButton("Fifth"));
frame.setSize(300, 200);
frame.show();
```

Especifique el numero de filas y columnas de la rejilla al constructor de la clase `GridLayout`. En este ejemplo, se usan tres filas y dos columnas. De nuevo, se han añadido cinco botones de prueba usando el método por defecto `add(Component comp)`. Observe el orden final de los botones: La secuencia de agregación linear se usa para rellenar las filas desde la parte superior a la inferior.Si este orden implícito no es adecuado para su programa, puede usar el segundo método `add()` de `container`. Toma un único argumento de posición numérica adicional, como el siguiente:

```
contentPane.add(new JButton("First"), 0);
contentPane.add(new JButton("Second"), 1);
contentPane.add(new JButton("Third"), 2);
contentPane.add(new JButton("Fourth"), 3);
contentPane.add(new JButton("Fifth"), 4);
```

El diseño resultante es exactamente el mismo que el de la figura **12.13.** En otras palabras, sólo ha separado el orden de adición desde la posición.

Advertencia

Si el numero de celdas solicitadas es muy distinto del real, nos podemos encontrar con un resultado inesperado. Por ejemplo, podemos solicitar una cuadricula de 3x3, pero solo añadimos cuatro componentes, con lo que obtendremos una cuadricula de 2x2. Es decir, en vez de aparecer tres botones en la primera fila y uno en la segunda, tendremos dos en la primera y otros dos en la segunda. Cuando se supera el número requerido total, se favorece el mantener el numero de filas requerido. Tambien podemos especificar una cuadricula de tamaiio cero, que se irá incrementando segun sea necesario.

Como las celdas de la cuadricula se suelen especificar utilizando coordenadas de dos dimensiones en vez de una unidimensional, es posible que nos preguntemos por qué no hemos visto el método `add(component comp, int x, int y)` por ningún lado. La respuesta es muy sencilla. Todos estos metodos no pertenecen a ningún administrador de diseiio, sino a la clase del contenedor que se ha definido. Esto se debe a que los métodos `add()` agregan en principio componentes a un contenedor. Durante la adición de componentes, el proceso de diseño carecerá de

importancia. En realidad, lo tendremos en cuenta en el momento en que vayamos a dibujar la interfaz. Para librarse de la limitación de no poder especificar la posición x e y en `GridLayout`, podemos utilizar el administrador `GridBagLayout`, del cual hablaremos más tarde, que es más flexible (y más complicado de utilizar) que `GridLayout`.

Administrador y cuadro BoxLayout

El administrador `BoxLayout` se parece a `FlowLayout` y `GridLayout`. Al primero porque trata de modificar el tamafio de los componentes basándose en las preferencias de los componentes. Pero se diferencia en que cuando se añaden componentes a una fila, no se volverán a ajustar. Y `BoxLayout` actua como lo haría `GridLayout`, si trabajase con una unica fila o columna. La diferencia entre los dos es que `BoxLayout` puede trabajar con su propio tamafio.

Entonces, ¿cómo podemos utilizar `BoxLayout`? Lo normal es que no podamos hacerlo. En su lugar utilizamos el contenedor `Box`, que se encargará de utilizar internamente `BoxLayout`. Lo primero que hacemos es crear un cuadro, especificando la dirección a travks de una de las constantes de `BoxLayout` (`X_AXIS` o `Y_AXIS`):

```
Box boxl = new Box(BoxLayout.X_AXIS); // horizontal
Box box2 = new Box(BoxLayout.Y_AXIS); // vertical
```

Ahora podemos agregar componentes como siempre:

```
boxl.add(new JButton("Left"));
boxl.add(new JButton("Center"));
boxl.add(new JButton("Right"));
box2.add(new JButton("Top"));
box2.add(new JButton("Center"));
box2.add(new JButton("Bottom"));
```

Si no queremos preocuparnos por las constantes de `BoxLayout`, podemos solicitar un cuadro independiente para determinar la dirección a travks de `createHorizontalBox()` o `createVerticalBox()`.

`Box` nos permite crear áreas de tamafio fijo:

```
public static Component createRigidArea(Dimension dim);
public static Component createHorizontalStrut(int width);
public static Companent createVerticalStrut(int height);
```

o utilizar los siguientes mktodos para crear áreas de tamafio ampliable:

```
public static Component createGlue();
public static Component createHorizontalGlue();
public static Component createVerticalGlue();
```

Para los mktodos `createRigidArea()`, `createHorizontalStrut()` y `createverticalStrut()`, creamos un componente de tamafio fijo y luego utili-

zamos el método add() para añadir otros dos componentes. De esta forma nos aseguramos de que tenemos una distancia fija entre ellos. En la figura **12.14** tenemos un ejemplo de este disefio.

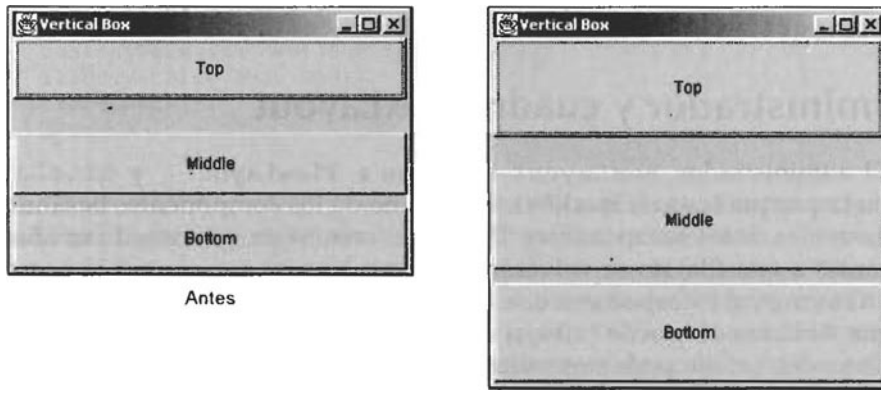


Figura 12.14. Box vertical con distancias fijas entre los componentes, antes y después de modificar sus tamaños

Este es el programa responsable de la creación de los cuadros de la figura **12.14.** Observe que el ancho de la zona rigida creada no se tiene en cuenta.

Listado 12.8. Disposición de una pantalla con un Box vertical y puntales

```
import java.awt.*;
import javax.swing.*;
public class VerticalBoxTest {
    public static void main(String args[]) {
        JFrame frame = new ExitableJFrame("Vertical Box");
        Container contentPane = frame.getContentPane();
        Box box = Box.createVerticalBox();
        box.add(new Button("Top"));
        box.add(Box.createVerticalStrut(25));
        box.add(new Button("Middle"));
        box.add(Box.createRigidArea(new Dimension (50000, 10)));
        box.add(new Button("Bottom"));
        contentPane.add(box, BorderLayout.CENTER);
        frame.setSize(300, 200);
        frame.show();
    }
}
```

Nota

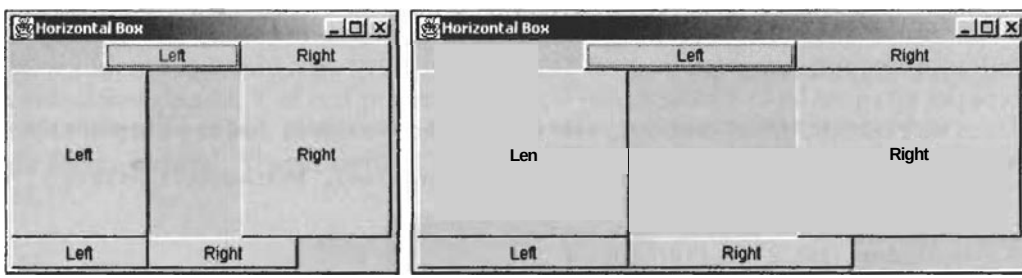
El programa del listado 12.8 usa el componente AWT Button en lugar del componente Swing JButton porque aun no ha aprendido la configuración de la alineacion de componentes.

Para los métodos `createGlue()`, `createHorizontalGlue()` y `createVerticalGlue()`, lo que crearnos en un cornponente cuyo tamaño se puede ajustar y que en el rnornento en que se aiiada, utilizará cualquier espacio adicional resultante de la modificación del tamaño de la ventana. En la figura 12.15 podemos ver el efecto al que hacernos referencia.

El listado 12.9 es el prograrna encargado de producir los cuadros de la figura 12.15. Observe que el lugar donde se agrega "Glue" deterrnina el espacio al que va el espacio extra:

Listado 12.9. Disposición de una pantalla con un Box horizontal y Glue

```
import java.awt.*;
import javax.swing.*;
public class HorizontalBoxTest {
    public static void main(String args[]) {
        JFrame frame = new ExitableJFrame("Horizontal Box");
        Container contentPane = frame.getContentPane();
        Box box = Box.createHorizontalBox();
        box.add(Box.createHorizontalGlue());
        box.add(new Button("Left"));
        box.add(new Button("Right"));
        contentPane.add(box, BorderLayout.NORTH);
        box = Box.createHorizontalBox();
        box.add(new Button("Left"));
        box.add(Box.createHorizontalGlue());
        box.add(new Button("Right"));
        contentPane.add(box, BorderLayout.CENTER);
        box = Box.createHorizontalBox();
        box.add(new Button("Left"));
        box.add(new Button("Right"));
        box.add(Box.createHorizontalGlue());
        contentPane.add(box, BorderLayout.SOUTH);
        frame.setSize(300, 200);
        frame.show();
    }
```



Antes Despues

Figura 12.15. Box horizontal antes y después de modificar sus tamaños

Alineación de los componentes Box

Esto nos lleva a los valores de alineamiento de componentes en un cuadro. El administrador `BoxLayout` intenta respetar los tamaños máximo y mínimo establecidos para los componentes del contenedor. Además, cada componente tiene un valor de alineamiento X y otro Y. Los valores `setXAlignment()` y `setYAlignment()` varian desde 0.0f hasta 1.0f y especifican el punto en el eje adecuado donde colocar los componentes cuando haya espacio disponible.

El alineamiento horizontal se hace a través del eje x. Hay tres constantes en la clase **`Component`** para determinados valores de alineamiento:

- **`LEFT_ALIGNMENT`** es 0.0f.
- **`CENTER_ALIGNMENT`** es 0.5f.

  `RIGHT_ALIGNMENT` es 1.0f.

Sin embargo, puede proporcionar cualquier valor entre 0 y 1 (como un **`float`**). Puede proporcionar diferentes valores para varios componentes, si es necesario, La figura 12.26 muestra algunos componentes que comparten el mismo valor para cada alineamiento. El listado 12.10 es el programa que genera esta figura.

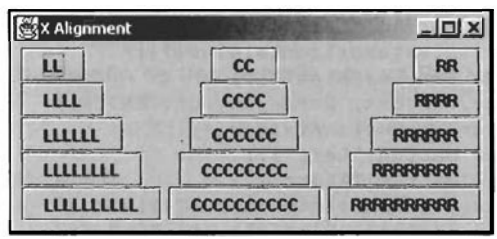


Figura 12.16. Diferentes alineamientos horizontales para componentes box

Listado 12.10. Trabajar con alineacion de componentes

```
import java.awt.*;
import javax.swing.*;

public class AlignX {

    private static Container makeIt(String labelChar, float alignment) {
        JPanel container = new JPanel();
        BoxLayout layout = new BoxLayout(container, BoxLayout.Y_AXIS);
        container.setLayout(layout);

        for (int i=1; i<6; i++) {
            String label = makeLabel(labelChar, i*2);
            JButton button = new JButton(label);
            button.setAlignmentX(alignment);
            container.add(button);
```

```
        return container;
    }

    private static String makeLabel(String s, int length) {
        StringBuffer buff = new StringBuffer(length);
        for (int i=0; i<length; i++) {
            buff.append(s);
        }
        return buff.toString();
    }

    public static void main(String args[]) {
        JFrame frame = new ExitableJFrame("X Alignment");
        Container contentPane = frame.getContentPane();

        Container panel1 = makeIt("R", Component.RIGHT_ALIGNMENT);
        Container panel2 = makeIt("C", Component.CENTER_ALIGNMENT);
        Container panel3 = makeIt("L", Component.LEFT_ALIGNMENT);

        contentPane.setLayout(new FlowLayout());
        contentPane.add(panel1);
        contentPane.add(panel2);
        contentPane.add(panel3);

        frame.pack();
        frame.show();
    }
}
```

El alineamiento vertical funciona igual que el alineamiento horizontal, con dos constantes diferentes y una compartida con el alineamiento horizontal:

- TOP_ALIGNMENT es 0.0f.
- BOTTOM_ALIGNMENT es 1.0f.
- La misma constante CENTER_ALINGMENT sigue siendo 0.5f.

Cuando los componentes no comparten el mismo alineamiento, suceden algunos comportamientos de alineamiento interesantes. En lugar de alinearlo al borde del contenedor, el alineamiento se produce a partir de una línea imaginaria a través del centro del contenedor. Como muestra la figura 12.17, el componente con el alineamiento ajustado a BOTTOM_ALIGNMENT tiene su parte inferior sobre su línea central imaginaria. Y el componente con TOP_ALIGNMENT tiene su parte superior sobre la línea central. El tercer componente con un CENTER_ALIGNMENT se centra de forma natural. Y aqui tiene el programa, el listado 12.11, que genera el figura 12.17.

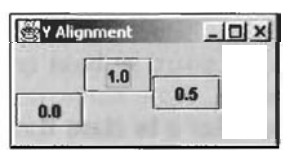


Figura 12.17. Mezcla de alineamientos en un diseño en el eje x

Listado 12.11. Disposición de una pantalla con diferentes alineamientos de componentes

```
import java.awt.*;
import javax.swing.*;

public class AlignY {

    public static void main(String args[]) {
        JFrame frame = new ExitableJFrame("Y Alignment");
        JPanel contentPane = (JPanel)frame.getContentPane();

        BoxLayout layout = new BoxLayout(contentPane, BoxLayout.X_AXIS);
        contentPane.setLayout(layout);

        JButton button = new JButton("0.0");
        button.setAlignmentY(Component.TOP_ALIGNMENT);
        contentPane.add(button);
        button = new JButton("1.0");
        button.setAlignmentY(Component.BOTTOM_ALIGNMENT);
        contentPane.add(button);
        button = new JButton("0.5");
        button.setAlignmentY(Component.CENTER_ALIGNMENT);
        contentPane.add(button);

        frame.setSize(200, 100);
        frame.show();
    }
}
```

SpringLayout

La incorporación más reciente a la familia de los administradores de diseño Java es `SpringLayout`, agregado con la versión Java 1.4. `SpringLayout` le permite adjuntar "fuentes" a los componentes para que se distribuyan de forma relativa a otros componentes. Por ejemplo, puede hacer que un boton aparezca unido al borde derecho, sin importar el tamaño de la pantalla.

Advertencia

No confunda las fuentes de `SpringLayout` con el "glue" de marcas de `BoxLayout`. Quizás parezca que tienen la misma funcion, pero son completamente diferentes.

El administrador `SpringLayout` se basa en `SpringLayout.Constraints` para los limites del componente.

Esto funciona de forma similar a la clase `GridBagConstraints` que complementa al administrador `GridBagLayout`. Es decir, cada componente agregado al

contenedor puede tener un `SpringLayout.Constraints` adjunto. Sin embargo, este es el fin de las similitudes.

Ahora miremos las diferencias. En primer lugar, normalmente no tiene que agregar los componentes con los limites. En su lugar, puede agregar el componente y adjuntar los limites por separado. Nada le impide agregar los limites con el componente, pero `SpringLayout.Constraints` no es una simple clase. Es una colección de objetos `Spring`, cada uno con un limite diferente en el componente.

Cuando usa un `Spring`, debe agregar el limite `spring` por separado para `SpringLayout.Constraints`. Los limites de `SpringLayout.Constraints` se agregan especificando limites en el borde de un componente.

Usando las cuatro constantes `SpringLayout` EAST, WEST, NORTH y SOUTH, se invoca al método `setContraints(String edge, Spring spring)` de `SpringLayout.Constraints`, donde `String` es una de las constantes limitadoras. Por ejemplo, si quiere agregar un componente en la parte superior izquierda de un contenedor, puede configurar dos fuentes de tamaiio constante combinandolos, y agregar el componente al contenedor con el conjunto combinado, como se muestra aqui:

```
Component left = ...;
SpringLayout layout = new SpringLayout();
JPanel panel = new JPanel(layout);
Spring xPad = Spring.constant(5);
Spring yPad = Spring.constant(25);
SpringLayout.Constraints constraint = new SpringLayout.Constraints();
constraint.setConstraint(SpringLayout.WEST, xPad);
constraint.setConstraint(SpringLayout.NORTH, yPad);
contentPane.add(left, constraint);
```

Esto no parece muy dificil, pero se complica cuando debe agregar el siguiente componente, a la derecha del anterior o debajo de 61. No puede limitarse a decir "agrega el componente n pixeles por encima". En realidad debe agregar el relleno al borde del anterior componente. Para encontrar el borde del anterior componente, pregunte al administrador de diseiio con `getConstraint()`, pasando el borde que desea y el componente. Esta instrucción, por ejemplo:

```
layout.getConstraint(SpringLayout.EAST, left)
```

obtendra la posición del borde derecho del primer componente. A partir de ahi, puede agregar el relleno necesario y adjuntarlo al borde de otro componente, como se muestra aqui.

```
Component right = ...;
Spring rightSideOfLeft = layout.getConstraint(SpringLayout.EAST, left);
Spring pad = Spring.constant(20);
Spring leftEdgeOfRight = Spring.sum(rightSideOfLeft, pad);
constraint = new SpringLayout.Constraints();
constraint.setConstraint(SpringLayout.WEST, leftEdgeOfRight);
constraint.setConstraint(SpringLayout.NORTH, yPad);
contentPane.add(right, constraint);
```

Funcionará perfectamente, pero se vuelve pesado a medida que el numero de elementos aumenta. Otro modo de hacerlo, de forma yue evitamos los pasos intermedios, es agregar los componentes sin límites. A continuación agregue cada limite por separado, conectando los componentes mediante el método `putConstraints()` de `SpringLayout`:

```
public void putConstraint(String e1, Component c1, int pad, String e2,
    Component c2)
public void putConstraint(String e1, Component c1, Spring s, String e2,
    Component c2)
```

Aquí, en lugar de preguntar por el borde y agregar el relleno uno mismo, la llamada a `putConstraint()` combina las tareas.

Nota

Los argumentos `string` de `putConstraint()` son las cuatro constantes `SpringLayout` mencionadas anteriormente.

Para demostrarlo, el siguiente fragmento agrega las mismas limitaciones al componente de la derecha, como acabamos de hacer. Aunque en este caso, estamos usando `putconstraint()` en lugar de `SpringLayout.Constraints` directamente.

```
Component left = ...;
Component right = ...;
SpringLayout layout = new SpringLayout();
JPanel panel = new JPanel(layout);
panel.add(left);
panel.add(right);
layout.putConstraint(SpringLayout.WEST, left, 5, SpringLayout.WEST, panel);
layout.putConstraint(SpringLayout.NORTH, left, 25, SpringLayout.NORTH,
panal);
layout.putConstraint(SpringLayout.NORTH, right, 25, SpringLayout.NORTH,
panel);
layout.putConstraint(SpringLayout.WEST, right, 5, SpringLayout.EAST,
left);
```

Truco

Cuando usamos `putConstraints()`, debemos asegurarnos de especificar la posición del componente que desconocemos en primer lugar y conectarlo a algo fijado o que pueda ser calculado, como el borde de la pantalla.

Para ayudarle a visualizar el uso de `SpringLayout`, Sun tiene una herramienta llamada BeanBuilder disponible en http://java.sun.com/products/javabeans/beanbuilder/. BeanBuilder se cred para ser usada básicamente con componentes

JavaBean. No nos detendremos en el uso especifico de esta herramienta, pero una de las cosas que le permite hacer es conectar componentes con `SpringLayout` y lo usaremos ahora para hacerlo. La figure **12.18** muestra el aspecto de la herramienta cuando la inicia.

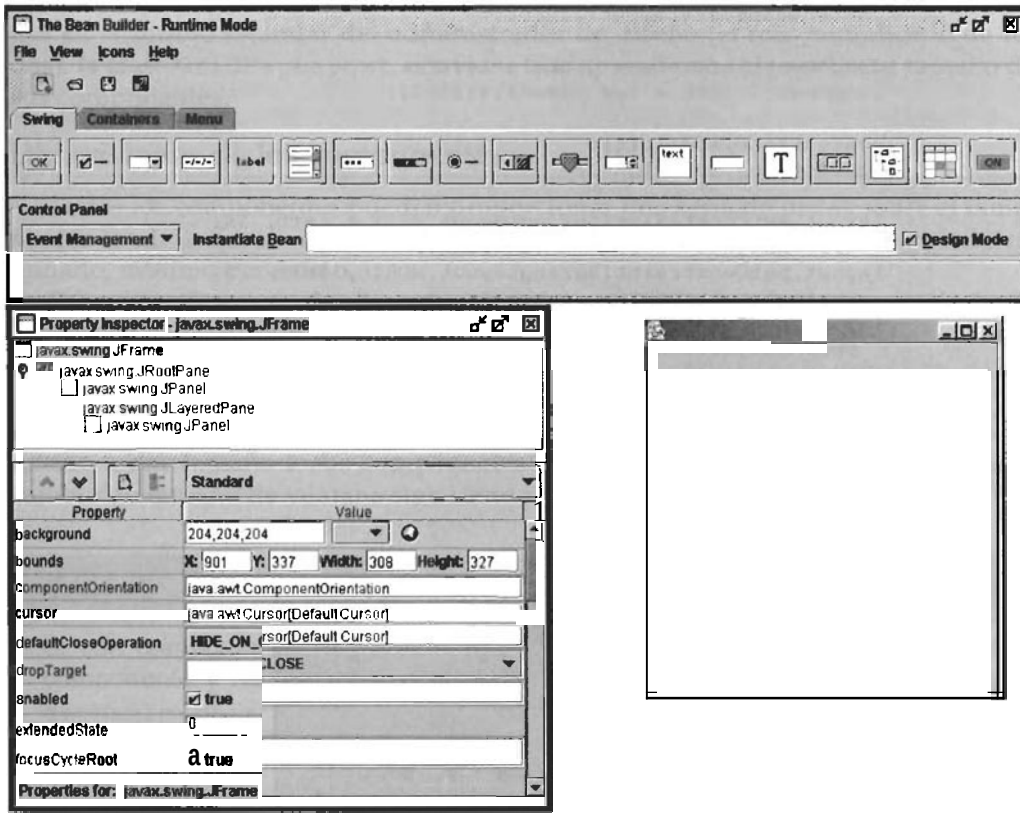


Figura 12.18. La herramienta BeanBuilder al iniciarse

Alrededor de los bordes de cada componente hay un conjunto de cuatro cuadros, cada uno para el norte, sur, este y oeste. Puede arrastrar una flecha fuera del cuadro y conectarlo a cualquier otro cuadro. Si la herramienta fuera un poco mds sofisticada, le permitiria especificar el tamaño de los espacios, pero tal y como es, la pantalla tendrd el mismo aspecto que la figura **12.19** durante el desarrollo. Conectará cada flecha de forma visual con una llamada especifica a `putConstraint()`.

Para demostrar el uso de `SpringLayout`, el listado **12.12** conecta las piezas que hemos explicado aqui, usando `putConstraints()`.

Listado 12.12. Disposición de una pantalla con SpringLayout

```
import java.awt.*;
import javax.swing.*;
```

```
public class SpringFormTest {
    public static void main(String args[]) {
        JFrame frame = new ExitableJFrame("Spring Form");
        Container contentPane = frame.getContentPane();

        SpringLayout layout = new SpringLayout();
        contentPane.setLayout(layout);

        Component left = new JLabel("Left");
        Component right = new JTextField(15);

        contentPane.add(left);
        contentPane.add(right);

        layout.putConstraint(SpringLayout.WEST,  left, 10,
          SpringLayout.WEST,  contentPane);
        layout.putConstraint(SpringLayout.NORTH, left, 25,
          SpringLayout.NORTH, contentPane);
        layout.putConstraint(SpringLayout.NORTH, right, 25,
          SpringLayout.NORTH,  contentPane);
        layout.putConstraint(SpringLayout.WEST, right, 20,
          SpringLayout.EAST, left);

        frame.setSize(300, 100);
        frame.show();
    }
}
```

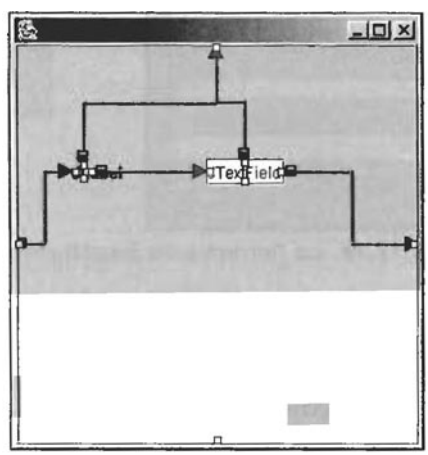


Figura 12.19. Uso de SpringLayout

Advertencia

Cuando se usa `SpringLayout`, dependiendo de los límites que use, puede que tenga que ajustar explícitamente el tamaño del contenedor.

GridBagLayout y GridBagContraints

Excepto SpringLayout, todos los ejemplos que hemos utilizado para demostrar los distintos administradores de diseño se han limitado a utilizar botones. Aun no hemos mostrado el dibujo de componentes con un tamaño fijo. Antes de centrarnos en el último estdndar del administrador de diseiio (el más complicado de todos), la clase GridBagLayout, conviene que aprendamos algo sobre el tamaño de sus componentes.

Dimensiones de Ins componentes

Todos los componentes pueden proporcionar una serie de pistas sobre el tamaño que tendrán dentro de la ventana. Se puede especificar tres tamaiios: predeterminado, mínimo y mdximo.

Para especificar uno de ellos para los componentes Swing, invocamos al método apropiado para ajustar las dimensiones: setMaximumSize(Dimension dim), setMinimumSize(Dimension dim) y setPreferredSize(Dimension dim). A su vez, los métodos receptores asociados (getPreferredSize(), getMinimumSize() y getMaximumSize()) ayudarán a los administradores de diseiio a dar tamaiio a sus componentes de modo que el resultado de cualquier cambio de tamaiio de ventana siga siendo aceptable.

Nota

Si usa componentes AWT, para modificar el tamaño debe subclasificar el componente y reemplazar getPreferredSize(), getMinimumSize() **y** getMaximumSize().

Intente ejecutar de nuevo la demostración de GridLayout y realizar pruebas con el tamaiio de la ventana. Observará que los cinco botones se pueden estirar o reducir hacia los extremos. Puede controlar esta flexibilidad ajustando el tamafio asociado de los componentes JButton y usando un administrador de disefio que respete estas pistas, como GridBagLayout. (GridLayout no lo hace.)

Nota

La mayoría de los administradoresde diseño proporcionados ignoran cualquier referencia sobre el tamaño maximo. Mientras que el valor que devuelve getMaximumSize() **es** new Dimension(Short.**MAX**_VALUE, Short.**MAX**_VALUE), **el administrador de disefio podra ignorar cualquier anulacion de esta configuración. En ciertos casos, al igual que ocurria con** BoxLayout, **el administrador de disefio respetara el maximo requerido.**

Trabajar con GridBagLayout

GridBagLayout es un administrador de diseiio especial que se utiliza en las clase de ayuda (GridBagConstraints) para especificar todos los parámetros del diseiio.

Normalmente, hay un ejemplo de clase por componente. La principal (pero no única) ventaja del administrador GridBagLayout sobre GridLayout es que permite la utilización de un componente en mhs de una celda.

El tamaño de un componente se puede especificar como el numero de celdas verticales y/o horizontales que puede ocupar un componente dentro de la cuadrícula. La clase de ayuda GridBagConstraints controla estos parámetros. Como indica la tabla 12.2, la clase contiene sobre todo constantes y variables de limites public:

Tabla 12.2. Composición de la clase GridBagConstraints

Variables/Métodos	Descripción
GridBagConstraints()	Crea un objeto GridBagConstraints
anchor	Área de cuadricula a donde se dirigira el componente cuando haya espacio extra
CENTER, EAST, NORTH, NORTHEAST, NORTHWEST, SOUTH, SOUTHEAST, SOUTHWEST y WEST	Valores para el limite delimitador
fill	Dirección en la que el componente crecera cuando haya espacio extra
NONE, BOTH, HORIZONTAL y VERTICAL	Valores para el limite de relleno
gridxgridy	Coordenadas en la cuadricula para el componente
gridwidthgridheight	Número de filas o columnas que pueden ocupar los cornponentes
RELATIVE y REMAINDER	Proporciona medios alternativos de especificar coordenadas, anchura y altura.
insets	Especifica el relleno externo alrededor de un componente
ipadxipady	Especifica el relleno interno alrededor de un componente
weightxweighty	Determina el modo en que se distribuye un componente cuando hay espacio disponible en el contenedor
clone()	Crea una copia de los lirnites

Esta clase se puede componer de una serie de variables y de una colección de constantes de clases (también tiene un constructor y una implementación del método `Object clone()`, que veremos más tarde).

Los campos más importantes de `GridBagConstraints` son `gridx`, `gridy`, `gridwidth` y `gridheight`. Controlan el tamaño y la ubicación de un componente dentro de la cuadricula. A diferencia de las limitaciones propias de las coordenadas de una dimension que vimos con el método `add()`, los campos `gridx` y `gridy` nos permiten especificar componentes `GridBagLayout` con dos coordenadas. Con `GridBagLayout` utilizamos una version especial de `add()` clue toma `GridBagConstraints` como segundo parámetro.

La figura 12.20 nos muestra un ejemplo del funcionamiento de `GridBagLayout` que muestra las ventajas que tiene sobre las clases `FlowLayout` y `GridLayout`. El listado 12.13 es el programa que produce el diseño GUI de la figura 12.20.

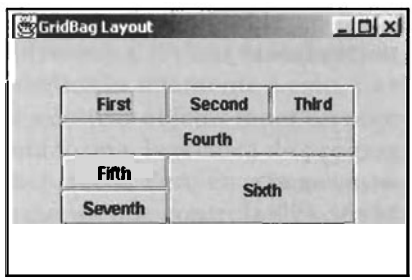


Figura 12.20. Un GridBagLayout

Listado 12.13. Disposición de una pantalla con GridBagLayout

```
import java.awt.*;
import javax.swing.*;
public class GridBagTest {
    public static void main(String args[]) {
        JFrame frame = new ExitableJFrame("GridBag Layout");
        Container contentPane = frame.getContentPane();

        GridBagLayout gb = new GridBagLayout();
        contentPane.setLayout(gb);

        GridBagConstraints gbc = new GridBagConstraints();
        JButton b;
        gbc.gridwidth    1;
        gbc.gridheight = 1;
        gbc.gridx = 0;
        gbc.gridy = 0;
        gbc.fill = GridBagConstraints.BOTH;
        b = new JButton("First");
        contentPane.add(b, gbc);
        b = new JButton("Second");
        gbc.gridx = 1;
        gbc.gridwidth = 2;
```

```
        contentPane.add(b, gbc);
        b = new JButton("Third");
        gbc.gridx = 3;
        gbc.gridwidth = 1;
        contentPane.add(b, gbc);
        b = new JButton("Fourth");
        gbc.gridx = 0;
        gbc.gridy++;
        gbc.gridwidth = 4;
        contentPane.add(b, gbc);
        b = new JButton("Fifth");
        gbc.gridy++;
        gbc.gridwidth = 1;
        contentPane.add(b, gbc);
        b = new JButton("Sixth");
        gbc.gridx = 1;
        gbc.gridwidth = 3;
        gbc.gridheight = 2;
        contentPane.add(b, gbc);
        b = new JButton("Seventh");
        gbc.gridx = 0;
        gbc.gridy++;
        gbc.gridwidth = 1;
        gbc.gridheight = 1;
        contentPane.add(b, gbc);
        frame.setSize(300, 100);
        frame.show();
    }
}
```

El método `main()` de este applet comienza creando un nuevo caso del objeto `GridBagLayout` y otro de ayuda `GridBagConstraints`. Después de configurar el disefio de `GridBagLayout`, el código inicia uno de los parámetros de limitación del primer objeto que se va a afiadir. Con esta finalidad se rellenan los campos de `GridBagcontraints`. Ajustamos el ancho y la altura de la celda a 1 y comenzamos a agregar componentes por la parte de arriba a la izquierda (la coordenada 0,0 de la cuadricula).

La siguiente linea configura el estilo de los componentes. Determina como rellenan el espacio que tienen a su disposición cuando han de exceder al tamafio preestablecido. Los valores legales para la variable son NONE, HORIZONTAL, VERTICAL y BOTH. A continuación asociamos estos parámetros con nuestros componentes invocando el método `add()` y entregándoselo a `GridBagConstraints` como segundo argumento. Resumiendo, para agregar un componente al contenedor del estilo `GridBagLayout` tenemos que dar dos pasos:

1. Determinar un conjunto de parámetros `GridBagConstraints`.
2. Invocar una versión de dos parámetros del método `add()` y utilizarlo para añadir componentes al contenedor.

Además del primer paso, tenemos otro punto interesante dentro del ejemplo del código anterior. Primero, iniciamos algunos de los campos de `GridBagCons-`

traints y luego añadimos los botones a su contenedor. De esta forma, el administrador GridBagLayout añadirá el primer boton al objeto GridBagConstraints durante su definición. A continuación modificaremos alguno de los valores de GridBagConstraints y volveremos a llamar a add() para albergar el segundo botón con GridBagConstraints (solo tenemos un objeto GridBagConstraints en todo el programa). Podemos cambiar las variables gridx y gridwidth, pero no tocaremos las variables gridy ni gridheight. Esto significa que hemos usado la mitad de los parámetros GridBagConstraints para el primer boton.

¿No deberiamos crear un objeto GridBagConstraints para cada elemento? Generalmente si, pero en este caso en concreto no. La razón la tenemos dentro de la propia clase GridBagLayout. El método setContraints() utiliza el método clone() para hacer una copia de los objetos GridBagConstraints que le entregan. Esta es la razón por la que podemos utilizar un único objeto GridBagConstraints para cualquier número de componentes.

El método clone() sólo estard disponible en las clases que trabajen con la interfaz Cloneable. Si volvemos a revisar la definición GridBagConstraints, veremos que esta clase se dedica exactamente a esto. La norma que se sigue es que los argumentos permitan a nuestros objetos tener un acceso total (compartido) a los objetos que se pasan. De esta forma, la tecnica de programación que se ha utilizado en nuestro ejemplo seria incorrecta. Pero en este caso, se puede utilizar para mejorar la legibilidad de las secciones que controla GridBagLayout.

Hay otro pardmetro de GridBagConstraints que hemos de estudiar. Para ver su efecto, tenemos que volver a compilar y ejecutar el applet eliminando la siguiente linea:

```
gbc.fill = GridBagConstraints.BOTH;
```

La figura 12.21 contiene un ejemplo del GridBagLayout resultante.

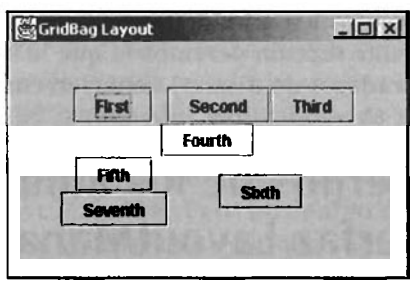


Figura 12.21. Un GridBagLayout con el relleno **ajustado** a NONE

De repente tenemos que el disefio de nuestra figura parece que ha abandonado el modelo de la cuadricula (si mira más de cerca, verá que los botones **Second** y **Sixth** no aparecen alineados con las líneas de cuadricula). La razón se debe a otro de los parámetros GridBagConstraints: anchor. Si no se determina un valor para fill, se utilizará el predeterminado NONE. Es decir, que cualquier componente tendrá su tamaño predeterminado (siempre que sea posible, claro). La extraña colo-

cación de los botones **Second** y **Six** se deben al estilo predeterminado `anchor`. El estilo predeterminado de nuestro ejemplo, centra los componentes dentro de la celda que ocupan (utilizando `gridwidth` y `gridheight`). Si añadimos la siguiente linea (donde eliminamos la linea donde se definia el estilo del relleno) obtendremos el resultado de la figura 12.22.

```
gb'.anchor = GridBagConstraints.NORTHWEST;
```

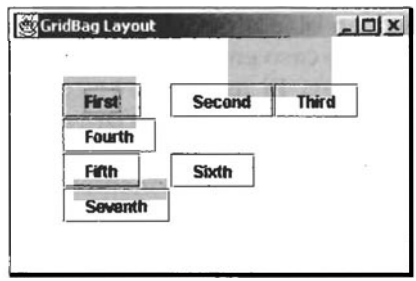


Figura 12.22. Un GridBagLayout con el relleno ajustado a NONE y el delimitador ajustado a NORTHWEST

Los valores viilidos para el parámetro `anchor` son CENTER, NORTH, EAST, NORTHEAST, SOUTHEAST, SOUTH, SOUTHWEST, WEST y NORTHWEST.

Nota

`GridBagLayout` es muy flexible y su revision solo ofrece una vista general de las funciones de la clase.

Ahora que ya conocemos los administradores de diseño de AWT, los veremos por dentro. En la siguiente sección veremos lo que la interfaz `LayoutManager` (lo que todos los adrninistradores de diseño) tienen en común entre ellos.

Elementos internos de los administradores de diseño: la interfaz LayoutManager

Los adrninistradores de diseño son unas potentes cajas negras que nos ayudan a diseñar nuestras interfaces GUI riipidamente. Si suele combinar varios administradores de diseño para obtener exactamente la pantalla que desea, puede que sea el momento de abandonar los administradores de diseño predefinidos y de crear los nuestros propios.

Si miramos a las definiciones de las clases para observar los adrninistradores de diseño, veremos que implementan la misma interfaz: `LayoutManager` (o la

subinterfaz `LayoutManager2`, ambas del paquete `java.awt`). Ésta es la clave para comprender los entresijos de los administradores de disefio. Aqui tenemos su definición:

```
public interface LayoutManager {
    public void addLayoutComponent(String name, Component comp);
    public void layoutContainer(Container parent);
    public Dimension minimumLayoutSize(Container parent);
    public Dimension preferredLayoutSize(Container parent);
    public void removeLayoutComponent(Component comp);
}
```

Los metodos que nos pedirá la interfaz se pueden agrupar de la siguiente forma:

- Agregar/eliminar componentes a un contenedor.
- Calcular el tamafio predeterminado y minimo de un contenedor.
- Efectuar cualquier operación de disefio.

Podemos ver el método `addLayoutComponent()` como análogo a `add()`. Pero esta afirmación sólo es correcta a medias. El unico que llamará al método `addLayoutComponent()` será `add(String constraint, Component comp)` y no `add(Component comp)`. Por ejemplo, los administradores `FlowLayout` y `GridLayout` implementan sus propios metodos `addLayoutComponent()` de la siguiente manera:

```
public void addLayoutComponent(String name, Component comp) {
}
```

Es verdad, un cuerpo vacio. Estos administradores no se hacen cargo de las colecciones de componentes que se afiaden a los contenedores asociados, porque no lo necesitan. El contenedor guarda un registro con todos los componentes que contiene (es decir, con la funcion principal). Y cuando se llama a `layoutContainer()`, el administrador de disefio recibe un controlador para que lo utilice con el contenedor. Gracias al controlador, el administrador de disefio podrá acceder a todos los componentes del contenedor utilizando los metodos de `container`, `getComponentCount()` y `getComponent()`. De esta forma algunos administradores de disefio pueden permitirse tener metodos `addLayoutComponent()` y `removeLayoutComponent()` vacios. Pero no es algo que ocurra siempre.

Además de implementar la interfaz `LayoutManager`, un administrador de disefio puede incorporar una subinterfaz `LayoutManager2`. Esta es su definición:

```
public interface LayoutManager2
        extends java.awt.LayoutManager {
    public void addLayoutComponent(Component comp, Object constraints);
    public float getLayoutAlignmentX(Container parent);
    public float getLayoutAlignmentY(Container parent);
    public void invalidateLayout(Container parent);
    public Dimension maximumLayoutSize(Container parent);
}
```

Los administradores de diseiio BorderLayout, BoxLayout, CardLayout y GridBagLayout no tienen vacios sus cuerpos para los mktodos anteriores. Esto se debe a que solicitan que se trabaje con add (Component comp, Object constraints) para acceder al método LayoutManager2 addLayoutComponent (Component comp, Object constraints).Mientras que el método add() sin etiqueta se limita a aiiadir un componente a la lista interna del contenedor, add() tambikn indica cuál es el administrador de diseiio del componente.

El método layoutContainer() es el corazón de cualquier administrador de diseiio. Es el punto donde todo tiene lugar. Aqui utilizará el administrador los componentes del contenedor y todos sus tamaiios para determinar con exactitud el tamaño y posición de cada componente.

Por ejemplo, el listado 12.14 es el layoutContainer() del administrador BorderLayout:

Listado 12.14. El método layoutContainer() del administrador BorderLayout

```
public void layoutContainer(Container target) {
    synchronized (target.getTreeLock()) {
      Insets insets = target.getInsets();
      int top = insets.top;
      int bottom = target.height - insets.bottom;
      int left = insets.left;
      int right = target.width - insets.right;

      boolean ltr = target.getComponentOrientation().isLeftToRight();
      Component c = nuil;

      if ((c=getChild(NORTH,ltr)) != null] {
         c.setSize(right - left, c.height);
         Dimension d = c.getPreferredSize();
         c.setBounds(left, top, right - left, d.height);
         top += d.height + vgap;

      if ((c=getChild(SOUTH,ltr)) != null) {
         c.setSize(right - left, c.height);
         Dimension d = c.getPreferredSize();
         c.setBounds(left, bottom - d.height, right - left, d.height);
         bottom -= d.height t vgap;

      if ((c=getChild(EAST,ltr)) != null) {
         c.setSize(c.width, bottam - top);
         Dimension d = c.getPreferredSize();
         c.setBounds(right - d.width, top, d.width, bottom - top);
         right -= d.width t hgap;

      if ((c=getChild(WEST,ltr)) != null) {
         c.setSize(c.width, bottom - top);
         Dimension d = c.getPreferredSize();
         c.setBounds(left, top, d.width, bottom - top);
         left += d.width t hgap;
      }
      if ((c=getChild(CENTER,ltr)) != null) {
```

```
            c.setBounds(left, top, right - left, bottom - top);

        }
    }
```

BorderLayout empieza determinando el área de trabajo que tiene a su disposición. Para ello resta el tamaiio de los elementos a la capacidad total del contenedor. A continuacidn, procede a componer los cinco componentes: NORTH, SOUTH, EAST, WEST y CENTER, por este orden. No olvidemos que los componentes de BorderLayout se han de aiiadir utilizando add con etiquetas o la versión limitada add (Component comp, Object obj). Por lo tanto, BorderLayout controla una lista de componentes almacenándolos directamente dentro de variables privadas. A continuación se le solicita al método `getChild()` que atrape los componentes apropiados de estas variables según la orientacidn del contenedor. (De esto se encarga el método `addLayoutComponent()`). El diseiio depende por completo del administrador. Todos los administradores han de tener en comun una cosa: el uso de los tamaños máximo, minimo y predeterminado de los componentes, para basar en ellos las decisiones. La posición y el tamaiio de los componentes se establecen con el método `setBounds()`.

Estos son los otros dos metodos de `LayoutManager`:

```
public Dimension preferredLayoutSize(Container parent)
public Dimension minimumLayoutSize(Container parent)
```

y este es otro para `LayoutManager2`:

```
public Dimension maximumLayoutSize(Container parent)
```

Los tres se parecen mucho a `layoutContainer()`, a excepción de que no tienen que diseñar los componentes. Su finalidad es determinar cuál seria el tamafio del contenedor si los elementos se construyesen a partir de su tamafio predeterminado, maximo o minimo. La lógica de estos metodos es prácticamente idéntica a la de `layoutContainer()`, exceptuando las llamadas `setSize()` y `setBounds()`. Además, el método `minimumLayoutSize()` invoca al método `getMinimumSize()` de los componentes en vez de calcular su tamafio, mientras que `maximumLayoutSize()` llama a `getMaximumSize()`.

Resumen

El concepto de administrador de diseiio confunde a muchas personas. Si no lo acaba de comprender del todo, quizás le ayude desarrollar GUI con un Entorno de desarrollo integrado (IDE) como la herramienta JBuilder que aparece en el CD de este libro. Esto no solamente elimina la necesidad de escribir manualmente el código para diseiiar la pantalla; también simplifica mucho la tarea de la configuración de componentes. Por ejemplo, ¿recuerda el administrador `GridBagLayout`, que tenia 11 limitadores diferentes para cada componente? Cada una de estas propieda-

des delimitadoras tenia varias constantes de clase diferentes que podia ser usada por sus valores. En lugar de tener que recordar todas las diferentes relaciones, puede usar una interfaz sencilla como la de la figura 12.23 para establecer las limitaciones de cada componente. Esta ventana de opciones incluso'llega a los valores de `Insets`, hasta sus cuatro componentes.

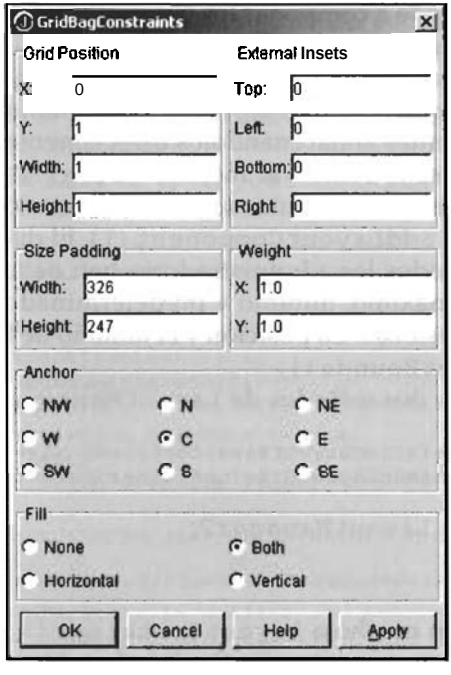
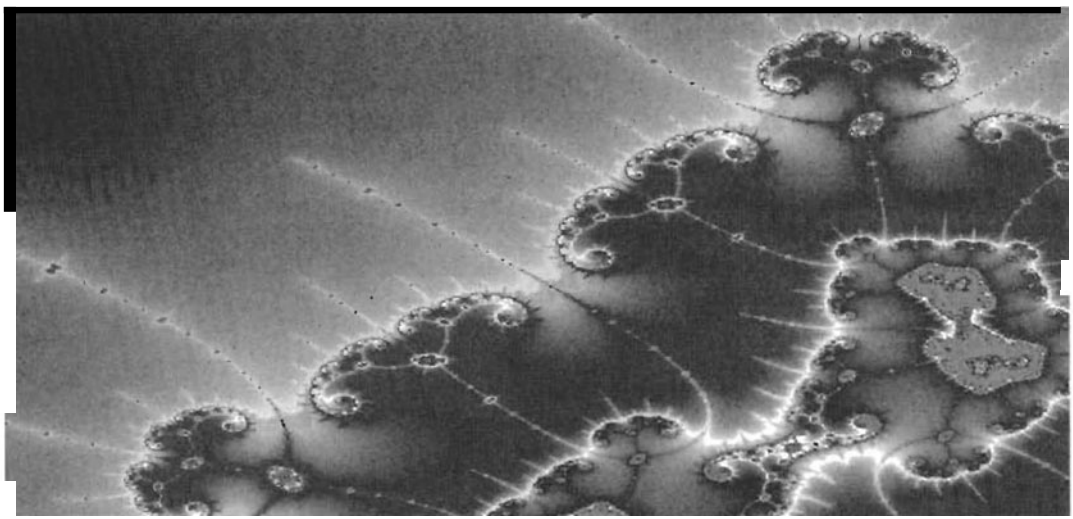


Figura 12.23. Trabajar con GridBagLayout en JBuilder

A mi me gusta JBuilder, pero esta herramienta no es la solución definitiva. Sin embargo, cuando cree interfaces de usuario, trabajar con cualquier herramienta IDE le simplificará bastante las cosas. Considere el usar cualquiera de las siguientes: VisualAge para Java de IBM, JBuilder de Borland, Forte para Java de Sun y VisualCafé de WebGain. Hay otras IDE disponibles, pero estas cuatro parecen las más populares y completas.



13

Componentes Swing para GUI

Las modernas interfaces GUI cuentan con una gran cantidad de elementos gráficos cuya finalidad es la de mejorar la productividad del usuario todo lo posible. Los botones y los menús siguen siendo elementos básicos, pero la verdad es que estan apareciendo nuevos modelos. Por ejemplo, hemos pasado de los clásicos botones, menus de comandos, los botones de opción, los menus emergentes, las barras de desplazamiento a otros elementos más complejos, como las listas y la vista del árbol de directorios, barras de herramientas y ventanas donde se muestra el estado de una operación. AWT de Java contiene dos conjuntos de componentes. El primero es una mezcla de los mas sencillos, procedentes de las versiones 1.0 y 1.1. El segundo conjunto, Swing, apareció con Java 1.2 y ha seguido mejorando. Este conjunto sustituye a los componentes AWT con sus propios elementos aún más complejos.

Nota

Aunque el conjunto de componentes Swing se presentó con la version 1.2 de Java 2 SDK, las versiones 1.2 se pueden usar con la version 1.1 de Java. Dado que no se incluian en la version 1.1, hay que obtener los componentes 1.2 por separado desde http://java.sun.com/products/jfc/. Y hay que hacer una importante advertencia al usarlos: Los componentesque se pueden usar con 1.1 son exactamente eso, la version 1.2 de los componentes. Se han realizado muchos cambios desde entonces a las bibliotecas Swing, incluyendo nuevos componentes y soluciones de errores. Estas correcciones y nuevos elementos no están disponibles en la versión 1.1.

Nota

Si estamos creando aplicaciones para el mundo real, deberiamos evitar la utilización de componentes **AWT** y centrarnos en el conjunto Swing

En este capitulo veremos algunos de los componentes Swing, como por ejemplo botones, etiquetas, botones de activación/desactivacidn, cuadro de diálogo, listas, barras de desplazamiento, controles deslizantes, texto, cuadros cuyo contenido se mueve, tablas y barras de herramientas.

Para acceder a estos componentes tendremos que utilizar las clases Swing. Antes de proceder a estudiar cada uno de estos componentes, nos centraremos en el componente más grande de todos: la propia clase Component.

La superclase Component

Los elementos de las interfaces GUI tienen muchas cosas en común. Y cuando las clases tienen elementos en común, lo primero que nos viene a la cabeza es la palabra "abstracto". La metodologia orientada a objetos requiere que haya una superclase que englobe a cualquier grupo de clases que identifiquen y compartan una serie de caracteristicas.

Una superclase tiene el código y las estructuras de datos que en cualquier otro caso tendriamos que duplicar y repartir por todos los sistemas (con lo que afectaríamos de forma negativa a la robustez del código).

- Las clases GUI AWT tienen una superclase: la clase Component, que estudiaremos en este apartado.
- Tras eso, veremos `JComponent`, superclase de las clases GUI Swing. En realidad, es una subclase de Component a través de `container`. Veremos primero la clase Component y luego `JComponent`.

La figura 13.1 muestra la jerarquia de clases de estos componentes.

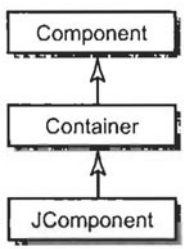


Figura 13.1. Jerarquia de las superclases Component/JComponent

La tabla 13.1 proporciona una lista de lo que cada componente tiene en comun con cada componente AWT (y Swing); en otras palabras, un resumen de los métodos de la clase `Component`.

Tabla **13.1.** Cornposicion de la clase Component

Variables/Método	Descripción
BOTTOM—ALIGNMENT	Valores de alinearniento horizontal y vertical
CENTER—ALIGNMENT	
LEFT—ALIGNMENT	
RIGHT—ALIGNMENT	
TOP—ALIGNMENT	
`Component` *()*	Crea un objeto **`Component`**
`add()`	Asocia un menú desplegable al cornponente
`addComponentListener()`	Asocia un receptor de eventos al cornponente
`addFocusListener()`	
`addHierarchyBoundsListener()`	
`addHierarchyListener()`	
`addInputMethodListener()`	
`addKeyListener()`	
`addMouseListener()`	
`addMouseMotionListener()`	
`addMouseWheelListener()`	
`addPropertyChangeListener()`	
`addNotify()`	Conecta un cornponente a un igual
`applyComponentOrientation()`	Aplica la orientación del cornponente a los componentes que contiene
`areFocusTraversalKeysSet()`	Cornprueba si se han establecido claves negativas
`checkImage()`	Cornprueba si se ha creado la irnagen asociada
`coalesceEvents()`	Puede hacer que el cornponente combine varios eventos
`contains``()`	Cornprueba si la posición especifica se encuentra dentro de un cornponente
`createImage()`	Crea una imagen

Variables/Método	Descripción
createVolatileImage()	Crea una irnagen volátil
disableEvents()	Desactiva el control de tipos de evento específicos
dispatchEvent()	Envía el evento al cornponente
doLayout()	Distribuye el cornponente
enableEvents()	Configura el tipo de eventos a los que responde el evento
enableInputMethods()	Establece la cornpatibilidad con el método de introducción
firePropertyChange()	Notifica el carnbio de propiedad a los oyentes de carnbio
getAccessibleContext()	Recupera el contexto actual
getAlignmentX()	Recupera el actual valor de alinearniento
getAlignmentY()	
getBackground()	Recupera el color con el que se dibuja en ese rnornento
getForeground()	
getBounds()	Recupera el rectangulo delimitador actual
getColorModel()	Recupera el rnodelo de color actual
getComponentAt()	Recupera el cornponente en la posición indicada del cornponente
getComponentListeners()	Recupera el conjunto de oyentes registrados
getFocusListeners()	
getHierarchyBoundsListeners()	
getHierarchyListeners()	
getInputMethodListeners()	
getKeyListeners()	
getListeners()	
getMouseListeners()	
getMouseMotionListeners()	
getMouseWheelListeners()	
getPropertyChangeListeners()	
getComponentOrientation()	Recupera la orientación actual

Variables/Método	Descripción
`getCursor()`	Recupera el cursor actual
`getDropTarget()`	Recupera el destino indicado actual
`getFocusCycleRootAncestor()`	Recupera el ascendiente del ciclo de foco
`getFocusTraversalKeys()`	Recupera el conjunto de claves negativas de foco actuales
`getFocusTraversalKeysEnabled()`	Comprueba si las claves negativas de foco estan activadas
`getFont()`	Recupera la fuente actual
`getFontMetrics()`	Recupera el tarnafio de fuente actual de la fuente
`getGraphics()`	Recupera el contexto grafico actual
`getGraphicsConfiguration()`	Recupera la configuración grafica actual
`getHeight()`	Recupera el tamaño actual
`getWidth()`	
`getSize()`	
`getIgnoreRepaint()`	Cornprueba si se deben pasar por alto las peticiones de volver a dibujar
`getInputContext()`	Recupera el contexto de entrada actual
`getInputMethodRequests()`	Recupera el controlador de peticiones de método de entrada
`getLocale()`	Recupera el lugar actual
`getLocation()`	Recupera la posición actual
`getLocationOnScreen()`	
`getX()`	
`getY()`	
`getMaximumSize()`	Recupera el tarnafio lirnite deseado
`getMinimumSize()`	
`getPreferredSize()`	
`getName()`	Recupera el nornbre
`getParent()`	Recupera el contenedor prirnario
`getToolkit()`	Recupera el conjunto de herramientas actual
`getTreeLock()`	Recupera el objeto de bloqueo de sincronizacion

Variables/Método	Descripción
hasFocus()	Comprueba si el componente tiene foco de entrada
imageUpdate()	Hace que el componente se dibuje de nuevo mientras se esta cargando la imagen asociada
invalidate()	Invalida lo que se muestra
isBackgroundSet()	Comprueba si se ha establecido un color de fondo
isCursorSet()	Comprueba si se ha establecido un cursor
isDisplayable()	Comprueba si el cornponente se puede mostrar
isDoubleBuffered()	Comprueba si esta activado el doble buffer
isEnabled()	Comprueba si el componente esta activo
isfocusable()	Comprueba si se puede enfocar
isFocusCycleRoot()	Comprueba si la raiz del foco es ciclica
isFocusOwner()	Comprueba si el foco tiene propietario
isFontSet()	Comprueba si se ha establecido la fuente
isForegroundSet()	Comprueba si se ha establecido un color de primer plano
isLightweight()	Comprueba si el componente es superficial
isOpaque()	Comprueba si el componente es transparente
isShowing()	Comprueba si se esta mostrando el componente
isValid()	Comprueba si el componente es valido
isVisible()	Comprueba si el componente es visible
list()	Escribe información del componente en el monitor
paint()	Dibuja el componente
paintAll()	
paramString()	Crea una cadena que representa el componente que mostrará toString()
prepareImage()	Inicia la imagen asociada para mostrarla
print()	Escribe el componente
printAll()	
processComponentEvent()	Procesa el evento

Variables/Método	Descripción
processEvent()	
processFocusEvent()	
processHierarchyBoundsEvent()	
processHierarchyEvent()	
processInputMethodEvent()	
processKeyEvent()	
processMouseEvent()	
processMouseMotionEvent()	
processMouseWheelEvent()	
remove()	Elimina la asociacion de menú desplegable
removeComponentListener()	Elimina la asociacion del oyente de eventos
removeFocusListener()	
removeHierarchyBoundsListener()	
removeHierarchyListener()	
removeInputMethodListener()	
removeKeyListener()	
removeMouseListener()	
removeMouseMotionListener()	
removeMouseWheelListener()	
removePropertyChangeListener()	
removeNotify()	Elimina la asociacion del componente con el igual nativo
repaint()	Dibuja de nuevo la visualización del componente
requestFocus()	Solicita que el componente consiga el foco de entrada
requestFocusInWindow()	
setBackground()	Cambia el color de fondo
setBounds()	Cambia el tamaño ylo la posición del componente
setLocation()	
setSize()	

Variables/Método	Descripción
`setComponentOrientation()`	Cambia la orientación del componente
`setCursor()`	Cambia el cursor sobre el componente
`setDropTarget()`	Cambia el objetivo indicado registrado para el componente
`setEnabled()`	Cambia el estado activado del componente
`setFocusable()`	Cambia el estado enfocable del componente
`setFocusTraversalKeys()`	Cambia la clave negativa para la operación negativa especifica dentro del componente
`setFocusTraversalKeysEnabled()`	Cambia si el componente responde a las claves negativas de foco
`setFont()`	Cambia la fuente para el componente
`setForeground()`	Cambia el color en primer plano para el componente
`setIgnoreRepaint()`	Cambia si el componente no hace caso de las peticiones de dibujado de nuevo
`setLocale()`	Cambia el lugar del componente
`setName()`	Cambia el nombre del componente
`setVisible()`	Cambia la visibilidad del componente
`toString()`	Devuelve la representación de cadena del estado del componente
`transferFocus()`	Mueve el foco
`transferFocusBackward()`	
`transferFocusUpCycle()`	
`update()`	Dibuja de nuevo la muestra del componente
`validate()`	Valida el componente

Una colección de métodos bastante grande, ¿verdad? Se debe a que de esta forma podemos modificar el comportamiento de los componentes Swing y AWT. Es decir, aunque la clase `Component` es enorme, la mayoría de los programas utilizan muy pocos métodos. La mayoría los invoca automáticamente el sistema, no el programador. Aun asi, la clase `Component` es básica en la programación con AWT y Swing. Tanto que estudiaremos los métodos de este grupo en orden. Las siguientes secciones están dedicadas a estos métodos. Recordemos que todos los componentes (es decir, todas las clases que descienden de `Component` y `JComponent`) pueden trabajar con ellos.

Nota

La clase Component implementa la interfaz `ImageObserver`, que únicamente especifica un método, `imageUpdate()`.

Métodos de control de eventos

El grupo más importante de métodos de Component es el especializado en el control de eventos. El listado 13.1 contiene los mecanismos de control de Java. Estos métodos tratan con los oyentes de eventos o, más concretamente, los oyentes responden a un evento cuando tiene lugar. Los oyentes de eventos forman parte del modelo de delegados de eventos de Java:

Listado 13.1. Métodos controladores de eventos

```
// Agrega oyentes
public void addComponentListener(ComponentListener l)
public void addFocusListener(FocusListener l)
public void addHierarchyBoundsListener(HierarchyBoundsListener l)
public void addHierarchyListener(HierarchyListener l)
public void addInputMethodListener(InputMethodListener l)
public void addKeyListener(KeyListener l)
public void addMouseListener(MouseListener l)
public void addMouseMotionListener(MouseMotionListener l)
public void addMouseWheelListener(MouseWheelListener l)
// Elimina oyentes
public void removeComponentListener(ComponentListener l)
public void removeFocusListener(FocusListener l)
public void removeHierarchyBoundsListener(HierarchyBoundsListener l)
public void removeHierarchyListener(HierarchyListener l)
public void removeInputMethodListener(InputMethodListener l)
public void removeKeyListener(KeyListener l)
public void removeMouseListener(MouseListener l)
public void removeMouseMotionListener(MouseMotionListener l)
public void removeMouseWheelListener(MouseWheelListener l)
// Notifica oyentes
protected void processComponentEvent(ComponentEvent e)
protected void processEvent(AWTEvent e)
protected void processFocusEvent(FocusEvent e)
protected void processHierarchyBoundsEvent(HierarchyEvent e)
protected void processHierarchyEvent(HierarchyEvent e)
protected void processInputMethodEvent(InputMethodEvent e)
protected void processKeyEvent(KeyEvent e)
protected void processMouseEvent(MouseEvent e)
protected void processMouseMotionEvent(MouseEvent e)
protected void processMouseWheelEvent(MouseWheelEvent e)
// Obtiene oyentes
public ComponentListener[] getComponentListeners()
public FocusListener[] getFocusListeners()
```

```
public HierarchyBoundsListener[] getHierarchyBoundsListeners()
public HierarchyListener[] getHierarchyListeners()
public InputMethodListener[] getInputMethodListeners()
public KeyListener[] getKeyListeners()
public EventListener[] getListeners(Class listenerType)
public MouseListener[] getMouseListeners()
public MouseMotionListener[] getMouseMotionListeners()
public MouseWheelListener[] getMouseWheelListeners()
// Admite el control de eventos
protected AWTEvent coalesceEvents(AWTEvent existingEvent, AWTEvent
newEvent)
public void dispatchEvent(AWTEvent e)
protected void disableEvents(long eventsToDisable)
protected void enableEvents(long eventsToEnable)
```

Los metodos de oyente para cambiar propiedades (listado 13.2) parecen métodos para controlar eventos; sin embargo, están más relacionados con los JavaBeans.

Listado 13.2. Oyentes para carnbiar propiedades

```
public void addPropertyChangeListener(PropertyChangeListener 1)
public void addPropertyChangeListener(String propertyName,
      PropertyChangeListener 1)
public void removePropertyChangeListener(PropertyChangeListener l)
public void removePropertyChangeListener(String propertyName,
      PropertyChangeListener l)
protected void firePropertyChange(String name, boolean oldValue,
      boolean newValue)
protected void firePropertyChange(String name, int oldValue, int newValue)
protected void firePropertyChange(String name, Object oldValue, Object
newValue)
public PropertyChangeListener[] getPropertyChangeListeners()
public PropertyChangeListener[] getPropertyChangeListeners(String name)
```

Métodos para mover componentes y modificar su tamaño

Los siguientes métodos de **Component** están relacionados con el cambio de tamaño de los elementos GUI:

```
public void setLocation(int x, int y)
public void setLocation(Point p)
public void setSize(int width, int height)
public void setSize(Dimension d)
public void setBounds(int x, int y, int width, int height)
public void setBounds(Rectangle r)
```

AWT tiene un modo novedoso de colocar y modificar el tamafio de los elementos GUI de una aplicación. Con Java, podemos dejar que sea el administrador de diseño (una clase) el encargado de dar a los componentes su aspecto final, asi como la posición y dimensiones. Los métodos que listamos en estas páginas se han dise-

```
public Color getBackground()
public Color getForeground()
public Toolkit getToolkit()
public void setFont(Font f)
public void setBackground(Color c)
public void setForeground(Color c)
pubiic boolean isFontSet()
public boolean isBackgroundSet()
public boolean isForegroundSet()
public GraphicsConfiguration getGraphicsConfiguration()
```

Métodos de los administradores de diseno

Algunos de los métodos de Component están pensados para los administradores de diseiio:

```
public Dimension getPreferredSize()
public Dimension getMinimumSize()
public Dimension getMaximumSize()
public void doLayout()
public float getAlignmentX()
public float getAlignmentY()
public ComponentOrientation getComponentOrientation()
public void setComponentOrientation(ComponentOrientation orientation)
public void applyComponentOrientation(ComponentOrientation orientation)
public Containe getParent()
```

Los métodos `getPreferredSize()`, `getMinimumSize()` y `getMaximumSize()` ya se vieron cuando estudiamos los administradores de diseiio. Con componentes AWT, tendremos que anular estos métodos para que los administradores de diseiio sepan qué tamaiio quiere que tenga un componente. Sin embargo, con componentes Swing, hay un método para establecerlo, de modo que no tiene que hacer al componente una subclase para ajustar los tamaiios.

El método `doLayout()` fuerza un diseiio inmediato para el componente. Nunca podremos invocar a este método por nuestra cuenta. De esto se encarga `validate()`.

Los administradores de diseiio utilizan los métodos `getAlignmentX()` y `getAlignmentY()` para ubicar los componentes con respecto al resto de elementos que se encuentran en el mismo área.

Los métodos `getComponentOrientation()`, `setComponentOrientation()` y `applyComponentOrientation()` se utilizan para los casos en los que hay que modificar la orientación del texto a causa del idioma utilizado y tienen cierto efecto sobre los componentes que controla el administrador de diseiio.

MCtodos para la administración de focos

El subsistema de administración de focos necesita mucha ayuda por parte de los componentes subyacentes:

ñado para que los utilicen los administradores de diseiio en vez de los usuarios. Es la regla generalizada, aunque podemos utilizar los métodos equivalentes de la clase Window y sus descendientes. Los administradores de diseiio no determinan dónde se va a colocar una ventana ni el tamaiio que les dará el usuario (su jurisdicción se limita al contenido de la ventana y no a la posición de ésta en la pantalla). El tamaiio de la ventana y su posición en la pantalla, es problema nuestro.

MCtodos de consulta de ubicacion y geometría

Los siguientes métodos de Component están relacionados con la consulta de la ubicación y la geometria de los componentes GUI. Dado que los administradores de diseiio tienen tanto control sobre la colocación y el tamaiio final de los componentes, estos métodos le descubrir qué han decidido los administradores de diseiio sobre la posición y el tamaiio, una vez que se ha realizado el diseiio.

```
public Rectangle getBounds()
public Rectangle getBounds(Rectangle returnValue)
public Point getLocation()
public Point getLocation(Point returnValue)
public Point getLocationOnScreen()
public Dimension getSize()
public Dimension getSize(Dimension returnValue)
public int getHeight()
public int getWidth()
public int getX()
public int getY()
public boolean contains(int x, int y)
public boolean contains(Point p)
public Component getComponentAt(int x, int y)
public Component getComponentAt(Point p)
```

Los métodos `getBounds()`, `getLocation()` y `getSize()` que tienen un pardmetro, guardan el valor que obtienen a partir del objeto que se les ha entregado en un objeto colocado previamente pasado como pardmetro, en lugar de colocar nuevos objetos en la pila. De esta forma se mejora considerablemente el rendimiento del sistema cuando se le llama con mucha frecuencia.

Métodos graficos y de representación

Los siguientes métodos de Component están relacionados con los gráficos y los componentes gráficos. Estos métodos tienen que ver con los atributos de dibujo del componente, como la fuente usada para cualquier representación de texto y los colores de fondo y en primer plano del componente.

```
public ColorModel getColorModel()
public Font getFont()
public FontMetrics getFontMetrics(Font font)
public Graphics getGraphics()
```

```
public boolean areFocusTraversalKeysSet(int id)
public Container getFocusCycleRootAncestor()
public Set getFocusTraversalKeys(int id)
public boolean getFocusTraversalKeysEnabled()
public boolean hasFocus()
public boolean isFocusable()
public boolean isFocusCycleRoot(Container container)
public boolean isFocusOwner()
public void requestFocus()
protected boolean requestFocus(boolean temporary)
public boolean requestFocusInWindow()
protected boolean requestFocusInWindow(boolean temporary)
public void setFocusable(boolean focusable)
public void setFocusTraversalKeys(int id, Set strokes)
public void setFocusTraversalKeysEnabled(boolean enabled)
public void transferFocus()
public void transferFocusBackward()
public void transferFocusUpCycle()
```

Si llega a Java **1.4** desde una versidn anterior, cuidado, la versidn **1.4** de SDK modifica la estructura de administracidn de foco considerablemente, de modo que asegurese de familiarizarse con la nueva distribucidn. Si antes creia saber lo que estaba pasando, ya no esté tan seguro.

Métodos de dibujo automático

Los siguientes métodos de Component son los encargados del dibujo automático:

```
public void paint(Graphics g)
public void paintAll(Graphics g)
public void repaint()
public void repaint(int x, int y, int width, int height)
public void repaint(long tm)
public void repaint(long tm, int x, int y, int width, int height)
public void print(Graphics g)
public void printAll(Graphics g)
public void update(Graphics g)
public boolean getIgnoreRepaint()
public void setIgnoreRepaint(boolean ignoreRequest)
```

Aqui, los métodos `paint()`, `update()` y `repaint()` son los más importantes. Todos los componentes Swing se dibujan automdticamente utilizando el método anuladopaint() de JComponent, que invoca a otros dos métodos: `paintComponent()` y `paintBorder()`.

Los metodos `repaint()` se utilizan para llamar a entidades externas (por ejemplo, el explorador) y para solicitar a Component que vuelva a dibujar en la pantalla. Una posible excepción la tendriamos en la variante `repaint()` con una lista de argumentos vacia. Podemos invocarla y obligarle a que utilice el método `update()` para actualizar el contenido de la ventana (es decir, que utilizard `paint()`), para que tenga lugar lo antes posible.

Métodos para el carnbio y la solicitud de estado

Los métodos de `Component` para el carnbio y la solicitud de estado son los siguientes:

```
public void setEnabled(boolean cond)
public void setVisible(boolean condj
public void validate()
public void invalidate()
public boolean isValid()
public boolean isVisible()
public boolean isShowing()
public boolean isEnabled()
public boolean isDisplayable()
public boolean isDoubleBuffered()
public boolean isLightweight()
public boolean isOpaque()
```

Cualquier elemento `Component` podrá encontrarse en cualquiera de estos estados: activado/desactivado, en pantalla/oculto, vrilido/no válido.

- Desactivamos los componentes cuando no queremos que el usuario los utilice en ciertas selecciones.
- También podremos ocultarlo para que no se acceda a él durante ciertos periodos de tiempo (desactivar un componente, implica que no se podrá acceder a él durante cierto período de tiempo).
- Por último, podemos invalidar componentes después de que modificarlos agregando componentes (con el método `add()`) o eliminarlos (con `remove()`). De esta forma, el administrador de diseiio del contendor del componente tendrri que volver a trabajar en el diseiio de los componentes del contenedor. Para validar el estado, tendremos que volver a llamar al `metodo validate()` y hacer que la pantalla se actualice en consecuencia. Un componente serri visible cuando aparece en la ventana.

Todos los componentes Swing son muy ligeros. No son componentes AWT que representan los elementos de la plataforma del usuario. Ademris, todos los Swing utilizan un doble buffer para dibujar, mientras que los componentes AWT no.

Un componente opaco es aquel que no permite que se vea lo que tiene detrris. Los elementos que se encuentran detrris de un elemento opaco quedan ocultos. Si un elemento no es opaco, se verá el fondo de la ventana en las zonas del componente donde no se haya dibujado.

Métodos relacionados con las imigenes

Los métodos que mostramos a continuación son los de la clase `Component` que estrin relacionados con las imágenes. Se pueden dibujar imágenes en componentes

como `JApplet` y `JFrame` y estos serán algunos de los métodos necesarios para crear y cargar imagenes.

```
public Image createImage(ImageProducer producer)
public Image createImage(int width, int height)
public VolatileImage createVolatileImage(int width, int height)
public VolatileImage createVolatileImage(int width, int height,
       ImageCapabilities caps)
public boolean imageUpdate(Image image, int flags, int x, int y, int w, int h)
public boolean prepareImage(Image image, int width, int height,
       ImageObserver observer)
public boolean prepareImage(Image image, ImageObserver observer)
public int checkImage(Image image, int width, int height,
       ImageObserver observer)
public int checkImage(Image image, ImageObserver observer)
```

Con la mayoría de los componentes Swing, podemos mostrar una imagen dentro de cada componente. Para ello necesitaremos la ayuda de la interfaz `Icon` y de la clase `ImageIcon` y no con estos metodos relacionados con imágenes.

Metodos relacionados con los metodos de entrada

El contexto y los metodos de entrada describen los mecanismos de comunicacion que tienen los componentes de texto. Se suelen utilizar con los idiomas cuyos caracteres no se pueden introducir a través de un teclado o no son alfabéticos. Los métodos que mostramos a continuación pertenecen a la clase `Component` y son los que están relacionados con los metodos de entrada:

```
public InputContext getInputContext()
public InputMethodRequests getInputMethodRequests()
public void enableInputMethods(boolean b)
```

Métodos de menú desplegable

Los componentes pueden tener menus desplegables o sensibles al contexto, asociados.

```
public void add(PopupMenu popup)
public void remove(MenuComponent popup)
```

Los objetos Swing `JComponent` también pueden tenerlos, pero por medio de un `JPopupMen`.

MCtodos para la comunicacion de igual a igual

La clase `Component` tiene dos métodos pensados para las comunicaciones:

```
public void addNotify()
public void removeNotify()
```

Todos los componentes tienen algunos metodos que les permiten comunicarse con su igual. Ya hemos visto que los componentes AWT se apoyan en los componentes nativos para conservar el aspecto de la plataforma y trabajar con las interfaces GUI. Los componentes Swing no se basan en iguales; sin embargo, los últimos contenedores de componentes, como `JFrame` o `JWindow`, aún confían en el control nativo de la plataforma.

Advertencia

Mantener la sincronizacion entre dos sistemas equivalentes es bastante difícil y arriesgado. Si las entidades pierden esta sincronia por alguna razon (por ejemplo, un error de programacion), todo el sistema se vendrá abajo. Swing se creo basicamente debido a los problemas de compatibilidad producidos por una implementación pobre entre los iguales de **AWT.** Como la mayoría de los componentes Swing no utilizan el sistema de comunicación de igual a igual de **AWT,** ya no tenemos ningun problema con la incompatibilidad de los sistemas en Java.

Los metodos como `addNotify()` y `removeNotify()` se utilizan para crear y eliminar los extremos de una comunicación de igual a igual. Los programadores de aplicaciones ya no suelen invocar a los metodos relacionados con estos tipos de comunicación. Son metodos de muy bajo nivel que utilizan los componentes para contactar con sus iguales. En concreto, si se llama a `addNotify()` o `removeNotify()` se interrumpirá el protocolo entre componentes abstractos/concretos, con lo que nos encontraremos con problemas de sincronia.

Y el resto

Los mktodos restantes de Component no se ajustan a la clasificacion general. Muchos de ellos estan disponibles para ayudarle en la eliminación de errores, como los mktodos `list()` que mostraran información sobre el componente. El método `DropTarget` tiene que ver con la compatibilidad para arrastrar y colocar. `Locales` esta relacionado con los idiomas.

```
public AccessibleContext getAccessibleContext()
public Cursor getCursor()
public boolean isCursorSet()
public void setCursor(Cursor cursor)
public DropTarget getDropTarget()
public void setDropTarget(DropTarget target)
public Locale getLocale()
public void setLocale(Locale locale)
public String getName()
public void setName(String name)
public void list()
```

```
public void list(PrintStream out)
public void list(PrintStream out, int indent)
public void list(PrintWriter out)
public void list(PrintWriter out, int indent)
protected String paramString()
public String toString()
public Object getTreeLock()
```

La superclase Swing de las clases GUI

Aunque `java.awt.Component` es la superclase de todos los componentes AWT (incluyendo el conjunto de componentes Swing), la clase `java.swing.JComponent` es más adecuada cuando se trabaja con los componentes Swing. La tabla 13.2 enumera los métodos y variables heredados de la clase **`Jcomponent`**:

Tabla 13.2. Composicidn de la clase JComponent

Variables/Método	Descripción
accessibleContext	Mantiene las propiedades internas del componente
listenerList	
ui	
TOOL—TIP—TEXT—KEY	Clave de la propiedad oyente para la propiedad del texto de ayuda emergente
UNDEFINED—CONDITION	Designa la condición de foco
WHEN—ANCESTOR—OF—FOCUSED—COMPONENT	
WHEN—FOCUSED	
WHEN—IN—FOCUSED—WINDOW	
`JComponent()`	Crea un objeto `JComponent`
`addAncestorListener()`	Asocia un evento oyente a un componente
`addPropertyChangeListener()`	
`addVetoableChangeListener()`	
`addNotify()`	Conecta un componente a su fuente
`contains()`	Comprueba si una posición especifica se encuentra en un componente
`createToolTip()`	Crea una ayuda de herramienta que se puede mostrar

Variables/Método	Descripción
`disable()` `enable()`	Cambia el estado de activación
`firePropertyChange()`	Inforrna a los oyentes de carnbios de propiedad de un carnbio
`fireVetoableChange()`	Inforrna a los oyentes de carnbios que se pueden vetar de un carnbio
`getAccessibleContext()`	Recupera el contexto actual
`getActionForKeyStroke()`	Busca una accion para una pulsación de tecla
`getActionMap()`	Recupera el rnapa de accion
`getAlignmentX()`	Recupera el valor actual de alinearniento
`getAlignmentY()`	
`getAncestorListeners()`	Recupera el conjunto de oyentes registrados
`getListeners()`	
`getPropertyChangeListeners()`	
`getVetoableChangeListeners()`	
`getAutoscrolls()`	Cornprueba si esta activado el autodesplazarniento
`getBorder()`	Recupera la zona alrededor de un cornponente para el borde
`getInsets()`	
`getBounds()`	Recupera el actual rectangulo delirnitador
`getClientProperty()`	Busca propiedades de cliente para la clave especificada
`get`*`ComponentGraphics()`*	Recupera el contexto grafico
`getConditionForKeyStroke()`	Busca una condición para una pulsación de tecla
`getDebugGraphicsOptions()`	Recupera opciones para depurar el contexto grafico
`getDefaultLocale()`	Recupera el lugar predeterrninado
`getGraphics()`	Recupera el contexto grafico actual
`getHeight()`	Recupera el tamaño del cornponente actual
`getSize()`	
`getWidth()`	
`getInputMap()`	Recupera el rnapa de entrada

Variables/Método	Descripción
`getInputVerifier()`	Recupera el verificador de entrada
`getLocation()`	Recupera la posición actual de un componente
`getX()`	
`getY()`	
`getMaximumSize()`	Recupera el límite de tamaño actual
`getMinimumSize()`	
`getPreferredSize()`	
`getRegisteredKeyStrokes()`	Recupera el conjunto de pulsaciones de tecla registrado
`getRootPane()`	Recupera el cuadro raiz de la fuente
`getToolTipLocation()`	Recupera la posición para el texto de la ayuda para herramientas
`getToolTipText()`	Recupera el texto de la ayuda para herramientas
`getTopLevelAncestor()`	Recupera el contenedor superior
`getTransferHandler()`	Recupera el controlador de arrastrar y colocar
`getUIClassID()`	Recupera el nombre de la clase que dibuja el componente
`getVerifyInputWhenFocusTarget()`	Recupera si se debe informar al verificador de entrada
`getVisibleRect()`	Recupera los limites visibles
`grabFocus()`	Recupera el foco de entrada
`isDoubleBuffered()`	Comprueba si esta activado el doble buffer
`isLightweightComponent()`	Comprueba si el componente es ligero
`isMaximumSizeSet()`	Comprueba si se ha inicializado el tamaño
`isMinimumSizeSet()`	
`isPreferredSizeSet()`	
`isOpaque()`	Comprueba si el componente es transparente
`isOptimizedDrawingEnabled()`	Comprueba si esta activado eldibujo optimizado; solo deberia estar activado si varios descendientes (componentes que contiene) no se pueden superponer
`isPaintingTile()`	Comprueba si el componente esta pintando un recuadro

Variables/Método	Descripción
isRequestFocusEnabled()	Cornprueba si un cornponente puede solicitar focos
isValidateRoot()	Cornprueba si este contenedor es raiz del arbol de cornponentes **revalidate()**
paint()	Dibuja un cornponente
paintImmediately()	
paintBorder()	Dibuja un fragmento de un componente
paintChildren()	
paintComponent()	
paramString()	Crea una cadena que representa un componente que debe rnostrar **toString()**
print()	Escribe el cornponente
printAll()	
printBorder()	Escribe un fragmento de un componente
printChildren()	
printComponent()	
processComponentKeyEvent()	Procesa un evento
processKeyBinding()	
processKeyEvent()	
processMouseMotionEvent()	
putClient**Property()**	Guarda el valor de la propiedad cliente
registerKeyboardAction()	Asocia una acción a la pulsación de una tecla
removeAncestorListener()	Elirnina la asociacion a un oyente de eventos
removePropertyChangeListener()	
removeVetoableChangeListener()	
removeNotify()	Elirnina la asociacion de un cornponente con su fuente
repaint()	Dibuja de nuevo el cornponente en pantalla
updateUI()	
requestFocus()	Solicita que un cornponente tenga foco de entrada
requestFocusInWindow()	
resetKeyboardActions()	Borra las acciones de teclado registradas

Variables/Método	Descripción
reshape()	Cambia el tamaño de un componente
revalidate()	Hace que se vuelvan a dibujar todos los componentes de un contenedor raiz
scrollRectToVisible()	Envía una petición a un contenedor
setActionMap()	Cambia el mapa de acciones
setAlignmentX()	Cambia la alineacion de los componentes
setAlignmentY()	
setAutoscrolls()	Cambia el valor del autodesplazamiento
setBackground()	Cambia el color de fondo
setBorder()	Cambia el borde
setDebugGraphicsOptions()	Cambia las opciones para depurar el contexto grafico
setDefaultLocale()	Cambia el lugar predeterminado para el contexto
setDoubleBuffered()	Cambia el estado de doble buffer
setEnabled()	Cambia el estado activado
setFont()	Cambia la fuente
setForeground()	Cambia el color en primer plano
setInputMap()	Cambia el mapa de pulsaciones de tecla de entrada
setInputVerifier()	Cambia el verificador de entrada
setMaximumSize()	Cambia el limite del tamaño deseado
setMinimumSize()	
setPreferredSize()	
setOpaque()	Cambia la transparencia del componente
setRequestFocusEnabled()	Cambia si el componente puede solicitar foco de entrada
setToolTipText()	Cambia el texto para la ayuda de herramientas
setTransferHandler()	Cambia el controlador para arrastrar y colocar
set*UI*()	Cambia el controlador UI
setVerifyInputWhenFocusTarget()	Cambia si debe informarse al verificador de entrada
setVisible()	Cambia la visibilidad

Variables/Método	Descripción
`unregisterKeyboardAction()`	Elimina la pulsación de tecla registrada
`update()`	Pinta de nuevo el cornponente que aparece

Algunos de estos mdtodos nos resultarán familiares, ya que `JComponent` aprovecha las ventajas de Java para anular los mdtodos ya existentes y proporcionar un comportamiento más personalizado. Además, JComponent añade la capacidad de que todos los componentes especifiquen su propio borde, el texto y la interfaz del usuario. Especificar la interfaz de usuario de un componente es parte de las capacidades avanzadas de Swing.

Agregar componentes a la interfaz GUI

En las próximas secciones se describen algunos de los componentes Swing que podemos utilizar para agregar a las applet y aplicaciones de Java. Ademds, en esta sección describiremos las clases e interfaces `Icon`, `ImageIcon`, `AbstractButton`, `Border`, `BorderFactory`, `JToolTip`, `ToolTipManager` y `ButtonGroup`.

`JButton`	`JSlider`
`JPanel`	`JTextField`
`JLabel`	`JPasswordField`
`JToggleButon`	`JTextArea`
`JCheckBox`	`JEditorPane`
`JRadioButton`	`JSpinner`
`JComboBox`	`JFormattedTextField`
`JScrollPane`	`JToolBar`
`JList`	`JInternalFrame`
`JScrollBar`	`JTable`

Agregar botones

Si abrimos cualquier aplicacidn en un equipo moderno nos encontraremos con botones en su interfaz. Hay ciertos tipos de botones que se utilizan para manipular ventanas en las distintas plataformas:

- Botón para modificar el tamaño.
- Botones para minimizar y maximizar ventanas.
- Botón para cerrar

También nos podemos encontrar botones dentro de otros elementos. Las barras de herramientas son una colección de botones. Las barras de desplazamiento se componen (por lo menos) de tres botones: dos para las flechas hacia arriba y hacia abajo (o izquierda y derecha) y un botón alargado para la zona de desplazamiento. Las barras de estado también contienen botones camuflados como simples etiquetas. Independientemente de su apariencia exacta, el botón suele estar asociado a una ventana rectangular que tiene un único comportamiento asociado. A partir de esta definición, todo lo que tenemos que hacer es seleccionar esta zona rectangular dibujando su contorno y agregar una etiqueta descriptiva o un icono que lo identifique. La idea de un botón en Java es precisamente ésta: un borde, una etiqueta o un icono y algunos métodos que se encarguen de controlar los eventos. La tabla 13.3 muestra la composición de la clase.

Tabla **13.3.** Composición de la clase JButton

Variables/Método	Descripción
`JButton()`	Crea un objeto `JButton`
`configurePropertiesFromAction()`	Configura un componente basado en `Action`
`getAccessibleContext()`	Recupera el context0 actual
`getUIClassID()`	Recupera el nombre de la clase que dibuja el componente
`isDefaultButton()`	Comprueba si este boton esta designado corno opcion predeterminada
`isDefaultCapable()`	Comprueba este boton puede designarse como opcion predeterminada
`paramString()`	Crea una cadena que representa un componente quese mostrara mediante `toString()`
`removeNotify()`	Elirnina la asociacion de un componente con su fuente; si es necesario, comprueba que el boton predeterminado no este disponible
`setDefaultCapable()`	Cambia que el boton pueda ser designado como la opcion por defecto
`updateUI()`	Dibuja otra vez el componente en pantalla

La mayor parte de los detalles de `JButton` se heredan de la clase `AbstractButton`. Antes de estudiarla, veamos como usar `JButton`. Todo lo que tiene que saber es cómo usar los constructores:

```
public JButton()
public JButton(Icon icon)
public JButton(String text)
```

```
public JButton(String text, Icon icon)
public JButton(Action a)
```

El constructor más usado para un JButton recibe un String, que aparecerá en el botón como su etiqueta. La figura 13.2 muestra un ejemplo de un componente JButton.

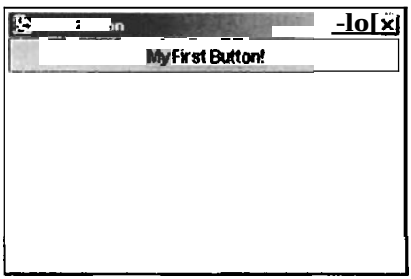


Figura 13.2. **Componente** de JButton

Agregar botones a applets o aplicaciones Java es fácil, como puede ver en el listado 13.3, que creó el botón de la figura 13.2.

Listado 13.3. Mostrar un boton con una etiqueta de texto

```
import java.awt.*;
import javax.swing.*;

public class ButtonTest {
    public static void main(String args[]) {
        JFrame frame = new ExitableJFrame("First Button");
        Container contentPane = frame.getContentPane();
        JButton b = new JButton("My First Button!");
        contentPane.add(b, BorderLayout.NORTH);
        frame.setSize(300, 200);
        frame.show();
```

Todo lo que hace este applet es crear un marco, obtener su cuadro de contenido, crear un nuevo objeto JButton y agregarlo (mediante el mktodo `add()`) al cuadro de contenido del marco.

La etiqueta del botón "My `First Button!`" se especificó al constructor JButton como argumento de cadena. Como puede ver en la figura 13.2, el boton se coloca en la parte superior de la pantalla. Esto se debe a que el administrador de diseño de los marcos es `BorderLayout` y utiliza la parte superior (north) como pariimetro del mktodo `add()`.

La primera declaración `import` hace que todas las clases `java.awt` estén disponibles para la unidad de compilación (el archivo fuente). En este caso, sólo tendria que usar las siguientes líneas en su lugar, porque en el programa sólo se usan clases AWT:

```
import java.awt.BorderLayout;
import java.awt.Container;
```

> **Nota**
>
> Al contrario de lo que se suele creer con Java, la importación de paquetes en Java no aumenta el tamaño de los ejecutables. La declaración `import` solo deja que el resto de clases puedan ver las definiciones con las que trabaja durante la compilacion. La vinculación se establece durante la ejecucion y no en la compilacion.

La segunda declaración import hace que se pueda trabajar con todas las clases `java.swing`. En este caso, el programa utiliza las clases `JApplet` y `JButton` de su paquete. Como ocurria con `BorderLayout`, se pueden importar:

```
import javax.swing.JButton;
import javax.swing.JFrame;
```

Con el método `setText()` (heredado de `AbstractButton`) podemos cambiar la etiqueta de un elemento `JButton` siempre que queramos. Del mismo modo, podemos averiguar el texto que aparece en un botón con `getText()`.

Mostrar imágenes en los botones

Para utilizar dos de los constructores de `JButton`, `public JButton(Icon icon)` y `public JButton(String label, Icon icon)`, tendremos que examinar la interfaz `Icon` que se especifica en estas líneas.

```
public interface Icon {
    public int getIconHeight();
    public int getIconWidth();
    public void paintIcon(Component c, Graphics g, int x, int y);
}
```

Para colocar una imagen dentro de un botón tenemos que suministrar una clase que trabaje con la interfaz `Icon`. Por ejemplo, el contenido de la figura 13.3 muestra un botón que tiene forma de tarta, con etiqueta y sin ella.

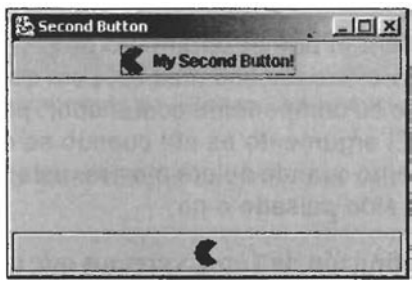


Figura 13.3. Dos cornponentes JButton con iconos

Listado 13.4. Presentacion de un boton con una etiqueta Icon

```
import java.awt.*;
import javax.swing.*;

public class ButtonIconTest {
    static class PieIcon implements Icon {
       Color color;
       public PieIcon(Color c) {
           color = c;
       }
       public int getIconWidth() {
           return 20;
       }
       public int getIconHeight() {
           return 20;
       }
       public void paintIcon(Component c, Graphics g, int x, int y) {
           g.setColor(color);
           g.fillArc(x, y, getIconWidth(), getIconHeight(), 45, 270);
       }
    }
    public static void main(String args[]) {
       JFrame frame = new ExitableJFrame("Second Button");
       Container contentPane = frame.getContentPane();
       Icon icon = new PieIcon(Color.red);
       JButton b = new JButton("My Second Button!", icon);
       contentPane.add(b, BorderLayout.NORTH);
       b = new JButton(icon);
       contentPane.add(b, BorderLayout.SOUTH);
       frame.setSize(300, 200);
       frame.show();
    }
}
```

En este programa, la clase `PieIcon` implementa la interfaz `Icon`. El icono tiene 20 píxeles de ancho (utilizamos el método `getIconWidth()`) y otros 20 de alto (utilizamos el método `getIconHeight()`). A continuación, el método `paintIcon()` describe cómo se va a dibujar el icono (en este caso, simplemente dibujando un arco relleno del color apropiado).

Truco

Normalmente se ignora el primer parametro de `paintIcon()`, Component. El componente proporciona al icono medios, para que un icono pueda conocer las caracteristicas de su componente contenedor; por ejemplo, el color con el que esta dibujado. El argumento es útil cuando se desea crear un icono que conoce su estado. como cuando quiere mostrar estados diferentes dependiendo de si el botón ha sido pulsado o no.

Si observamos la definición de `Icon`, veremos que no se hace ninguna mención a las imágenes. Podremos dibujar cualquier cosa como un `Icon` para un componente

Swing (siempre y cuando especifiquemos cómo se hace). Si queremos suministrar una imagen como **Icon** para un componente, podremos utilizar la clase `ImageIcon`. Con **ImageIcon**, todo lo que tenemos que hacer es saber cómo crearlo. Para ello tenemos estos nueve constructores:

```
public ImageIcon();
public ImageIcon(byte[] data);
public ImageIcon(byte[] data, String description);
public ImageIcon(Image image);
public ImageIcon(Image image, String description);
public ImageIcon(String filename);
public ImageIcon(String filename, String description);
public ImageIcon(URL location);
public ImageIcon(URL location, String description);
```

Truco

Podemos especificar una descripción para un **ImageIcon** que se utilizara con las nuevas tecnologias de asistencia para disminuidos fisicos, como los lectores de pantalla. Estas descripciones puntuales deben ser cortas y precisas; deben ayudar a las personas, no retardar su trabajo.

Podemos utilizar el siguiente código para crear el icono:

```
Icon icon = new ImageIcon("logo.gif", "Sybex Corporate Logo");
```

Una vez que tenemos el objeto `Icon`, podemos asociarlo a un componente que se encargue de mostrar una imagen con un icono dentro de un componente Swing. En la figura **13.4** tenemos un archivo que tiene el icono de un botón, con etiqueta y sin ella y el listado **13.5** es el programa que lo crea.

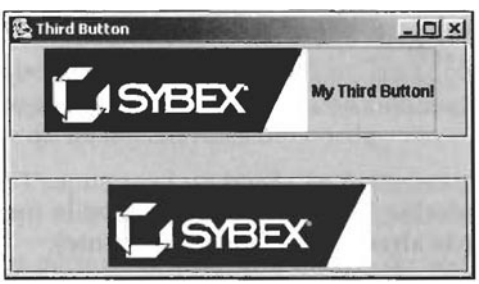


Figura 13.4. Cornponente de JButton con una irnagen corno icono

Listado 13.5. Mostrar un boton con una irnagen

```
import java.awt.*;
import javax.swing.*;
public class ButtonImageTest {
```

```
    public static void main(String args[]) {
        JFrame frame = new ExitableJFrame("Third Button");
        Container contentPane = frame.getContentPane();
        Icon icon = new ImageIcon("logo.gif", "Sybex Corporate Logo");
        JButton b = new JButton("My Third Button!", icon);
        contentPane.add(b, BorderLayout.NORTH);
        b = new JButton(icon);
        contentPane.add(b, BorderLayout.SOUTH);
        frame.setSize(350, 200);
        frame.show();
    }
}
```

Truco

Cuando utilicemos `ImageIcon`, **no nos tendremos que preocupar de la asincronia propia del proceso de carga de Image. Una imagen que se vaya a mostrar en un elemento** `ImageIcon` **tendra una carga sincronizada (es decir, nunca nos encontraremos con una imagen que se haya cargado a medias). De hecho, la imagen podría ser animada.**

Definición dc hordes y contornos para los botones

Como veremos en breve, en Swing, cuando no nos guste el aspecto de algo, podremos cambiarlo. Por ejemplo, podemos volver a definir el tipo de borde que aparecerá alrededor de los botones de cualquier subclase `JComponent`. Con la excepción de la clase `BorderFactory` que se encuentra dentro del paquete `javax.swing`, todos los bordes se encuentran dentro del paquete `javax.swing.border`. La figura 13.5 muestra la jerarquia de las clases de bordes.

La interfaz `Border` es la base de todos los bordes:

```
public interface Border {
    public Insets getBorderInsets(Component c);
    public boolean isBorderOpaque();
    public void paintBorder(Component c, Graphics g, int x, int y, int w,
int h);
}
```

Al igual que la interfaz `Icon`, `Border` describe la forma en que se escribe algo (en este caso, un borde alrededor de un componente).

- Recordará que el capitulo 12 se estudió a los insets como la zona en la que un administrador de diseño no coloca componentes. Respecto a los bordes, el valor devuelto `getBorderInsets()` describe exactamente dónde se dibujará el borde.
- El método `isBorderOpaque()` determina si el borde ocultará el color del fondo por completo o no.

- El método `paintBorder()` hace las funciones de la interfaz, describiendo cómo se ha de dibujar el borde.

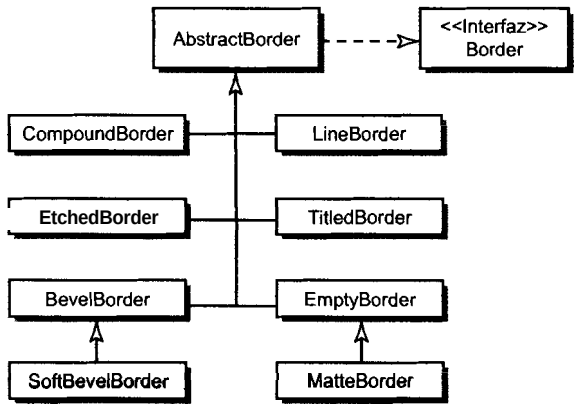


Figura 13.5. Árbol de jerarquías de la clase javax.swing.border

En la mayoría de los casos no hará falta que implementemos el borde nosotros mismos. Todo lo que tenemos que hacer es utilizar uno de los métodos ya existentes:

- **BevelBorder:** Utiliza líneas de distintos colores para crear una sensación tridimensional.
- **CompoundBorder:** Nos permite combinar dos bordes para crear un tercero.
- **EmptyBorder:** Borde que sólo ocupa un espacio.
- **EtchedBorder:** Proporciona un borde que parece tiene cierta profundidad.
- **LineBorder:** Nos permite especificar cualquier color o grosor para el borde que vamos a crear.
- **MatteBorder:** Un borde que nos permite contornear cualquier elemento. Con un color sólido, se diferencia de `LineBorder` en que puede tener distintos tamaños en cada uno de los laterales del borde.
- **SoftBevelBorder:** Proporciona un borde en 3 dimensiones donde los colores utilizados no tienen un contrataste tan fuerte como con `BevelBorder`.
- **TitledBorder:** Nos permite colocar una cadena de texto en el borde.

Además, cuando trabajamos con bordes, nunca tendremos que conocer la clase de borde especifica con la que estamos trabajando. Todo lo que hemos de hacer es pedirle a `BorderFactory` que cree el borde apropiado:

```
Border b = BorderFactory.createRaisedBevelBorder();
aJComponent.setBorder(b);
```

Truco

Al utilizar `BorderFactory` (en vez de especificar directamente la clase del borde con el que vamos a trabajar), el método puede optimizar el comportamiento del borde. En muchos casos, si se solicita varias veces el mismo borde, el borde se creara una unica vez. Cada vez que se solicita un nuevo borde se le pasa la referencia al borde anterior.

Para ver como funcionan los bordes nos podemos fijar en la figura 13.6, donde tenemos un icono que hace las funciones de borde y una etiqueta. Hemos utilizado el mismo `ImageIcon` queen la figura 13.4. El listado 13.6 esla fuente para la figura 13.6.

Figura 13.6. Un cornponente JButton rodeado por un MatteBorder

Listado 13.6. Mostrar un boton con un borde

```
import java.awt.*;
import javax.swing.*;
import javax.swing.border.*;

public class ButtonBorderTest {
    public static void main(String args[]) {
        JFrame frame = new ExitableJFrame("Fourth Button");
        Container contentPane = frame.getContentPane();
        Icon icon = new ImageIcon("logo.gif");
        JButton b = new JButton("My Fourth Button!");
        Border bored = BorderFactory.createMatteBorder(10, 5, 10, 5, icon);
        b.setBorder(bored);
        contentPane.add(b, BorderLayout.CENTER);
        frame.setSize(350, 200);
        frame.show();
    }
```

Cuando ejecutamos el programa, vemos que no tenemos exactamente lo que nos esperábamos. Tenemos el mismo borde tanto si seleccionamos el botón, como si no. Para la mayoría de los elementos `JComponent`, este efecto no es ningún problema. Siempre tendremos el mismo borde. Pero con `JButton` implementaremos una

interfaz Border que ofrezca dos bordes, uno para el botón en estado de reposo y otro que aparecerd cuando se pulse.

AbstractButton

La clase principal de JButton es AbstractButton. La clase AbstractButton también es una superclase de JMenuItem y sus clases secundarias (los componentes de los menus), así como de JToggleButton y sus subclases JCheckBox y JRadioButton. Bdsicamente, para comprender cualquiera de estas clases tendremos que comprender cómo funciona la compartición de AbstractButton.

Mientras que varios de estos métodos están relacionados con el control de eventos, merece la pena que nos fijemos en tres de ellos:

- **Iconos:** Estos métodos incluyen `setDisabledIcon()`, `setDisabledSelectedIcon()`, `setIcon()`, `setPressedIcon()`, `setRolloverIcon()`, `setRolloverSelectedIcon()` y `setSelectedIcon()`.
- **Ubicación:** Podemos determinar la posición del texto y de los iconos en cualquier punto del botón: `setHorizontalAlignment(int alignment)`, `setHorizontalTextPosition(int textPosition)`, `setVerticalAlignment(int alignment)` y `setVerticalTextPosition(int textPosition)`.
- **Nemotecnicos:** Podemos asociar atajos de teclado para los botones mediante `setMnemonic(char mnemonic)` y `setMnemonic(int mnemonic)`. Y, si el mismo carácter está presente varias veces y no quiere que se señale el primer caso, use `setDisplayedMnemonicIndex(int index)`,dondeindex es la posición del principio del texto de la etiqueta.

Cambiar los iconos de un botón

Cuando trabajamos con cualquiera de los elementos de AbstractButton, podremos especificar un icono para cada caso del botón. Los estados que se utilicen dependerán de una subclase en particular. Por ejemplo, con JButton, el método que se encargará de la configuración predeterminada del botón será `setIcon()`. Este es el icono que se utilizard con los botones que se limitan a aparecer en la pantalla sin hacer nada, mientras que el usuario trabaja en otra aplicación. Por otro lado, si queremos que aparezca otro icono cuando hacemos clic sobre el botón, tendremos que utilizar el método `setPressedIcon()`. Incluso tenemos un método para cambiar el aspecto del botón cuando el cursor se encuentra sobre el botón. Se trata de `setRolloverIcon()`. Pero antes de nada tendremos que activar este comportamiento, para lo cual utilizaremos el método `setRolloverEnable()` ya que tendremos que determinar la nueva posición del cursor cada vez que se mueve el ratón. Los otros métodos de los iconos funcionan de forma similar, pero responden a distintos comportamientos.

Por otra parte, si no quiere iconos distintos para cada uno de los tres estados descritos anteriormente, puede crear un icono que responda a su estado. El método `paintIcon()` de la interfaz **`Icon`** pasa el `JComponent`. Puede preguntar al componente cual es su estado actual y basarse en un sólo icono. La elección del mecanismo dependerá del tipo de icono con el que esté trabajando. Si el icono es una imagen, puede tener diferentes imágenes de cada estado. Si el icono se dibuja de forma manual, puede tener un icono dibujado de forma ligeramente diferente para cada estado. La figura 13.7 nos muestra los cambios que se pueden aplicar sobre un icono `JButton`. Tenemos tres estados diferentes de un programa. Se dibujarán formas, y no colores, para cada uno de los estados. El icono base muestra un triángulo que apunta hacia arriba. Cuando pulsamos el botón, el tridngulo mirará hacia abajo. Y cuando pasamos el cursor por encima, el triángulo apuntará a la derecha.

Figura 13.7. Un elemento JButton con distintos iconos

El listado 13.7 es el programa de ejemplo de la figura 13.7.

Listado 13.7. Vista de un boton con diferentes iconos

```
import java.awt.*;
import javax.swing.*;

public class ChangingButtonIconTest {
    static class TriangleIcon implements Icon {
        String name;
        static class State {
            public static final int NORMAL    = 0;
            public static final int PRESSED   = 1;
            public static final int ROLLOVER  = 2;
            private State() {}
        }
        int state;
        Color color;
```

```
        public TriangleIcon(Color c, int state) {
            color = c;
            this.state = state;

        public int getIconWidth() {
            return 20;

        public int getIconHeight() {
            return 20;
        }
        public void paintIcon(Component c, Graphics g, int x, int y) {
            g.setColor(color);
            Polygon p = new Polygon();
            if (state == State.NORMAL) {
               p.addPoint(x+(getIconWidth() / 2), y);
               p.addPoint(x, y+getIconHeight()-1);
               p.addPoint(x+getIconWidth()-1, y+getIconHeight()-1);
            } else if (state == State.PRESSED) {
               p.addPoint(x, y);
               p.addPoint(x+getIconWidth()-1, y);
               p.addPoint(x+(getIconWidth() / 2), y+getIconHeight()-1);
            } else {
               p.addPoint(x, y);
               p.addPoint(x, y+getIconHeight()-1);
               p.addPoint(x+getIconWidth()-1, y+(getIconHeight() / 2));
            }
            g.fillPolygon(p);
        }
    }

    public static void main(String args[]) {
       JFrame frame = new ExitableJFrame("Fifth Button");
       Container contentPane = frame.getContentPane();
       Icon normalIcon = new TriangleIcon(Color.red,
TriangleIcon.State.NORMAL);
       Icon pressedIcon = new TriangleIcon(Color.red,
TriangleIcon.State.PRESSED);
       Icon rolloverIcon = new TriangleIcon(Color.red,
           TriangleIcon.State.ROLLOVER);
       JButton b = new JButton("My Fifth Button!", normalIcon);
       // o b = new JButton("My Fifth Button!");
       //    b.setIcon(normalIcon);
       b.setPressedIcon(pressedIcon);
       b.setRolloverIcon(rolloverIcon);
       b.setRolloverEnabled(true);
       contentPane.add(b, BorderLayout.NORTH);
       frame.setSize(300, 100);
       frame.show();
    }
}
```

Ubicación de texto e iconos

La interfaz `SwingConstants` se encarga de controlar las claves de la ubicación de texto e iconos dentro de `AbstractButton`. Como `AbstractIcon` implementa

la interfaz, las constantes disponibles serán `AbstractButton`, por lo que no tendremos que preocuparnos de la clase `SwingConstants`.

Los métodos de posicionamiento tienen un efecto mayor cuando aparecen dentro del botón etiquetas de texto e iconos. Por defecto, la etiqueta de texto aparecerá a la derecha del icono y ambos estarán centrados verticalmente. Si modificamos el comportamiento predeterminado, ambos elementos podrán combinar posiciones; aunque probablemente quiera mover la etiqueta de texto alrededor del icono.

Para colocar texto e iconos, utilizamos el método **`setVerticalTextPosition`**(), que aceptará los pardmetros **`CENTER`**, **`TOP`** y **`BOTTOM`**. Inicialmente utilizard **`CENTER`**. El método `setHorizontalTextPosition()` acepta los valores **`CENTER`**, **`RIGHT`** y **`LEFT`**. Inicialmente utilizará **`RIGHT`**. Entonces, con los valores de **`TOP`** y **`RIGHT`**, el texto se podra colocar en la parte superior derecha del icono y con **`TOP`** y **`CENTER`** se podra colocar justo sobre el icono.

Los métodos de alineación de `setVerticalAlignment(int alignment)` y **`setHorizontalAlignment`**`(int alignment)` especifican la posición que ocupard el texto o el icono cuando haya más espacio disponible. Por defecto, estos valores son **`CENTER`**. Pero, la alineación vertical tambien puede tener otros valores (**`TOP`** y **`BOTTOM`**), mientras que la horizontal puede ser **`RIGHT`** o **`LEFT`**.

En la figura **13.8** tenemos el `PieIcon` del listado **13.4** colocado en diferentes zonas de un **`JButton`**. El listado **13.8** es la demostración del programa

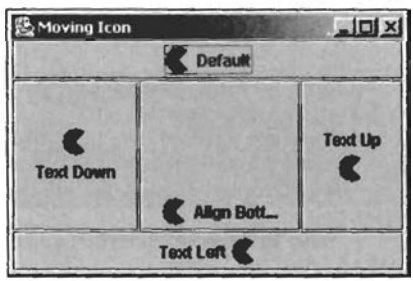


Figura 13.8. Un JButton con iconos desperdigados

Listado 13.8. Muestra de un boton con etiquetas diferentemente alineadas

```
import java.awt.*;
import javax.swing.*;

public class MovingIconTest {
    static class PieIcon implements Icon {
        Color color;
        public PieIcon(Color c) {
           color = c;
        }
        public int getIconWidth() {
           return 20;
        }
        public int getIconHeight() {
```

```
            return 20;
        }
        public void paintIcon(Component c, Graphics g, int x, int y) {
            g.setColor(color);
            g.fillArc(x, y, getIconWidth(), getIconHeight(), 45, 270);
        }
    }

    public static void main(String args[]) {
        JFrame frame = new ExitableJFrame("Moving Icon");
        Container contentPane = frame.getContentPane();
        JButton b;
        Icon icon = new PieIcon(Color.red);
        b = new JButton("Default", icon);
        contentPane.add(b, BorderLayout.NORTH);
        b = new JButton("Text Left", icon);
        b.setHorizontalTextPosition(JButton.LEFT);
        contentPane.add(b, BorderLayout.SOUTH);
        b = new JButton("Text Up", icon);
        b.setHorizontalTextPosition(JButton.CENTER);
        b.setVerticalTextPosition(JButton.TOP);
        contentPane.add(b, BorderLayout.EAST);
        b = new JButton("Text Down", icon);
        b.setHorizontalTextPosition(JButton.CENTER);
        b.setVerticalTextPosition(JButton.BOTTOM);
        contentPane.add(b, BorderLayout.WEST);
        b = new JButton("Align Bottom Left", icon);
        b.setHorizontalAlignment(JButton.LEFT);
        b.setVerticalAlignment(JButton.BOTTOM);
        contentPane.add(b, BorderLayout.CENTER);
        frame.setSize(300, 200);
        frame.show();
    }
}
```

Configuración de los botones nemotécnicos

A la hora de crear botones o elementos del menú, a ciertos programadores les gusta asociar ciertas reglas nemotécnicas de tal forma que los usuarios puedan navegar por los elementos de la pantalla sin llegar a levantar las manos del teclado. No son los atajos de teclado que tienen la mayoría de las aplicaciones, como **Control-P,** que en las aplicaciones de Microsoft envía el documento a la impresora. Un botón nemotécnico nos permite asociar, por ejemplo, **Alt-F** para abrir el menú Archivo. Cuando se le ha asociado una de estas reglas a un botón, la letra que se utiliza en la combinación, aparecerá subrayada en la etiqueta de su texto.

Truco

Si un boton no tiene etiqueta de texto y se le ha asociado una regla nemotecnica, la combinación de las teclas funcionara correctamente. Si el carácter que se utiliza en la combinación de teclas no aparece en la etiqueta

del boton, tambien funcionara. Como podemos ver, en cualquier caso funcionara la regla nemotecnica, aunque el usuario no tenga ninguna forma de saber que combinación ha de utilizar.

En la figura **13.9** tenemos un botón con una regla nemotécnica, otro con una regla que no es válida con una etiqueta de texto y una regla nemotécnica, pero sin etiqueta. Obsérvese cómo se ha subrayado la t del botón que se encuentra en la parte superior de la página. El **listado 13.9** es el código fuente.

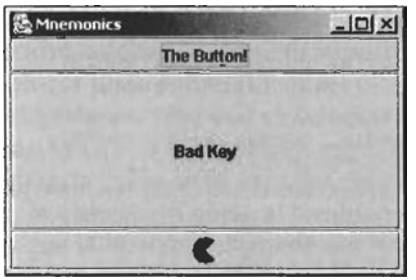


Figura **13.9.** Botones JButtons con un conjunto de reglas nemotecnicas

Listado 13.9. Trabajar con botones nemotécnicos

```
import java.awt.*;
import. java.awt.event.*;
import javax.swing.*;

public class ButtonMnemonicTest {
    static class PieIcon implements Icon {
        Color color;
        public PieIcon(Color c) {
            color = c;
        }
        public int getIconWidth() {
            return 20;
        }
        public int getIconHeight() {
            return 20:
        }
        public void paintIcon(Component c, Graphics g, int x, int y) {
            g.setColor(color);
            g.fillArc(x, y, getIconWidth(), getIconHeight(), 45, 270);
      }
    }

    public static void main(String args[]) {.
      JFrame frame = new ExitableJFrame("Mnemonics");
      Container contentPane = frame.getContentPane();
      JButton h;
      Icon icon = new PieIcon(Color.red);
```

```
        b = new JButton("The Button!");
        b.setMnemonic(KeyEvent.VK_T);
        // 0 es T, 1 es h, ...
        b.setDisplayedMnemonicIndex(7);
        contentPane.add(b, BorderLayout.NORTH);
        b = new JButton("Bad Key");
        b.setMnemonic(KeyEvent.VK_I!;
        contentPane.add(b, BorderLayout.CENTER);
        b = new JButton(icon);
        b.setMnemonic(KeyEvent.VK_B);
        contentPane.add(b, BorderLayout.SOUTH);
        frame.setSize(300, 200);
        frame.show();

}
```

Trabajar con la informacidn sobre herramientas

Antes de empezar con otras clases de componentes Swing, veremos algo que comparten todas ellas: la informacion sobre las herramientas. Podemos especificar el texto que se mostrará cuando el usuario coloque el cursor sobre un componente determinado. La clase encargada de hacerlo es `JToolTip`, pero en realidad ni siquiera hace falta saber que existe. Para configurar la informacion sobre herramientas tan sólo hemos de llamar al método `setToolTipText(String tip)` de `JComponent`. Una vez que se ha determinado el texto que se va a utilizar, colocaremos el cursor sobre el elemento deseado. La figura 13.10 nos muestra la apariencia de la ayuda para herramientas para un **`JButton`** y el listado 13.10 es el programa asociado.

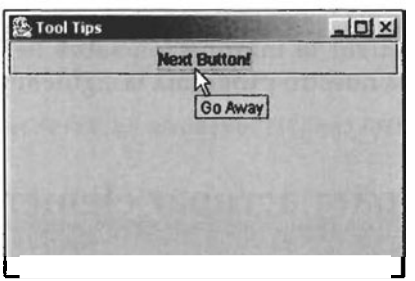


Figura 13.10. JButton que muestra informacion sobre herrarnientas

Listado 13.10. Trabajar con texto de ayuda para herrarnientas

```
import java.awt.*;
import javax.swing.*;

public class ButtonTipTest {
    public static void main(String args[]) {
        JFrame frame = new ExitableJFrame("Tool Tips");
        Container contentPane = frame.getContentPane();
```

```
        JButton b = new JButton("Next Button!");
        b.setToolTipText("Go Away");
        contentPane.add(b, BorderLayout.NORTH);
        frame.setSize(300, 200);
        frame.show();
    }
}
```

Configuración del administrador de informacion sobre herramientas

Aunque la configuracidn y el uso de informacidn sobre herramientas es muy sencilla, hay ocasiones en las que no nos gusta la velocidad con la que aparece (o desaparece) el texto en pantalla. O simplemente queremos desactivar esta función. Para configurar estas opciones necesitaremos la ayuda de la clase `ToolTipManager`.

Para hacernos con el administrador de informacidn sobre herramientas del programa, tendremos que pedir al `ToolTipManager` su `sharedInstance()`, un método estático. Una vez que tengamos el administrador, podremos utilizar cualquiera de los siguientes métodos para modificar los parámetros deseados:

```
public int getDismissDelay() // inicialmente 4 segundos
public int getInitialDelay() // inicialmente 3/4 segundo
public int getReshowDelay()  // inicialmente 1/4 segundo
public boolean isEnabled()   // inicialrnente true
```

O cambiar las modificaciones con otro. En cualquier caso, obtendremos los resultados en milisegundos:

```
public void setDismissDelay(int delay)
public void setEnabled(boolean b)
public void setInitialDelay(int delay)
public void setReshowDelay(int delay)
```

Así que, para desactivar la informacidn sobre herramientas de un programa, tendremos que añadir a nuestro programa la siguiente linea:

```
ToolTipManager.sharedInstance().setEnabled(false);
```

Uso de JPanel para agrupar elementos

Los elementos de la clase `JPanel` tienen una clase contenedora ligera. Podemos utilizar `JPanel` o `Container` para agrupar componentes en la pantalla. La ventaja de utilizar `JPanel` sobre `Container` es que los componentes del primero utilizan el doble buffer para realizar su dibujo y pueden incluir un borde. En la figura 13.11 tenemos componentes `JPanel` con borde. (Si no le interesa ninguna de estas funciones, puede seguir con `Container`.) Lo unico realmente necesario de la definición son los cuatro constructores:

```
public JPanel()
public JPanel(boolean isDoubleBuffered)
public JPanel(LayoutManager layout)
public JPanel(LayoutManager layout, boolean isDoubleBuffered)
```

El listado que se muestra acontinuación, es la demostración del programa que produce la figura 13.11.

Listado 13.11. Muestra de paneles con bordes

```
import java.awt.*;
import javax.swing.*;
import javax.swing.border.*;

public class PanelTest {
    private static void fill(Container c) {
        for (int i=0;i<3;i++) {
            c.add(new JButton("" + i));
        }
    }
    public static void main[String args[]) {
        JFrame frame = new ExitableJFrame("JPanels");
        Container contentPane = frame.getContentPane();
        JPanel jp = new JPanel();
        fill(jp);
        Border bored = BorderFactory.createTitledBorder("Hello");
        jp.setBorder[bored);
        contentPane.add(jp, BorderLayout.NORTH) :
        jp = new JPanel(new GridLayout(3, 1));
        fill(jp);
        bared = BorderFactory.createBevelBorder(BevelBorder.RAISED,
            Color.red, Color.blue);
        jp.setBorder(bored);
        contentPane.add(jp, BorderLayout.CENTER);
        jp = new JPanel(new GridLayout(1, 3));
        fill(jp);
        bored = BorderFactory.createLineBorder(Color.green);
        jp.setBorder(bored);
        cantent-Pane.add(jp, BorderLayout.EAST);
        frame.setSize(300, 200);
        frame.show();
    }
}
```

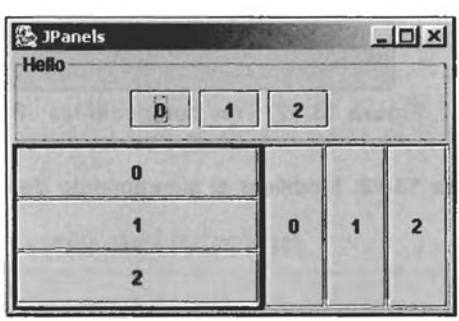


Figura 13.11. Tres cornponentes JPanel con bordes y distintos adrninistradores de diseño

Agregar etiquetas a los elementos

Un componente `JLabel` no es más que una sencilla línea de texto pasiva (es decir, que no interactua en absoluto) que se encuentra dentro de un icono. La mayor parte de la definición es parecida a la de `AbstractButton`. Como `JLabel` no es un botón que se pueda seleccionar, no puede ser una subclase de `AbstractButton`. Aparte del comportamiento, los métodos son idénticos.

La excepción m6s notoria en cuanto a comportamiento es la existencia de botones nemotécnicos para etiquetas. Cuando utilizamos el método `setDisplayMnemonic()` tendremos que asociar la etiqueta a un componente mediante `setLabelFor()`. A continuación, cuando el usuario utiliza la combinación de teclas establecida, el foco de entrada se desplaza hasta el componente asociado. Como mejor funciona este sistema es cuando trabaja con componentes de texto, que no pueden tener su propia regla nemotécnica.

Cuando queremos crear un JLabel, usamos uno de estos seis constructores:

```
public JLabel()
public JLabel(Icon icon)
public JLabel(Icon icon, int horizontalAlignment)
public JLabel(String text)
public JLabel(String text, int horizontalAlignment)
public JLabel.(String text, Icon icon, int horizontalAlignment)
```

Al igual que sucede con el movimiento de iconos y de texto dentro de un botón, las constantes de alineamiento de `JLabel` proceden de `SwingConstants`. La figura **13.12** muestra los tres componentes `JLabel` con diferentes alineaciones. El listado **13.12** es el programa asociado.

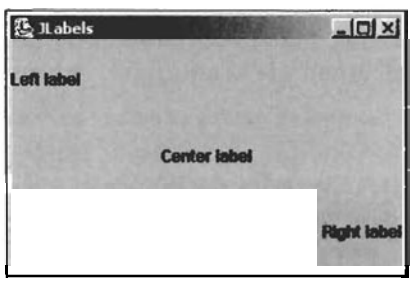


Figura 13.12. Tres cornponentes JPanel

Listado 13.12. Modificar el alinearniento de las etiquetas

```
import java.awt.*;
import javax.swing.*;

public class LabelTest {
    public static void main(String args[]) {
        JFrame frame = new ExitableJFrame("JLabels");
```

```
        Container contentPane = frame.getContentPane();
        contentPane.setLayout(new GridLayout(3,1));
        JLabel label = new JLabel("Left label");
        contentPane.add(label); // justificado a la izquierda predeterminad
        label = new JLabel("Center label", JLabel.CENTER);
        contentPane.add(label); // centro
        label = new JLabel("Right. label", JLabel.RIGHT);
        contentPane.add(label); // justificado a la derecha
        frame.setSize(300, 200);
        frame.show();
    }
```

Para la alineacidn horizontal podremos utilizar cualquiera de los siguientes parámetros: **LEFT, CENTER** o **RIGHT.**

Se pueden solicitar las propiedades de la alineacidn y el texto de la etiqueta utilizando los siguientes métodos:

```
public int getHorizontalAlignment()
public int getHorizontalTextPosition()
public int getVerticalAlignment()
public int getVerticalTextPosition()
public String getText()
```

o se pueden cambiar con los métodos `setXXX()`.

Nota

La alineacion de una etiqueta únicamente tendra un efecto visual si el administrador de diseño que la controla permite que los componentes utilicen otras dimensiones aparte de las especificadas en el método `getMinimum-Size()`. Por ejemplo, si cambiamos la alineacion de las etiquetas que se encuentran dentro de `FlowLayout` no veremos ningún resultado, porque `FlowLayout` siempre aplica las mínimas dimensiones a sus componentes.

Además del texto y de su alineación, `JLabel` también tiene un icono. Su uso es idéntico al que hemos visto con `JButton`. Configuramos el icono en el constructor; lo modificamos con `setIcon(Icon icon)` o `setDisabledIcon(Icon icon);` y efectuamos una consulta usando `getIcon()` y `getDisabledIcon()`. Incluso podemos especificar el espacio que se dejará entre el texto y el icono mediante `setIconTextGap(int gap).`

Agregar botones de activación

Un componente `JToggleButton` es un boton que tiene dos estados y que suele utilizarse en las barras de herramientas, para indicar el estado de una opción que se puede alternar (activada/desactivada), como los atributos de fuente negrita, cursi-

va y subrayado que pueden estar activados en un procesador de texto. Estos son sus constructores:

```
public JToggleButton()
public JToggleButton(Icon icon)
public JToggleButton(Icon icon, boolean selected)
public JToggleButton(String text)
public JToggleButton(String text, boolean selected)
public JToggleButton(String text, Icon icon)
public JToggleButton(String text, Icon icon, boolean selected)
public JToggleButton(Action action)
```

La figura 13.13 muestra un ejemplo de componentes JToggleButton y el listado 13.13 *es* el programa que lo creo.

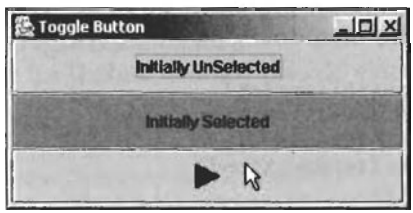


Figura 13.13. Cornponentes JToggleButton

Listado 13.13. Trabajar con botones de activación

```
import java.awt.*;
import javax.swing.*;

public class ToggleTest {
    static class TriangleIcon implements Icon {
        String name;
        static class State {
            public static final int NORMAL   = 0;
            public static final int PRESSED  = 1;
            public static Einal int ROLLOVER = 2;
            public static final int SELECTED = 3;
            private State() {}

        int state;
        Color color;
        public TriangleIcon(Color c, int stat,?! {
            color = c;
            this.state = state;
        }
        public int getIconWidth() {
            return 20;
        }
        public int getIconHeight() {
            return 20;

        public void paintIcon(Component c, Graphics g, int x, int y) {
            g.setColor(color);
```

```
            Polygon p = new Polygon();
            if (state == State.NORMAL) {
               p.addPoint(x+(getIconWidth() / 2), y);
               p.addPoint(x, y+getIconHeight()-1);
               p.addPoint(x+getIconWidth()-1, y+getIconHeight()-1);
            } else if (state == State.PRESSED) {
               p.addPoint(x, y);
               p.addPoint(x+getIconWidth()-1, y);
               p.addPoint(x+(getIconWidth() / 2), y+getIconHeight()-1);
            } else if (state == State.SELECTED) {
               p.addPoint(x+getIconWidth()-1, y);
               p.addPoint(x+getIconWidth()-1, y+getIconHeight()-1);
               p.addPoint(x, yt(getIconHeight() / 2));
            } else {
               p.addPoint(x, y);
               p.addPoint(x, y+getIconHeight()-1);
               p.addPoint(x+getIconWidth()-1, y+(getIconHeight() / 2));
            }
            g.fillPolygon(p);
         }
      }

      public static void main(String args[]) {
         JFrame frame = new ExitableJFrame("Toggle Button");
         Container contentPane = frame.getContentPane();
         contentPane.setLayout(new GridLayout(0, 1));
       Icon normalIcon = new TriangleIcon(Color.red, TriangleIcon.State.NORMAL];
         Icon pressedIcon = new TriangleIcon(Color.red,
TriangleIcon.State.PRESSED);
         Icon selectedIcon = new TriangleIcon(Color.red,
            TriangleIcon.State.SELECTED);
         Icon rolloverIcon = new TriangleIcon(Color.red,
            TriangleIcon.State.ROLLOVER);
         JToggleButton b = new JToggleButton("Initially UnSelected");
         contentPane.add(b);
         b = new JToggleButton("Initially Selected", true);
         contentPane.add(b);
         b = new JToggleButton(normalIcon);
         b.setPressedIcon(pressedIcon);
         b.setSelectedIcon(selectedIcon);
         b.setRolloverIcon(rolloverIcon);
         b.setRolloverEnabled(true);
         contentPane.add(b);
         frame.setSize(300, 150);
         frame.show();
      }
```

Nota

Observe que la clase `JToggleButton` tambien admite un estado seleccionado. Dado que se puede seleccionar el boton de activación, ademasde pulsarlo, puede proporcionar otro icono mas.

Agregar casillas de verificación

Un componente JCheckbox es un botón que tiene dos estados y que se suele utilizar con los elementos seleccionables de las interfaces GUI (por ejemplo, las opciones en los formularios de votación o las opciones en las preferencias del navegador). Estos son sus constructores:

```
public JCheckBox()
public JCheckBox(Icon icon)
public JCheckBox(Icon icon, boolean selected)
public JCheckBox(String text)
public JCheckBox(String text, boolean selected)
public JCheckBox(String text, Icon icon)
public JCheckBox(String text, Icon icon, boolean selected)
public JCheckBox(Action action)
```

Su utilización es la misma que la de JToggleButton. La única diferencia es su aspecto fisico. En vez de seleccionar todo el botón, unicamente aparecerd seleccionado el icono asociado con JCheckBox. Si no se ha especificado ningun icono, aparecerá el predeterminado. En la figura **13.14** tenemos un ejemplo de un componente JCheckBox y el listado **13.14** es el programa asociado.

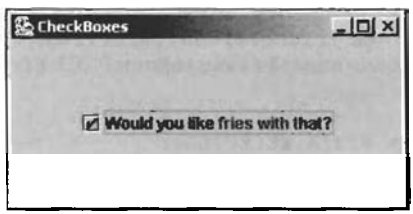


Figura 13.14. Un cornponente JCheckBox

Listado 13.14. Trabajar con botones de activación

```
import java.awt.*;
import javax.swing.*;

public class CheckBoxTest {
    public static void main(String args[]) {
        JFrame frame = new ExitableJFrame("CheckBoxes");
        Container contentPane = frame.getContentPane();
        JCheckBox cb = new JCheckBox("Would you like fries with that?");
        cb.setHorizontalAlignment(JCheckBox.CENTER );
        contentPane.add(cb, BorderLayout.CENTER);
        frame.setSize(300, 150);
        frame.show();
    }
```

El programa declara y construye un objeto JCheckBox al que etiqueta con la cadena que se entrega como argumento al constructor JCheckBox. A continuación,

por estética, centramos el elemento en la pantalla. A continuación añadimos JCheckBox utilizando el método **add()** de getContenctPane(). Por defecto, el estado inicial de una casilla de verificación es **false** (desactivada). Para modificar el estado inicial de una de estas casillas, utilizaremos una versión alternativa del constructor: JCheckbox **(String, boolean)**. El parámetro booleano determina el estado inicial de la casilla.

Hay otros constructores adicionales que se utilizan para determinar qué icono se utilizará con JCheckBox.

Añadir botones de opción en grupos de botones

Para definir un grupo de casilla de verificación donde Qnicamente se pueda activar una de ellas, utilizamos la clase JRadioButton en vez de **JCheckBox**. Sus ocho constructores se parecen a los de JToggleButton y **JCheckBox:**

```
public JRadioButton()
public JRadioButton(Icon icon)
public JRadioButton(Icon icon, boolean selected)
public JRadioButton(String text)
public JRadioButton(String text, boolean selected)
public JRadioButton(String text, Icon icon)
public JRadioButton(String text, Icon icon, boolean selected)
public JRadioButton(Action action)
```

También necesitará otra clase de ayuda: ButtonGroup. La tabla 13.4 contiene su definición. La finalidad de **ButtonGroup** es proporcionar el alcance del grupo exclusivo entre sí y administrar su estado. Sólo se puede seleccionar uno de los botones agregados al grupo. Un elemento seleccionado queda deseleccionado cuando se selecciona otro elemento del grupo.

Tabla 13.4. Composición de la clase ButtonGroup

Variables/Métodos	Descripción
buttons	Contiene referencias a los botones contenidos
ButtonGroup()	Crea un objeto ButtonGroup
add()	Agrega un boton al grupo
getButtonCount()	Recupera el recuento de botones de un grupo
getSelection()	Recupera el modelo del boton seleccionado
isSelected()	Comprueba si el modelo del boton esta actualmente seleccionado
remove()	Elimina un boton del grupo
setSelected()	Cambia el estado de selección del modelo del boton

Truco

Observe que ButtonGroup trabaja con los componentes de AbstractButton, no solo con JRadioButton. Por ejemplo, también utilizará ButtonGroup con JRadioButtonMenuItem. Técnicamente hablando, podemos poner cosas como JToggleButton dentro de un grupo e botones, si ese es el comportamiento que deseamos o podemos mezclar botones de activación, botones normales o botones de menu en un grupo. Como cada una de estas clases es una subclase de AbstractButton, sólo puede haber una seleccionada cada vez.

A continuación tenemos un programa que utiliza un caso de ButtonGroup para definir una colección de botones de opción:

Listado 13.15. Trabajar con JRadioButton

```
import java.awt.*;
import javax.swing.*;

public class RadioTest {
    public static void main(String args[]) {
        JFrame frame = new ExitableJFrame("Radio Group");
        Container contentPane = frame.getContentPane();
        contentPane.setLayout(new GridLayout(0, 1));
        ButtonGroup group = new ButtonGroup();
        JRadioButton option = new JRadioButton("French Fries", true);
        group.add(option);
        contentPane.add(option);
        option = new JRadioButton("Onion Rings", false);
        group.add(option);
        contentPane.add(option);
        option = new JRadioButton("Ice Cream", false);
        group.add(option);
        contentPane.add(option);
        frame.setSize(200, 200);
        frame.show();

}
```

Como se puede ver, tendremos que añadir los botones de opción al grupo uno a uno. No podemos especificar el nombre del grupo durante el proceso de creación. El segundo argumento del constructor es el estado inicial que tendrán las casillas de verificación: false para las desactivadas y true para las activas. Sólo podremos seleccionar uno de los elementos de los grupos de botones de opción.

La figura 13.15 muestra el resultado del programa del listado 13.15. Observernos los botones de la figura. Este formato es el resultado obtenido al cambiar el diseño predeterminado BorderLayout a GridLayout.

Además, con JCheckBox y JToggleButton podemos especificar nuestros propios iconos.

Figura 13.15. Botones de opción

Agregar listas desplegables

La clase JCornboBox engloba todas las opciones de las listas desplegables. Puede crear este componente con uno de los cuatro constructores; puede proporcionar inicialmente el conjunto de elecciones o no.

```
public JComboBox()
public JComboBox(ComboBoxModel model)
public JComboBox(Object object[])
public JComboBox(Vector vector)
```

Vamos a ver cómo podemos crear JComboBox, agregar opciones y permitir su edición cuando las opciones disponibles no sean suficientes.

El componente JComboBox permite que el usuario seleccione uno de los elementos (u opciones) disponibles en un lugar para ahorrar espacio. Cuando la interfaz se satura, podemos sustituir un grupo de botones de opción por un elemento JComboBox compacto. En la figura 13.16 tenemos un ejemplo donde podemos ver la lista desplegable que se ha creado con este componente.

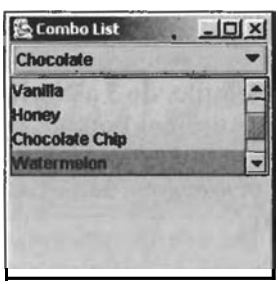


Figura 13.16. Componente JComboBox

Listado 13.16. Escoger elecciones de un JComboBox

```
import java.awt.*;
import. javax.swing.*;

public class ComboTest {
```

```
public static void main(String args[]) {
    JFrame frame = new ExitableJFrame("Combo List");
    Container contentPane = frame.getContentPane();
    String flavors[] = {"Chocolate", "Strawberry", "Vanilla", "Honey",
    "Chocolate Chip", "Watermelon", "Rasberry", "Banana"};
    JComboBox jc = new JComboBox(flavors);
    jc.setMaximumRowCount(4);
    contentPane.add(    BorderLayout.NORTH);
    frame.setSize(200, 200);
    frame.show();
```

A partir de una cadena `string` este applet inicia toda la lista de opciones de `JComboBox`. Si tenemos que aiiadir una entrada, utilizaremos el método `addItem()`. Cuando se ha especificado la lista de opciones, se configura el numero mixiimo de opciones que aparecerin en la ventana con el valor de cuatro mediante `setMaximunRowCount()`. Esto indica que se deben mostrar cuatro elecciones como mixiimo cuando aparezca la lista desplegable. Por ultimo, la lista se aiiade al panel de control del applet con el método `add()`.

La clase `JComboBox` suministra una serie de métodos que nos permitirán acceder y/o configurar los elementos seleccionados que aparecen en la lista:

- `getSelectedItem()`: Devuelve el elemento del objeto.
- `getSelectedIndex()`: Devuelve la posición que ocupa el elemento dentro de la lista.
- `getSelectedObjects()`: Devuelve una matriz que contiene el elemento.
- `setSelectedIndex(int pos)`: Selecciona un elemento a partir de su posición.
- `setSelectedItem(Object anObject)`: Selecciona un elemento a partir de su nombre (usa el método `equals()` para comprobar la igualdad).

Los componentes `JComboBox` se suelen utilizar para las listas pequeñas de elementos seleccionables (por ejemplo, de 3 a 20 elementos). Pero si sólo tenemos dos opciones, puede que sea mejor utilizar botones de opción (a menos que el grupo de opciones sea dinámico, es decir, que crezca y llegue a tener más de dos opciones). Si por otro lado, la lista de opciones es demasiado larga, hemos de considerar la posibilidad de utilizar `JList`.

Truco

En caso de que no sepamos la cantidad de opciones disponibles que vamos a agregar a la lista `JComboBox`, podremos permitir al usuario que introduzca una opcion mas. No añadiremos otra opcion a la lista, sino que permitiremos que escriba una nueva. Para ello invocamos al método `setEditable()` con el valor `true`.

Añadir paneles deslizantes

El componente `JScrollPane` es muy interesante. Actua como un contenedor de un componente, proporcionando la capacidad de desplazarse por dicho componente, cuando el tamaño de éste es mayor que es espacio disponible dentro de `JScrollPane`. Estos son sus constructores:

```
public JScrollPane()
public JScrollPane(int vsbPolicy, int hsbPolicy)
public JScrollPane(Component view)
public JScrollPane(Component view, int vsbPolicy, int hsbPolicy)
```

Las normas para la barra de desplazamiento vertical y horizontal proceden de `ScrollPaneConstants` que implementa `JScrollPane`. Estos son los valores:

```
VERTICAL_SCROLLBAR_ALWAYS
VERTICAL_SCROLLBAR_AS_NEEDED
VERTICAL_SCROLLBAR_NEVER
HORIZONTAL_SCROLLBAR_ALWAYS
HORIZONTAL_SCROLLBAR_AS_NEEDED
HORIZONTAL_SCROLLBAR_NEVER
```

El siguiente ejemplo nos muestra un `JLabel` con una imagen `ImageIcon` muy grande. Debido a su gran tamaño, las barras de desplazamiento aparecerán automdticamente.

Figura 13.17. Elernento JScrollPane con una imagen JLabel

Listado 13.17. Desplazando un cornponente grande en un JScrollPane

```
import java.awt.*;
import javax.swing.*;

public class ScrollTest {
    public static void main(String args[]) {
```

```
        JFrame frame = new ExitableJFrame("Scroll Pane");
        Container contentPane = frame.getContentPane();
        Icon icon = new ImageIcon("photoshop.jpg");
        JLabel lab = new JLabel(icon);
        JScrollPane pane = new JScrollPane(lab);
        contentPane.add(pane, BorderLayout.CENTER);
        frame.setSize(300, 300);
        frame.show();

}
```

Agregar listas

Considere `JList` el equivalente pesado al componente `JComboBox`. Contiene un conjunto similar de cuatro constructores:

```
public JList()
public JList(ListModel model)
public JList(Object object[])
public JList(Vector vector)
```

Las siguientes son las diferencias existentes entre los componentes de `JList` y JComboBox:

- Con los conjuntos de elementos más grandes se utilizan las listas.
- Cuando hay que efectuar una serie de selecciones, se utilizan las listas.
- La capacidad de las listas suele ser mayor que el estado de la interfaz GUI.
- La clase `JList` contiene más métodos para su funcionalidad.
- La clase `JList` sólo nos permite seleccionar elementos de su conjunto de opciones.

La figura 13.18 muestra un ejemplo de un componente `JList` y el listado 13.18 es el programa que la genera.

Listado 13.18. Selección de una lista desplegable

```
import java.awt.*;
import javax.swing.*;
public class ListTest {
    public static void main(String args[]) {
        JFrame frame = new ExitableJFrame("Color List");
        Container contentPane = frame.getContentPane();
        String colors[] = {"Red", "Orange", "Yellow",
        "Green", "Blue", "Indigo", "Violet"\;
        JList colorList = new JList(colors);
        colorList.setVisibleRowCount(5);
        JScrollPane pane = new JScrollPane(colorList);
        contentPane.add(pane, BorderLayout.NORTH);
        frame.setSize(200, 200);
```

```
        frame.show();
    }
}
```

Figura 13.18. Componente **JList**

Lo primero que tenemos que aprender a la hora de utilizar la clase `JList` es que tenemos que proporcionar soporte para que se puedan aplicar las propiedades de desplazamiento. Siempre que queramos ofrecer un panel `JScrollPane` por cuyo contenido nos podamos desplazar, tendremos que colocar `JList` dentro de él. Si no lo hacemos asi, toda la parte de la ilustración que no quepa en la pantalla, se perdera. Esto ocurre con todos los componentes Swing, no sólo con `JList`.

Los componentes `JList` vienen en tres variantes interactivas: una que nos permite efectuar varias selecciones (el modo predeterminado), otra que nos permite efectuar un único intervalo de selecciones y una más que se comporta como un componente `JComboBox` o `JRadioButton` que Cinicamente aceptará una selección. Para configurar y procesar varias selecciones, disponemos de los siguientes métodos:

- void **setSelectionMode(int** mode): Nos permite cambiar el comportamiento de un elemento `JList` cuando se efectuan varias selecciones. Los modos que tenemos a nuestra disposición son los siguientes: **`SINGLE_SELECTION`, `SINGLE_INTERVAL_SELECTION` y `MULTIPLE_INTERVAL_SELECTION`**

Nota

Por defecto, un componente `JList` es el modo **`MULTIPLE_INTERVAL_SELECTION`.**

- **int[] getSelectedIndices()**: Devuelve una matriz de indices que especifica varias selecciones.
- **Object[] getSelectedValues()**: Parecido a **`get`**`SelectedIndexes()`, con la salvedad de que devuelve una matriz con los objetos seleccionados.
- int **getSelectionMode()**: Nos indica si el elemento `JList` puede trabajar con varias selecciones.

Advertencia

No conviene que utilicemos `setSelectionMode()` para cambiar el comportamiento de la lista despues de activar el componente `JList`. Si se hace, la respuesta que se obtendrá al efectuar varias selecciones puede llegar a confundir al usuario.

Los elementos de una lista pueden mostrarse horizontal o verticalmente. Por defecto, se muestran verticalmente. Para modificar la orientación, pase al método `setLayoutOrientation()` uno de los siguientes valores:

- **VERTICAL:** Muestra las celdas en una sola columna (por defecto).
- **HORIZONTAL-WRAP:** Muestra las celdas horizontalmente, si es necesario por la falta de espacio, continua en una nueva fila.
- **VERTICAL-WRAP:** Muestra las celdas verticalmente, si es necesario por la falta de espacio, continua en una nueva columna.

La figura 13.19 muestra los resultados de agregar una llamada de `setLayoutOrientation(JList.VERTICAL_WRAP)` al programa del listado 13.18. En la lista se muestran cinco filas debido a la llamada a `setVisibleRowCount()`.

Figura 13.19. Componente JList con varias columnas

Suministrar barras de desplazamiento

Cuando queremos que los usuarios de nuestras interfaces introduzcan valores numéricos pertenecientes a un intervalo determinado (que se puedan modificar en cualquier momento) podemos suministrarles barras de desplazamiento para que los controlen con más eficacia. La clase `JSlider` permite que Java pueda trabajar con las herramientas de desplazamiento. Tiene seis constructores:

```
public JSlider()
public JSlider(int orient)
public JSlider(int min, int max)
public JSlider(int min, int max, int value)
```

```
public JSlider(int orient, int min, int max, int value)
public JSlider(BoundedRangeModel brm)
```

Para que la introducción de un valor dentro de un rango sea más sencilla, `JSlider` puede mostrar unas marcas en las barras de desplazamiento. Incluso puede usar etiquetas en las marcas. Además, podemos hacer que la barra se detenga solamente en un resultado determinado. Gracias a estas propiedades, el componente `JSlider` es mas agradable para el usuario. En la figura 13.20 tenemos dos componentes `JSlider`, uno vertical y otro horizontal. El **listado 13.19** es el programa que lo genera.

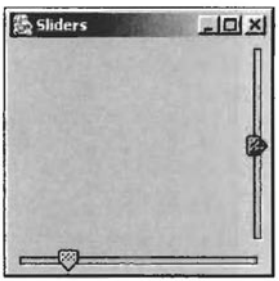


Figura 13.20. Cornponentes JSlieder vertical y horizontal

Listado 13.19. Trabajo con JSlider

```
import java.awt.*;
import javax.swing.*;

public class SliderTest {
    public static void main(String args[]) {
        JFrame frame = new ExitableJFrame("Sliders");
        Container contentPane = frame.getContentPane();
        JSlider upDown, leftRight;
        upDown = new JSlider(JSlider.VERTICAL, 0, 100, 50);
        LfftRight = new JSlider(JSlider.HORIZONTAL, 0, 10, 2);
        contentPane.add(upDown, BorderLayout.EAST);
        contentPane.add(leftRight, BorderLayout.SOUTH);
        frame.setSize(200, 200);
        frame.show();

}
```

El programa crea dos objetos `JSlider`: uno vertical (para lo cual se utiliza la constante de la clase `JSlider JSlider.VERTICAL)` y otro horizontal (con `JSlider.HORIZONTAL)`. Los argumentos numéricos adicionales que se le entregan al constructor `Scrollbar` son los siguientes:

- **Valores minimo y máximo:** Valores minimo y máximo que puede representar la barra de desplazamiento. Por ejemplo 0-100 (para porcentajes), 6-120 (para las edades), 1-31 (para las fechas), etc.

- **Valor inicial:** Punto de inicio del deslizador (algún valor comprendido entre el máximo y el minimo).

Aunque el programa `SliderTest` funciona correctamente, un usuario no podrá ver dónde se encuentran los distintos valores. Con un poco de trabajo, podemos aiiadir una serie de marcas y etiquetas para que el aspecto de la barra de deslizamiento sea más agradable. La figura 13.21 asi nos muestra la versión mejorada, con varias opciones.

En el listado 13.20 podemos ver en negrita las líneas agregadas para conseguir estas mejoras.

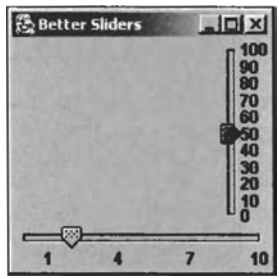


Figura 13.21. JSlider configurada vertical y horizontalrnente

Listado 13.20. Personalizar el componente JSlider

```
import java.awt.*;
import javax.swing.*;
import java.util.*;

public class SliderTest2 {
    public static void main(String args[]) {
        JFrame frame = new ExitableJFrame("Better Sliders");
        Container contentPane = frame.getContentPane();
        JSlider upDown, leftRight;
        upDown = new JSlider(JSlider.VERTICAL, 0, 100, 50);
        upDown.setMajorTickSpacing(10);
        upDown.setPaintLabels(true);
        leftRight = new JSlider(JSlider.HORIZONTAL, 0, 10, 2);
        Hashtable h = new Hashtable();
        h.put(new Integer(1), new JLabel("1"));
        h.put(new Integer(4), new JLabel("4"));
        h.put(new Integer(7), new JLabel("7"));
        h.put(new Integer(10), new JLabel("10"));
        leftRight.setLabelTable(h);
        leftRight.setPaintLabels(true);
        contentPane.add(upDown, BorderLayout.EAST);
        contentPane.add(leftRight, BorderLayout.SOUTH);
        frame.setSize(200, 200);
        frame.show();
    }
}
```

Además de `JSlider`, el conjunto de componentes Swing también nos ofrece un elemento `JScrollBar`. Su funcionamiento es parecido a `JSlider`. Este elemento se utiliza principalmente para definir la región de deslizamiento. Debido a la poca frecuencia de su uso (por la disponibilidad de `JScrollPane`), no veremos su descripción.

Agregar campos de texto

`JtextField` implementa campos de texto de una única línea. Tiene cinco constructores:

```
public JTextField()
public JTextField(int columns)
public JTextField(String text)
public JTextField(String text, int columns)
public JTextField(Document doc, String text, int columns)
```

El comportamiento de los campos de texto es el mismo que el de una clase relacionada `JTextArea`. Ambas provienen de su clase principal, llamada `JTextComponent`. Estos componentes albergarán el texto del que se tienen que hacer cargo y dibujar. Se puede permitir que el usuario edite este texto y utilice el cursor para seleccionar parte de 61. Esta capacidad se suele utilizar en los procesos copiar/cortar/pegar del sistema operativo. Los componentes `JTextField` y `JTextArea` comparten la mayoría de los métodos definidos en la clase `JTectComponent` relacionados con estos componentes de texto.

La figura **13.22** muestra un componente `JTextField` incrustado dentro de un marco al que se le ha aplicado una justificacion a la derecha, posiblemente para que se introduzca un valor numérico. El listado **13.21** es el programa del ejemplo.

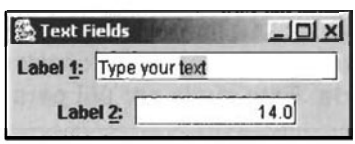


Figura 13.22. Componente JTextField con un texto seleccionado y un campo con justificacion a la derecha

Listado 13.21. Trabajar con diferentes campos de texto

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class TextFieldTest {
    public static void main(String args[]) {
        JFrame frame = new ExitableJFrame("Text Fields");
        Container contentPane = frame.getContentPane();
        // En primer lugar
        JPanel panel = new JPanel();
```

```
JLabel label = new JLabel("Label 1: ");
JTextField text = new JTextField("Type your text", 15);
label.setDisplayedMnemonic(KeyEvent.VK_1);
label.setLabelFor(text);
panel.add(label);
panel.add(text);
contentPane.add(panel, BorderLayout.NORTH);
// En segundo lugar
panel = new JPanel();
label = new JLabel("Label 2: ");
text = new JTextField("14.0", 10);
text.setHorizontalAlignment(JTextField.RIGHT);
label.setDisplayedMnemonic(KeyEvent.VK_2);
label.setLabelFor(text);
panel.add(label);
panel.add(text);
contentPane.add(panel, BorderLayout.SOUTH);
frame.pack();
frame.show();
```

Agregar un componente `JTextField` es parecido a agregar cualquier componente Swing. Podemos llamar a su constructor y utilizar el método `add()` para colocar el objeto resultante en el contenedor. En este ejemplo, el primer conjunto `JTextField` configura la cadena de texto "Type your text" dentro de una columna con un ancho de 15 caracteres. El segundo parámetro no limita la cantidad de caracteres que se pueden introducir, únicamente su representacidn en pantalla (aproximadamente 15 caracteres).

Truco

Los componentes `JTextField` tienen una propiedad muy interesante que nos puede resultar de gran ayuda: podemos centrar el texto o justificarlo a la derecha o a la izquierda. Esto suele ser útil para aplicaciones numericas.

El programa del listado 13.21 también muestra un `JLabel` junto a cada campo de texto, con el nemotécnico de la etiqueta configurado para mover el foco de entrada hasta el componente de texto asociado. Asociar una etiqueta a un campo de texto no es obligatorio, pero hace que el uso del teclado sea más sencillo para el usuario.

Ocultar campos de contraseñas

La clase `JTextField` tiene una subclase, `JPasswordField` que permite que se oculte el texto que aparece en la pantalla para evitar que nadie lo pueda leer. `JPasswordField` proporciona un conjunto de constructores similar a la de `JTextField`.

```
public JPasswordField()
public JPasswordField(int columns)
public JPasswordField(String text)
public JPasswordField(String text, int columns)
public JPasswordField(Document doc, String text, int columns)
```

La principal aplicación de esta propiedad es con las contraseiias de entrada, tal y como se puede ver en la figura **13.23** y en el listado **13.22.**

Figura 13.23. Dos JPasswordField

Listado 13.22. Introducción de texto oculto

```
import java.awt.*;
import javax.swing.*;

public class PasswordTest {
    public static void main(String args[]) {
        JFrame frame = new ExitableJFrame("Password Fields");
        Container contentPane = frame.getContentPane();
        contentPane.setLayout(new GridLayout(2, 2));
        contentPane.add(new JLabel("Enter password:"));
        JPasswordField passwordField = new JPasswordField(10);
        contentPane.add(passwordField);
        contentPane.add(new JLabel("Enter secondary password:"));
        passwordField = new JPasswordField(10);
        passwordField.setEchoChar('#');
        contentPane.add(passwordField);
        frame.pack();
        frame.show();
    }
}
```

El carácter "echo" se utiliza para ocultar los caracteres que aparecen en la pantalla. Los sustituimos por asteriscos en la primera contraseiia y por almohadillas en la segunda.

```
passwordField.setEchoChar('#');
```

Ubicación de las áreas de texto

El componente `JTextArea` se utiliza en los casos en que hay que colocar una gran cantidad de texto en un campo. Éste componente tiene seis constructores:

```
public JTextArea()
public JTextArea(int rows, int cols)
public JTextArea(String text)
```

```
public: JTextArea(String text, int rows, int cols)
public JTextArea(Document doc)
public JTextArea(Document doc, String text, int rows, int cols)
```

La figura 13.24 nos muestra el típico componente **`JTextArea`** y el listado 13.23 es el programa asociado.

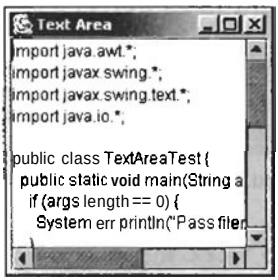


Figura 13.24. Un componente JTextArea

Listado 13.23. Trabajar **con** JTextArea

```
import java.awt.*;
import javax.swing.*;
import javax.swing.text.*;
import java.io.*;

public class TextAreaTest {
    public static void main(String args[]) throws Exception {
        if (args.length == 0) {
           System.err.println("Pass filename to read");
        }
        JFrame frame = new ExitableJFrame("Text Area");
        Container contentPane = frame.getContentPane();
        JTextArea ta = new JTextArea();
        JScrollPane pane = new JScrollPane(ta);
        contentPane.add(pane, BorderLayout.CENTER);
        // Lee la entrada
        Reader reader = new FileReader(args[0]);
        ta.read(reader, null);
        frame.setSize(200, 200);
        frame.show();

}
```

Nota

Cuando se ejecuta este programa, debemos asegurarnos de pasar el nombre del archivo que queremos cargar. El segundo argumento para el método **`read()`** usado en el listado 13.23 es una descripción que pueden aprovechar las tecnologias de ayuda a usuarios con necesidades especiales.

Mostrar HTML

JEditorPane se usa para mostrar y editar texto con formato especial como HTML o Texto con formato enriquecido (RTF). Al igual que JTextField y JTextArea, JEditorPane es una subclase de JTextComponent. Sin embargo, JEditorPane tiene cuatro constructores:

```
public JEditorPane()
public JEditorPane(String type, String text)
public JEditorPane(String url)
public JEditorPane(URL initialPage)
```

En este caso, el argumento constructor es la pdgina que se va a mostrar. Podemos indicar al constructor especificamente lo que se va a mostrar, tanto tipo como contenido:

```
JEditorPane ep = new JEditorPane("text/html", "<H1>Help</H1><H2>Me!</H2>");
```

O podemos pasar al constructor la URL para la pdgina que se va a mostrar y el componente determinard el tipo.

```
JEditorPane ep = new JEditorPane("http://www.google.com");
```

Observe que el componente que muestra el HTML no estd preparado para ser usado como navegador con todas las de la ley ni como editor HTML; sólo es un componente ligero. JEditorPane admite funciones HTML 3.2 como tablas y formularios. Incluso admite hojas en estilo de cascada (CSS).Sin embargo, el componente no admite JavaScript o DHTML. En otras palabras, es un buen visualizador de ayuda.

Para demostrar su uso, el programa del listado 13.24 muestra en modo de sólo lectura cualquier URL que le pase desde la linea de comandos. La figura 13.25 muestra el componente JEditorPane al visualizar la pdgina de Yahoo!. (Hasta que aprenda sobre el control de eventos en el siguiente capitulo, no podrd seguir ningún vínculo con el visor.)

Listado 13.24. Muestra de HTML

```
import java.awt.+;
import javax.swing.*;

public class HTMLTest {
    public static void main(String args[]) throws Exception {
        if (args.length == 0) {
          System.err.println("Pass UKL to read");
        }
        JFrame frame = new ExitableJFrame("HTML Viewer");
        Container contentPane = frame.getContentPane();
        JEditorPane ep = new JEditorPane(args[0]);
        ep.setEditable(false);
        JScrollPane pane = new JScrollPane(ep);
```

```
        contentPane.add(pane, BorderLayout.CENTER);
        frame.setSize(640, 480);
        frame.show();
    }
}
```

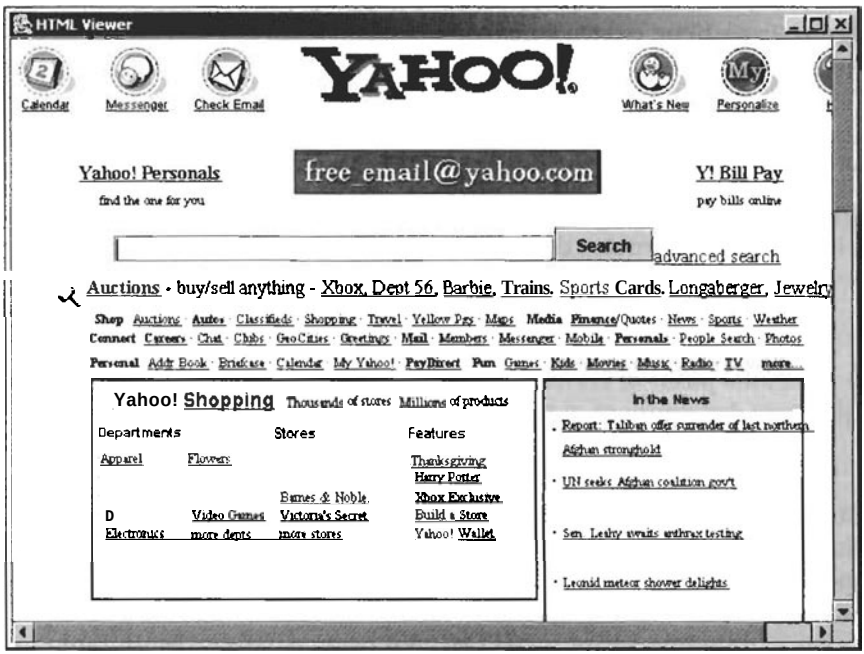


Figura 13.25. Ejernplo de un cornponente JEditorPane

Rotar campos de entrada

En la versión 1.4 del SDK se present6 el cornponente **`JSpinner`**. Si se proporciona al cornponente un conjunto fijo de valores, el usuario puede usar las teclas **arriba** y **abajo** conectadas al componente para desplazarse por las diferentes opciones. Aunque debernos hacer una advertencia: el usuario no puede ver ninguna opción aparte de las que estén seleccionadas. De modo que, a rnenos que conozca las opciones de antemano (por ejemplo, los meses del aiio) este componente puede no ser la rnejor opción para obtener una entrada. La figura 13.26 rnuestra el componente **`JSpinner`**, en el que las opciones son los rneses del año.

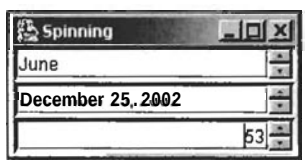


Figura 13.26. Cornponente JSpinner

El componente tiene dos constructores:

```
public JSpinner()
public: JSpinner(SpinnerModel model)
```

SpinnerModel es parte de la arquitectura Model/View/Controller (MVC) de los componentes Swing. Por el momento, todo lo que debe saber es que puede crear un objeto que rota sin el modelo y asignar el modelo posteriormente o sólo iniciar con el modelo. Pero debe crear un modelo en alguna parte y en ese modelo se colocaran las opciones.

Hay tres modelos predefinidos con los que trabajar:

- **SpinnerDateModel:** Para rotar por un rango de fechas. Se especifica el formato de la fecha con la ayuda de la clase `JSpinner.DateEditor`. Por ejemplo, **"MMMMM dd, yyyy"** mostrará el nombre del mes completo, a continuacion dos numeros para el dia del mes y cuatro numeros para el aiio (como en Diciembre 25, 2002). También puede especificar cuánto debe rotar la figura cuando se pulse una de las dos teclas de rotación.

Nota

Para los caracteres de formato usados con `SpinnerDateModel`, las letras "M", "d" e "y" estan definidas junto a las otras opciones disponibles en la clase `java.text.SimpleDateFormat`. Revise el javadoc de la clase `JSpinner` para ver los patrones disponibles.

- **SpinnerListModel:** Para rotar por una matriz de valores, proporcione la matriz; no es necesaria ninguna otra configuración.
- **SpinnerNumberModel:** Para rotar por un rango de numeros defina el rango y el incremento de rotación.

El listado 13.25 es el programa que genera la figura 13.26. Utiliza los tres tipos de rotadores.

Listado 13.25. Demostración del componente JSpinner

```
import java.awt.*;
import javax.swing.*;
import java.util.*;

public class SpinnerTest {
    public static void main(String args[]) {
        JFrame frame = new ExitableJFrame("Spinning");
        Container contentPane = frame.getContentPane();
        String months[] = {"January", "February", "March",
            "April", "May", "June", "July", "August",
            "September", "October", "November", "December"};
```

```
        SpinnerModel model = new SpinnerListModel(months);
        JSpinner spinner = new JSpinner(model);
        contentPane.add(spinner, BorderLayout.NORTH);

        SpinnerDateModel model2 = new SpinnerDateModel();
        model2.setCalendarField(Calendar.WEEK_OF_MONTH);
        JSpinner spinner2 = new JSpinner(model2);
        JSpinner.DateEditor editor2 = new JSpinner.DateEditor(
            spinner2, "MMMMM dd, yyyy");
        spinner2.setEditor(editor2);
        frame.getContentPane().add(spinner2, BorderLayout.CENTER);

        SpinnerNumberModel model3 = new SpinnerNumberModel(50, 0, 100, 5);
        JSpinner spinner3 = new JSpinner(model3);
        frame.getContentPane().add(spinner3, BorderLayout.SOUTH);

        frame.setSize(200, 100);
        frame.show();
    }
```

Formato de campo de texto

`JFormattedTextField` es el otro componente nuevo introducido en la versión 1.4 de la plataforma Java 2. En todo caso, este componente funciona igual que `JSpinner`, pero en lugar de ofrecer la opción de rotar, es el usuario el que tiene que modificar el valor. Sólo tiene que crear el componente y asignar al componente un formato; a partir de ahi sólo se podrán introducir datos con el formato válido.

Hay seis constructores para el componente `JFormattedTextField`:

```
public JFormattedTextField()
public JFormattedTextField(Object value)
public JFormattedTextField(Format format)
public JFormattedTextField(JFormattedTextField.AbstractFormatter formatter)
public JFormattedTextField(JFormattedTextField.AbstractFormatterFactory factory)
public JFormattedTextField(JFormattedTextField.AbstractFormatterFactory factory,
        Object currentValue)
```

En la mayoría de los casos, sólo hay que pasar un valor al componente, desde el constructor o mediante el método `setValue()`. Entonces el componente comprueba el tipo de dato del valor y usa ese tipo de datos como formato válido para el campo. Puede ocurrir que desee especificar un formato aparte del predefinido para un tipo de datos, como para la fecha. En ese caso, debe buscar en las subclases `Format`, que incluyen el `SimpleDateFormat` anteriormente mencionado.

Por ejemplo, para conseguir un campo de entrada numkrico, use el siguiente código para que acepte cualquier numero válido (con o sin decimales):

```
new JFormattedTextField(new Integer(100))
```

Ahora examinemos el listado 13.26, que es un programa que muestra `JFormattedTextField` con campos de entrada de datos y numkrico.

Listado 13.26. Aceptar entradas con formato

```
import java.awt.*;
import javax.swing.*;
import java.util.*;

public class FormattedTest {
    public static void main(String args[]) {
        JFrame frame = new ExitableJFrame("Formatted");
        Container contentPane = frame.getContentPane();
        JFormattedTextField ftf1 = new
            JFormattedTextField(new Integer(0));
        contentPane.add(ftf1, BorderLayout.NORTH);
        JFormattedTextField ftf2 = new
            JFormattedTextField(new Date());
        contentPane.add(ftf2, BorderLayout.SOUTH);
        frame.setSize(200, 100);
        frame.show();
    }
}
```

Cómo ya ha visto el aspecto de un campo de texto, no le ofreceremos una captura de pantalla.

Barras de herramientas

El componente `JToolBar` tiene un contenedor al que se pueden arrastrar elementos y soltarlos alli. Estos son sus constructores:

```
public JToolBar()
public JToolBar(int orientation)
public JToolBar(String name)
public JToolBar(String name, int orientation)
```

El elemento `orientation` es una de las constantes de clase heredadas de `SwingConstants`: HORIZONTAL o VERTICAL. Y el name es para el título de la ventana, cuando se arrastra la barra de herramientas fuera del marco principal.

La figura 13.28 muestra el típico uso de `JToolBar`: ubicar una serie de componentes de tamaño normal dentro de una barra de herramientas colocando un separador entre ellos. El programa tras esta ilustración es el listado 13.27. Cuando se ejecuta el programa `ToolbarTest`, hay que asegurarse de arrastrar la herramienta para obtener las otras configuraciones de barra de tarea que aparecen en la figura 13.27.

Nota

Se puede colocar cualquier componente en una `JToolBar`.

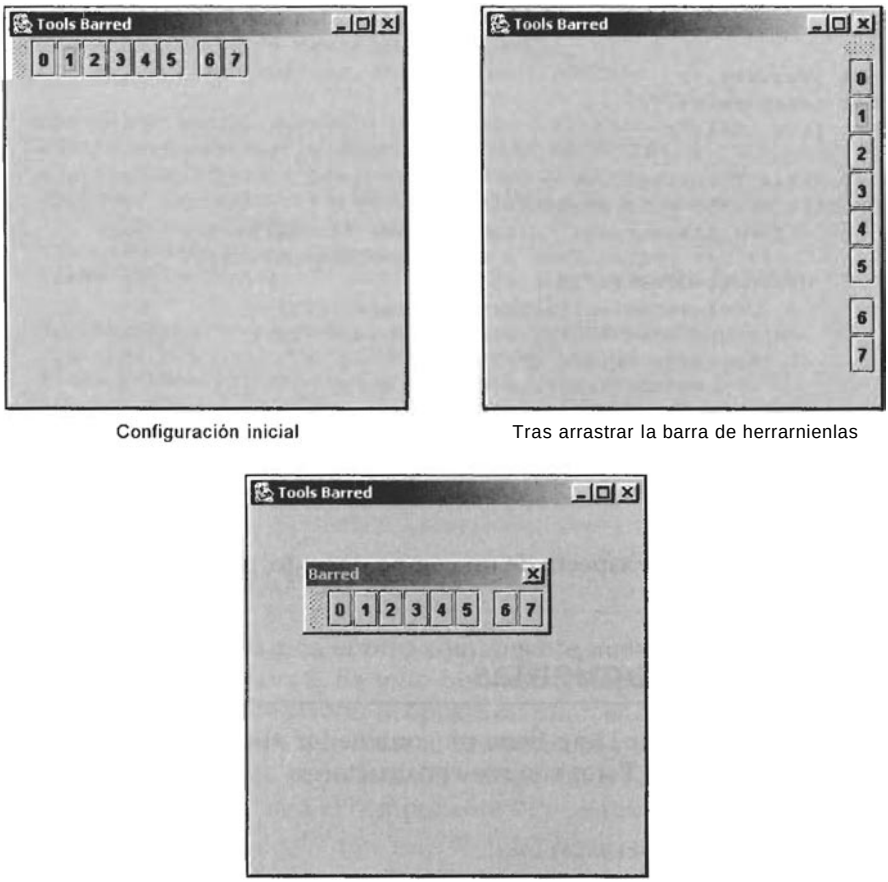


Figura 13.27. Un ejemplo de **JToolBar**

Listado 13.27. Trabajar con una barra de herramientas

```
import java.awt.*;
import javax.swing.*;

public class ToolBarTest {
    public static void main(String args[]) {
        JFrame frame = new ExitableJFrame("Tools Barred");
        Container contentPane = frame.getContentPane();
        JToolBar bar;
        bar = new JToolBar("Barred");
        JToggleButton jb;
        for (int i=0; i < 8; i++) {
           jb = new JToggleButton("" + i);
           bar.add(jb);
           if (i == 5) {
```

```
                bar.addSeparator();
            }
        }
        contentPane.add(bar, BorderLayout.NORTH);
        frame.setSize(300, 300);
        frame.show();
    }
}
```

Trabajar con marcos internos

JIternalFrame es un componente que permite que se pueda modificar el tamaño, agregar iconos, arrastrar la ventana, dentro de los confines de un contenedor externo. La ventana no se puede colocar fuera de él.

El contenedor externo suele ser JDesktopPane, que es una subclase de JLaye**redPane**, pero puede ser cualquier cosa. La definición de la clase JInternelFrame es la siguiente:

```
public JInternalFrame();
public JInternalFrame(String title);
public JInternalFrame(String title, boolean resizable);
public JInternalFrame(String title, boolean resizable, boolean closable);
public JInternalFrame(String title, boolean resizable, boolean closable,
      boolean rnaxirnizable);
public JInternalFrame(String title, boolean resizable, boolean closable,
      boolean rnaxirnizable, boolean iconifiable);
```

La figura **13.28** nos muestra un ejemplo de un elemento InternalFrame. Los tres componentes JInternalFrame se encuentran dentro de JDesktopPane. El listado **13.28** es el programa que lo genera.

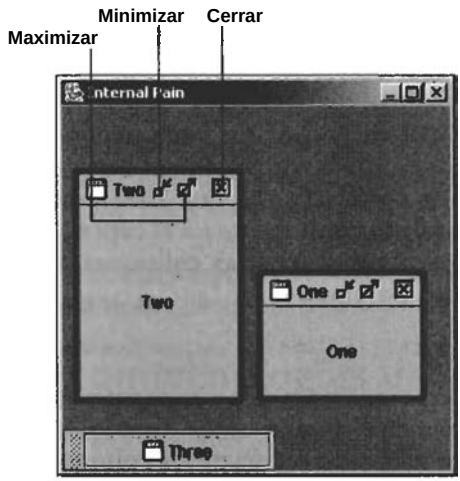


Figura 13.28. Panel JDesktop con tres elementos JInternalFrame.

Listado 13.28. Trabajar con marcos internos en el escritorio

```
import java.awt.*;
import javax.swing.*;

public class InternalTest {
    public static void main(String args[]) {
        JFrame frame = new ExitableJFrame("Internal Pain");
        Container contentPane = frame.getContentPane();
        JLayeredPane desktop = new JDesktopPane();
        desktop.setOpaque(false);
        desktop.add(createLayer("One"), JLayeredPane.POPUP_LAYER);
        desktop.add(createLayer("T o"), JLayeredPane.DEFAULT_LAYER);
        desktop.add(createLayer("Three"), JLayeredPane.PALETTE_LAYER);
        contentPane.add(desktop, BorderLayout.CENTER);
        frame.setSize(300, 300);
        frame.show();
    }
    static JInternalFrame createLayer (String label) {
        return new SelfInternalFrame(label);
    }
    static class SelfInternalFrame extends JInternalFrame {
        public SelfInternalFrame(String s) {
            getContentPane().add (new JLabel(s, JLabel.CENTER),
               BorderLayout.CENTER);
            setBounds(50, 50, 100, 100);
            setResizable(true);
            setClosable(true);
            setMaximizable(true);
            setIconifiable(true);
            setTitle(s);
            setVisible(true);
        }

}
```

Configuración de tablas

El componente JTable es uno de los componentes Swing más completos. De hecho, tiene un paquete completo para él: java.awt.swing.table. Sin embargo, hay un modo sencillo de crear JTable que trataremos aquí. El resto del uso de JTable se estudiará más profundamente en el capítulo 15.

La clase JTable nos permite mostrar columnas de datos. JTable tiene un constructor que toma dos parámetros:

```
public JTable(Object data[][], Object columnHeaders[]).
```

El parámetro de dos dimensiones data hace referencia a los datos que se muestran en la tabla. El parámetro de una unica dimension columnHeader se utiliza para determinar el texto de las cabeceras de las columnas. Al ser ambos pardmetros matrices String, podemos crear un objeto JTable sin tener que hacer nada especial.

En la figura 13.29 tenemos un ejemplo que hemos creado con este procedimiento y el listado 13.29 es el programa que lo genera.

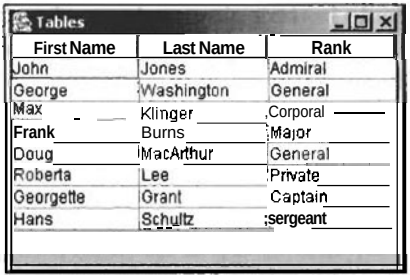


Figura 13.29. Ejemplo JTable

Listado 13.29. Trabajar con el cornponente de tablas

```
import java.awt.*;
import javax.swing.';

public class TableTest {
    public static void main(String args[]) {
        JFrame frame = new ExitableJFrame("Tables");
        Container contentPane = frame.getContentPane();
        String data[][] = {
            {"John", "Jones", "Admiral"},
            {"George", "Washington", "General"},
            {"Max", "Klinger", "Corporal"},
            {"Frank", "Burns", "Major"},
            ["Doug", "MacArthur", "General"},
            ("Roberta", "Lee", "Private"),
            {"Georgette", "Grant", "Captain"},
            {"Hans", "Schultz", "Sergeant"]
        };
        String columns [] = {"First Name", "Last Name", "Rank");
        JTable jt = new JTable(data, columns);
        JScrollPane pane = new JScrollPane(jt);
        contentPane.add(pane, BorderLayout.CENTER);
        frame.setSize(300, 200);
        frame.show();
    }
}
```

Al igual que sucedia con `JList` y `JTextArea`, el componente `JTable` exige que se coloque la tabla en `JScrollPane`.

Compatibilidad con tecnologías de ayuda a discapacitados

Como se mencionó anteriormente, la tecnologia de ayuda a discapacitados está especializada en dispositivos de entrada/salida como lectores de pantalla para

invidentes. Por lo general, estos dispositivos son ayudas para personas con todo tipo de minusvalías, pero también se pueden emplear para proporcionar un mecanismo de entrada secundario cuando no se puede usar el teclado. Los componentes Swing admiten por defecto las tecnologías de ayuda a discapacitados. Sin embargo, puede hacer muchas cosas para que sean aún más útiles.

Por ejemplo, en este capítulo ya hemos examinado el proceso para establecer una descripción para `ImageIcon`:

```
Icon icon = new ImageIcon("logo.gif", "Sybex Corporate Logo");
```

Podría usar una interfaz para leer "Sybex Corporate Logo" en lugar de mostrar el archivo logo.gif en la pantalla. El lector no necesitaría ninguna información especial para leer las etiquetas de texto o las opciones de texto de `JComboBox`; tan sólo diría la etiqueta específica de cada elemento.

El uso de la ayuda para herramientas es otro modo de proporcionar descripciones de los componentes.

Para establecer explícitamente la descripción de un componente, se obtiene el `AccessibleContext` de ese componente y se invoca al método `setAccessibleDescription()`, como en el siguiente ejemplo:

```
AccessibleContext context = jComponent.getAccessibleContext();
context.setAccessibleDescription("Clicking this button will submit the order");
```

Advertencia

Todos los componentes Swing admiten esta tecnología, de modo que si quiere que sus aplicaciones también la admitan, no debe usar componentes o contenedores AWT.

Creación de nuevos componentes

Cuando se crea una subclase de un componente superior como `JWindow` o `JFrame`, se anula el método `paint()` para dibujar en la zona del componente.

Para crear su propio componente de bajo nivel, puede hacer subclases de `JPanel` o `JComponent`. Aunque, en lugar de anular el método `paint()`, anularíamos el método `paintComponent()`. Como ya se ha mencionado, `paint()` invoca a `paintComponent()` y `paintBorder()` (y también `paintChildren()`, si hay componentes en el container). De modo que, suponiendo que quiere que el nuevo componente admita bordes, debe anular el método `paintComponent()` en su lugar.

Para demostrarlo, la definición de la clase `RedBlueBox` del listado 13.30 representa un componente que divide diagonalmente la zona que se muestra en dos regiones de colores, una roja y una azul.

Listado 13.30. Demostracion del componente Box

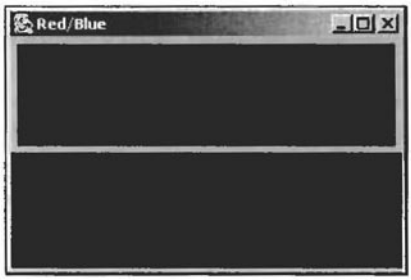
```
import java.awt.*;
import javax.swing.*;

public class RedBlueBox extends JComponent {
//  public void paint(Graphics g) {
    public void paintComponent(Graphics g) {
        Insets insets = getInsets();
        int endX = getWidth() - insets.right;
        int endY = getHeight() - insets.bottom;
        // obtiene la esquina superior izquierda
        int x = insets.left;
        int y = insets.top;

        g.setColor(Color.RED);
        Polygon p = new Polygon();
        p.addPoint(x, y);
        p.addPoint(endX, y);
        p.addPoint(x, endY);
        g.fillPolygon(p);
        g.setColor(Color.BLUE);
        p.reset();
        p.addPoint(endX, y);
        p.addPoint(x, endY);
        p.addPoint(endX, endY);
        g.fillPolygon(p);
    }
}
```

Ahora, agregaremos el siguiente método `main()` para probar el programa en RedBlueBox para agregar dos cuadros a lo que vemos en pantalla, uno con borde y otro sin él.

Deje sin comentarios el método `paint()` anterior para ver la diferencia (y comente la línea `paintComponent()`).

```
public static void main(String args[]) {
      JFrame frame = new ExitableJFrame("Red/Blue");
      Container contentPane = frame.getContentPane();
      contentPane.setLayout(new GridLayout(0, 1));
      JComponent comp = new RedBlueBox();
      comp.setBorder(BorderFactory.createMatteBorder(
          5, 5, 5, 5, Color.PINK));
      contentPane.add(cornp);
      comp = new RedBlueBox();
      contentPane.add(comp);
      frame.setSize(300, 200);
      frame.show();
    }
```

La figura 13.30 muestra las pantallas resultantes. Con `paintComponent()`, se dibuja el borde. Si se anula directamente `paint()`, no se invocará a `paintBorder()`, por lo que no se verá el borde.



Anulandocorrectamente paintComponent() Anulando incorrectamente paintComponent()

Figura 13.30. Componente personalizado, con y sin borde

Componentes del sistema de menu

Los componentes del sistema de menu Swing no son muy distintos a cualquier otro componente. De hecho, proceden de componentes `AbstractButton` como `JButton` y `ToggleButton`. La superclase de todos los componentes de men6 es `JMenuItern`. Sus subclases son **`JMenu`**, `JCheckBoxMenuItern` y `JRadioButtonMenuItem`. Además de poder añadir estos componentes en cualquier sitio donde podamos agregar el resto de componentes Swing, puede colocarlo en un objeto **`JMenuBar`** y colocar éste en un `JApplet` o `JFrame`. El resto del componente es `JPopupMenu` que veremos brevemente antes de que termine el capítulo.

Nota

`JMenuBar` es sólo otro `JComponent`, de modo que tambien puede colocarse en cualquier punto en el que se pueda colocar un componente. A pesar de esta flexibilidad, puede parecer un poco extraño si no agregamos `JMenuBar` en algun sitio donde el usuario espere encontrarlo.

Agregar barras de menus

El propósito principal de la clase **`JMenuBar`** es agrupar una colección de casos **`JMenu`**. A este respecto, su método **`add()`** es el corazón del método (igual que los métodos **`add()`** de la clase **`Container`**). La clase **`JMenuBar`** sólo tiene un constructor, **`public JMenuBar()`**.

Agregar menús

La finalidad de la clase **`JMenu`** es agrupar una colección de `JMenuItem` y otros submenus. El corazon del método **`JMenu`** es el método **`add()`**, que se utiliza para

especificar (de forma incremental) una lista de elementos de un menu (o menus). Hay tres constructores:

```
public JMenu();
public JMenu(String label);
public JMenu(String label, boolean tearOff);
```

La clase `JMenu` también incluye una propiedad que no soportan todas las plataformas: los menús desplegables. Este menú desplegable se puede quedar abierto mientras el usuario lo desee, con lo que los elementos se seleccionan más fácilmente. Obviamente, la plataforma en la que se ejecute la aplicación que incluya este elemento ha de poder trabajar con él. En caso contrario, no tendra ningún efecto.

También podemos añadir un separador a un menú, colocándolo en cualquier punto de la lista. Para ello utilizaremos el método `setSeparator()`. Es muy útil, sobre todo cuando los menus son muy largos. Con estos elementos conseguimos ordenar los elementos de los menus y evitar que la lista de opciones se convierta en un caos.

Nota

Cuando utilizamos el método `addSeparator()` para agregar un separador, en realidad estamos añadiendo otro elemento `JComponent` al menu. En este caso, el componente es `JSeparator`. No se suele utilizar la clase de forma directa. Pero existe y se puede utilizar para crear líneas verticales en la pantalla.

Agregar elementos de menú

La clase `JMenuItem` (asi como `JCheckboxMenuItem` y `JRadioButtonMenuItem`) recoge el ultimo elemento del menu. Estos son sus constructores:

```
public JMenuItem()
publlc JMenuItem(Icon icon)
public JMenuItem(String label)
public JMenuItem(String label, int mnemonic)
public JMenuItem(String label, Icon icon)
public JMenuItem(Action action)
```

Los elementos del menú se pueden activar y desactivar sobre la marcha para que reflejen el estado de una aplicación. Por ejemplo, el men6 **Edicion** suele escribir en gris **Cortar** y **Copiar** (es decir, que están desactivados) cuando no se ha seleccionado ningun elemento (texto, imagen, etc.)

En el mismo momento en que el usuario selecciona una parte del texto, estas opciones del men6 **Edicion** se activarán, es decir, que estarán a disposición del usuario.

Truco

JMenuItem es una subclase de AbstractionButton, es decir, que podemos hacer que aparezca un icono en lugar de un menú.

La clase JCheckBoxMenuItem cuenta con el estado activo/desactivo (que se muestra grdficamente por medio de una marca de verificación o cualquier otro gráfico similar). Éstos son sus constructores:

```
public JCheckBoxMenuItem()
public JCheckBoxMenuItem(Icon icon)
public JCheckBoxMenuItem(String label)
public JCheckBoxMenuItem(String label, boolean selected)
public JCheckBoxMenuItem(String label, Icon icon)
public JCheckBoxMenuItem(String label, Icon icon, boolean selected)
public JCheckBoxMenuItem(Action action)
```

La clase JRadioButtonMenuitem se parece a JRadioButton, con la salvedad que aparece dentro de un menú. Incorpora el estado activo/desactivo, pero como estos botones aparecen en grupos, sólo se puede activar uno de ellos. Éstos son sus constructores:

```
public JRadioButtonMenuItem()
public JRadioButtonMenuItem(Icon icon)
public JRadioButtonMenuItem(Icon icon, boolean selected)
public JRadioButtonMenuItem(String label)
public JRadioButtonMenuItem(String label, boolean selected)
public JRadioButtonMenuItem(String label, Icon icon)
public JRadioButtonMenuItem(String label, Icon icon, boolean selected)
public JRadioButtonMenuItem(Action action)
```

Agregar menus emergentes

La clase JPopupMenu es una subclase de JComponent. Construimos y publicamos un men6 desplegable igual que un mend convencional. La única diferencia entre ambos tipos de menus es que no adjuntamos el men6 desplegable a un elemento de la barra de menus. Lo que hacemos es mostrar temporalmente el men6 desplegable por encima de cualquier otro componente. Dicho componente es el que efectúa la llamada y su configuración corre a cargo del método setInvoker(). Cuando el usuario efectúa una selección, el men6 emergente desaparecerd.

Para mostrar un men6 desplegable utilizamos el siguiente método:

```
show(Component source, int x, int y)
```

El pardmetro Component es el componente sobre el que se basard el men6 desplegable. Los dos partimetros enteros son las coordenadas x e y que determinan la posición donde aparecerá el menú. Por lo general, estas coordenadas son la posición del puntero donde el usuario desea ver el menú emergente.

Un programa para construir una barra de menus

La siguiente aplicación utiliza las clases de los menús para construir una barra de menú adjunta a una ventana. Como hay tantas clases para los menús, no podemos utilizar un ejemplo para verlas todas:

Listado 13.31. Uso de menús

```
import java.awt.*;
import javax.swing.*;

public class MenuTest extends ExitableJFrame {
    public MenuTest() {
        super("Menus Menus");
        ImageIcon icon = new ImageIcon("smalllogo.gif");
        Image image = icon.getImage();
        setIconImage(image);
        JMenu fileMenu = new JMenu("File");
        fileMenu.add(new JMenuItem("Open"));
        fileMenu.add(new JMenuItem("Close"));
        fileMenu.add(new JMenuItem("Exit"));
        JMenu helpMenu = new JMenu("Help");
        helpMenu.add(new JMenuItem("About MenuTest"));
        helpMenu.add(new JMenuItem("Class Hierarchy"));
        helpMenu.addSeparator();
        helpMenu.add(new JCheckBoxMenuItem("Balloon Help"));
        JMenu subMenu = new JMenu("Categories");
        JRadioButtonMenuItem rb;
        ButtonGroup group = new ButtonGroup();
        subMenu.add(rb = new JRadioButtonMenuItem("A Little Help", true));
        group.add(rb);
        subMenu.add(rb = new JRadioButtonMenuItem("A Lot of Help"));
        group.add(rb);
        helpMenu.add(subMenu);
        JMenuBar mb = new JMenuBar();
        mb.add(fileMenu);
        mb.add(helpMenu);
        setJMenuBar(mb);
    }
    public static void main(String args[]) {
        JFrame frame = new MenuTest();
        frame.setSize(300, 300);
        frame.show();
    }
}
```

Nota

Este programa `MenuTest` tambien cambia el icono de marco. Asi que, en lugar del icono de la taza de cafe que se usa por defecto, podemos seleccionar otro icono personalitado.

En la clase `MainWindow`, para construir la barra de menús se crea un nuevo objeto JMenuBar y se utiliza el método `add()` para adjuntarle los dos menús. Cada menu individual se construye por separado. Para vincularlo a la ventana, utilizamos el método `setJMenuBar()` de `JFrame`. pasando el caso JMenuBar. La figura 13.31 nos muestra la barra de menús en acción.

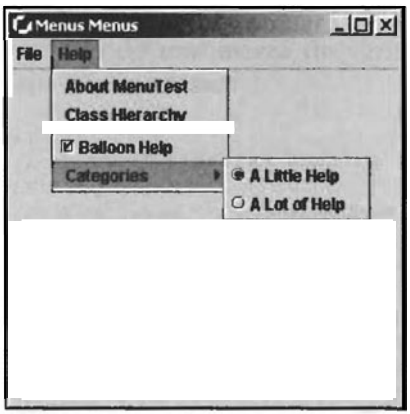


Figura 13.31. JMenuBar con elementos Jmenus y JMenuItemComponente (y un submenu)

Resumen

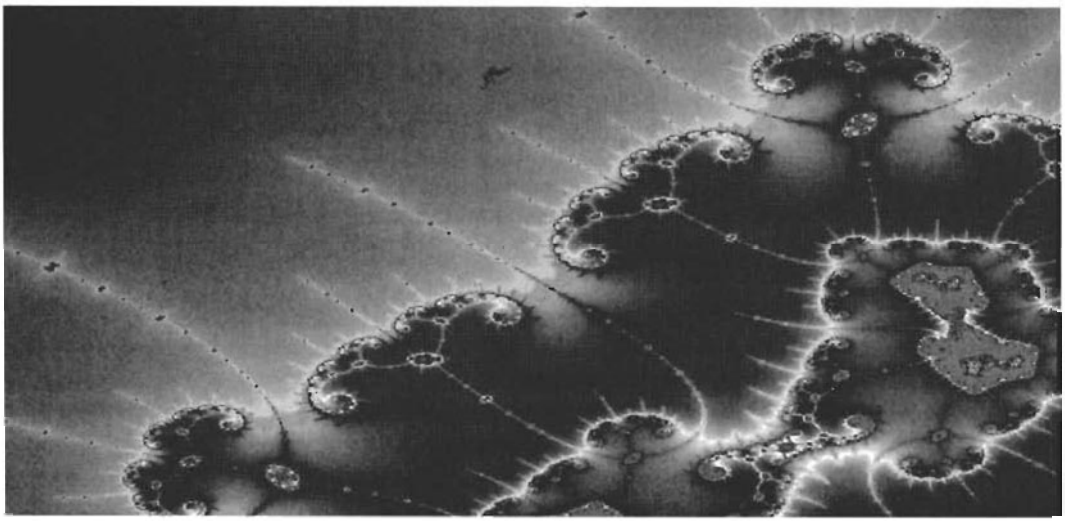
El conjunto de componentes Swing descrito en este capitulo es sólo una pequeña parte de es las Java Foundation Classes (JFC).Si agrega los dos niveles de funciones de dibujo y de administradores de diseiio, tendra un conjunto de herramientas muy variado. Por supuesto, hay mucho más por aprender. Hasta ahora sólo hemos visto cómo crear y colocar los componentes en la pantalla. Todavia hay que aprender a interactuar con ellos. Por sí mismas, el tamaño de las bibliotecas Swing es mayor que el de la plataforma Java 1.0 original. Para ayudarle a entender mejor este amplio conjunto de componentes, la tabla 13.5 enumera cada componente y su función. Aunque aun no ha visto todos, la lista le ayudará a tener una visión general.

Tabla 13.5. Resumen de componentes Swing

Componente	Descripción
`Box`	Container that uses a BoxLayout
`JApplet`	Clase base para applets Swing
`JButton`	Componente seleccionable que admite presentación de texto/imágenes

Componente	Descripción
JCheckBox	Componente seleccionable que muestra el estado al usuario
JCheckBoxMenuItem	Componente seleccionable para un menu; muestra el estado al usuario
JColorChooser	Para seleccionar colores
JComboBox	Para seleccionar de una lista desplegable de opciones
JComponent	Clase base para componentes Swing
JDesktopPane	Contenedor para marcos internos
JDialog	Clase base subventanas desplegables
JEditorPane	Para editar y mostrar contenido con formato
JFileChooser	Para seleccionar archivos y directorios
JFormattedTextField	Para editar y mostrar una sola línea de texto con formato
JFrame	Clase base para ventanas de nivel superior
JInternalFrame	Clase base para ventana de nivel superior internas
JLabel	Para mostrar textolimagenes
JLayeredPane	Contenedor que admite componentes superpuestos
JList	Para ofrecer una lista que se puede desplazar de opciones
JMenu	Componente seleccionable para contener elementos de menu; admite presentacion de textolimagenes
JMenuBar	Para contener menús
JMenuItem	Componente seleccionable que admite presentacion de textolimagenes
JOptionPane	Para crear mensajes emergentes
JPanel	Componente contenedor basico
JPasswordField	Para editar y mostrar una contraseiia
JPopupMenu	Para contener elementos de menu y aparecer sobre los componentes
JProgressBar	Para mostrar el progreso de una operacion del usuario
JRadioButton	Componente seleccionable que muestra el estado al usuario; incluido en `ButtonGroup` para asegurar que solo se selecciona un boton
JRadioButtonMenuItem	Componente seleccionable para menus; muestra el estado al usuario; incluido en `ButtonGroup` para asegurar que sólo se selecciona un boton

Componente	Descripción
JRootPane	Contenedor interno usado por JFrame, JApplet y otros
JScrollBar	Para controlar una zona que se puede desplazar
JScrollPane	Para proporcionar la posibilidad de desplazamiento a otro componente
JSeparator	Para colocar una línea separadora en un menu o barra de herramientas
JSlider	Para seleccionar entre un valor de un rango numérico de valores
JSpinner	Para seleccionar un valor entre un conjunto de valores, entre una lista, entre un rango numérico o entre un rango de datos
JSplitPane	Contenedor que permite al usuario seleccionar la cantidad de espacio para cada dos componentes
JTabbedPane	Contenedor que permite que se muestren varios contenedores; cada contenedor aparece en una ficha
JTable	Para mostrar datos tabulados
JTextArea	Para editar y mostrar contenido de texto con un solo atributo
JTextField	Para editar y mostrar contenido de texto con un solo atributo en una sola línea
JTextPane	Para editar y mostrar contenido de texto con varios atributos
JToggleButton	Componente seleccionable que admite presentación de texto/imágenes; al seleccionarlo el componente permanece dentro
JToolBar	Contenedor que se puede arrastrar
JToolTip	Usado de forma interna para mostrar ayuda de herramientas sobre los componentes
JTree	Para mostrar datos jerarquicos
JViewport	Contenedor para contener un componente demasiado grande para la zona en la que se muestra
JWindow	Clase base para ventanas emergentes



14
Control de eventos

Ya conoce la administración de diseño y los componentes. En este capitulo estudiaremos la tercera parte de la triada GUI. Un evento en Java es una notificación cuando sucede algo interesante. Por tanto, el control de eventos es la respuesta a esa notificación. Si no está interesado en un evento, no tiene por qué responder. Por otro lado, agregar a los programas código para que reconozca y controle los eventos en los que estamos interesados es parte de la creación de interfaces que respondan al usuario. Probablemente quiera realizar alguna acción en respuesta a que un usuario hace clic en un botón.

Este capitulo le enseñará a crear interfaces que respondan a las acciones del usuario. En primer lugar, veremos de forma general el modelo de control de eventos basado en delegados, donde los componentes delegan las acciones de control de eventos a otros. (En realidad, éste es el segundo método que tiene Java para controlar eventos. El primero ya no se usa y no lo estudiaremos aqui.) Tras esta revisión, veremos muchos de los eventos más usados, los componentes con los que trabajan y cómo responder cuando suceda un evento.

Control de eventos basado en delegación

El mecanismo de control de eventos permite el registro de objetos con componentes. El registrante recibe una notificación cuando sucede algo interesante en el componente. Si conoce los patrones de disefio, el mecanismo de control de eventos de Java sigue el patrón observer.

El patrón `Observer` funciona de esta manera: El lugar donde sucede el evento recibe el nombre de sujeto. En el mundo **GUI,** este sujeto es el componente. El sujeto advierte de que se han apreciado ciertos eventos. En Java, la capacidad de observar se anuncia proporcionando un método de registro para dicho evento. Este método de registro tendrá un nombre parecido a `addEventListener()`, donde `Event` es el tipo especifico soportado. (Se debe proporcionar un método de cancelación del registro, llamado `removeEventListener()`.) Cuando algo se registra como un oyente para un sujeto, ese algo recibe el nombre de observador. Así, cuando el evento sucede en el sujeto, se avisa a todos los observadores registrados. La figura 14.1 le ayudara a visualizar este proceso.

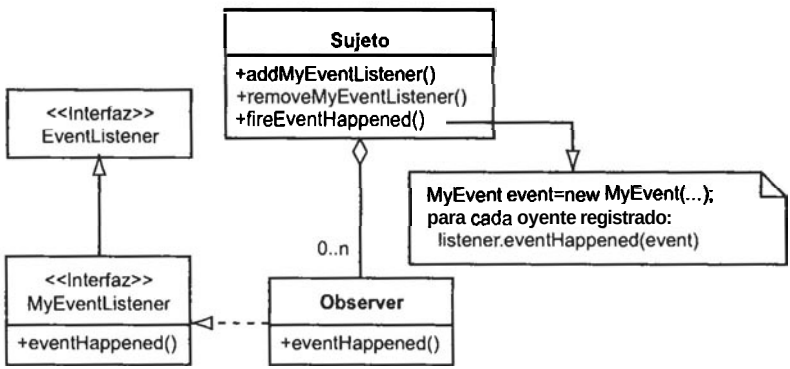


Figura 14.1. El patron de diseño Observer

Nota

Como muestra la figura 14.1, todos los observadores deben implementar una interfaz que sea subclase de la interfaz `java.util.EventListener`. Este requisito es en beneficio de la arquitectura de componentes JavaBeans.

Para demostrar este mecanismo de delegación, trabajaremos con un ejemplo de respuesta a la selección de un componente de botón, `JButton`. Parte del proceso de aprendizaje del trabajo con componentes Swing es descubrir qué registro de oyente se necesita para crear una respuesta interesante. Por ejemplo, si miramos la clase `JButton` no veremos métodos que se parezcan a `addEventListener()`. En ese caso, debe ascender en la jerarquia de clases hasta la clase `AbstractButton`. Aqui, encontrará estos tres métodos:

```
addActionListener()
addChangeListener()
addItemListener()
```

Si ascendemos aún más, aparecerán 13 oyentes más que pueden registrarse. Procedente de `javax.swing.JComponent`:

```
addAncestorListener()
addPropertyChangeListener()
addVetoableChangeListener()
```

De `java.awt.Container`:

```
addContainerListener()
addPropertyChangeListener() [otra vez]
```

Y de `java.awt.Component`:

```
addComponentListener()          addKeyListener()
addFocusListener()              addMouseListener()
addHierarchyBoundsListener()    addMouseMotionListener()
addHierarchyListener()          addMouseWheelListener()
addInputMethodListener()        addPropertyChangeListener() [por tercera vez]
```

Con todas ellas, hay 16 observadores diferentes que se pueden registrar con un componente `JButton`. Por desgracia, no hay un modo fdcil de saber qué oyente responde a un evento interno concreto. Algunos nombres, como el oyente de la rueda del ratón, pueden sugerir asociaciones a su correspondiente controlador de eventos; pero otros nombres pueden producir confusion (por ejemplo, `addAncestorListener()`).

En el caso del componente `JButton`, los oyentes de acciones registradas controlan las respuestas a la selección. Asi, el observador debe implementar la interfaz `ActionListener` y registrarse a si mismo con el sujeto (el botón) mediante el método `addActionListener()`.

La interfaz `ActionListener` consiste en un sólo método:

```
public interface ActionListener extends EventListener {
    public void actionPerformed(ActionEvent e)
}
```

Asi, si queremos observar cuando se produce la selección de un botón, debemos crear una clase que implemente la interfaz. A continuación tiene una de estas implementaciones que escribe un mensaje:

```
import java.awt.event.*;

public class MyActionListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        System.out.println("Ouch!");
    }
}
```

Volviendo al ejemplo `ButtonTest` del ultimo capitulo, podemos agregar el código de respuesta creando un caso de `MyActionListener` y asocidndolo con el botón mediante `addActionListener()`, como se indica aqui:

```
JButton b = ...
ActionListener listener = new MyActionListener();
b.addActionListener(listener);
```

Observar eventos de componente no suele limitarse a un sólo oyente. En la mayoría de los casos, podemos incluir tantos como queramos. Por ejemplo, en nuestro ejemplo JButton, si tenemos dos observadores, podemos crear una segunda implementación ActionListener y asociarla, también, al botón. Sin embargo, en lugar de crear una nueva clase, podemos usar una clase interna anónima para el oyente. Éste será su aspecto:

```
b.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent actionEvent) {
        System.out.println("Hey, stop that!");
    }
});
```

Nota

En algunos casos especiales, las clases pueden limitar el numero de observadores a uno solo.

Todo esto junto produce el listado 14.1.

Listado 14.1. Escuchando la selección JButton

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class ButtonTest {
    public static void main(String args[]) {
        JFrame frame = new ExitableJFrame("First Button");
        Container contentPane = frame.getContentPane();
        JButton b = new JButton("My First Button!");
        ActionListener listener = new MyActionListener();
        b.addActionListener(listener);
        b.addActionListener(new ActionListener() {
           public void actionPerformed(ActionEvent actionEvent) {
               System.out.println("Hey, stop that!");
           }
        });
        contentPane.add(b, BorderLayout.NORTH);
        frame.setSize(300, 200);
        frame.show();
```

Advertencia

Cuando trabaje con varios observadores para un solo sujeto, es importante saberque el orden en el que los observadores se agregan al sujeto no repercute en el orden en el que son notificados. Nada nos garantiza que el observador registrado en primer lugar sea el primero en ser notificado.

Basicamente, eso es todo lo que hay que saber respecto al modelo de control de eventos basado en delegación del conjunto de componentes Swing. Lo complicado es comprender lo que son los diferentes eventos y qué oyentes de eventos deben registrarse si queremos observar lo que ocurre en un evento concreto. Y eso es lo que haremos en el resto del capitulo.

Tipos de eventos y oyentes

El codigo para tratar con eventos aparece en parejas. Hay una interfaz para registrar un observador y hay una clase para el tipo de evento que estamos observando. La mayoría de los eventos que deberemos controlar al trabajar con componentes Swing se encuentra en el paquete `java.awt.event`. Se pueden encontrar más tipos de eventos y oyentes en el paquete `javax.swing.event`.

Del mismo modo que la interfaz `EventListener` es la raiz de todos los oyentes de eventos, hay una clase que es la raiz de todos lo tipos de eventos: `java.util.EventObject`. La clase proporciona un entorno basico desde el que trabajan todos los eventos. Como indica la tabla 14.1, este entorno común sólo indica que todos los eventos deben tener una fuente.

Tabla 14.1. Composición de la clase EventObject

Variables/Métodos	Descripción
source	Punto de origen
`EventObject()`	Crea un objeto `EventObject`
`getSource()`	Recupera el punto de origen
`toString()`	Devuelve una cadena que representa el estado del evento

En el mundo de los componentes Swing, muchos eventos interesante son subclases de la clase `java.awt.AWTEvent` y se encuentran en el paquete `java.awt.event` package. Observe la figura 14.2.

Como se indico anteriormente, no todos los eventos se encuentran en el paquete de eventos AWT. Cuando se introdujo por primera vez Swing, tenia que contenerse a si mismo. Los eventos especificos del conjunto de componentes Swing permanecieron en los paquetes Swing y alli siguen.

La figura **14.3** muestra todos los eventos del paquete `javax.swing.event`. Las clases relacionadas incluyen el nombre de su paquete entre comillas invertidas (« y »).

Con todas estas clases de eventos, no es de extraiiar que la parte más complicada del control de eventos para componentes sea aprender qué oyente de eventos se necesita para asociarlo con el componente fuente para observar el comportamiento

que nos interesa. Las siguientes secciones son descripciones de los tipos de eventos y pares de oyentes más comunmente usados, junto a donde cómo se usan.

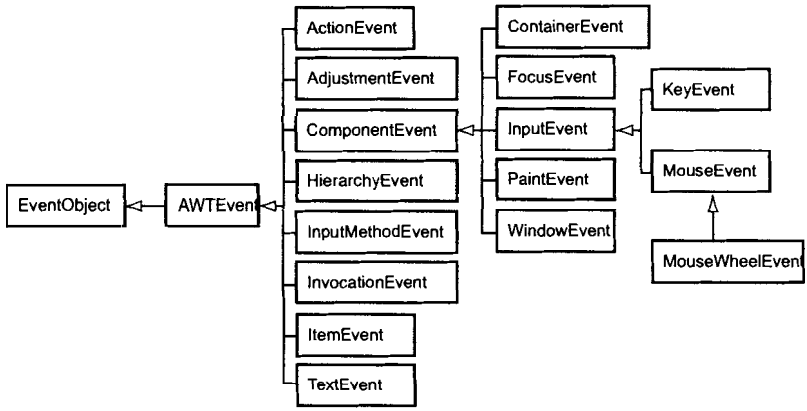


Figura 14.2. Jerarquia de clases de eventos AWT

Algunos eventos y sus oyentes asociados se explicaran mas tarde porque están asociados a componentes que aun no hemos estudiado a fondo. Entre ellos están los siguientes pares: `ListSelectionEvent-ListSelectionListener`, `TableModelEvent-TableModelListener` y `TreeModelEvent-TreeModelListener`.

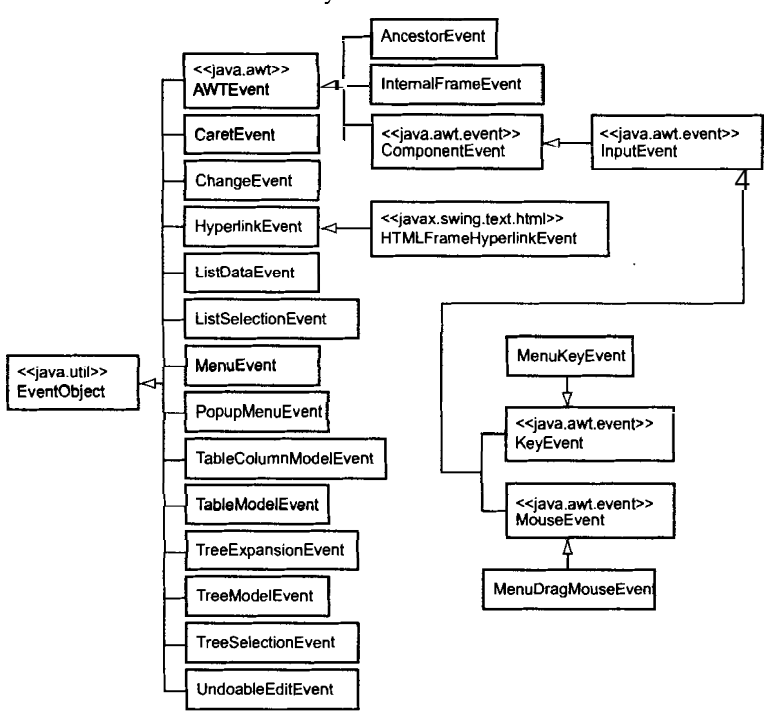


Figura 14.3. Jerarquia de clases de eventos Swing

Eventos de acción

La combinación `ActionListener-ActionEvent` es probablemente el par de control de eventos más usado. De hecho, ya ha visto cómo funciona. Trabaja con casi todos los diferentes componentes seleccionables para eventos. Por parte de Swing, encontraremos habitualmente la combinación **`ActionListener-ActionEvent`** usada en los siguientes casos:

- **`AbstractButton`** (y todas sus subclases, como `JButton`), cuando se selecciona el componente.
- `JComboBox`, cuando se realiza una selección o se ha introducido una selección que se puede editar.
- `JTextField`, cuando se pulsa la tecla **Intro** dentro del componente.

Ya hemos visto el aspecto de la interfaz `ActionListener` y en la tabla 14.2 están los elementos que componen esta clase de eventos.

Nota

Observara que muchas de las clases de eventos tiene constantes para el rango de ID del evento, corno **`ACTION_FIRST`** y **`ACTION_LAST`** en la anterior tabla. Estas constantes se proporcionan por si necesita hacer una subclase del evento para crear un tipo de evento mas especifico. Por ejemplo, necesitara una ID tras **`ACTION_LAST`**. Todas las subclases de **`java.awt.`**`AWTEvent` necesitan una ID.

Tabla 14.2. Cornposicion de la clase ActionEvent

Variables/Métodos	Descripción
ACTION—FIRST	Tipo de evento
ACTION—LAST	
ACTION—PERFORMED	
`ALT_MASK`	Modificador para indicar si una tecla del teclado estaba pulsada cuando sucedio el evento
`CTRL_MASK`	
ME`TA_MAS`K	
SHIFT— MASK	
`ActionEvent()`	Crea un objeto `ActionEvent`

Variables/Métodos	Descripción
`getActionCommand()`	Recupera una cadena de comando
`getModifiers()`	Recupera las teclas rnodificadoras pulsadas
`getWhen()`	Recupera la hora del evento
`paramString()`	Crea una cadena que representa el evento, se rnostrara con `toString()`

Los modificadores trabajan juntos. Si se pulsan varios modificadores cuando sucede el evento, se combinarán en un entero. A continuación, el método `getModifiers()` devolverá como un int. Deberá usar el operador `&` para ver los que se han pulsado. Este es el marco de código que busca modificadores:

```
int modifiers = actionEvent.getModifiers();
if ((modifiers & ActionEvent.ALT_MASK) == ActionEvent.ALT_MASK) {
    System.out.println("Alt key pressed");
}
```

El comando de acción es una parte de información especifica de cada componente que se puede establecer a nivel de componente o se puede usar su valor por defecto. Para un botón abstracto, el comando por defecto es la etiqueta de texto del botón. Para un campo de texto, el comando por defecto es el contenido. Para un cuadro combinado, es el tipo del evento. El listado 14.2 muestra como usar **`ActionListener`** con un `JComboBox` que se puede editar, un `JButton` y un `JTextField`. Se usa el mismo oyente para los tres y muestra los modificadores y el comando de acción. Mostrar el contenido de `JComboBox` seleccionado requiere que se agregue un caso especial al oyente. Si se ejecuta el programa `ActionTest` se genera la pantalla que aparece en la figura 14.4.

Listado 14.2. Dernostracion de ActionListener

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class ActionTest {

    public static void main(String args[]) {
        JFrame frame = new ExitableJFrame("Action Listener");
        Container contentFane = frame.getContentPane();
        ActionListener listener = new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                System.out.println("Command: " + e.getActionCommand());
                System.out.println("Modifiers: ");
                int modifiers = e.getModifiers();
                System.out.println("\tALT  : " +
                    checkMod(modifiers, ActionEvent.ALT_MASK));
                System.out.println("\tCTRL : " +
                    checkMod(modifiers, ActionEvent.CTRL_MASK));
```

```
                System.out.println("\tMETA : " +
                    checkMod(modifiers, ActionEvent.META_MASK));
                System.out.println("\tSHIFT: " +
                    checkMod(modifiers, ActionEvent.SHIFT_MASK));
                Object source = e.getSource();
                if (source instanceof JComboBox) {
                    JComboBox jb = (JComboBox)source;
                    System.out.println("Combo: " + jb.getSelectedItem());
                }
            }
            private boolean checkMod(int modifiers, int mask) {
                return ((modifiers & mask) == mask);
            }
        };

        String flavors[] = {"Chocolate", "Strawberry", "Vanilla", "Honey",
            "Chocolate Chip", "Watermelon", "Rasberry", "Banana"};
        JComboBox jc = new JComboBox(flavors);
        jc.setMaximumRowCount(4);
        jc.setEditable(true);
        jc.addActionListener(listener);
        contentPane.add(jc, BorderLayout.NORTH);

        JButton b = new JButton("My First Button!");
        b.addActionListener(listener);
        contentPane.add(b, BorderLayout.CENTER);

        JPanel panel = new JPanel();
        JLabel label = new JLabel("Label 1: ");
        JTextField text = new JTextField("Type your text", 15);
        text.addActionListener(listener);
        label.setDisplayedMnemonic(KeyEvent.VK_1);
        label.setLabelFor(text);
        panel.add(label);
        panel.add(text);
        contentPane.add(panel, BorderLayout.SOUTH);

        frame.pack();
        frame.show();
    }
}
```

Figura 14.4. Resultados del programa de prueba ActionListener

Eventos de ajuste

Cuando queremos saber si se mueve `JScrollBar`, registramos `AdjustmentListener` con el componente. Esto incluye el movimiento de las barras de

desplazamiento aisladas, además de uno de los componentes JScrollBar de JScrollPane. Cuando el evento tiene lugar, se envía AdjustmentEvent al único método oyente, describiendo el movimiento. A continuación tiene la definición de la interfaz para el oyente. La composición de clase de AdjustmentEvent se encuentra en la tabla 14.3.

```
public interface AdjustmentListener extends EventListener {
    public void adjustmentValueChanged(AdjustmentEvent e)
```

Tabla **14.3.** Cornposicion de la clase **AdjustmentEvent**

Variables/Métodos	Descripción
BLOCK – DECREMENT	Tipo de ajuste
BLOCK – INCREMENT	
TRACK	
UNIT_DECREMENT	
UNIT – INCREMENT	
ADJUSTMENT – FIRST	Tipo de evento
ADJUSTMENT_LAST	
ADJUSTMENT – VALUE – CHANGED	
`AdjustmentEvent()`	Crea un objeto `AdjustmentEvent`
`getAdjustable()`	Recupera la fuente del evento
`getAdjustmentType()`	Recupera el tipo de ajuste
`getValue()`	Recupera el valor de la fuente
`getValueIsAdjusting()`	Recupera la si la fuente aún esta moviendose activamente
`paramString()`	Crea una cadena que representa el evento para que la muestre `toString()`

Normalmente, no escucharia el movimiento de la barra de tareas desde un JScrollPane. Sin embargo, si cree que es necesario controlar ese suceso, el listado 14.3 muestra esta función. En JScrollPane, encontrará un JLabel con una imagen.

Listado 14.3. Demostracion de AdjustmentListener

```
import java.awt.*;
import java.awt.event.*;
```

```
import javax.swing.*;

public class AdjustmentTest {
    public static void main(String args[]) {
        JFrame frame = new ExitableJFrame("Adjustment Listener");
        Container contentPane = frame.getContentPane();
        Icon icon = new ImageIcon("examnotes.gif", "Java 2 Exam Notes");
        JButton b = new JButton(icon);
        JScrollPane pane = new JScrollPane(b);
        AdjustmentListener hListener = new AdjustmentListener() {
            public void adjustmentValueChanged(AdjustmentEvent e) {
                System.out.println("Horizontal: ");
                dumpInfo(e);
            }
        };
        JScrollBar hBar = pane.getHorizontalScrollBar();
        hBar.addAdjustmentListener(hListener);
        AdjustmentListener vListener = new AdjustmentListener() {
            public void adjustmentValueChanged(AdjustmentEvent e) {
                System.out.println("Vertical: ");
                dumpInfo(e);
            }
        };
        JScrollBar vBar = pane.getVerticalScrollBar();
        vBar.addAdjustmentListener(vListener);
        contentPane.add(pane, BorderLayout.CENTER);
        frame.setSize(300, 200);
        frame.show();
    }
    private static void dumpInfo(AdjustmentEvent e) {
        System.out.println("\tValue: " + e.getValue());
        String type = null;
        switch (e.getAdjustmentType()) {
            case AdjustmentEvent.TRACK:
                type = "Track";
                break;
            case AdjustmentEvent.BLOCK_DECREMENT:
                type = "Block Decrement";
                break;
            case AdjustmentEvent.BLOCK_INCREMENT:
                type = "Block Increment";
                break;
            case AdjustmentEvent.UNIT-DECREMENT:
                type = "Unit Decrement";
                break;
            case AdjustmentEvent.UNIT_INCREMENT:
                type = "Unit Increment";
                break;
        }
        System.out.println("\tType: " + type);
    }
}
```

Cuando ejecute el programa, observe que no sólo se avisa a los oyentes cuando se mueven las barras de desplazamiento, si no también cuando se inicia el programa.

Eventos padre

Los eventos padre pertenecen exclusivamente a `JComponent` y todas sus subclases (bdsicamente, todos los componentes Swing, pero ningun componente AWT o ventana de nivel superior). Si registra un `AncestorListener` con cualquier componente le permitirá descubrir cuando ese componente (o alguno de los contenedores en los que se encuentra) se vuelve visible o invisible, se mueve o se agrega o elimina de una jerarquia de componentes. La siguiente interfaz consiste en un método para cada uno de los tres tipos de eventos padre. La composición de `AncestorEvent` se encuentra en la tabla 14.4.

```
public interface AncestorListener extends EventListener {
    public void ancestorAdded(AncestorEvent e)
    public void ancestorMoved(AncestorEvent e)
    public void ancestorRemoved(AncestorEvent e)
}
```

Nota

Cambiar el tamaiio de un componente padre se considera una operacion de movimiento porque parte del componente padre se mueve como parte de la operacion de cambio de tamaño.

Tabla 14.4. Composición de la clase AncestorEvent

Variables/Métodos	Descripción
ANCESTOR—ADDED	Tipo de evento
ANCESTOR—MOVED	
ANCESTOR—REMOVED	
`AncestorEvent()`	Crea un objeto `AncestorEvent`
`getAncestor()`	Recupera el contenedor en el que ocurrio el evento
`getAncestorParent()`	Recupera un contenedor del contenedor en el que ocurrió el evento
`getComponent()`	Recupera el componente desde el que algo esta escuchando

Como verd en la siguiente sección, podemos averiguar cuando un componente determinado se vuelve visible o se mueve mediante `ComponentListener`. Además, el menos usado `AncestorListener` puede ser útil cuando queremos conocer cuando el marco de un componente aparece, desaparece o se mueve. El listado

14.4 muestra AncestorListener proporcionando un botón que ocultará el marco durante cinco segundos.

Listado 14.4. Dernostracion de AncestorListener

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
import java.util.Timer;
import java.util.TimerTask;

public class AncestorTest {
    public static void main(String args[]) {
        final JFrame frame = new ExitableJFrame("Ancestor Listener");
        frame.setName("The Frame");
        Container contentPane = frame.getContentPane();
        JButton b = new JButton("Hide for 5");
        b.setName("The Button");
        ActionListener action = new ActionListener() {
            public void actionPerformed(ActionEvent actionEvent) {
                frame.setVisible(false);
                TimerTask task = new TimerTask() {
                    public void run() {
                        frame.setVisible(true);
                    }
                };
                Timer timer = new Timer();
                timer.schedule(task, 5000);
            }
        };
        b.addActionListener(action);
        AncestorListener ancestor = new AncestorListener() {
            public void ancestorAdded(AncestorEvent e) {
                System.out.println("Added");
                dumpInfo(e);
            }
            public void ancestorMoved(AncestorEvent e) {
                System.out.println("Moved");
                dumpInfo(e);
            }
            public void ancestorRemoved(AncestorEvent e) {
                System.out.println("Removed");
                dumpInfo(e);
            }
            private void dumpInfo(AncestorEvent e) {
                System.out.println("\tAncestor: " +
name(e.getAncestor()));
                System.out.println("\tAncestorParent: " +
name(e.getAncestorParent()));
                System.out.println("\tComponent: " +
name(e.getComponent()));
            }
            private String name(Container c) {
                return (c == null) ? null : c.getName();
```

```
            }
        });
        b.addAncestorListener(ancestor);
        contentPane.add(b, BorderLayout.NORTH);
        frame.setSize(300, 200);
        frame.show();
    }
}
```

Nota

Observe en el listado 14.4 que el ocultamiento se realiza con un subproceso secundario temporizador. Esto se debe a que el control de eventos podría bloquearse durante cinco segundos si lo hicieramos con una llamada a `Thread.sleep(5000)` directamente en el oyente de acciones. Específicamente, si lo hubiera puesto en estado de letargo directamente en el oyente de acciones, no se habria avisado al oyente padre de que el marco se habia ocultado hasta después de que el marco pareciera de nuevo.

Eventos de componente

Los eventos de componentes pertenecen exclusivamente a la clase **`Component`** y sus subclases. Esto incluye todos los componentes AWT y Swing, incluyendo ventanas superiores.

`ComponentListener` se parece a `AncestorListener` en que se puede percibir cuando un componente se vuelve visible o invisible, se mueve o cambia su tamaño. Sin embargo, con `ComponentListener` no obtenemos el aspecto "agregado o eliminado de la jerarquia de componentes" de `AncestorListener`.

Ese comportamiento se encuentra en `ContainerListener`, descrito a continuacion.

El oyente consta de cuatro métodos separados, uno para cada una de las operaciones que necesitan una respuesta. A cada método se le envía un **`ComponentEvent`**, como se enumera en la tabla 14.5.

```
public interface ComponentListener extends EventListener {
    public void componentHidden(ComponentEvent e)
    public void componentMoved(ComponentEvent e)
    public void componentResized(ComponentEvent e)
    public void componentShown(ComponentEvent e)
}
```

El listado 14.5 muestra la escucha del componente. Se añaden dos botones a `JSplitPane`, cada uno de ellos tiene un **`ComponentListener`**. El botón izquierdo también tiene un `ActionListener` que alterna el estado de visibilidad del botón derecho.

Tabla 14.5. **Composición** de la **clase** ComponentEvent

Variables/Métodos	Descripción
COMPONENT – FIRST	Tipo de **evento**
COMPONENT – LAST	
COMPONENT_HIDDEN	
COMPONENT – MOVED	
COMPONENT – RESIZED	
COMPONENT – SHOWN	Tipo de **evento**
`ComponentEvent()`	Crea un **objeto** `ComponentEvent`
`getComponent()`	**Recupera** la fuente del **evento**
`paramString()`	Crea una cadena que representa el **evento** para que la **muestre** `toString()`

Listado 14.5. Demostracion de **ComponentListener**

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class ComponentTest {
    public static void main(String args[]) {
        JFrame frame = new ExitableJFrame("Component Listener");
        Container contentPane = frame.getContentPane();

        ComponentListener comp = new ComponentListener() {
            public void componentHidden(ComponentEvent e) {
                dump("Hidden", e);
            }
            public void componentMoved(ComponentEvent el {
                dump("Moved", e);
            }
            public void componentResized(ComponentEvent e) {
                dump("Resized", e);
            }
            public void componentShown(ComponentEvent e) {
                dump("Shown", e);
            }
            private void dump(String type, ComponentEvent e) {
              System.out.println(e.getComponent().getName() + " : " + type);
            }
        };

        JButton left = new JButton("Left");
        left.setName("Left");
```

```
        left.addComponentListener(comp);

        final JButton right = new JButton("Right");
        right.setName("Right");
        right.addComponentListener(comp);

        ActionListener action = new ActionListener() {
             public void actionPerformed(ActionEvent e) {
                right.setVisible(!right.isVisible());
             }
        };
        left.addActionListener(action);

        JSplitPane pane = new JSplitPane(
             JSplitPane.HORIZONTAL_SPLIT, true, left, right);

        contentPane.add(pane, BorderLayout.CENTER);

        frame.setSize(300, 200);
        frame.show();
    }
}
```

Nota

Cuando ejecute el programa del listado 14.5, observe que al mover el divisor entre los componentes notifica al componente derecho de un movimiento o un cambio de tamaño, pero el componente izquierdo solo obtiene el cambio de tamaño. Esto se debe a que el origen del componente izquierdo no cambia con el cambio de tamaño, lo que no sucede con el componente derecho.

Eventos de contenedor

En la clase **container** tenemos los eventos de contenedor. Desde la clase JComponent, la clase base de todos los componentes Swing es una subclase de **Container**, esto significa que todos los componentes Swing avisan del evento. Aunque está disponible para todos los componentes, el par oyente de eventos sólo es realmente relevante para componentes que contienen otros componentes, de ahi el tipo de contenedor.

El oyente tiene dos métodos, para que se le avise cuando se agreguen o eliminen componentes del contenedor. Cada método recibe un ContainerEvent, cuya descripción aparece en la tabla 14.6.

```
public interface ContainerListener extends EventListener {
    public void componentAdded(ContainerEvent e)
    public void componentRemoved(ContainerEvent e)
}
```

Recuerde que en la figura 14.2 se mostraba que `ContainerEvent` es una subclase de `ComponentEvent`. Aqui, `getContainer()` y `getComponent()` devuelven el mismo objeto, al igual que `getSource()`. Cada método devuelve el objeto como un tipo diferente. Además, cuando se compara `ContainerEvent` con `AncestorEvent`, la fuente del evento en un `ContainerEvent` es el contenedor. Para `AncestorEvent`, el evento fuente es el componente.

Tabla **14.6.** Composición de la clase ContainerEvent

Variables/Métodos	Descripción
CONTAINER—FIRST	Tipo de evento
CONTAINER—LAST	
COMPONENT—ADDED	
COMPONENT—REMOVED	
`ContainerEvent()`	Crea un objeto **`ContainerEvent`**
`getChild()`	Recupera el componente que se esta agregando o eliminando
`getContainer()`	Recupera el contenedor desde el que se esta agregando o eliminado el componente
`paramString()`	Crea una cadena que representa el evento para que la muestre `toString()`

Para demostrarlo, el listado 14.6 agrega indirectamente oyentes de acción a una serie de botones. Aunque, en lugar de agregar el oyente directamente a los botones, el contenedor registra al oyente cuando el botón se agrega al contenedor.

Listado 14.6. Demostracion del ContainerListener

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class ContainerTest {
    public static void main(String args[]) {
        JFrame frame = new ExitableJFrame("Container Listener");
        Container contentPane = frame.getContentPane();

        ContainerListener cont = new ContainerListener() {
            ActionListener listener = new ActionListener() {
                public void actionPerformed(ActionEvent e) {
                    System.out.println("Selected: " + e.getActionCommand());
                }
            };
            public void componentAdded(ContainerEvent e) {
```

```
            Component c = e.getChild();
            if (c instanceof JButton) {
                JButton b = (JButton)c;
                b.addActionListener(listener);
            }
        }
        public void componentRemoved(ContainerEvent e) {
            Component c = e.getChild();
            if (c instanceof JButton) {
                JButton b = (JButton)c;
                    b.removeActionListener(lister)   ;
                }
            }
        };

        contentPane.addContainerListener(cont);

        contentPane.setLayout(new GridLayout(3, 2));
        contentPane.add(new JButton("First"));
        contentPane.add(new JButton("Second"));
        contentPane.add(new JButton("Third"));
        contentPane.add(new JButton("Fourth"));
        contentPane.add(new JButton("Fifth"));

        frame.setSize(300, 200);
        frame.show();
    }
}
```

Eventos focales

Otro evento procedente de Component y todas sus subclases es el **evento** focal. Los eventos focales tienen lugar cuando el componente gana o pierde foco de entrada. El oyente para los eventos focales tiene un **método** para cada uno de los dos tipos de **evento:**

```
public interface FocusListener extends EventListener {
    public void focusGained(FocusEvent e)
    public void focusLost(FocusEvent e)
}
```

Cuando se avisa a `FocusListener`, cada **método** de interfaz recibe un `FocusEvent`, otro subtipo de `ComponentEvent`. La tabla **14.7** enumera la **composición** de este tipo de **evento.**

Tabla 14.7. **Composición** de la clase **FocusEvent**

Variables/Métodos	Descripción
FOCUS—FIRST	Tipo de evento
FOCUS—LAST	

Variables/Métodos	Descripción
`FOCUS_GAINED`	
`FOCUS_LOST`	
`FocusEvent()`	Crea un objeto **`FocusEvent`**
`getOppositeComponent()`	Recupera el componentesecundario involucrado en un evento
`isTemporary()`	Recupera si el cambio defoco es temporal, lo contrario de permanente (ver la nota)
`paramString()`	Crea una cadena que representa el evento para que la muestre `toString()`

Nota

En un carnbio de foco temporal, el foco de entrada se pierde debido a un evento como que otra ventana reciba un foco de entrada. Cuando la ventana principal obtiene el foco de nuevo, también lo hara el componente.

El listado 14.7 muestra el componente que está escuchando. Se han agregado dos botones a `JSplitPane`; cada uno tiene un `ComponentListener`. El botón izquierdo también tiene un `ActionListener` que activa o desactiva el estado de visibilidad del boton derecho.

Listado 14.7. Demostracion de FocusListener

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class FocusTest {
    public static void main(String args[]) {
        JFrame frame = new ExitableJFrame("Focus Listener");
        Container contentPane = frame.getContentPane();

        FocusListener listener = new FocusListener() {
            public void focusGained(FocusEvent e) {
                dumpInfo(e);

            public void focusLost(FocusEvent e) {
                dumpInfo(e);
            }
            private void dumpInfo(FocusEvent e) {
                System.out.println("Source   : " t
                    name(e.getComponent()));
                System.out.println("Opposite : " +
```

```
                name(e.getOppositeComponent()));
            System.out.println("Temporary: " +
                e.isTemporary());
        }
        private String name(Component c) {
            return (c == null) ? null : c.getName();
        }
    };

    // Primero
    JPanel panel = new JPanel();
    JLabel label = new JLabel("Label 1: ");
    JTextField text = new JTextField("Type your text", 15);
    text.setName("First");
    text.addFocusListener(listener);
    label.setDisplayedMnemonic(KeyEvent.VK_1);
    label.setLabelFor(text);
    panel.add(label);
    panel.add(text);
    contentPane.add(panel, BorderLayout.NORTH);

    // Segundo
    panel = new JPanel();
    label = new JLabel("Label 2: ");
    text = new JTextField("14.0", 10);
    text.setName("Second");
    text.addFocusListener(listener);
    text.setHorizontalAlignment(JTextField.RIGHT);
    label.setDisplayedMnemonic(KeyEvent.VK_2);
    label.setLabelFor(text);
    panel.add(label);
    panel.add(text);
    contentPane.add(panel, BorderLayout.SOUTH);

    frame.pack();
    frame.show();
```

Nota

El uso de `FocusListener` es sólo un pequeño modo de interactuar con el subsistema Focus Subsystem.

Eventos de hipervínculo

Los eventos de hipervinculo pertenecen exclusivamente al componente `JEditorPane`. Podemos registrar un oyente cuando el componente está mostrando HTML y podemos descubrir cuando se introduce, se saca o se activa un hipervinculo en el componente.

```
public interface HyperlinkListener extends EventListener {
    public void hyperlinkUpdate(HyperlinkEvent e)
}
```

Cuando se avisa a `HyperlinkListener`, el método de interfaz recibe un `HyperlinkEvent`. La tabla 14.8 enumera la composición de este tipo de evento.

Tabla 14.8. Composición de la clase HyperlinkEvent

Variables/Métodos	Descripción
`HyperlinkEvent()`	Crea un objeto `HyperlinkEvent`
`getDescription()`	Recupera la descripción de un vínculo
`getSourceElement()`	Recupera la fuente del evento como `Element`
`getURL()`	Recupera la URL asociada

`getEventType()` devuelve el tipo real del evento. El valor devuelto será una de las tres constantes de la clase interna `HyperlinkEvent.EventType`:

- ACTIVATED
- ENTERED
- EXITED

El listado 14.8 completa el ejemplo del listado 13.24 del capitulo 13. El código agregado hace que `JEditorPane` siga los hipervinculos cuando el usuario los selecciona.

Listado 14.8. Demostración de HyperlinkListener

```
import java.awt.*;
import javax.swing.*;
import javax.swing.event.*;
import java.io.*;

public class HyperlinkTest {
    public static void main(String args[]) {
        if (args.length == 0) {
            System.err.println("Pass URL to read");
            System.exit(-1);
        }

        JFrame frame = new ExitableJFrame("Hyperlink Listener");
        Container contentPane = frame.getContentPane();

        final JEditorPane ep = new JEditorPane();

        try {
            ep.setPage(args[0]);
        } catch (IOException e) {
            System.err.println("Bad URL: " + e);
```

```
            System.exit(-1);
        }

        HyperlinkListener listener = new HyperlinkListener() {
            public void hyperlinkUpdate(HyperlinkEvent e) {
                if (e.getEventType() == HyperlinkEvent.EventType.ACTIVATED) {
                    try {
                        ep.setPage(e.getURL());
                    } catch (IOException ioe) {
                        System.err.println("Error loading: " + ioe);
                    }
                }
            }

        };

        ep.addHyperlinkListener(listener);
        ep.setEditable(false);
        JScrollPane pane = new JScrollPane(ep);
        contentPane.add(pane, BorderLayout.CENTER);
        frame.setSize(640, 480);
        frame.show();
    }
```

Eventos de marcos internos

Los eventos de rnarcos internos son otro tipo de evento que es especifico de un sólo cornponente. En este caso, ese componente es `JInternalFrame`. Como muestra la siguiente definición, hay siete subtipos de eventos que se pueden escuchar desde un `InternalFrameListener`:

```
public interface InternalFrameListener extends EventListener {
    public void internalFrameActivated(InternalFrameEvent e)
    public void internalFrameClosed(InternalFrameEvent e)
    public void internalFrameClosing(InternalFrameEvent e)
    public void internalFrameDeactivated(InternalFrameEvent e)
    public void internalFrameDeiconified(InternalFrameEvent e)
    public void internalFrameIconified(InternalFrameEvent e)
    public void internalFrameOpened(InternalFrameEvent e)
}
```

Cuando se avisa a un `InternalFrameListener`, cada método recibe un `InternalFrameEvent`. La tabla 14.9 enumera la composición de este tipo de evento.

Tabla 14.9. Composición de la clase InternalFrameEvent

Variables/Métodos	Descripción
INTERNAL_FRAME_FIRST	Tipo de evento
INTERNAL – FRAME – LAST	

Variables/Métodos	Descripción
INTERNAL – FRAME – ACTIVATED	
INTERNAL – FRAME – CLOSED	
INTERNAL – FRAME – CLOSING	
INTERNAL – FRAME – DEACTIVATED	
INTERNAL – FRAME – DEICONIFIED	
INTERNAL – FRAME – ICONIFIED	
INTERNAL – FRAME – OPENED	
`InternalFrameEvent()`	Crea un objeto `InternalFrameEvent`
`getInternalFrame()`	Recupera la fuente del evento como `JInternalFrame`
`paramString()`	Crea una cadena que representa el evento para que lo muestre `toString()`

El listado 14.9 amplia el ejemplo `JInternalFrame` (listado 13.27) del capítulo 13 para que muestre un mensaje cuando sucede cada tipo de evento de marco interno.

Listado 14.9. Demostracion de InternalFrameListener

```
import java.awt.*;
import javax.swing.*;
import javax.swing.event.*;

public class InternalFrameTest {
    public static void main(String args[]) {
        JFrame frame = new ExitableJFrame("Internal Frame Listener");
        Container contentPane = frame.getContentPane();
        JLayeredPane desktop = new JDesktopPane();
        desktop.setOpaque(false);
        desktop.add(createLayer("One"), JLayeredPane.POPUP_LAYER);
        desktop.add(createLayer("Two"), JLayeredPane.DEFAULT_LAYER);
        desktop.add(createLayer("Three"), JLayeredPane.PALETTE_LAYER);
        contentPane.add(desktop, BorderLayout.CENTER);
        frame.setSize(300, 300);
        frame.show();
    }
        static JInternalFrame createLayer(String label) {
            return new SelfInternalFrame(label);
        }
    static class SelfInternalFrame extends JInternalFrame {
        InternalFrameListener listener = new InternalFrameListener() {
            public void internalFrameActivated(InternalFrameEvent e) {
                dumpInfo("Activated", e);
```

```
        public void internalFrameClosed(InternalFrameEvent e) {
            dumpInfo("Closed", e);
        }
        public void internalFrameClosing(InternalFrameEvent e) {
            dumpInfo("Closing", e);

        public void internalFrameDeactivated(InternalFrameEvent e) {
            dumpInfo("Deactivated", e);
        }
        public void internalFrameDeiconified(InternalFrameEvent e) {
            dumpInfo("Deiconified", e);

        public void internalFrameIconified(InternalFrameEvent e) {
            dumpInfo("Iconified", e);

        public void internalFrameOpened(InternalFrameEvent e) {
            dumpInfo("Opened", e);
        }
        private void dumpInfo(String s, InternalFrameEvent e) {
            System.out.println("Source: " +
                e.getInternalFrame().getName() + " : " + s);
        }
    };
    public SelfInternalFrame(String s) {
        getContentPane().add (new JLabel(a, JLabel.CENTER),
            BorderLayout.CENTER);
        setName(s);
        addInternalFrameListener(listener);
        setBounds(50, 50, 100, 100);
        setResizable(true);
        setClosable(true);
        setMaximizable(true);
        setIconifiable(true);
        setTitle(s);
        setVisible(true);
    }

}
```

Eventos de elemento

Al pasar a los eventos de elemento llegamos a las subclases de componente `JComboBox` y `AbstractButton`. Un evento de elemento tiene lugar cuando se selecciona un elemento dentro del componente `JComboBox`, o cuando se selecciona un boton abstracto. Podría pensar que podemos responder a la selección de un boton asociando un `ActionListener`, pero el `ItemListener` es una elección mas natural para componentes que tienen estado, como `JCheckBox`. Usando la pareja `ItemListener`/`ItemEvent`, le indica al evento el estado del componente que se puede seleccionar. (Por otro lado, con una pareja `ActionListener`/ `ActionEvent`, deberiamos volver y preguntar por el estado a la fuente del evento.) Sólo hay un método en la definición de `ItemListener`:

```
public interface ItemListener extends EventListener {
    public void itemStateChanged(ItemEvent e)
}
```

Aquí, el método oyente recibe un **ItemEvent** cuando ocurre el evento. La clase **ItemEvent** aparece descrita en la tabla 14.10.

Tabla **14.10.** Composición de la clase ItemEvent

Variables/Métodos	Descripción
ITEM - FIRST	Tipo de evento
ITEM_LAST	
ITEM_STATE_CHANGED	
DESELECTED - SELECTED	Tipo de carnbio de estado
ItemEvent()	Crea un objeto ItemEvent
getItem()	Recupera el elemento afectado
getItemSelectable()	Recuperalafuente del evento como ItemSelectable
getStateChange()	Recupera el tipo de cambio de estado
paramString()	Crea una cadena que representa el evento para que lo rnuestre toString()

Para demostrar **ItemEvent** y ItemListener, el listado 14.10 muestra un grupo de componentes JRadioButton en un ButtonGroup y un JComboBox. Todos estos componentes están asociados a un oyente que vuelca información sobre el evento.

Listado 14.10. Demostracion de ItemListener

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class ItemTest {
    public static void main(String args[]) {
        JFrame frame = new ExitableJFrame("Item Listener");
        Container contentPane = frame.getContentPane();

        ItemListener listener = new ItemListener()
           public void itemStateChanged(ItemEvent e) {
               System.out.println("Source: " + name(e.getSource()));
               System.out.println("Item: " + name(e.getItem()));
               int state = e.getStateChange();
               System.out.println("State: " +
                  ((state == ItemEvent.SELECTED) ?
```

```
                    "Selected" : "Deselected"));
            }
            private String name(Object o) {
                if (o instanceof JComponent) {
                    JComponent comp = (JComponent)o;
                    return comp.getName();
                } else {
                    return o.toString();
                }
            }
        };

        JPanel panel = new JPanel(new GridLayout(0, 1));
        ButtonGroup group = new ButtonGroup();
        JRadioButton option = new JRadioButton("French Fries", true);
        option.setName(option.getText());
        option.addItemListener(listener);
        group.add(option);
        panel.add(option);
        option = new JRadioButton("Onion Rings", false);
        option.setName(option.getText());
        option.addItemListener(listener);
        group.add(option);
        panel.add(option);
        option = new JRadioButton("Ice Cream", false);
        option.setName(option.getText());
        option.addItemListener(listener);
        group.add(option);
        panel.add(option);
        contentPane.add(panel, BorderLayout.NORTH);

        String flavors[] = {"Chocolate", "Strawberry", "Vanilla", "Honey",
                "Chocolate Chip", "Watermelon", "Rasberry", "Banana"};
        JComboBox jc = new JComboBox(flavors);
        jc.setName("Combo");
        jc.addItemListener(listener);
        jc.setMaximumRowCount(3);
        contentPane.add(jc, BorderLayout.SOUTH);

        frame.pack();
        frame.show();
    }
}
```

Si se ejecuta el programa `ItemTest` se generará la pantalla que aparece en la figura 14.5.

Figura 14.5. Los resultados del prograrna de prueba ItemListener

Cuando ejecute el programa, observe que para cambio de estado ocurren dos eventos de elemento. El primero es una indicación que el componente actualmente seleccionado ha sido deseleccionado. El segundo evento de elemento indica la selección del segundo componente.

Eventos de tecla

Ahora volvamos al tema de los eventos que salen de todas las subclases `Component`; éstos son los eventos de tecla. Se generan cuando se pulsa o se deja de pulsar alguna de las teclas del teclado. El oyente para los eventos de tecla tiene un método para la pulsación de una tecla, uno para cuando se deja de pulsar y un tercero para una tecla escrita:

```
public interface KeyListener extends EventListener {
    public void keyPressed(KeyEvent e)
    public void keyReleased(KeyEvent e)
    public void keyTyped(KeyEvent e)
}
```

Las diferencias entre `keyPressed()` y `keyTyped()` requiere alguna explicación. Supongamos que queremos introducir una **A** mayuscula en un campo de texto; esto genera cinco eventos de tecla diferentes.

1. En primer lugar, se pulsa la tecla **Mayús** para indicar que queremos una letra maytiscula.
2. Esto activa una llamada a `keyPressed()` para la tecla **Mayús**.
3. A continuación, se pulsa la tecla **A** para indicar que queremos escribir la letra **A.**
4. Esto activa una llamada a `keyPressed()` para la tecla **A.**
5. Esto también activa otra llamada a `keyTyped()` que indica que el carácter escrito era una A maytiscula. Cuando se deja de pulsar las teclas, se llamara a `keyReleased()` para las teclas **A** y **Mayús**.

El evento de tecla en el que estamos interesados determina lo que estamos escuchando. Para algo como la pulsación de una tecla de función **F1**, F2, F3, etc. considera que estas teclas no se han escrito, sólo se han pulsado. Avisar al oyente es la parte sencilla, pero una vez que tenemos el oyente debemos examinar el caso `KeyEvent` para descubrir información sobre el evento concreto. La tabla **14.11** enumera la composición de `KeyEvent`.

`KeyEvent` y `MouseEvent` son subclases de `InputEvent`. Aunque no usaremos `InputEvent` de forma directa, necesitará saber algunos de los métodos disponibles. Su componente de clase aparece en la tabla **14.12.**

El listado **14.11** muestra el evento de tecla escuchando. El programa sólo contiene un `JTextField` con un `KeyListener`. El oyente envía información sobre el evento.

Tabla 14.11. Composición de la clase KeyEvent

Variables/Métodos	Descripcidn
KEY – FIRST	Tipo de evento
KEY – LAST	
KEY – PRESSED	
KEY – RELEASED	
KEY_TYPED	
KEY – LOCATION – LEFT	Posición de la tecla pulsada o dejada de pulsar
KEY_LOCATION_NUMPAD	
KEY – LOCATION – RIGHT	
KEY_LOCATION – STANDARD	
KEY_LOCATION – UNKNOWN	
CHAR – UNDEFINED	Constante usada para indicar un caracter sin equivalente Unicode
VK_0	Hay casi 200 codigos de tecla para indicar las teclas pulsadas o dejadas de pulsar
VK_1	
VK_2	
...	
VK_RIGHT	
VK_RIGHT – PARENTHESIS	
VK_ROMAN – CHARACTERS	
...	
VK_X	
VK_Y	
VK_Z	
KeyEvent ()	Crea un objeto **KeyEvent**
getKeyChar()	Recupera un caracter
getKeyCode()	Recupera el codigo de tecla
getKeyLocation()	Recupera la posición de una tecla
getKeyModifiersText()	Convierte una cadena de texto que describe modificadores a entero

Variables/Métodos	Descripción
`getKeyText()`	Convierte el codigo de tecla a texto que describe la tecla
`isActionKey()`	Cornprueba si la tecla es una tecla de acción
`paramString()`	Crea una cadena que representa el evento para que lo rnuestre `toString()`
`setKeyChar()`	Cambia el caracter de la tecla
`setKeyCode()`	Cambia el codigo de tecla
`setModifiers()`	Cambia los rnodificadores para la tecla

Tabla **14.12.** Cornposicion de la clase InputEvent

Variables/Métodos	Descripción
`ALT_DOWN_MASK`	Mascaras del modificador
`ALT_GRAPH_DOWN_MASK`	
`ALT_GRAPH_MASK`	
`ALT_MASK`	
`BUTTON1_DOWN_MASK`	
`BUTTON1_MASK`	
`BUTTON2_DOWN_MASK`	
`BUTTON2_MASK`	
`BUTTON3_DOWN_MASK`	
`BUTTON3_MASK`	
`CTRL_DOWN - MASK`	
`CTRL_MASK`	
`META_DOWN_MASK`	
`META_MASK`	
`SHIFT_DOWN_MASK`	
`SHIFT - MASK`	
`consume()`	Marca el evento cómo ya controlado
`getModifiers()`	Obtiene un conjunto de máscaras
`getModifiersEx()`	Obtiene un conjunto de mascaras extendidas

Variables/Métodos	Descripción
getModifiersExText()	Convierte las mascaras de modificador a texto que describe modificadores
getWhen()	Obtiene la marca de hora del evento
isAltDown()	Comprueba el estado de un modificador concreto
isAltGraphDown()	
isControlDown()	
isMetaDown()	
isShiftDown()	
isConsumed()	Comprueba si el evento esta marcado como controlado

Listado 14.11. Demostracion de KeyListener

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class KeyTest {
    public static void main(String args[]) {
        JFrame frame = new ExitableJFrame("Key Listener");
        Container contentPane = frame.getContentPane();

        KeyListener listener = new KeyListener() {
            public void keyPressed(KeyEvent e) {
                dumpInfo("Pressed", e);
            }
            public void keyReleased(KeyEvent e) {
                dumpInfo("Released", e);
            }
            public void keyTyped(KeyEvent e) {
                dumpInfo("Typed", e);

            private void dumpInfo(String s, KeyEvent e) {
                System.out.println(s);
                int code = e.getKeyCode();
                System.out.println("\tCode: " +
KeyEvent.getKeyText(code));
                System.out.println("\tChar: " + e.getKeyChar());
                int mods = e.getModifiersEx();
                System.out.println("\tMods: " +
KeyEvent.getModifiersExText(mods));
                System.out.println("\tLocation: " +
                    location(e.getKeyLocation()));
                System.out.println("\tAction? " + e.isActionKey());
```

```
            private String location(int location) {
                switch(location) {
                    case KeyEvent.KEY_LOCATION_LEFT:
                        return "Left";
                    case KeyEvent.KEY_LOCATION_RIGHT:
                        return "Right";
                    case KeyEvent.KEY_LOCATION_NUMPAD:
                        return "NumPad";
                    case KeyEvent.KEY_LOCATION_STANDARD:
                        return "Standard";
                    case KeyEvent.KEY_LOCATION_UNKNOWN:
                    default:
                    return "Unknown";
                }
            }
        };

        JTextField text = new JTextField();
        text.addKeyListener(listener);
        contentPane.add(test, BorderLayout.NORTH);
        frame.pack();
        frame.show();
    }
}
```

Cuando ejecutamos el programa KeyTest, en realidad no importa lo que escribamos en el campo de texto. Sólo debemos asegurarnos de observar los resultados de cada tecla pulsada y soltada. Si mira cuidadosamente, observará que cuando escribe una tecla, se trabaja con el carácter; cuando la tecla es pulsada o soltada, trabajamos con el código de tecla. Asegurese de probar el proceso de diferenciar entre las teclas **Mayús izquierda** y **Mayús derecha** y los numeros del teclado principal comparados con los del teclado numérico. Antes de Java 1.4, no se podian detectar estas diferencias.

Eventos de menú

A continuacibn: eventos de menu y su oyente asociado, `MenuListener`. Los eventos de menu pertenecen en exclusiva al componente **`JMenu`**, que es el componente que contiene los elementos de menu. Al registrar un **`MenuListener`** con un **`JMenu`**, podemos descubrir cuando el menú está seleccionado, deseleccionado o cancelado. Cada uno de estos sucesos tiene un método en la definicibn de **`MenuListener`**:

```
public interface MenuListener extends EventListener {
    public void menuCanceled(MenuEvent e)
    public void menuDeselected(MenuEvent e)
    public void menuSelected(MenuEvent e)
}
```

El evento asociado al oyente, **`MenuEvent`**, es poco interesante. La definición de clase sólo tiene un constructor.

Para mostrar MenuEvent y MenuListener, el listado 14.12 toma el marco del ejemplo de menús del capitulo 13 (listado 13.31)y asocia el oyente a los dos elementos superiores JMenu y al submenú.

Listado 14.12. Dernostracion de MenuListener

```
import java.awt.*;
import javax.swing.*;
import javax.swing.event.*;

public class MenuTest extends ExitableJFrame {
    public MenuTest() {
        super("Menu Listener");

        MenuListener listener = new MenuListener() {
            public void menuCanceled(MenuEvent e) {
               dumpInfo("Canceled", e);
            }
            public void menuDeselected(MenuEvent e) {
               dumpInfo("Deselected", e);
            }
            public void menuSelected(MenuEvent e) {
               dumpInfo("Selected", e);

            private void dumpInfo(String s, MenuEvent e) {
               JMenu menu = (JMenu)e.getSource();
               System.out.println(s t ": " + menu.getText());
            }
        };

        JMenu fileMenu = new JMenu("File");
        fileMenu.addMenuListener(listener);
        fileMenu.add(new JMenuItem("Open"));
        fileMenu.add(new JMenuItem("Close"));
        fileMenu.add(new JMenuItem("Exit"));
        JMenu helpMenu = new JMenu("Help");
        helpMenu.addMenuListener(listener);
        helpMenu.add(new JMenuItem("About MenuTest"));
        helpMenu.add(new JMenuItem("Class Hierarchy"));
        helpMenu.addSeparator();
        helpMenu.add(new JCheckBoxMenuItem("Balloon Help"));
        JMenu subMenu = new JMenu("Categories");
        subMenu.addMenuListener(listener);
        JRadioButtonMenuItem rb;
        ButtonGroup group = new ButtonGroup();
        subMenu.add(rb = now JRadioButtonMenuItem("A Little Help", true));
        group.add(rb);
        subMenu.add(rb = new JRadioButtonMenuItem("A Lot of Help"));
        group.add(rb);
        helpMenu.add(subMenu);
        JMenuBar mb = new JMenuBar();
        mb.add(fileMenu);
        mb.add(helpMenu);
        setJMenuBar(mb);
```

```
}
public static void main(String args[]) {
    JFrame frame = new MenuTest();
    frame.setSize(300, 300);
    frame.show();
```

En los metodos `MenuListener`, la operación de conversión `(JMenu)e.getSource()` se puede realizar sin comprobar si la fuente del evento es un caso de `JMenu` (porque un JMenu sólo es el tipo de objeto que puede generar un `MenuEvent`).

Truco

Una funcion muy útil de `MenuListener` es esperar hasta que un `JMenu` esté a punto de aparecer antes de rellenar la lista de elementos del menú. Esto es lo que podemos realizar respondiendo a `menuSelected()`.

Eventos de ratón

Otro evento disponible para todas las subclases Component es el evento de ratón. Sdlo hay un `MouseEvent`, pero hay dos opentes que se ocupan de él: `MouseListener` para eventos de ratón sencillos y `MouseMotionListener` para eventos de movimiento.

```
public interface MouseListener extends EventListener {
    public void mouseClicked(MouseEvent e)
    public void mouseEntered(MouseEvent e)
    public void mouseExited(MouseEvent e)
    public void mousePressed(MouseEvent e)
    public void mouseReleased(MouseEvent e)
}
public interface MouseMotionListener extends EventListener {
    public void mouseDragged(MouseEvent e)
    public void mouseMoved(MouseEvent e)
```

Los dos oyentes están separados para mejorar el rendimiento. Los eventos de movimiento suceden con frecuencia, cada vez que mueve el ratón. En la mayoría de los casos, no nos interesa saber cuando se mueve un ratón, sólo cuando se pulsan sus botones. De ahi que se encuentren separados.

MouseEvent informa de la posición y cuando se hace clic. Estos aspectos de la clase se describen en la tabla 14.13. El método `isPopupTrigger()` es necesario porque el evento desencadenante es especifico para cada plataforma. Para un ratón estándar para diestros, en Windows el evento desencadenante podría ser soltar el botón derecho del ratón. Sin embargo, en plataformas Unix el evento seria pulsar el botón izquierdo del ratón.

Tabla 14.13. Cornposicion de la clase MouseEvent

Variables/Métodos	Descripción
BUTTON1	Indicador del boton
BUTTON2	
BUTTON3	
NOBUTTON	
MOUSE— FIRST	Tipo de evento
MOUSE—LAST	
MOUSE—CLICKED	
MOUSE—DRAGGED	
MOUSE—ENTERED	
MOUSE—EXITED	
MOUSE—MOVED	
MOUSE—PRESSED	
MOUSE—RELEASED	
`MouseEvent()`	Crea un objeto `MouseEvent`
`getButton()`	Recupera el indicador de carnbio de un estado del boton
`getClickCount()`	Recupera el número de clic de ratón asociados
`getMouseModifiersText()`	Convierte una cadena de texto que describelos modificadores en un entero
`getPoint()`	Recupera la posición en la fuente del evento
`getX()`	
`getY()`	
`isPopupTrigger()`	Cornprueba si el evento de ratón activara un menu desplegable si hay alguno disponible
`paramString()`	Crea una cadena que representa el evento para que lo muestre `toString()`
`translatePoint()`	Convierte el espacio de coordenada agregando los offset que se le han pasado

El listado 14.13 es un ejemplo más interesante que la mayoría de los que hemos visto hasta ahora. En lugar de volcar la información sobre un evento, el marco admite dos tareas diferentes. En primer lugar, el marco mostrará un men6 desplegable si se genera el evento de ratón adecuado. En segundo lugar, dibujará una

linea a medida que arrastra el ratón dentro del componente. El método que dibuja la linea no tiene memoria, de modo que no recordará las líneas dibujadas anteriormente.

Listado 14.13. Demostracion de MouseListener y MouseMotionListener

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class MouseTest extends ExitableJFrame {
    int startX, startY, endX, endY;
    Color color = Color.BLACK;
    public MouseTest() {
        super("Mouse Listener");

        final JPopupMenu pop = new JPopupMenu();
        pop.add(new JMenuItem("Cut"));
        pop.add(new JMenuItem("Copy"));
        pop.add(new JMenuItem("Paste"));
        pop.addSeparator();
        pop.add(new JMenuItem("Select All"));
        pop.setInvoker(this);

        MouseListener popup = new MouseListener() {
              public void mouseClicked(MouseEvent e) {
              }
              public void mouseEntered(MouseEvent e) {
              }
              public void mouseExited(MouseEvent e) {

              public void mousePressed(MouseEvent e) {
                  if (e.isPopupTrigger()) {
                    showPopup(e);


              public void mouseReleased(MouseEvent e) {
                  if (e.isPopupTrigger()) {
                    showPopup(e);

              }
              private void showPopup(MouseEvent e) {
                  pop.show(e.getComponent(), e.getX(), e.getY());
              }
        };
        addMouseListener(popup);

        MouseListener drawing1 = new MouseListener() {
              public void mouseClicked(MouseEvent e) {
              }
              public void mouseEntered(MouseEvent e) {
              }
              public void mouseExited(MouseEvent e) {
              }
              public void mousePressed(MouseEvent e) {
```

```
            color = Color.RED;
            startX = endX = e.getX();
            startY = endY = e.getY();
            repaint();
        }
        public void mouseReleased(MouseEvent e) {
            color = Color.BLACK;
            repaint();
        }
};
addMouseListener(drawing1);

MouseMotionListener drawing2 = new MouseMotionListener() {
        public void mouseDragged(MouseEvent e) {
            endX = e.getX();
            endY = e.getY();
            repaint();
        }
        public void mouseMoved(MouseEvent e) {

};
addMouseMotionListener(drawing2);
        }
        public void paint(Graphics g) {
            super.paint(g);
            g.setColor(color);
            g.drawLine(startX, startY, endX, endY);

        public static void main(String args[]) {
            JFrame frame = new MouseTest();
            frame.setSize(300, 300);
            frame.show();
        }
```

Nota

En las implementaciones de la interfaz oyente del ratón, la mayoría de la implementación es stub. En lugar de crear muchas clases que proporcionen metodos que no hacen nada, varios de los oyentes de eventos tienen clases adaptadoras que proporcionan los stub. Este concepto se estudiara en la última sección "Adaptadores".

La figura 14.6 muestra el programa del listado 14.13 en funcionamiento, con una linea dibujada y el menú desplegable visible.

Eventos de rueda de ratón

Otro evento de ratón disponible para Component y todas sus subclases es `MouseWheelEvent`. Una novedad de Java 1.4, este evento agrega soporte para escuchar si el usuario está moviendo la rueda del ratón.

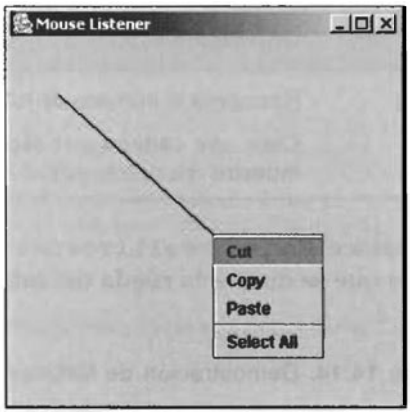


Figura **14.6.** El prograrna de prueba MouseListener

Nota

En la mayoría de los casos, no necesitaremos usar este par evento-oyente, aunque esta disponible. Componentes utiles como `JScrollPane` ahora tiene soporte integrado para responder al desplazamiento con la rueda del ratón.

Puede escuchar el evento registrando un `MouseWheelListener`. La interfaz consiste en un sólo método:

```
public interface MouseWheelListener extends EventListener {
    public void mouseWheelMoved(MouseWheelEvent e)
}
```

Cómo subclase de `MouseEvent`, **`MouseWheelEvent`** ya tiene un buen bagaje. Además, la tabla 14.14 enumera más información disponible para este evento.

Tabla 14.14. Cornposicion de la clase MouseWheelEvent

Variables/Métodos	Descripción
WHEEL—BLOCK—SCROLL	Tipo de desplazamiento
WHEEL—UNIT—SCROLL	
`MouseWheelEvent()`	Crea un objeto `MouseWheelEvent`
`getScrollAmount()`	Recupera la cantidad de desplazamiento; solo es valido cuando el tipo es "units"
`getScrollType()`	Recupera el tipo de desplazamiento
`getUnitsToScroll()`	Recupera el numero de rotaciones de rueda rnultiplicado por la cantidad de desplazamiento

Variables/Métodos	Descripción
`getWheelRotation()`	Recupera el numero de rotaciones de rueda
`paramString()`	Crea una cadena que representa el evento para que la muestre `toString()`

El listado 14.14 muestra el `MouseWheelListener` cambiando el color de fondo de un marco. A medida que se mueve la rueda del ratón, los colores rotan por una matriz predefinida.

Listado 14.14. Demostracion de MouseWheelListener

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class MouseWheelTest extends ExitableJFrame {
    private static final Color colors[] = {
        Color.BLACK,
        Color.BLUE,
        Color.CYAN,
        Color.DARK_GRAY,
        Color.GRAY,
        Color.GREEN,
        Color.LIGHT_GRAY,
        Color.MAGENTA,
        Color.ORANGE,
        Color.PINK,
        Color.RED,
        Color.WHITE,
        Color.YELLOW
    };

    public MouseWheelTest() {
        super("Mouse Wheel Listener");
        final Container contentPane = getContentPane();

        MouseWheelListener listener = new MouseWheelListener() {
            int colorCounter;

            private static final int UP = 1;
            private static final int DOWN = 2;

            public void mouseWheelMoved(MouseWheelEvent e) {
                int count = e.getWheelRotation();
                int direction = (Math.abs(count) > 0) ? UP : DOWN;
                changeBackground(direction);
            }
            private void changeBackground(int direction) {
                contentPane.setBackground(colors[colorCounter]);
                if (direction == UP) {
```

```
                colorCounter++;
            } else {
                --colorCounter;


            if (colorCounter == colors.length) {
                colorCounter = 0;
            } else if (colorCounter < 0) {
                colorCounter = colors.length-1;
            }
        }
    };
    contentPane.addMouseWheelListener(listener);
}

public static void main(String args[]) {
    JFrarne frame = new MouseWheelTest();
    frame.setSize(300, 300);
    frame.show();
}
}
```

Eventos de menú desplegable

El siguiente evento es el evento de menu desplegable. Sorprendentemente, estos eventos no son especificos del control `JPopupMenu`. Además de asociar los `PopupMenuListener` relacionados con los menus desplegables, podemos asociar el oyente con un `JComboBox`. Esta ultima opción le permite retrasar la carga de opciones en un `JComboBox` hasta que el usuario acuda a verlas. Si la operación es muy exigente, esta inicialización "perezosa" puede ser beneficiosa.

La interfaz `PopupMenuListener` consta de tres métodos. El `PopupMenuEvent` es muy sencillo, como `MenuEvent` y sólo tiene un constructor:

```
public interface PopupMenuListener extends EventListener {
    public void popupMenuCanceled(PopupMenuEvent e)
    public void popupMenuWillBecomeInvisible(PopupMenuEvent e)
    public void popupMenuWillBecomeVisible(PopupMenuEvent e)
}
```

El listado 14.15 muestra `PopupMenuListener` con un `JComboBox` y no inicializa las opciones hasta mds tarde.

Listado 14.15. Dernostracion de PopupMenuListener

```
import java.awt.*;
import javax.swing.*;
import javax.swing.event.*;

public class PopupTest {
    public static void main(String args[]) {
        JFrame frame = new ExitableJFrame("Popup Menu Listener");
        Container contentPane = frame.getContentPane();
```

```
        final String flavors[] = {"Chocolate", "Strawberry", "Vanilla",
           "Honey", "Chocolate Chip", "Watermelon", "Rasberry", "Banana"};
        PopupMenuListener iistener = new PopupMenuListener() {
           boolean initialized = false;
           public void popupMenuCanceled(PopupMenuEvent e) {

           public void popupMenuWillBecomeInvisible(PopupMenuEvent e) {
           }
           public void popupMenuWillBecomeVisible(PopupMenuEvent e) {
               if (!initialized) {
                  JComboBox combo = (JComboBox)e.getSource();
                  ComboBoxModel model = new DefaultComboBoxModel(flavors);
                  combo.setModel(model);
                  initialized = true;
               }
           }
        };

        JComboBox jc = new JComboBox();
        jc.addPopupMenuListener(listener);
        jc.setMaximumRowCount(4);
        jc.setEditable(true);
        contentPane.add(jc, BorderLayout.NORTH);
        frame.pack();
        frame.show();

}
```

Eventos de ventana

Naturalmente, los eventos de ventana son específicos de `java.awt.Window` y todas sus subclases, incluidas `JWindow`, Frame, `JFrame`, Dialog, `JDialog` y `FileDialog`. Los eventos de ventana se envían cuando se produce algún tipo de cambio en una ventana. Puede ser una tarea como, por ejemplo, abrir, centrar o minimizar la ventana.

Hay tres oyentes asociados a los eventos de ventana: `WindowListener`, `WindowFocusListener` y `WindowStateListener`, cada uno proporciona un conjunto de tipos de eventos a los que escuchar:

```
public interface WindowListener extends EventListener {
    public void windowActivated(WindowEvent w)
    public void windowClosed(WindowEvent w)
    public void windowClosing(WindowEvent w)
    public void windowDeactivated(WindowEvent w)
    public void windowDeiconified(WindowEvent w)
    public void windowIconified(WindowEvent w)
    public void windowOpened(WindowEvent w)
}
public interface WindowFocusListener extends EventListener {
    public void windowGainedFocus(WindowEvent w)
    public void windowLostFocus(WindowEvent w)
}
```

```
public interface WindowStateListener extends EventListener {
    public void windowStateChanged(WindowEvent w)
}
```

`WindowListener` ha estado en Java desde el principio. `WindowFocusListener` y `WindowStateListener` son novedades de Java 1.4. Los tres tratan con el mismo `WindowEvent`, definido en la tabla 14.15.

Tabla 14.15. Cornposicion de la clase WindowEvent

Variable/Métodos	Descripcidn
WINDOW_FIRST	Tipo de evento
WINDOW—LAST	
WINDOW_ACTIVATED	
WINDOW_CLOSED	
WINDOW_CLOSING	
WINDOW_DEACTIVATED	
WINDOW_DEICONIFIED	
WINDOW_GAINED_FOCUS	
WINDOW_ICONIFIED	
WINDOW_LOST—FOCUS	
WINDOW_OPENED	
WINDOW_STATE_CHANGED	
`WindowEvent()`	Crea un objeto `WindowEvent`
`getNewState()`	Recupera un nuevo estado deventana para cambios de estado
`getOldState()`	Recupera el antiguo estado de la ventana para los cambios de estado
`getOppositeWindow()`	Recupera la ventana secundaria involucrada en el evento
`getWindow()`	Recupera la fuente de evento como ventana
`paramString()`	Crea una cadena que representa el evento para que lo muestre `toString()`

Ya ha visto un uso indirecto de `WindowListener`. En la clase `ExitableJFrame` definida en el capítulo 10, la operacion por defecto para cerrar el marco se establecia en EXIT_ON_CLOSE. Podemos proporcionar el comportamiento EXIT_ON_CLOSE

adjuntando un `WindowListener` y cerrando en `windowClosing()`. El listado **14.16** es una clase que hace precisamente eso.

Listado 14.16. Dernostracion de WindowListener

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class WindowTest {
    public static void main(String args[]) {
        JFrame frame = new JFrame("Window Listener");
        WindowListener listener = new WindowListener() {
            public void windowActivated(WindowEvent w) {
            }
            public void windowClosed(WindowEvent w) {

            public void windowClosing(WindowEvent w) {
                System.exit(0);
            }
            public void windowDeactivated(WindowEvent w) {
            }
            public void windowDeiconified(WindowEvent w) {
            }
            public void windowIconified(WindowEvent w) {
            }
            public void windowOpened(WindowEvent w) {

        };
        frame.addWindowListener(listener);
        frame.setSize(300, 300);
        frame.show();
    }
}
```

Nota

En el listado 14.16, si el oyente de ventana no estuviera adjuntado al marco, el marco se ocultaria pero la aplicacion nunca terminaria.

Adaptadores

En esta sección veremos otro ejemplo de anexar un `WindowListener` a un marco. En el listado **14.16,** habrá observado algunos métodos stub para el `WindowListener`. Esto nos lleva a lo que se conoce como clases adaptadoras. En Java, las clases adaptadoras proporcionan las implementaciones stub vacías para las interfaces de oyente. Asi, en lugar de implementar una interfaz y proporcionar

stub para métodos en los que no necesitamos hacer nada, hacemos una subclase de la clase adaptadora y anulamos los métodos que no necesitamos definir. El listado 14.17 muestra el uso de este adaptador para el anterior listado 14.16. Observe que en esta version el código es mucho más limpio.

Listado 14.17. Demostracion de WindowListener con un WindowAdapter

```
imprrt java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class WindowAdapterTest {
    public static void main(String args[]) {
        JFrame frame = new JFrame("Window Listener");
        WindowListener listener = new WindowAdapter() {
            public void windowClosing(WindowEvent w) {
                System.exit(0);

        };
        frame.addWindowListener(listener);
        frame.setSize(300, 300);
        frame.show();
    }
}
```

No todos los oyentes de eventos de los paquetes `java.awt.event` y `javax.swing.event` tienen adaptadores. Aquí tiene una lista de las interfaces con adaptadores:

`ComponentAdapter`	`KeyAdapter`
`ContainerAdapter`	`MouseAdapter`
`FocusAdapter`	`MouseMotionAdapter`
`HierarchyBoundsAdapter`	`MouseInputAdapter` (combina `MouseAdapter` y `MouseMotionAdapter`)
`InternalFrameAdapter`	`WindowAdapter`

Noh

Ahora que hemos visto los adaptadores, intente rescribir el ejemplo `MouseTest` del listado 14.13 para usar adaptadores.

Resumen

En este capitulo hemos explicado las bases del control de eventos con componentes Swing. Ahora deberia comprender bastante bien los eventos primarios dis-

ponibles y dónde adjuntar un oyente cuando necesitemos saber cuando ocurre un evento.

Si no está satisfecho con respecto a los eventos que no se han estudiado en este capitulo, aqui tiene un breve resumen de cuatro que podrian interesarle:

- **PaintEvent:** Para este evento, observe que no hay un `PaintListener` correspondiente. Se anula `paint()` o `paintcomponent()` para encontrar qué componente necesita ser pintado.

  InputMethodEvent: El paquete `java.awt.im` proporciona un marco de mktodos de introducción para introducir texto sin emplear el teclado. Esto suele ser necesario en lenguajes como el chino, coreano y japonés, que usan ideogramas para los caracteres que no tienen teclas que los representen en el teclado.

- **TextEvent:** Este evento funciona con componentes de texto AWT. Su uso no conlleva componentes de texto Swing.

- **UndoableEditEvent:** Swing proporciona un marco completo para operaciones de deshacer/rehacer en el paquete `javax.swing.undo`. Con la ayuda de un `UndoManager`, las modificaciones realizadas a los componentes de texto se pueden deshacer y rehacer. `UndoManager` es el `UndoableEditListener`; no se define uno propio, aunque podemos hacerlo.

11

Capacidades de Swing avanzadas

En los capitulos anteriores, se han mostrado los componentes más simples orientados a formularios y cómo responder a las acciones del usuario con el manejador de eventos de Java. En este capitulo se detallarán mas capacidades para manejar objetos, introducidas por el conjunto de componentes de Swing. También se analizara la funcion de algunos cuadros de diálogo comunes utilizados para mostrar mensajes o para introducir datos por parte del usuario. Posteriormente se repasarán algunos de los componentes de Swing vistos hasta aqui y se mostrarán distintas maneras de utilizarlos. También se podrán ver algunos controles nuevos, describiendo todas sus posibilidades, relacionadas con varias clases interrelacionadas para cada control.

Al examinar estos nuevos componentes, se analizará un concepto llamado Controlador / Vista / Modelo (o arquitectura MVC), que fue introducido por primera vez en los años setenta por otro lenguaje de programación orientado a objetos, el Smalltalk. Veremos cómo utilizar componentes Swing que permiten cambiar la representación visual de componentes sin cambiar la manera de utilizarlos.

Manejador de eventos Swing avanzado

Además de utilizar los mecanismos manejadores de JDK 1.1 para trabajar con eventos, existen varias opciones incorporadas en Swing que incrementan mucho las capacidades existentes, a la vez que ofrecen nuevas alternativas. Esta sección describe las opciones existentes, que incluyen:

- Una clase para facilitar el manejo de listas de listener personalizadas mediante la simplificación a la hora de crear componentes, proporcionando únicamente el manejador.
- Un proceso especial de manejo para eventos de teclado.

Más adelante se explican también algunas consideraciones especiales para utilizar componentes Swing en aplicaciones multitareas, asi como el subsistema de control de focos.

Administración de listas de Listener

Al crear componentes nuevos, será necesario suministrar algún soporte para administrar listas de eventos dentro de éstos. Por ejemplo, supongamos que se quiere definir un componente como un calendario de modo que genere un `ActionEvent` cuando el usuario haga doble clic en la fecha de hoy. Se puede hacer de un modo muy fácil con ayuda de la clase `AWTEventMulticaster`, la cual se utiliza de manera interna para aiiadir y eliminar listener, tal y como hacen otros componentes automáticamente. El siguiente paso consiste en facilitar un método que reinicie la lista, cuándo ocurra un evento especifico. Los siguientes códigos muestran este comportamiento:

```
ActionListener actionListener = null;
// Añade un listener a la lista
public void addActionListener(ActionListener l) {
  actionListener = AWTEventMulticaster.add(actionListener, l);
}
// Elimina un listener de la lista
public void removeActionListener(ActionListener l) {
  actionListener = AWTEventMulticaster.remove(actionListener, l);
}
// Propaga un evento a todos los listener
public void fireActionPerformed(ActionEvent e) {
  if (actionListener != null) {
    actionListener.actionPerformed(e);
```

Nota

El nombre de algunos de estos metodos se describira con mas detalle cuando se vean los JavaBean.

La clase `AWTEventMulticaster` es muy útil cuando se maneja una lista de eventos AWT especificos. Sin embargo, si se crea una lista de eventos propia, no es posible utilizarla para administrarlos directamente, sino que debe personalizarse, utilizarla como subclase o emplear un modo alternativo para administrar la lista.

Por ello, Sun decidió aiiadir una clase que se pudiese utilizar con cualquier lista de listeners, siempre que éstos partan de la interfaz `EventListener`. La clase se encuentra en el paquete `javax.swing.event` y se denomina `EventListenerList`. El código para la operación anterior será:

```
EventListenerList listenerList = new EventListenerList();
public void addActionListener(ActionListener 1) {
  listenerList.add(ActionListener.class, 1);
}
public void removeActionListener(ActionListener 1) {
  listenerList.remove(ActionListener.class, 1);
}
protected void fireActionPerformed(ActionEvent e) {
  EventListener listeners[] = listenerList.getListeners(
    ActionListener.class);
  for (int i = 0, n=listeners.length; i<n; i++) {
    ((ActionListener)listeners[i]).actionPerformed(e);
  }
```

EL beneficio aiiadido de utilizar `EventListenerList` es que trabaja con cualquier tipo de listener, y no sólo con los de Swing o `AWT`. La unica diferencia entre `EventListenerList` y `AWTEventMulticaster` es el método `fireActionPerformed()`. Mientras que ambas clases notifican cada uno de los listener, con `EventListenerList` se tendrá que comprobar manualmente la lista en busca de tipos especificos de listerner, por dos motivos: para permitirle compartir la lista con multiples tipos de listener y porque la clase no sabe el método de listener que debe invocar. Debido a la generalidad funcional esta clase funciona con todos los eventos listener. Se requiere, además, que se ignoren los métodos `add()` y `remove()`, y que se incluya un parámetro adicional que defina los tipos concretos de clase de listener.

Abstraction Action Listening

La interfaz `Action` del paquete `javax.swing` ofrece un `ActionListener` especial con una implementación por defecto de la clase `AbstratAction`. Con `AbstractAction`, se puede crear el listener y asociarlo a muchos componentes. Si se quiere quitar un evento especifico, basta sólo con deshabilitar `AbstratAction` y éste se lo notificará al componente. A continuación se muestra la definición de la interfaz `Action`:

```
public interface Action extends ActionListener {
  public static final String ACCELERATOR-KEY;
  public static final String ACTION-COMMAND-KEY;
  public static final String DEFAULT;
  public static final String LONG-DESCRIPTION;
  public static final String MNEMONIC-KEY;
  public static final String NAME;
  public static final String SHORT-DESCRIPTION;
```

```
    public static final String SMALL_ICON;
    public void addPropertyChangeListener(PropertyChangeListener l);
    public Object getValue(String key);
    public boolean isEnabled();
    public void putValue(String key, Object value);
    public void removePropertyChangeListener(PropertyChangeListener l);
    public abstract void setEnabled(boolean b);
}
```

La tabla 15.1 contiene la definición de la clase `AbstractAction` que describe las funciones de la interfaz `Action`

Tabla **15.1.** Composición de la clase AbstractAction

Variables/Métodos	Descripción
changeSupport	Define la lista de los objetos registrados de PropertyChangeListener
enabled	Define el estado que se encuentra habilitado
`AbstractAction()`	Construye un objeto AbstractAction
`addPropertyChangeListener()`	Añade un listener a la acción
clone()	Crea una copia de la acción
firePropertyChange()	Notifica los listener registrados
`getKeys()`	Recupera las teclas de los atributos asociados con la accion
`getPropertyChangeListeners()`	Muestra los listener activados
`getValue()`	Recupera el valor de teclas concretas asociadas con la accion
`isEnabled()`	Comprueba la accion que se encuentra activada
`putValue()`	Almacena un par de valores de las teclas para la accion
`removePropertyChangeListener()`	Cambia el estado de la accion

Lo primero que se debe tener en cuenta es que aunque `AbstratAction` implementa `ActionListener`, por medio de `Action`, no existe un método `actionPerformed()`. Esto es debido a que la clase es abstracta. Cuando se crean objetos `AbstractAction`, se define qué acción realizar.

Además de la rutina perdida y de los métodos `getValue()` y `putValue()` de la clase `AbstractAction`, se deben notificar ocho constantes en la interfaz `Action`. Las constantes de `Action` son las teclas utilizadas por los métodos `getValue()` y `putValue()`. Una vez que se ha dado un valor para `Action`, cuando se asocie éste

con el JComponent ayropiado, le preguntará por los valores y se utilizarán para crear una etiqueta, un icono o incluso un cuadro de ayuda.

Cuando deseemos utilizar una Action debe pasarse la Action a un creador de componente, tal como JButton, o JMenuItem, para después utilizar el comando add() para añadir el componente creado a un contenedor JMenu, JPopMenu, o JToolBar. Asi, si se necesita la misma funcionalidad en dos de estos controles, se permitirá un mayor control y flexibilidad describiendo el ActionListener como una AbstratAction. Otra posibilidad consiste en crear primero el componente, establecer la acción mediante SetAction(), y añadirlo.

Para demostrar estas posibilidades, la figura 15.1 muestra el resultado de la misma acción en múltiples barras de herramientas. (Por lo general, se utilizará la misma acción en una barra de herramientas y en un mend.) En este ejemplo, el botón central conmuta el estado de la acción, que, a su vez, conmuta los componentes de las dos barras de herramientas asociadas con ella. El icono ovalado que aparece en los botones de las barras de herramientas cambia de lleno a vacío segun el estado del componente. El listado 15.1 contiene el código de este ejemplo.

Figura 15.1. Ejemplo de Action simplificado

Listado 15.1. Acción compartida entre componentes JButton

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class ActionTester {
  static class MyAction extends AbstractAction {
    public MyAction(String name, Icon icon) {
      super(name, icon);
    }
    public void actionPerformed(ActionEvent e) {
      System.out.println("Selected: " + getValue(Action.NAME));
    }
  }
  public static void main(String args[]) {
    JFrame f = new ExitableJFrame("Hello World");
    Container pane = f.getContentPane();
    Icon icon = new RedOvalIcon();
    final Action action = new MyAction("Hello", icon);
    // Añade una herramienta de ayuda
```

```
    action.putValue(Action.SHORT_DESCRIPTION, "World");
    JToolBar jt1 = new JToolBar();
    JButton b1 = new JButton(action);
    jt1.add(b1);
    pane.add(jt1, BorderLayout.NORTH);
    JToolBar jt2 = new JToolBar();
    JButton b2 = new JButton(action);
    jt2.add(b2);
    pane.add(jt2, BorderLayout.SOUTH);
    JButton jb = new JButton("Toggle Action");
    jb.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        action.setEnabled(!action.isEnabled());
      }
    });
    pane.add(jb, BorderLayout.CENTER);
    f.setSize(200, 200);
    f.show();
  }
  static class RedOvalIcon implements Icon {
    public void paintIcon(Component c, Graphics g, int x, int y) {
      g.setColor(Color.RED);
      if (c.isEnabled()) {
        g.fillOval(x, y, getIconWidth(), getIconHeight());
      } else {
        g.drawOval(x, y, getIconWidth(), getIconHeight());
      }
    }
    public int getIconWidth() {
      return 20;
    }
    public int getIconHeight() {
      return 30;
    }
  }
```

Hay dos modos de manejar las acciones de teclado para componentes. Se puede aiiadir un `KeyListener` y observar las teclas que son pulsadas hasta que se pulse la tecla especifica que interese.

A pesar de que este método funciona bien, los medios para comprobar el evento para una tecla especifica puede ser mucho más sencillo con la ayuda de la clase `KeyStroke` que se describe en la tabla 15.2.

Tabla **15.2.** Composición de la clase Keystroke

Variables/Métodos	Descripción
`getKeyStroke()`	Método preestablecido para crear pulsación de teclas
`getKeyStrokeForEvent()`	Método preestablecido para recuperar pulsacion de teclas desde un evento de teclas

La clase `Keystroke` no es demasiado funcional por sí misma. El método `getKeyStroke()`, sin embargo, dispone de seis funciones diferentes (una de ellas, bastante discutida) que nos permite crear una pulsación de tecla desde un carácter (la letra J, por ejemplo) o desde su combinación de teclas preferida ("**Control-Alt-**Supr") La clase `java.awt.event.KeyEvent` contiene más de 200 constantes que nos permiten crear pulsaciones mediante un `KeyCode`. La clase `java.awt.event.InputEvent`, por su parte, dispone de constantes para teclas especiales, tales como Control, Alt y Maylis.

El mecanismo `String` es el más util y flexible para crear objetos `Keystroke`. Bbsicamente, consiste en combinar uno o mas modificadores con una expresión opcional de tecla pulsada o escrita. La tecla escrita puede representar una "A" mayuscula, que se compone en realidad de cuatro pulsaciones (dos pulsaciones de la tecla Maylis. y dos de la tecla "a", incluyendo el acto de liberar la tecla). El siguiente listado contiene los simbolos que podemos utilizar en una combinación de pulsaciones:

```
<modifiers>* (<typedID> | <pressedReleasedID>)
modifiers := shift | control | ctrl | meta | alt | button1 | button2 |
button3
typedID := typed <typedKey>
typedKey := string of length 1 giving Unicode character
pressedReleasedID := (pressed | released) key
key := KeyEvent key code name, i.e. the name following "VK_"
```

Una vez creada la pulsación, se debe registrar la pulsación con el componente que generará la respuesta. Esto implica aiiadir la pulsación al `InputMap` del componente, para después asociarla con una Acción en su Mapa de acción (`ActionMap`), lo que se realiza mediante String, como se muestra en el siguiente ejemplo:

```
component.getInputMap().put(keystroke, stringKey);
component.getActionMap().put(stringKey, action);
```

Con ello, el componente respondera a la pulsación con la acción designada.

Los parámetros del método `getInputMap()` controlan si una pulsación se encuentra activa. Algunas de las opciones disponibles son `WHEN_FOCUSED` (por defect~)`WHEN_IN_FOCUSED_WINDOW`, y `WHEN ANCESTOR_OF_FOCUSED_COMPONENT`.

Como demostración, el programa de la–lista 15.2 asocia diversos listeners de acciones, para B, Control-C y **Mayús-D**, a diferentes botones dentro de un `JFrame`. La tecla B responde cuando el componente especifico tiene foco; las teclas **Control-**C responden cuando el foco aparece en cualquier lugar de la ventana que contiene el componente. Maylis-D reacciona cuando se libera la pulsación y sólo si el componente o el ancestro de contención contiene el foco de entrada. En otras palabras, la pulsación no respondera si es otro botón el que tiene el foco de entrada.

Listado 15.2. Respuesta a pulsaciones

```
import java.awt.*;
import java.awt.event.*;
```

```
import javax.swing.*;
public class KeyTester {
  static class MyActionListener extends AbstractAction {
    MyActionListener(String s) {
      super(s);
    }
    public void actionPerformed(ActionEvent e) {
      System.out.println(getValue(Action.NAME));
    }
  }
  public static void main(String args[]) {
    String actionKey = "theAction";
    JFrame f = new ExitableJFrame("Key Tester");
    JButton jbl = new JButton("<html><center>B<br>Focused/Typed");
    JButton jb2 = new JButton("<html><center>Ctrl-C<br>Window/Pressed");
    JButton jb3 = new JButton(
      "<html><center>Shift-D<br>Ancestor/Released");
    Container pane = f.getContentPane();
    pane.add(jbl, BorderLayout.NORTH);
    pane.add(jb2, BorderLayout.CENTER);
    pane.add(jb3, BorderLayout.SOUTH);
    Keystroke stroke = KeyStroke.getKeyStroke("typed B");
    Action action = new MyActionListener("Action Happened");
    // Cambia a JComponent.WHEN_FOCUSED map
    InputMap inputMap = jbl.getInputMap();
    inputMap.put(stroke, actionKey);
    ArtionMap actionMap = jbl.getActionMap();
    actionMap.put(actionKey, action);

    stroke = Keystroke.getKeyStroke("ctrl C");
    action = new MyActionListener("Action Didn't Happen");
    inputMap = jb2.getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW);
    inputMap.put(stroke, actionKey);
    actionMap = jb2.getActionMap();
    actionMap.put(actionKey, action);
    stroke = KeyStroke.getKeyStroke("shift released D");
    action = new MyActionListener("What Happened?");
    inputMap =
jb3.getInputMap(JComponent.WHEN_ANCESTOR_OF_FOCUSED_COMPONENT);
    inputMap.put(stroke, actionKey);
    actionMap = jb3.getActionMap();
    actionMap.put(actionKey, action);
    f.setSize(200, 200);
    f.show();
  }
```

Interaccion con el subsistema de focos

Antes de nada, es importante definir qué es un foco; Un foco define el componente que estd recibiendo los eventos de pulsaciones, es decir, el componente con el que el usuario estd trabajando. Antes de la comercialización de la versión 1.4 del SDK, el subsistema de control de focos era bastante complicado, ya que tenia algu-

nos fallos y además dependia de algunas plataformas. La versión **1.4** incorpora un nuevo subsistema de focos en AWT, y a su vez, en Swing. Aunque no tenga experiencia previa en Java, no tiene que preocuparse por que se mezclen el subsistema antiguo y el nuevo, o preguntarse cuáles de los antiguos fallos se han subsanado, ya que el subsistema es completamente nuevo.

En el capitulo **14** se han descrito los métodos `FocusEvent` y `WindowEvent` para obtener focos. En esta sección, nos centraremos en tres de los aspectos mas avanzados del nuevo subsistema:

- La clase KeyboardFocusManager.
- El desplazamiento del foco.
- El desplazamiento programable.

La clase KeyboardFocusManager

La clase KeyboardFocusManager es la base fundamental de la gestion de focos. Por lo general, existe un sólo gestor global para las aplicaciones, pero con applets es diferente, ya que los buscadores separan los contextos applet segiin el lugar de dónde provienen, es decir, segiin de dónde provenga su código. Aunque cada cargador de clase (o contexto) puede disponer de su propio gestor de focos, nunca puede haber más de una ventana o componente activo.

Para obtener el gestor de focos que estamos utilizando basta con preguntar con el método `getCurrentKeyboardFocusManager()` de la siguiente clase:

```
keyboardFocusManager manager =
KeyboardFocusManager.getCurrentKeyboardFocusManager()
```

Una vez que disponemos del gestor, podemos invocar cualquiera de los métodos que aparecen en la tabla **15.3.**

> **Nota**
>
> **Aunque el constructor** `java.awt.KeyboardFocusManager` **es público, algunos de los metodos son abstractos, por lo que no podemos crear un nuevo gestor de focos sin utilizar subclases.**

Tabla 15.3. Composición de la clase KeyboardFocusManager

Variables/Métodos	Descripción
BACKWARD – TRAVERSAL – KEYS	**Identificador de los desplazamientos**
DOWN – CYCLE – TRAVERSAL – KEYS	

Variables/Métodos	Descripción
FORWARD – TRAVERSAL – KEYS	
UP – CYCLE – TRAVERSAL – KEYS	
`KeyboardFocusManager()`	Construye un objeto KeyboardFocusManager
`addKeyEventDispatcher()`	Registra el encargado de enviar las cadenas
`addKeyEventPostProcessor()`	Registra el procesador en la cadena posterior al procesamiento
`addPropertyChangeListener()`	Registra el listener apropiado
`addVetoableChangeListener()`	
`clearGlobalFocusOwner()`	Elimina el propietario del foco global
*`dequeueKeyEvents()`**	Hace que el gestor cancele los envios atrasados de eventos de pulsacion
*`discardKeyEvents()`**	
`dispatchEvent()`*	Despacha eventos **AWT** al gestor de focos para el encargado de enviar eventos **AWT**
`dispatchKeyEvent()`*	
`downFocusCycle()`	Mueve el ciclo del foco hacia arriba o hacia abajo
`upFocusCycle()`	
*`enqueueKeyEvents()`**	Hace que el gestor comience a enviar los eventos de pulsacion atrasados
`firePropertyChange()`	Notifica los cambios a los listener registrados
`fireVetoableChange()`	
`focusNextComponent()`	Desplaza el ciclo de focos al componente anterior o posterior
`focusPreviousComponent()`	
`getActiveWindow()`	Devuelve la ventana activa
`getGlobalActiveWindow()`	
`getCurrentFocusCycleRoot()`	Devuelve la raíz del ciclo de focos actual
`getGlobalCurrentFocusCycleRoot()`	
`getCurrentKeyboardFocusManager()`	Devuelve el gestor actual

Variables/Métodos	Descripción
getDefaultFocusTraversalKeys()	Devuelve los rnovirnientos de teclas del foco
getDefaultFocusTraversalPolicy()	Devuelve las reglas de rnovirniento del foco
getFocusedWindow()	Devuelve la ventana de foco actual
getGlobalFocusedWindow()	
getFocusedOwner()	Devuelve el cornponente de foco actual
getGlobalFocusedOwner()	
getKeyEventDispatcher()	Devuelve los distribuidores actuales
getKeyEventPostProcessors()	Devuelve los post-procesadores actuales
getPermanentFocusOwner()	Devuelve el propietario de foco actual
getGlobalPermanentFocusOwner()	
getPropertyChangeListeners()	Devuelve el grupo de listener actual
getVetoableChangeListeners()	
postProcessKeyEvent()*	Gestiona los accesos directos del menú
processKeyEvent()*	Procesa el evento si la tecla en cuestión es una de las desplazadas del componente
redispatchEvent()	Solicita que se gestione el thread AWT
removeKeyEventDispatcher()	Elimina el registro de envio de las cadenas a enviar
removeKeyEventPostProcessor()	Elirnina el registro de procesamiento de la cadena de eventos de post-procesamiento
removePropertyChangeListener()	Elirnina el registro del listener apropiado
removeVetoableChangeListener()	
setCurrentKeyboardFocusManager()	Cambia el gestor actual
setDefaultFocusTraversalKeys()	Cambia el grupo de teclas utilizadas en el desplazamiento
setDefaultFocusTraversalPolicy()	Cambia las reglas de desplazamiento del foco
setGlobalActiveWindow()	Cambia la ventana activa
setGlobalCurrentFocusCycleRoot()	Cambia la raiz del ciclo del foco
setGlobalFocusedWindow()	Cambia la ventana del foco

Variables/Métodos	Descripción
`setGlobalFocusOwner()`	Cambia los componentes del foco
`setGlobalPermanentFocusOwner()`	

Los melodos que tienen un aslerisco (') son los que normalmente invoca la larea AWT o corresponden al flujo de dalos de una llamada de tarea AWT, y no de un desarrollador.

Es necesario aclarar algunos detalles de los términos que aparecen en la tabla anterior. Por ejemplo, a la diferencia entre una ventana activa y una ventana de foco es que la primera puede corresponder a la ventana de foco o al marco principal de la ventana de foco. La diferencia entre global y no global estriba en que los métodos no globales devuelven "nulo" si otro contexto de tarea tiene un foco, mientras que los métodos globales devuelven el componente apropiado del otro contexto de tarea. El ciclo de foco se explica en la siguiente sección.

Incluyendo los métodos cancelados, la clase `KeyboardFocusManager` cuenta en total con 58 métodos diferentes, lo cuál es una gran ventaja. En las siguientes secciones, veremos la manera de utilizarlos.

Desplazamiento del foco

El desplazamiento de foco constituye el concepto más importante de la clase `KeyboardFocusManager`, y se basa en la manera en que el foco de entrada se desplaza hacia adelante o hacia atrás entre los componentes de un contenedor. El orden de movimiento se denomina ciclo del foco.

Si todos los componentes se encuentran en el mismo contenedor no tenemos que preocuparnos, ya que el `FocusTransversalPolicy` controla el orden de movimiento, como se muestra en la figura 15.2. Las teclas que se utilizan en el desplazamiento del foco dependen de la implementación, aunque suelen ser la tecla de Tabulador para avanzar, y las teclas May&.-Tab. para retroceder, sin que haya valores predeterminados para los movimientos hacia arriba o hacia abajo.

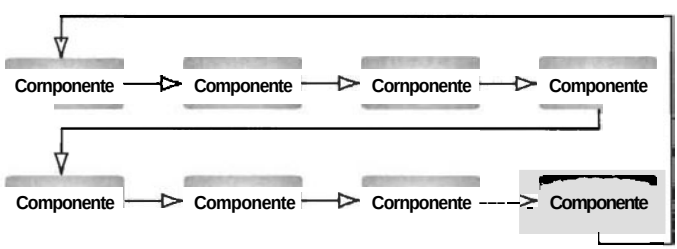


Figura 15.2. Orden predeterminado de los focos

Podemos alterar el comportamiento del ciclo predeterminado de los focos colocando algunos de los componentes en un Contenedor y ajustando el valor de la propiedad `focusCycleRoot` de dicho contenedor a `true`. Si el valor es verdadero, el foco de entrada permanece dentro del contendor al pulsar las teclas de movi-

miento del foco. Este nuevo comportamiento se ilustra en la figura 15.3, en la que hemos situado algunos de los componentes anteriores dentro de un contenedor `JPanel`.

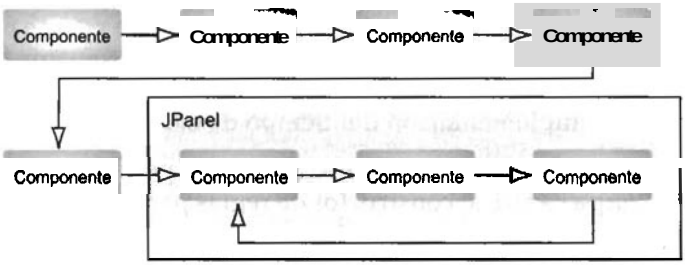


Figura 15.3. Orden restrictivo de los focos

El método `FocusTraversalPolicy` se encarga de determinar los componentes anteriores y posteriores, aunque siempre podemos crear nuestras propias reglas en el caso de que necesitemos modificar o restringir el orden predeterminado. Por lo general, esto no ocurre ya que hay disponibles cuatro implementaciones que nos permiten personalizar el orden. La figura 15.4 representa un diagrama de jerarquias que refleja dichas implementaciones, que se explican con más detalle en los párrafos siguientes. Para personalizar las reglas predeterminadas, sólo tenemos que utilizar el método `set-Focusable(boolean)`, que se encargará de conmutar la capacidad de foco de un componente para asi aiiadirlo o eliminarlo de un ciclo.

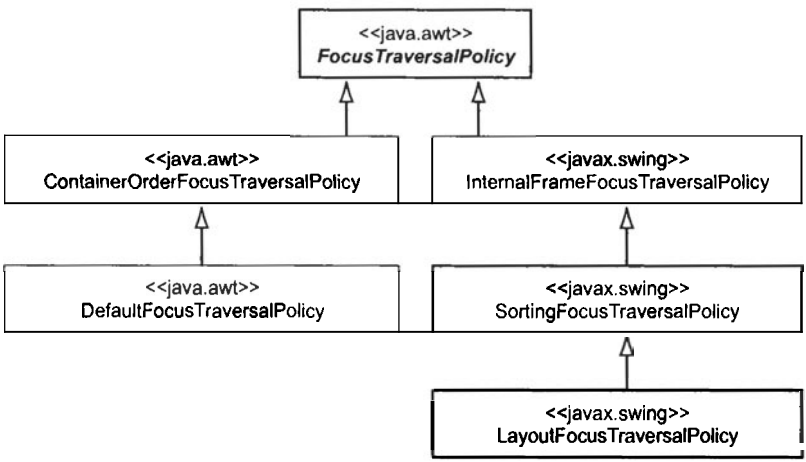


Figura 15.4. Diagrama jerarquico de la clase FocusTraversalPolicy

- El ciclo `ContainerOrderFocusTraversalPolicy`: Sigue el mismo orden en el que los componentes se aiiadieron al contenedor. Los componentes pueden estar en el ciclo siempre que sean visibles, activos, se muestren y

tengan capacidad de foco, cuatro de las distintas propiedades de los componentes.

- El `DefaultFocusTraversalPolicy`: Contiene las reglas de movimiento predeterminadas para los programas AWT. Extiende la comprobación de idoneidad realizada por el método `ContainerOrderFocusTraversalPolicy`, para comprobar con el sistema operativo que el componente no haya establecido explicitamente la capacidad de foco. Dicha capacidad depende de la implementación del tiempo de ejecución de Java.
- El `SortingFocusTraversalPolicy`: Nos permite proporcionar un argumento `Comparator` al constructor de reglas para asi definir el orden del ciclo de foco.
- El `LayoutFocusTraversalPolicy`: Define las reglas predeterminadas de los programas Swing. Tiene en cuenta el tamaño del componente y se desplaza de arriba a abajo y de izquierda a derecha, asumiendo una orientación por defecto utilizada en EE.UU. (`ComponentOrientation`), que está basada en las posiciones de los componentes segun el gestor de distribución.

Desplazamiento programable

Además de diseñar una ventana según las reglas de desplazamiento, hay ocasiones en las que necesitamos controlar de una manera programada la elección del componente para obtener el foco de entrada. Esto se puede realizar de diferentes maneras, pero la más sencilla es en la que el componente mismo se encarga de solicitar la obtención del foco de entrada, mediante el método `requestFocusInWindow()`. No es recomendable emplear el método `requestFocus()`, ya que no funciona bien entre plataformas cuando la ventana que contiene el componente no cuenta además con el foco de entrada.

Para desplazar el ciclo de foco hasta el componente siguiente o el anterior, sólo tenemos que incluir los siguientes métodos en el `KeyboardFocusManager`:

Método	Instrucciones
`focusNextComponent()` `focusNextComponent(Component)`	Se desplaza hasta el siguiente componente o hasta el situado tras el componente especificado
`focusPreviousComponent()` `focusPreviousComponent(Component)`	Se desplaza hasta el componente anterior o hasta el situado delante del componente especificado
`upFocusCycle()` `upFocusCycle(Component)`	Se desplaza un nivel hacia arriba en el ciclo de foco
`downFocusCycle()` `downFocusCycle(Container)`	Se desplaza un nivel hacia abajo en el ciclo de foco

Advertencia

Aunque clase `Component` contiene duplicados de todos estos metodos para solicitar de manera directa el cambio de foco, es conveniente utilizar el gestor de foco.

Truco

Si queremos conocer de antemano los cambios de foco que van a rechazarse, solo tenemos que añadir el método `VetoableChangeListener` al `KeyboardFocusManager`.

Trabajando con Thread múltiples

El acceso a las propiedades de los componentes Swing deberia estar, por diseño, limitado al thread que despacha los eventos. Esto significa que mientras que se puede preguntar al `JTextField` por su contenido, en otro thread con `getText()` el valor que se obtenga puede no ser el valor actual correcto. Normalmente esto no es un problema ya que el acceso a la mayoría de los componentes se hace desde este thread.

Por ejemplo, si se desea imprimir el contenido de un campo de texto cuando el usuario selecciona un botón, se está ya en el thread que despacha el evento para responder a la selección del botón.

¿Cuándo se convierte esto en un problema? Esto ocurre en cualquier ocasión en que un programa responda a eventos que no sean generados por AWT, o si se desea generar un thread aparte para hacer una operación mds larga y de este modo que la interfaz permanezca sensible. En ambos casos se requiere un trabajo extra para actuar correctamente cuando se necesite acceso a los componentes Swing desde otro thread.

¿Cómo resolver este problema? La clase `EventQueue` (además de la `SwingUtilities`) suministra dos métodos para ayudar a controlar este problema de accesibilidad: `invokeLater()` e `invokeAndWait()`. Ambos necesitan un objeto ejecutable, que se ejecutard en el thread. Todo lo que se necesita hacer es definir una clase que implemente `Runnable`, normalmente como una clase andnima interna, y dársela luego a uno de los dos métodos. El método `invokeLater` regresa inmediatamente permitiendo al thread actual continuar. Con `invokeAndWait()`, el thread original se bloquea hasta que el objeto `Runnable` finaliza su operación. Por tanto, si se desea tan sólo imprimir el contenido de un `JTextField`, se utilizard `invokeLater()`.

Sin embargo, si se necesitan procesar los resultados deberá llamarse a `invokeAndWait()`. Estos dos ejemplos se muestran a continuación.

```
final JTextField jt …;
// Proceso ejecutándose en un thread de no evento
EventQueue.invokeLater(new Runnable() {
  public void run() {
    System.out.println(jt.getText());

});
final String curValue[] = {""};
try {
  EventQueue.invokeAndWait(new Runnable() {
    public void run() {
      curValue[0] = jt.getText();

  });
} catch (InvocationTargetException e) {
} catch (InterruptedException e) {
}
processString(curValue[0]);
```

Nota

Los métodos `repaint()`, `revalidate()`, e `invalidate()` pueden invocarse desde cualquier thread ya que no tienen acceso directo a los componentes.

Cuadros de diálogo para usuarios

Swing ofrece un modo muy práctico de mostrar mensajes, diálogos pop-up y solicitar una entrada de usuario desde cuadros de diálogo modnles. Los tres principales componentes para este propósito son `JOptionPane`, `JColorChooser`, y `JFileChooser` para entradas generales o confirmaciones, colores y nombres de archivos, respectivamente.

Nota

Cuando un cuadro de dialogo modal se muestra dentro de un método, el método no continua hasta que el usuario cierra el cuadro.

Cuadros de diálogo para mensajes

`JOptionPane` es el principal componente para suministrar información a los usuarios o para solicitarles una entrada. La tabla 15.4 muestra todos los métodos y constantes de esta clase.

Tabla **15.4.** Composición de la clase JoptionPane

Variables/Métodos	Descripción
CANCEL – OPTION	Indica el boton seleccionado
CLOSED – OPTION	
NO – OPTION	
OK – OPTION	
YES – OPTION	
DEFAULT – OPTION	Tipos de boton que se muestran
OK – CANCEL – OPTION	
YES – NO – CANCEL – OPTION	
YES – NO – OPTION	
ERROR – MESSAGE	Tipos de icono que se muestran en el mensaje
INFORMATION – MESSAGE	
PLAIN – MESSAGE	
QUESTION – MESSAGE	
WARNING_MESSAGE	
ICON – PROPERTY	Nombre de las propiedades que sevan a utilizar para propiedades ligadas dentro de los componentes
INITIAL_SELECTION_VALUE_ PROPERTY	
INITIAL – VALUE – PROPERTY	
INPUT – VALUE – PROPERTY	
MESSAGE_PROPERTY	
MESSAGE – TYPE – PROPERTY	
OPTIO_TYPE_PROPERTY	
OPTIONS – PROPERTY	
SELECTION_VALUES_PROPERTY	
VALUES – PROPERTY	
WANTS – INPUT – PROPERTY	
icon	Variables para almacenar los diferentes estados de un componente
initialSelectionValue	

Variables/Métodos	Descripción
initialValue	
inputValue	
message	
messageType	
options	
optionType	
selectionValues	
value	
wantsInput	
UNITIALIZED_VALUE	Valor que indica que el usuario no ha seleccionado nada
`JOptionPane()`	Construye un objeto JoptionPane
`createDialog()`	Construye un objeto en el que incluir un componente
`createInternalFrame()`	
`getAccessibleContext()`	Devuelve el contexto accesible actual
`getDesktopPaneForComponent()`	Devuelve el panel de escritorio para un componente
`getFrameForComponent()`	Devuelve el marco actual para un componente
`getIcon()`	Devuelve el icono actual
`getInitialSelectionValue()`	Devuelve el valor inicial de la entrada
get`InitialValue()`	Devuelve el valor inicial
get`InputValue()`	Devuelve el valor introducido por el usuario
`getMaxCharactersPerLineCount()`	Devuelve la duración maxima del mensaje
`getMessage()`	Devuelve el mensaje
`getMessageType()`	Devuelve el tipo de mensaje
`getOptions()`	Devuelve las opciones para el usuario
`getOptionType()`	Devuelve el tipo de opciones visibles
`getRootFrame()`	Devuelve el marco raiz cuando no hay ningún otro disponible
`getSelectionValues()`	Devuelve losvalores seleccionados por el usuario
`getUI()`	Devuelve el componente UI

Variables/Métodos	Descripción
`getUIClassID()`	Devuelve el nombre de clase de UI
`getValue()`	Devuelve el valor seleccionado resultante del valor seleccionado del usuario
`getWantsInput()`	Devuelve la peticion al componente para aceptar la entrada
`paramString()`	Devuelve la representación del estado del componente
`selectInitialValue()`	Hace que el componente seleccione el valor inicial
`setIcon(Icon newIcon)`	Cambia el icono
`setInitialSelectionValue()`	Cambia el valor de entrada inicial
`setInitialValue()`	Cambia el componente de entrada inicial
`setInputValue()`	Cambia el valor de entrada
`setMessage()`	Cambia el mensaje
`setMessageType()`	Cambia el tipo de rnensaje
`setOptions()`	Cambia las opciones visibles
`setOptionType()`	Cambia el tipo de opción
`setRootFrame()`	Cambia el marco raiz
`setSelectionValues()`	Cambia las opciones de selección de entrada
`setUI()`	Cambia el componente UI
`setValue()`	Cambia el valor elegido
`setWantsInput()`	Cambia la peticion al cornponente de obtener la entrada
`showConfirmDialog()`	Muestra la confirrnacion emergente dentro de JDialog o de JinternalFrame
`showInternalConfirmDialog()`	
`showInputDialog()`	Muestra la entrada emergente dentro de JDialog o JinternalFrame
`showInternelInputDialog()`	
`showMessageDialog()`	Muestra mensajes emergente dentro de JDialog o JinternalFrame
`showInternalMessageDialog()`	
`showOptionDialog()`	Muestra las opciones ernergentes dentro de JDialog o JinternalFrame

Variables/Métodos	Descripción
`showInternalOptionDialog()`	
`updateUI()`	Actualiza el componente visible tras un cambio de aspecto y sensibilidad

Existen dos maneras de trabajar con `JOptionPane`. Se podría crear un `JOptionPane` utilizando diversos constructores y métodos publicos, aunque este modo es bastante pesado. El modo más sencillo que hay de trabajar con `JOptionPane` es usar diversos métodos factory, creando tipos especificos con `showXXXDialog`. Estos métodos permiten configurar todas las opciones necesarias y mostrar los dialogos de una manera sencilla, devolviendo la selección del usuario cuando es conveniente.

Nota

Los metodos Factory se utilizan para crear un objeto sin tener que utilizar el constructor de su clase. Además permiten ocultar detalles de implementación, especialmente en subclases, mediante interfaces bien definidas, como se vio en el capitulo 13.

Existen cuatro tipos de cuadros de diiilogo : mensaje, confirmacicin, entrada y opción. Cada uno ofrece ligeras diferencias en su comportamiento, pero todas siguen el mismo patrón general. Como muestra la figura 15.5, cada cuadro dispone de cuatro zonas, una para un icono, otra para un mensaje, otra para entrada de datos y otra para opciones. Los tipos de cuadro de diiilogo utilizados determinan qué se encuadra dentro de cada cuadrante. Se puede especificar si el cuadro de diiilogo creado es un diiilogo interno (`JInternalFrame`) o externo (`JDialog`). El tipo que se utilice dependerii del comportamiento que se desee.

Figura 15.5. Apariencia del cuadro de diálogo general

Cuadros de diálogo para mensajes

Los cuadros de diálogo para mensajes se usan para mostrar información de carácter general. Por defecto, incluyen los siguientes elementos:

- Un icono especifico para el tipo de mensaje (información, error, advertencia, o pregunta).
- Un drea para el mensaje.
- Un botón de "Aceptar" u "OK".

El método `showMessageDialog(component parent, Object msg, String title, int msgType)` es el método mds sencillo y el que se utiliza normalmente para crear diálogos de mensajes. Sus pardmetros actúan de la siguiente manera:

`parent`: Se usa para posicionar el diálogo. Si es nulo, el diálogo se centrard en la pantalla, si no el diálogo se centra según el valor de parent.

- `msg`: Es el mensaje a mostrar. Éste es un **`Object`**, no un **`String`**. Si el parimetro no es un **`String`**, se llamará a su método `toString()` para obtener uno de este tipo.
- **`Title`**: Es titulo que mostrará la batra de titulo del cuadro de diálogo
- `msgType`: Indica el tipo de mensaje. Se usa para indicar qué icono se debe mostrar. Los valores vdlidos son **`ERROR_MESSAGE`**, **`INFORMATION_MESSAGE`**, **`WARNING_MESSAGE`**, **`QUESTION_MESSAGE`** y **`PLAIN_MESSAGE`**. En cuadros de diálogo para mensajes, `QUESTION_MESSAGE` no es realmente una opción apropiada, ya que el usuario sólo puede seleccionar **Aceptar** y no entre Sí/ **No.**

Si omitimos los parametros **`title`** y **`msgtype`**, el cuadro de diálogo tendra el titulo "Mensaje" y un icono informativo. La figura 15.6 muestra la apariencia de un cuadro de diálogo para mensajes de información.

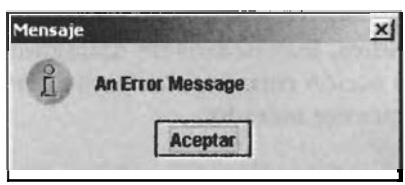


Figura **15.6.** Cuadro de dialogo para mensaje de inforrnacion

Los dos distintos segmentos de código que se muestran a continuación representan las dos maneras posibles de mostrar los cuadros de diálogo para mensajes. El primer modo es claramente el más sencillo para trabajar con didlogos, y, de hecho, los demds ejemplos de cuadros de diálogo de este capítulo se han creado utilizando esta técnica.

```
JOptionPane.showMessageDialog (f, "Show me the money", "Window Title",
JOptionPane.INFORMATION_MESSAGE);
```

El segundo mecanismo se emplea cuando necesitamos personalizar alguna parte del cuadro, tal como el tamaño de la ventana del mensaje. Ya que no podemos utilizar el método `setMaxCharactersPerLineCount()` u otro argumento opcional, es necesario crear subclases. Llegados a este punto, los métodos **`factory`** no son de gran ayuda ya que no nos permiten crear subclases especializadas.

```
JOptionPane pane = new JOptionPane ();
pane.setMessage ("Show me the money");
pane.setMessageType (JOptionPane.INFORMATION_MESSAGE);
JDialog dialog = pane.createDialog (f, "Window Title");
dialog.show();
```

La figura 15.7 muestra el mismo cuadro de diálogo pero con otros tipos de mensajes.

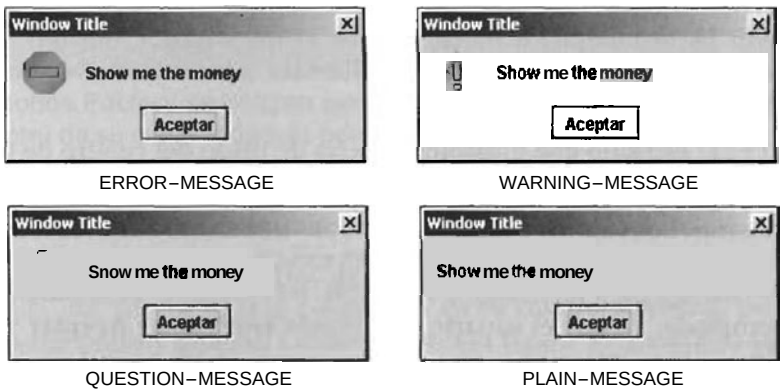


Figura 15.7. Otros cuadros de diálogo para mensajes

Cuadros de diálogo de confirmación

Como su nombre indica, los cuadros de diálogo de confirmación se usan para aceptar o rechazar una acción concreta. La forma principal de crear estos cuadros de diálogo es con el siguiente método:

```
showConfirmationDialog(Componentparent, Object msg, String title, int
optType)
```

El tipo `optType` determina cuáles de las siguientes opciones podemos seleccionar:

`YES-NO-CANCEL-OPTION`	Botones **Sí**, **No** y **Cancelar**.
`YES-NO-OPTION`	Botones **Sí** y **No**.
`OK-CANCEL-OPTION`	Botones **Aceptar** y **Cancelar.**

También se puede indicar el tipo de mensaje que se desea mediante el siguiente método:

```
showConfirmDialog(Component parent, Object msg, String Title, int optType,
int msgType)
```

En este método, los tipos son los mismos que se usan para el parámetro `msgType` de los cuadros de ditilogo para mensajes. Sin el partimetro `msgType`, el valor por defecto será QUESTION_MESSAGE. Como los cuadros de diálogo de confirmacidn requieren la respuesta del usuario, el método `showConfirmDialog()` devuelve el valor que el usuario ha seleccionado en uno de estos modos: YES_OPTION, NO OPTION, OK_OPTION, CANCEL_OPTION y CLOSED_OPTION. Todos, excepto CL0SED_OPTION se corresponden con sus respectivos botones, es decir, con **Si,** No, **Aceptar** y **Cancelar.** Con CLOSED_OPTION el usuario cierra la ventana sin seleccionar nada. Esta opcidn deberia tratarse del mismo modo que la operación Cancelar, o, en su defecto, que la opcidn No. La figura 15.8 muestra un ejemplo de cuadro de ditilogo de confirmación.

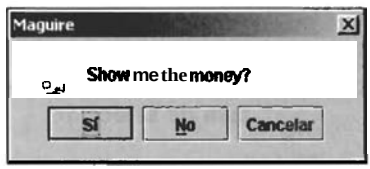


Figura 15.8. Cuadro de diálogo de confirrnacion

El código usado para crear la figura anterior y manejar las respuestas es el siguiente:

```
int returnValue = JOptionPane.CANCEL-OPTION;
while (returnValue != JOptionPane.YES_OPTION) {
  returnValue = JOptionPane.showConfirmDialog(f, "Show me the money?",
    "Maguire", JOptionPane.YES_NO_CANCEL_OPTION);
  if (returnValue == JOptionPane.YES_OPTION) {
    System.out.println("Congratulations for saying yes.");
  } else if (returnValue == JOptionPane.NO_OPTION) {
    System.out.println("How dare you say no?");
  } else { // Cerrado / Cancelar
    System.out.println("Get real!");
  }
}
```

Cuadros dc diálogo para la entrada de datos

Los cuadros de diálogo para la entrada de datos se usan cuando se pretende que el usuario introduzca unos datos ya sea mediante un campo de texto o con una lista de opciones (`JComboBox` o `JList`, dependiendo del numero de opciones). Para que los datos se introduzcan por medio de un cuadro de texto, el método más común es el siguiente:

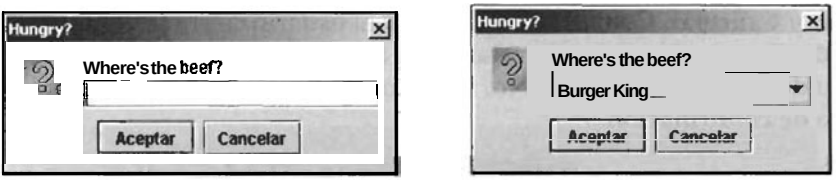
```
showInputDialog(Component parent, Object msg, String title, int msgType)
```

El `msgType` o tipo de mensaje de este método será probablemente `QUESTION_MESSAGE`. Si se quiere suministrar una lista de opciones para que el usuario pueda elegir entre una de ellas entonces se utilizará:

```
InputDialog(Component parent, Object msg, String title, int msgType, Icon
icon, Object options[], Object initialValue).
```

Para utilizar el icono por defecto de un tipo mensaje bastará con darle un valor nulo al parámetro **`icon`**.

Los dos ejemplos de la figura 15.9 contienen botones **Aceptar** y **Cancelar** y devuelven el valor introducido o seleccionado. Si el usuario selecciona **Cancelar** o cierra el cuadro de diálogo, el valor que devuelve es nulo.



Entrada en un cuadro de texto

Entrada segun una lista de selección

Figura 15.9. Cuadros de dialogo de entrada de datos mediante un campo de texto y una lista de seleccion

A continuación se muestra el codigo usado para generar y manejar los dos cuadros de la figura 15.9. Hacen falta unas 20 opciones para que el parámetro `JList` muestre el cuadro de diálogo con lista de selección.

```
String input = null;
while ((input == null)) {
input = JOptionPane.showInputDialog(f, "Where's the beef?", "
    "Hungry?", JOptionPane.QUESTION_MESSAGE);
  if ((input != null) && (input.length() == 0))
    input = null;

System.out.println("Entered: " + input);
// Segundo
input = null;
while(input == null) {
  String options[] = {"Burger King", "McDonalds", "Outback", "Wendy's"};
  input = (String)JOptionPane.showInputDialog(f, "Where's the beef?",
    "Hungry?", JOptionPane.QUESTION_MESSAGE, null, options, options[0]);
}
System.out.println("Entered: " input);
```

Cuadros de diálogo para opciones

Los cuadros de diálogo para opciones son una variedad general y no sólo para un propósito especifico y configurable. De hecho todos los otros cuadros de diálogo son para opciones más especificas. Con los cuadros de diálogo para opciones se obtiene un control total a través de un método sencillo:

```
int showOptionDialog(Component parentComponent, Object message, String
title, int optionTyoe, int messageType, Icon icon, Object[]options, Object
initialValue)
```

Hay dos cosas a tener en cuenta al usar directamente este tipo de diálogos. Primero, la lista de opciones se convierte en botones para que el usuario realice su selección mediante éstos. El valor que se devuelve es la posición del objeto seleccionado. Lo segundo que se debe tener en cuenta es el pardmetro **`message`**. En el caso bdsico de un **`Component`**, **`Icon`**, o un objeto que no sea un array, el objeto del parámetro **`message`** se coloca justo dentro del **`pane`**. Para un **`Icon`**, éste se sitúa en un `JLabel`. Para un objeto (corno un **`String`**) también se coloca, con la ayuda de `toString()`, en un `JLabel`. Pero si el parámetro es un array de elementos, éstos se tratan como una serie de mensajes y se sitlian en un cuadro vertical, unos sobre otros. Por ejemplo, al mostrar un área de texto con un String para que el usuario acepte un acuerdo de licencia, manualmente se configuran los componentes de la opción **`pane`**, como se muestra en la figura 15.10.

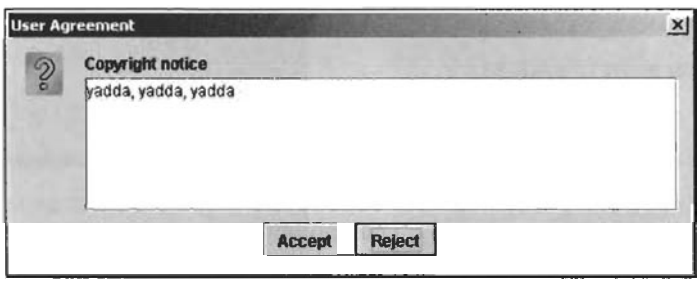


Figura 15.10. Panel genérico de opciones con mensaje y area de **texto**

El siguiente código se puede utilizar para asegurar que el usuario acepta el acuerdo de licencia en un cuadro de diálogo como el de la figura 15.10.:

```
String choices[] = new String [] {"Accept", "Reject"};
String licenseLabel = "Copyright notice";
String license = "yadda, yadda, yadda";
JScrollPane sp = new JScrollPane(new JTextArea(license, 6, 40));
Object msgs[] = new Object [] {licenseLabel, sp};
int returnValue2 = JOptionPane.CLOSED_OPTION;
while(returnValue2 != 0) {
  returnValue2 = JOptionPane.showOptionDialog(f, msgs,
    "User Agreement", JOptionPane.OK_CANCEL_OPTION,
    JOptionPane.QUESTION_MESSAGE, null, choices, choices[1]);
}
```

Cuadros de selección de colores

Cuando `JOptionPane` está disponible para mostrar mensajes de información y para la adquisición de entradas generales de usuarios, el componente **`JColor-Chooser`** es un mecanismo de entrada especializado para que los usuarios puedan

seleccionar un color. Por lo general, sin embargo, se crea y muestra la seleccion en un solo paso con el siguiente código:

```
showDialog(Component parent, String total, Color initialValue)
```

Existen otras muchas maneras de manipular el componente, que se detallan a continuación, en la tabla 15.5. Al igual que la opción `JOptionPane`, no existe un **`JColorChooser`** por defecto dentro de una ventana emergente.

Tabla **15.5.** Cornposicion de la clase JColorChooser

Variables/Métodos	Descripción
accesibleContext	Contexto accesible para el componente
CHOOSER – PANEL – PROPERTY	Nornbre de propiedad utilizado para propiedades ligadas dentro de un rnisrno componente
PREVIEW – PANEL – PROPERTY	
SELECTION MODEL – PROPERTY	
`JColorChooser``()`	Construye un objeto JColorChooser
`AddChooserPanel()`	Añade un nuevo panel de eleccion para rnostrar
`createDialog()`	Construye un cuadro de dialogo en el que incluir un componente
`getAccessibleContext()`	Devuelve el contexto accesible actual
`getChooserPanels()`	Devuelve el grupo actual de paneles de eleccion
`getColor()`	Devuelve el color seleccionado
`get``DragEnabled()`	Devuelve una indicación si la acción de arrastrar esta activada
`getPreviewPanel()`	Devuelve el panel de previsualizacion actual
`getSelectionModel()`	Devuelve el rnodelo de seleccion
`getUI()`	Devuelve el cornponente UI
`getUIClassID()`	Devuelve el nornbre de clase de UI
paramString()	Devuelve la representación string del estado del componente
`removeChooserPanel()`	Elirnina el panel de seleccion de la pantalla
`setChooserPanels()`	Cambia el conjunto de paneles de seleccion
`setColor()`	Cambia el color seleccionado
`setDragEnabled()`	Cambia el modo de arrastre actual
`setPreviewPanel()`	Cambia el cornponentede previsualizacion del panel

Variables/Métodos	Descripción
`setSelectionModel()`	Cambia el modelo de la selección
`setUI()`	Cambia el cornponente UI
`showDialog()`	Muestra una ventana ernergente con los colores dentro de JDialog
`updateUI()`	Actualiza el componente visible tras un cambio de aspecto y sensibilidad en Java

Cada ficha `JColorChooser` tiene su propio estilo de entrada. Los tres paneles por defecto del componente se muestran en la figura 15.11.

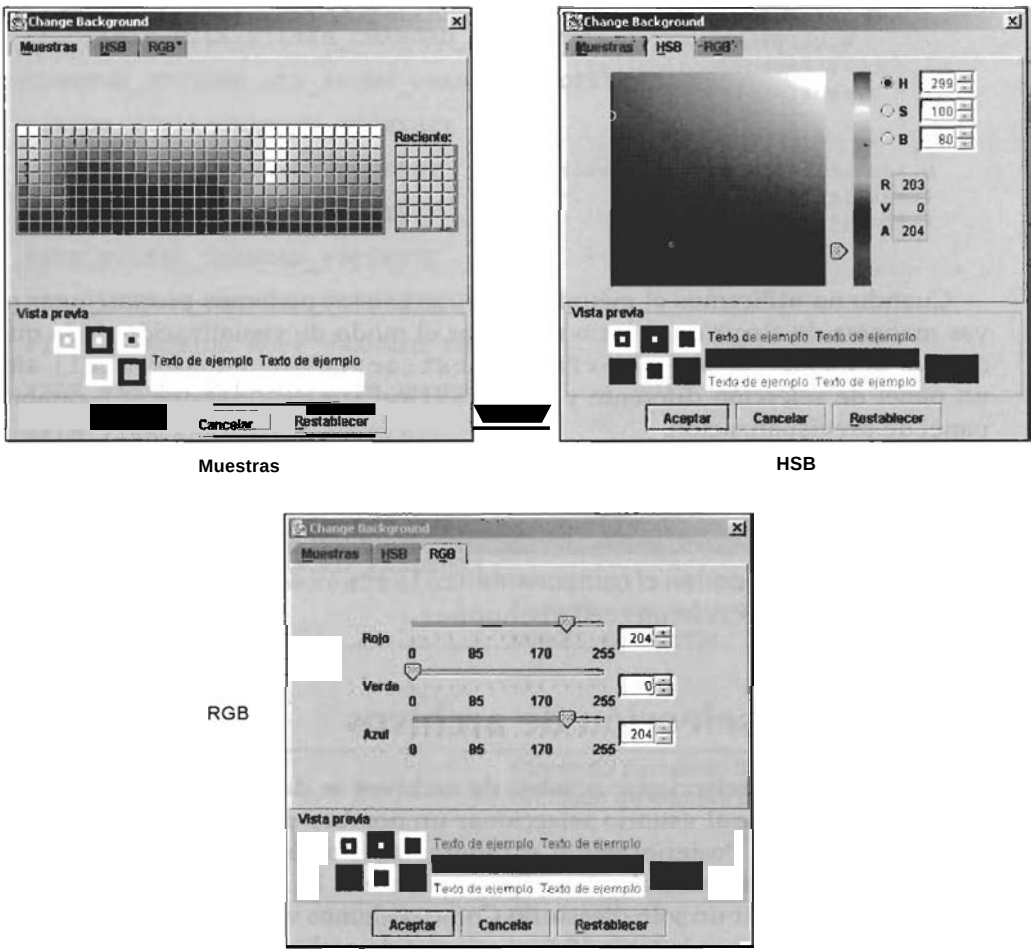


Figura 15.11. Mecanisrnos de entrada de JColorChooser

Para demostrar las propiedades de **JColorChooser**, el siguiente programa coloca un botón dentro de un marco. Cuando se selecciona el botón, se muestra un **JColorChooser** para cambiar el color de fondo del botón.

Listado 15.3. Selección de color con JColorChooser

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class ColorTester {
  public static void main(String args[]) {
    final JFrame f = new ExitableJFrame("Color Tester");
    final JButton b = new JButton("Change Button Background");
    f.getContentPane().add(b, BorderLayout.CENTER);
    ActionListener listener = new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        Color color = JColorChooser.showDialog(f, "Change Background",
          b.getBackground());
        b.setBackground(color);
        b.repaint();

    };
    b.addActionListener(listener);
    f.setSize(300, 100);
    f.show();
```

Cuando no utilicemos el método `showDialog()`, podemos proporcionar nuevas maneras de elegir un color o modificar el modo de visualización de la nueva opción. El método `addChooserPanel(AbstractColorChooserPanel)` añade un panel de seleccidn diferente y `setPreviewPanel(JComponent)` cambia el panel de previsualización.

Nota

Las clases que soportan el componente **JColorChooser** están disponibles en el paquete `javax.swing.colorchooser`.

Cuadros de selección de archivos

El diálogo para seleccionar nombre de archivos se denomina `JFileChooser`. Este cuadro permite al usuario seleccionar un nombre de un archivo a la hora de abrirlo o guardarlo. Posteriormente, el programador obtendra la información seleccionada y la utilizará para abrir o salvar un archivo. El mismo mecanismo se emplea para seleccionar un solo directorio Como podemos ver en la composición de la clase que se muestra en la tabla 15.6, el selector de archivos dispone de gran cantidad de opciones de personalización.

Tabla 15.6. Cornposicion de la clase JColorChooser

Variables/Métodos	Descripción
accessibleContext	Contexto accesible para un cornponente
ACCEPT – ALL – FILE – FILTER – USED – CHANGED – PROPERTY	Nornbre de las propiedades utilizadas para propiedades vinculadas dentro del componente
ACCESSORY_CHANGED_PROPERTY	
APPROVE – BUTTON – MNEMONIC – CHANGED – PROPERTY	
APPROVE – BUTTON – TEXT – CHANGED – PROPERTY	
APPROVE_BUTTON_TOOL_TIP_TEXT_CHANGED_PROPERTY	
CHOOSABLE_FILE_FILTER_CHANGED_PROPERTY	
CONTROL – BUTTONS – ARE – SHOWN – CHANGED – PROPERTY	
DIALOG – TITLE – CHANGED – PROPERTY	
DIALOG_TYPE_CHANGED_PROPERTY	
DIRECTORY – CHANGED – PROPERTY	
FILE – FILTER – CHANGED – PROPERTY	
FILE_HIDING_CHANGED_PROPERTY	
FILE – SELECTION – MODE – CHANGED – PROPERTY	
FILE – SYSTEM – VIEW – CHANGED – PROPERTY	
FILE – VIEW – CHANGED – PROPERTY	
MULTI – SELECTION – ENABLED – CHANGED – PROPERTY	
SELECTED – FILE – CHANGED – PROPERTY	
SELECTED_FILES_CHANGED_PROPERTY	
APPROVE – OPTION	Devuelve el valor que indica corno se cerro el cuadro ernergente
CANCEL – OPTION	
ERROR – OPTION	
APPROVE – SELECTION	String del comando del evento para aceptar o rechazar de rnanera prograrnada
CANCEL – SELECTION	
DIRECTORIES – ONLY	Especifica el elemento a rnostrar en el cuadro de diálogo
FILES – AND – DIRECTORIES	

Variables/Métodos	Descripción
FILES_ONLY	
CUSTOM – DIALOG	Especifica el tipo de cuadro de diálogo que se va a mostrar (texto del boton de accion)
OPEN – DIALOG	
SAVE – DIALOG	
`JFileChooser()`	Construye un objeto `JFileChooser`
accept ()	Comprueba si el archivo debe mostrarse o no
`addActionListener()`	Añade el controlador de eventos del boton de seleccion
`addChoosableFileFilter()`	Añade el filtro al componente
`approveSelection()`	Conmuta la seleccion Aceptar
`cancelSelection()`	Conmuta la seleccion Cancelar
`changeToParentDirectory()`	Hace que el directorio actual suba un nivel
createDialog()	Construye el cuadro de diálogo en el que incluir el componente
`ensureFileIsVisible()`	Asegura que el archivo se muestre dentro de la región visible del componente
fireActionPerformed()	Notifica la seleccion a los controladores de accion listening
`getAcceptAllFileFilter()`	Devuelve el filtro especifico de plataforma Aceptar todos (*.*, por ejemplo)
`getAccessibleContext()`	Devuelve el contexto accesible actual
`getAccessory()`	Devuelve el accesorio de componente
`getActionListeners()`	Devuelve el conjunto actual de listeners
`getApproveButtonMnemonic()`	Devuelve el acceso directo para el boton de aceptacion
`getApproveButtonText()`	Devuelve la etiqueta de texto del boton de aceptacion
`getApproveButtonToolTipText()`	Devuelve el mensaje de ayuda del boton de aceptación
`getChoosableFileFilters()`	Devuelve el conjunto de filtros de opción para el usuario
`getControlButtonsAreShown()`	Indica si los botones de control aparecen en el cuadro
`getCurrentDirectory()`	Devuelve el directorio actual

Variables/Métodos	Descripción
`getDescription()`	Devuelve la descripcion del archivo
`getDialogTitle()`	Devuelve el título del cornponente
`getDialogType()`	Devuelve el tipo de cornponente
`getDragEnabled()`	Indica si la opcion de arrastrar esta activada
`getFileFilter()`	Devuelve el filtro actual
`getFileSelectionMode()`	Devuelve el rnodo de seleccion
`getFileSystemView()`	Devuelve la vista del sisterna de archivos
`getFileView()`	Devuelve la vista de los archivos
`getIcon()`	Devuelve el icono del archivo
`getName()`	Devuelve el nornbre del archivo
`getSelectedFile()`	Devuelve el archivo seleccionado
`getSelectedFiles()`	Devuelve el conjunto de archivos seleccionados
`getTypeDescription()`	Devuelve el tipo de descripcion del archivo
`getUI()`	Devuelve el componente UI
`getUIClassID()`	Devuelve el nornbre de clase de UI
`isAcceptAllFileFilterUsed()`	Indica si se está utilizando el filtro Aceptar todos
`isDirectorySelectionEnabled()`	Indica si la seleccion de directorios esta activada
`isFileHidingEnabled()`	Indica si la opcion de ocultar archivos esta activada
`isFileSelectionEnabled()`	Indica si la seleccion de archivos esta activada
`isMultiSelectionEnabled()`	Indica si se pueden seleccionar varias entradas
`isTraversable()`	Indica si el archivo puede desplazarse (un directorio)
`paramString()`	Devuelve la representación del estado del componente
`removeActionListener()`	Elirnina el controlador de eventos del boton de seleccion
`removeChoosableFileFilter()`	Elirnina el filtro
`rescanCurrentDirectory()`	Hace que se actualice la presentación del directorio actual

Variables/Métodos	Descripción
`resetChoosableFileFilters()`	Reestablece el conjunto de filtros
`setAcceptAllFileFilterUsed()`	Añade o elimina la opcion de aceptar todos los filtros del conjunto de filtros disponibles para el usuario
`setAccessory()`	Cambia el accesorio del componente
`setApproveButtonMnemonic()`	Cambia el acceso directo del boton de aceptacion
`setApproveButtonText()`	Cambia la etiqueta de texto del boton de aceptacion
`setApproveButtonToolTipeText()`	Cambia el mensaje de ayuda del boton de aceptacion
`setControlButtonsAreShown()`	Cambia si se muestran los botones de control
`setCurrentDirectory()`	Cambia el directorio actual
`setDialogTitle()`	Cambia el titulo del componente
`setDialogType()`	Cambia el tipo de componente
`setDranEnabled()`	Cambia el modo de arrastre actual
`setFileFilter()`	Cambia el filtro actual
`setFileHidingEnabled()`	Cambia si la opción de ocultar archivos esta activada
`setFIleSelectionMode()`	Cambia el modo de selección del archivo
`setFileSystemView()`	Cambia la vista del sistema de archivos
`setFileView()`	Cambia la vista del archivo
`setMultiSelectionEnabled()`	Cambia el modo de selección
`setSelectedFile()`	Cambia el archivo seleccionado
`setSelectedFiles()`	Cambia los archivos seleccionados
`setup()`	Configura el componente segun los ajustes actuales
`showDialog()`	Muestra el componente en el cuadro de diálogo modal
`showOpenDialog()`	
`showSaveDialog()`	
`updateUI()`	Actualiza los componentes visualizados debido a un cambio de aspecto y sensibilidad

No hay que abrumarse ante la longitud de la definición de la clase **JFileChooser**. En su modo más sencillo, se crea el **JFileChooser** y se muestra. Posteriormente, se observa el informe de estado devuelto, por si el usuario selecciona **Cancelar** o no. Si no lo selecciona, se abre o salva el fichero seleccionado. Observe el siguiente ejemplo:

```
JFileChooser chooser = new JFileChooser();
int status = chooser.showOpenDialog(aFrame);
// o int status = chooser.showSaveDialog(aFrame);
if (status == JFileChooser.APPROVE_OPTION) {
  File f = chooser.getSelectedFile();
  processFile(f);
}
```

Cuando se muestra el selector de archivos, aparece la ventana que vemos en la figura **15.12,** comenzando seguramente en un directorio distinto.

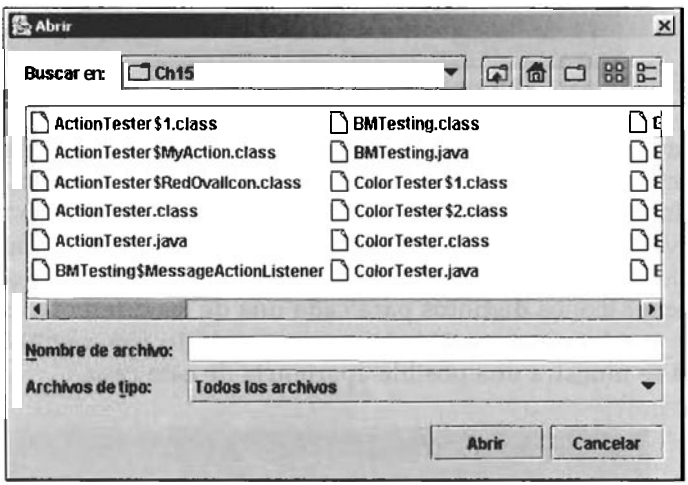


Figura 15.12. JFileChooser basico con la apariencia y sensibilidad de Metal o Java

Como todos los componentes Swing, cada uno puede tener multiples apariencias visuales. Por ejemplo, si la apariencia está configurada a Windows, se verá el diálogo estándar de Windows al mostrar un **JFileChooser** (figura **15.13).**

Para comenzar **JFileChooser** en un directorio concreto, sólo tenemos que incluir el directorio en el constructor.

Por ejemplo, para comenzar en el directorio actual, debemos incluir un punto, como se muestra en el ejemplo siguiente:

```
JFileChooser fileChooser = new JFileChooser(".");
```

A través de la clase, no es posible acceder al campo nombre de archivo dentro del JFileChooser. Para ello, debemos introducirnos en el componente UI de la clase y acceder el campo desde alli, acción que puede ser necesaria cuando deseemos

inicializar el campo o eliminarlo al reutilizar el componente. La tarea de limpieza se realiza con una operacidn como la siguiente:

```
BasicFuileChooserUI ui = (BasicFileChooserUI)fileChooser.getUI();
ui.setFileName("");
```

Figura 15.13. JFileChooser de Windows

Tal y como veremos muy pronto con otros componentes Swing, se puede personalizar prácticamente todo con el **`JFileChooser`**. Por ejemplo, si se desea tener filtros de archivos seleccionables, de tal modo que el usuario solamente pueda elegir archivos de imágenes, Java o HTML, entonces se debe asociar un `FileFilter` con `addChoosableFileFilter()` que muestre sólo archivos GIF y JPEG. Si deseamos tener iconos distintos para cada una de las diferentes extensiones de los archivos se puede crear un `FileView` y conectarlo con **`set`**`FileView()`. En la figura 15.14 se muestra una posible apariencia de este caso.

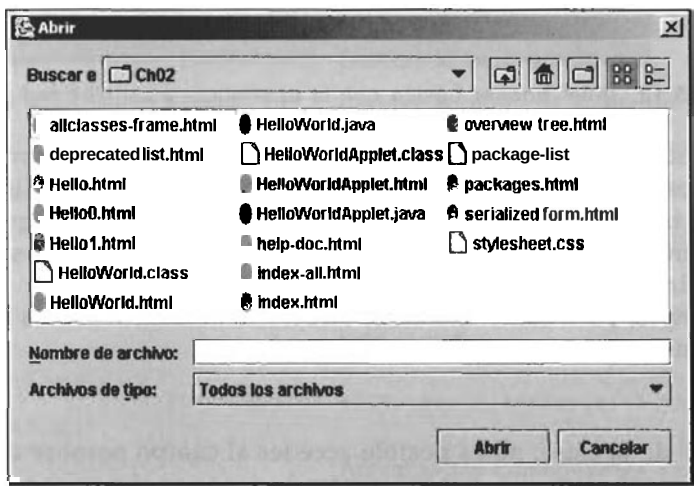


Figura 15.14. JFileChooser mejorado

La lista 15.4 muestra el programa que se ha utilizado para personalizar el `JFileChooser` de la figura 15.14. El objeto personalizado `FileView` se llama `IconView` y muestra diferentes ejemplos de `AnOvalIcon` basados en la extensión de los archivos. La clase `ExtensionFilter` permite especificar qué extensiones de archivos podrán seleccionarse por un `FileFilter`.

Listado 15.4. Personalización de JFileChooser

```
import java.awt.*;
import java.io.File;
import java.util.HashMap;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.filechooser.*;
public class EnhancedFileTester extends JPanel {
  class AnOvalIcon implements Icon {
    Color color;
    public AnOvalIcon(Color c) {
      color = c;
    }
    public void paintIcon(Component c, Graphics g,
        int x, int y) {
      g.setColor(color);
      g.fillOval(x, y, getIconWidth(), getIconHeight());
    }
    public int getIconWidth() {
      return 10;
    }
    public int getIconHeight() {
      return 15;
    }

  public class IconView extends FileView {
    private HashMap hash = new HashMap();
    public IconView() {
      hash.put("htm", new AnOvalIcon(Color.RED));
      hash.put("html", new AnOvalIcon(Color.GREEN));
      hash.put("java", new AnOvalIcon(Color.BLUE));

    public String getName(File f) {
      String s = f.getName();
      if (s.length() == 0) {
        s = f.getAbsolutePath();
      }
      return s;
    }
    public String getDescription(File f) {
      return f.getName();
    }
    public String getTypeDescription(File f) {
      return f.getAbsolutePath();
    }
    public Icon getIcon(File f) {
      String path = f.getAbsolutePath();
```

```
    int pos = path.lastIndexOf( '.' );
    if ((pos >= 0) && (pos < (path.length() - 1)) ) {
      String ext = path.substring(pos + 1).toLowerCase();
      return(Icon)hash.get(ext);

    return null;

  public Boolean isTraversable(File file) {
    return(new Boolean(file.isDirectory()));
  }

public class ExtensionFilter extends FileFilter {
  private String extensions[];
  private String description;
  public ExtensionFilter(String description, String extension) {
    this(description, new String[] {extension});

  public ExtensionFilter(String description, String extensions[]) {
    this.description = description;
    this.extensions = (String[])extensions.clone();

  public boolean accept(File file) {
    if (file.isDirectory()) {
      return true;

    int count = extensions.length;
    String path = file.getAbsolutePath();
    for (int i =0;i < count;i++) {
      String ext = extensions[i];
      if (path.endsWith(ext) &&
          (path.charAt(path.length()-ext.length()) == '.')) {
        return true;
      }

    return false;

  public String getDescription() {
    return(description == null ? extensions[0] : description);
  }
}
 public EnhancedFileTester() {
   JButton jb = new JButton("Open File Viewer");
   add(jb);
   ActionListener listener = new ActionListener() {
     public void actionPerformed(ActionEvent e) {
       JFileChooser chooser = new JFileChooser(".");
       FileFilter type1 = new ExtensionFilter(
         "Java source", ".java");
       FileFilter type2 = new ExtensionFilter(
         "Image files", new String[] {".jpg", ".gif", "jpeg", "xbm"});
       FileFilter type3 = new ExtensionFilter(
         "HTML files", new String[] {".htm", ".html"});
       chooser.addChoosableFileFilter(type1);
       chooser.addChoosableFileFilter(type2);
       chooser.addChoosableFileFilter(type3);
       chooser.setFileFilter(type2); // ajuste inicial del filtro
```

```
        FileView view = new IconView();
        chooser.setFileView(view);
        int status = chooser.showOpenDialog(EnhancedFileTester.this);
        if (status == JFileChooser.APPROVE_OPTION) {
          File f = chooser.getSelectedFile();
          System.out.println(f);
        }
      }
    };
    jb.addActionListener(listener);
  }
  public static void main(String args[]) {
    JFrame f = new ExitableJFrame("Enhanced File Example");
    JPanel j = new EnhancedFileTester();
    f.getContentPane().add(j, BorderLayout.CENTER);
    f.setSize(300, 200);
    f.show();
  }
}
```

Truco

Hay que tener cuidado para no confundir las clases `FileFilter` en los paquetes `java.io` y `javax.swing.filechooser`.

Algo que no se ha demostrado aqui pero que podemos hacer, es aiiadir un accesorio (como un previsualizador de archivos o un reproductor de archivos de audio) mediante el siguiente método:

```
setAccessory(JComponent accessory)
```

Este accesorio se mostraria a la derecha de la lista de archivos. Para que funcione se debe registrar el componente como un `PropertyChangeListener` con el `JFileChooser`:

```
chooser.addPropertyChangeListener(theAccessory)
```

y también fijarse en `JFileChooser.SELECTED_FILE_CHANGED_PROPERTY` `PropertyChange-Event`. Esta propiedad, utilizada para cambiar eventos, funciona de manera similar a como lo hacian los eventos de AWT excepto en que cuando una propiedad se cambia para un objeto, normalmente se hace por medio de un método `setXXX()`. Este concepto se explicará más profundamente cuando se hable de los Beans de Java. El siguiente ejemplo demuestra la manera en que un panel de accesorio responde al modificarse el archivo seleccionado:

```
public void propertyChange(PropertyChangeEvent e) {
  String prop = e.getPropertyName();
  if (prop == JFileChooser.SELECTED_FILE_CHANGED_PROPERTY) {
    f = (File)e.getNewValue();
    if (isShowing()) {
```

```
            // cambia los ajustts para mostrar la nueva imagen
            changeStuff(f);
            repaint();
        }
    }
}
```

Arquitectura Modelo/Vista/Controlador

La arquitectura Modelo/Vista/Controlador (MVC) es la segunda forma en la que se pueden utilizar los componentes Swing. Tal y como su nombre indica, la arquitectura estd compuesta por tres partes:

- El modelo define el estado de un sistema: Se trata de la representación lógica subyacente.
- La vista define la forma en que el usuario observa el modelo: La representacion visual.
- El controlador define cómo el usuario interactua con el modelo.

Para ayudar a comprender estos conceptos, pensemos en una cuadricula de numeros. Los numeros que constituyen las posiciones de las filas y columnas definen un modelo. Por ejemplo, en la posición (4,2) el valor puede ser -24.5. La vista define la manera en que el usuario ve los datos. De esta manera, para un sistema de contabilidad, un numero negativo se puede mostrar en rojo o entre paréntesis, o puede que se desee mostrar el numero tan sólo con un tipo de fuente diferente. El modelo en sí mismo no cambia, el valor es siempre -24.5. Únicamente la vista del modelo por parte del usuario es lo que cambia. Es posible que el modelo termine siendo una tabla. De nuevo, el modelo de numeros no cambia. Lo unico que cambia es el modo en que el usuario ve la cuadricula al cambiar de modelo. Cada vista de éste se adaptard automáticamente. Por ejemplo, si el valor -24.5 cambia a 3.14, éste no se mostrará más en rojo y la vista de la tabla se actualizard.

La tercera parte, el controlador, define como el usuario interactua con el modelo. Esto está muy unido a la vista. Si una vista es un diagrama de barras, y el usuario arrastra un barra hacia abajo, el valor del numero asociado a esta disminuiria. Debido a la estrecha unión entre el controlador y la vista, estos dos conceptos se combinan a menudo formando un objeto, llamado *delegate*. En Swing, también se le suele llamar objeto UI o UI delegate (*User Interface* o Interfaz del usuario).

Ya se ha visto anteriormente como se trabaja con MVC. Cuando se describió el `JFileChooser` y se instal6 un `FileView` nuevo en el listado 15.5, la lista de los nombres de ficheros nunca cambió. Lo unico que cambió fue el icono. También se puede cambiar el nombre mostrado modificando el método `getName()`. Mientras que esta acción cambia el nombre del fichero que se muestra en la pantalla, el método `getSelectedFile()` sigue devolviendo el nombre real y no el nombre mostrado. Es asi como funciona el MVC.

Nota

Aunque normalmente se asocia MVC a componentes visuales, este no esta restringido a este entorno.

Con cada componente Swing, existe siempre un modelo y posiblemente dos. El segundo modelo aparece cuando hay un mecanismo de selección especial. Por ejemplo, un `ListModel` que describe los datos que se encuentran detrás de un componente `JList`. El `ListModel` indica qué elementos se muestran en un `JList`. El segundo modelo asociado con un `JList` es su mecanismo de selección, es decir, si utiliza un modo selección individual o múltiple. Esto también es un modelo y se representa por `ListSelectionModel`. Los modelos de selección tienden a estar relacionados con las representaciones gráficas o a interaccionar con un componente.

Utilización del modelo de boton

Para entender mejor los conceptos anteriores, es importante definir el componente más simple de todos: el `JButton`. El `JButton`, como anteriormente se dijo, puede tener varios iconos diferentes asociados. Cada uno de estos iconos representa un estado diferente de un botón. Estos estados se describen en el modelo de botón y no en el de los iconos. Debido a que la etiqueta de texto es sólo una parte visual del botón, ésta no es parte del modelo. Se ve claramente en la definición de la clase `ButtonModel`:

```
public interface ButtonModel extends ItemSelectable {
  public void addActionListener(ActionListener l);
  public void addChangeListener(ChangeListener l);
  public void addItemListener(ItemListener l);
  public String getActionCommand();
  public int getMnemonic();
  public boolean isArmed();
  public boolean isEnabled();
  public boolean isPressed();
  public boolean isRollover();
  public boolean isSelected();
  public void removeActionListener(ActionListener l);
  public void removeChangeListener(ChangeListener l);
  public void removeItemListener(ItemListener l);
  public void setActionCommand(String command);
  public void setArmed(boolean b);
  public void setEnabled(boolean b);
  public void setGroup(ButtonGroup group);
  public void setMnemonic(int key);
  public void setPressed(boolean b);
  public void setRollover(boolean b);
  public void setSelected(boolean b);
}
```

El DefaultButtonModel es el modelo especifico que se utiliza por defecto. Se encarga, básicamente, de notificar cualquier vista cuando cambia el estado del modelo.

Para demostrar la conexión entre modelos y botones, el siguiente listado muestra la creación de dos objetos JButton o vistas que comparten el mismo modelo. Gracias a ello, cuando se selecciona uno también lo hace el otro, notificandose ambos en los dos ActionListeners.

Listado 15.5. Modelos comunes entre botones

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class BMTesting {
  static class MessageActionListener implements ActionListener {
    String message;
    public MessageActionListener(String msg) {
      message = msg;
    }
    public void actionPerformed(ActionEvent e) {
      System.out.println(message);
    }
  }
  public static void main(String args[]) {
    JFrame f = new ExitableJFrame("Button Model Tester");
    JButton jb1 = new JButton("Hello");
    ButtonModel bm = jb1.getModel();
    JButton jb2 = new JButton("World");
    jb2.setModel(bm);
    Container c = f.getContentPane();
    c.add(jb1, BorderLayout.NORTH);
    c.add(jb2, BorderLayout.SOUTH);
    jb1.addActionListener(new MessageActionListener("Selected One"));
    jb2.addActionListener(new MessageActionListener("Selected Two"));
    f.pack();
    f.show();
  }
}
```

Objeto delegate de botones

Los objetos delegate para un JButton se denominan ButtonUI o más concretamente, BasicButtonUI. Describen caracteristicas como el color, el tamaño y la forma en que se dibujan los bordes, los iconos y el texto. También están definidos los márgenes, para describir de esta manera el area seleccionable. Cuando el delegate detecta una acción dentro de sus bordes, se lo notifica al modelo. Como resultado de esto, el modelo comprueba si es necesario cambiar, y si lo es, el objeto UI lo hará. Todo esto ocurre, principalmente, detrás de la escena. El objeto UI no almacena información como el texto de la etiqueta actual. Sin embargo, describe cdmo dibujar ese texto. Se puede cambiar la apariencia fisica de un JButton cam-

biando su apariencia y sensibilidad (L&F, del inglés Look and Feel). Esto nos conduce al Look and Feel modificable (PLAF o PL&F) de Swing.

Look and Feel modificable

Gracias a la separación entre el modelo y la vista del MVC, es muy fácil cambiar el modo de presentar un objeto a un usuario sin tener que cambiar el estado de ese objeto. De hecho, Swing viene con Look and Feel para Windows, Motif, Macintosh y Metal. Para este último, se conoce con el nombre de Java Look and Feel.

> El L&F de Macintosh (Aqua) solo esta disponible en el Java de Apple para la plataforma Mac OS X.

Todo esto significa que se puede cambiar de apariencia con unas pocas llamadas a métodos sencillos. La apariencia Metal es la que aparece por defecto al ejecutarse las aplicaciones de Java. Para obtener el L&F local en lugar del dado por defecto, debemos modificar la apariencia de la manera pertinente. Lo único que cambia es la apariencia de la vista de cada objeto, sin cambiar el resto de las partes del programa. Hay que tener en cuenta que no se refiere a un programa especifico, sino al sentido general de cualquier programa que utilice componentes Swing.

En la figura 15.15 se muestra cómo pueden aparecer múltiples look and feel en una aplicación, aunque por lo general no los utilizaremos juntos. El listado 15.6 corresponde al programa que generó la figura 15.15.

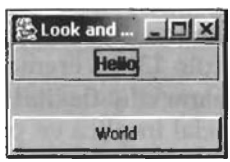


Figura 15.15. Carnbiando Look and Feels

LF

Listado 15.6. Intercarnbio de Look & Feel

```
import java.awt.*;
import javax.swing.*;
public class Local    {
  public static void main(String args[]) {
    JFrame f = new ExitableJFrame("Look and Feel");
    JButton jbl = new JButton("Hello");
    try {
      UIManager.setLookAndFeel(
        UIManager.getSystemLookAndFeelClassName());
    } catch(Exception ex) {
```

```
            System.err.println("Could not swap LookAndFeel");
        }
        JButton jb2 = new JButton("World");
        Container c = f.getContentPane();
        c.add(jb1, BorderLayout.NORTH);
        c.add(jb2, BorderLayout.SOUTH);
        f.setSize(150, 100);
        f.show();
    }
}
```

Nota

El Look and Feel de Windows solo esta disponible en la plataforma del mismo nombre. Existe un bloqueo de seguridad de software que evita su utilización en otras plataformas.

Componentes Swing avanzados

En capítulos anteriores únicamente se ha demostrado la forma de interactuar con componentes, utilizando el mecanismo de distribución de eventos estdndar. Gracias a MVC, cada componente soporta un segundo mecanismo. En esta sección aprenderemos a utilizar componentes Swing de una manera mbs orientada hacia MVC. Aunque este mecanismo secundario puede aparentar más dificultad de utilización y configuración, debido a que los programas van siendo cada vez más complejos, es fbcil darse cuenta que se hace mucho mds sencillo mantener aplicaciones orientadas a MVC.

Más concretamente, repasaremos las funciones de los componentes `JList` y `JcomboBox` vistos en el capítulo 13, y veremos por primera vez los componentes `JTable` y `JTree` y su gran potencial y flexibilidad, utilizados en el entorno MVC. Aunque un aumento de potencial implica en principio una mayor complejidad, lo que MVC exige, en realidad, es enfrentarse al problema desde una perspectiva distinta.

Cuadros de lista y listas desplegables

Tanto si se trabaja con objetos `JList` (cuadros de lista) como con **`JComboBox`** (cuadros de lista desplegables), se necesita proporcionar un conjunto de elementos para mostrarlos. Cada uno de estos objetos controla su propio conjunto de elementos internamente mediante un **`List`**`Model` o un **`ComboBoxModel`** respectivamente. Un **`ComboBoxModel`** es tan sólo un `ListModel` especializado. El método `ListCellRenderer` es el encargado de mostrar cada elemento de `JList` o `JComobBox`. La parte de selección del controlador de **`JComboBox`** se controla me-

diante **ComboBoxModel,** mientras que `JList` controla la selección mediante `ListSelectionModel`.

Una vez presentadas todas las piezas, sólo resta unirlas para crear un **JList** y un **JComboBox** que compartan un mismo modelo de datos. Los datos de los ejemplos siguientes constituyen la lista de ingredientes de un mago, además de la cantidad y precio. Este método proporciona además una manera muy sencilla de incrementar la lista, de modo que **JList** y **JComboBox** aumentan al mismo tiempo que lo hace la lista. Empecemos por desarrollar el modelo de los dos componentes.

ListModel/ComboBoxModel

El **ListModel** se compone de cuatro métodos:

```
public interface ListModel {
  public void addListDataListener(ListDataListener l);
  public Object getElementAt(int index);
  public int getSize();
  public void removeListDataListener(ListDataListener l);
}
```

mientras que el `ComboBoxModel` aiiade dos más.

```
public interface ComboBoxModel extends ListModel {
  public Object getSelectedItem();
  public void setSelectedItem(Object item);
}
```

Para crear nuestro propio modelo de datos, se creará una clase que implemente la interfaz `ComboBowModel` para satisfacer ambas necesidades, ya que implementa ambas interfaces. Sin embargo, el comportamiento para manejar la lista de listeners será común para todos los modelos de datos. Existe una clase que controla este proceso automáticamente: la clase **AbstractListModel**. En la tabla 15.7, esta clase aiiade varios métodos más que van más all6 de la interfaz.

Tabla **15.7.** Composición de la clase AbstractListModel

Variables/Métodos	Descripción
listenerList	Contiene la lista de los listener de eventos
`AbstractListModel()`	Construye un objeto AbstractListModel
`addListDataListener()`	Añade un listener al modelo
fireContentsChanged()	Notifica a los listener de los cambios en un rnodelo
fireIntervalAdded()	
fireInternalRemoved()	
`getElementAt()`	Obtiene un elemento del modelo

Variables/Métodos	Descripción
`getSize()`	Informa del numero de elementos del modelo
`removeListDataListener()`	Elimina un listener del modelo

Como el manejo de la lista de objetos en el modelo de datos es casi siempre igual, existe una clase que lo facilita: `DefaultListModel`, que se detalla a continuación, en la lista 15.8.

Tabla **15.8.** Composición de la clase DefaultListModel

Variables/Métodos	Descripción
`DefaultListModel()`	Construye un objeto DefaultListModel
`add()`	Añade un elemento al modelo
`addElement()` `insertElementAt()`	
`capacity()`	Devuelve la capacidad del modelo
`clear()`	Elimina todos los elementos del modelo
`removeAllElements()`	
`contains()`	Comprueba si el modelo contiene algun elemento
`copyInto()`	Copia un array de elementos dentro del modelo
`elementAt()`	Recupera un elemento del modelo
`get()`	
`getElementAt()`	
`elements()`	Devuelve una enumeración de los elementos del modelo
`ensureCapacity()`	Asegura que el modelo sostener un cierto número de elementos, modificando su tamaño si fuera necesario
`firstElement()`	Devuelve el primer elemento del modelo
`getSize()`	Devuelve el numero de elementos del modelo
`size()`	
`indexOff()`	Devuelve las posición de un elemento dentro del modelo
`isEmpty()`	Comprueba si el modelo esta vacio
`lastElement()`	Devuelve el ultimo elemento del modelo

Variables/Métodos	Descripción
lasIndexOff()	Devuelve la posición de un elemento dentro del rnodelo, cornenzando la búsqueda por el final
remove()	Elimina un elernento del modelo
removeElement()	
removeElementAt()	
removeRange()	Elimina una variedad de elernentos del rnodelo
set()	Cambia un elernento dentro del rnodelo
setElementAt()	
setSize()	Cambia el tamaño del rnodelo
toArray()	Copia los elementos del modelo a un array
toString()	Devuelve una representación en cadena del estado del cornponente
trimToSize()	Ajusta el tamaño del rnodelo segun la capacidad

Así, para crear un modelo de datos para utilizar tanto con `JList` como con `JComboBox`, se debe extender `DefaultListModel` y añadirlo en los métodos **`ComboBoxModel`**. El método clave para utilizar en **`DefaultListModel`** es `addElement()`, el cual se utiliza para añadir cada elemento dentro del modelo. Para la parte del **`ComboBoxModel`**, tan sólo se necesita almacenar el valor en algún lado y notificar cada uno de ellos por medio de **`fire`**`ContentsChanged()` cuando éste cambie. Esto se consigue mediante el siguiente código:

```
// Datos y modelo
final String partsList[][] = {
  ("Eye of Newt", "10", "99¢"},
  ["Toe of Frog", "12", "$18.99"},
  ["Wool of Bat", "1", "$10.00"},
  ["Tongue of Dog", "25", "$1.25"},
  {"Wolf Nipple Chips", "15", "$12.95"},
  ("Spleen of Ocelot", "18", "$2.50"},
  {"Black Acorn", "150", "75¢"},
  ("Pinch of Salt", "5000", "15¢"},
  {"Sphere of Power", "2", "$50.00"}
};
class PartsListModel extends DefaultListModel
    implements ComboBoxModel {
  Object currentValue;
  public PartsListModel() {
    for (int i=0, n=partsList.length;i<n;i++) {
      addElement(partsList[i]);
    }
```

```
  }
  // Métodos de ComboBoxModel
  public Object getSelectedItem() {
    return currentValue;
  }
  public void setSelectedItem(Object anObject) {
    currentValue = anObject;
    fireContentsChanged(this, -1, -1);
```

Para observar los cambios en la lista o en el modelo combo box, se registra un `ListDataListener`. Posteriormente, cuando el evento sucede, se notifica con un `ListDataEvent`, como se define en la tabla 15.9.

```
public interface ListDataListener extends EventListener {
  public void contentsChanged(ListDataEvent e);
  public void intervalAdded(ListDataEvent e);
  public void intervalRemoved(ListDataEvent e);
```

Tabla **15.9.** Cornposicion de la clase ListDataEvent

Variables/Métodos	Descripción
CONTENTS—CHANGED	Tipo de modificación realizada al rnodelo
INTERVAL—ADDED	
INTERVAL—REMOVED	
`ListDataEvent()`	Construye un objeto ListDataEvent
`getIndex0()`	Devuelve el índice desde el principio del tipo de elemento al que se le ha realizado el carnbio
`getIndex1()`	Devuelve el índice desde el final del tipo de elernento al que se le ha realizado el carnbio
`getType()`	Devuelve el tipo de evento
`toString()`	Devuelve la representación en cadena del estado del evento

ListCellRenderer

Si se desea usar este modelo en el programa, JList o JComboBox mostrarian cada elemento de un modo ilegible, como `[Ljava.lang.String;@704ad524`. Básicamente, nos indica que cada elemento es un array String, lo cual es absolutamente correcto. Para mejorar este resultado y darle sentido, todo lo que se debe hacer es definir un `ListCellRenderer` propio.

```
public interface ListCellRenderer {
public interface ListCellRenderer {
  public Component getListCellRendererComponent(JList list,
    Object value, int index, boolean isSelected, boolean cellHasFocus);
}
```

Normalmente, para un renderer se subclasifica `JLabel`, ya que suele resultar una manera apropiada de renderizar. El valor de `getListCellRendererComponent()` se convierte asi en algo que puede mostrarse. Por ejemplo, si el valor fuera un nombre de archivo, se podría acceder a diferentes vistas, ya sean como texto o como imagen. En este caso particular, el valor añadido arriba por `addElement()` es un array String, y sólo necesita ser convertido en algo legible. El componente también necesita hacerse opaco para que el color de fondo no sea trasparente. El siguiente código constituye el utilizado para visualizar el elemento `MyLabelRenderer`:

```
class MyLabelRenderer extends JLabel implements ListCellRenderer {
  public MyLabelRenderer() {
    setOpaque(true);
  }
  public Component getListCellRendererComponent(JList list,
      Object value, int index, boolean isSelected,
      boolean cellHasFocus) {
    if (value != null) {
      String values[] = (String[])value;
      String setting = values[0] t " / " t values[1] t " / " +
        values[2];
      setText(setting);

      setBackground(isSelected ? Color.BLUE : Color.WHITE);
      setForeground(isSelected ? Color.WHITE : Color.BLUE);
    return this;
  }
}
```

ListSelectionModel

El siguiente elemento a considerar en cuanto al aspecto de `JList` y `JComobBox` es el modelo de selección, que corresponde a una de las tres constantes de `ListSelectionModel`: `SINGLE_SELECTION`, `SINGLE_INTERVAL_SELECTION` o `MULTIPLE_INTERVAL_SELECTION`.

El último de ellos es el valor por defecto. Como `DefaultListSelectionModel` es la implementación por defecto del `ListSelectionModel` utilizado, las otras dos definiciones de clase no son del todo necesarias. Todo lo que necesitamos conocer para cambiar el modelo de selección son las tres constantes. Si se desea poner una lista en un modo de selección individual, debemos realizar los siguientes pasos.

```
ListSelectionModel lsrn = jl.getSelectionModel();
lsm.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
```

Si deseamos saber cuándo se seleccionó un rango, aiiadiriamos un `ListSelectionListener` al `ListSelectionModel` o `JList`. Este método, sin embargo, no nos dice directamente qué elementos están seleccionados. Se tiene que utilizar un índice suministrado por su `ListSelectEvent` para obtener la posición en el modelo. A continuación se muestra la definición de la clase para el `ListSelectionListener`y, en la tabla 15.10, la composición de `ListSelectionEvent`.

```
public interface ListSelectionListener extends EventListener {
  public abstract void valueChanged(ListSelectionEvent e);
```

Tabla 15.10. Composición de la clase ListSelectionEvent

Variables/Métodos	Descripción
CONTENTS—CHANGED	Tipo de cambio realizado al modelo
INTERVAL—ADDED	
INTERVAL—REMOVED	
`ListSelectionEvent()`	Construye un objeto ListSelectionEvent
`getFirstIndex()`	Devuelve el índice desde el principio del tipo de elemento al que se le ha realizado el cambio
`getLastIndex()`	Devuelve el índice desde el final del tipo de elemento al que se le ha realizado el cambio
`getValueIsAdjusting()`	Devuelve el tipo de evento
`toString()`	Devuelve la representación en cadena del estado del evento

Llegados a este punto, para conectar el listener al componente, sólo tenemos que aiiadirlo. En este caso concreto, el controlador sólo muestra el elemento seleccionado, y únicamente si el usuario no está arrastrando el ratón para cambiar el valor.

```
JList jl = ...;
jl.addListSelectionListener(new ListSelectionListener() {
  public void valueChanged(ListSelectionEvent e) {
    if (!e.getValueIsAdjusting()) {
      String element[] = (String[])pcm.getElementAt(e.getFirstIndex());
      System.out.println (element[0] + " : " + element[1] + " : " +
        element[2]);
    }
  }
});
```

La figura 15.16 nos muestra cómo se veria la lista de ingredientes del mago. Como vemos, cada componente mantiene la selección por separado, aunque la lista de elementos es siempre la misma, incluso después de pulsar el botón Add Mer-

chandise para aiiadir más ingredientes al modelo. El listado 15.7 contiene el código correspondiente al ejemplo. Se han incluido de nuevo algunas partes del código anterior para asi disponer de todo el código completo, que facilita la tarea de probar este ejemplo y ver el funcionamiento de todas las piezas.

Figura 15.16. Un JList y un JComboBox compartiendo el rnisrno modelo

Listado 15.7. JList y JComboBox cornpartiendo el mismo modelo

```
import javax.swing.*;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;
public class ListIt {
  final String partsList[][] = {
    {"Eye of Newt", "10", "99¢"},
    {"Toe of Frog", "12", "$18.99"},
    {"Wool of Bat", "1", "$10.00"},
    {"Tongue of Dog", "25", "$1.25"},
    {"Wolf Nipple Chips", "15", "$12.95"},
    {"Spleen of Ocelot", "18", "$2.50"},
    {"Black Acorn", "150", "75¢"},
    {"Pinch of Salt", "5000", "15¢"},
    {"Sphere of Power", "2", "$50.00"}
  };
  class PartsListModel extends DefaultListModel
      implements ComboBoxModel {
    Object currentValue;
    public PartsListModel() {
      for (int i=0, n=partsList.length;i<n;i++) {
        addElement(partsList[i]);
      }
    }
    // Métodos ComboBoxModel
    public Object getSelectedItem() {
      return currentValue;
    }
    public void setSelectedItem(Object anObject) {
      currentValue = anObject;
```

```
    fireContentsChanged(this, -1, -1);
  }
}
class MyLabelRenderer extends JLabel implements ListCellRenderer {
  public MyLabelRenderer() {
    setOpaque(true);
  }
  public Component getListCellRendererComponent(JList list,
      Object value, int index, boolean isSelected,
      boolean cellHasFocus) {
    if (value != null) {
      String values[] = (String[])value;
      String setting = values[0] + " / " + values[1] +
        " / " + values[2];
      setText(setting);
    }
    setBackground(isSelected ? Color.BLUE : Color.WHITE);
    setForeground(isSelected ? Color.WHITE : Color.BLUE);
    return this;
  }
}
public ListIt() {
  JFrame f = new ExitableJFrame("List Models");
  final PartsListModel pcm = new PartsListModel();
  ListCellRenderer lcr = new MyLabelRenderer();
  JList jl = new JList(pcm);
  jl.setCellRenderer(lcr);
  ListSelectionModel lsm = jl.getSelectionModel();
  lsm.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
  jl.addListSelectionListener(new ListSelectionListener() {
    public void valueChanged(ListSelectionEvent e) {
      if (!e.getValueIsAdjusting()) {
        String element[] = (String[])pcm.getElementAt(
          e.getFirstIndex());
        System.out.println(element[0] + " : " + element[1] +
          " . " + element[2]);
      }
    }
  });
  JScrollPane jsp = new JScrollPane(jl);
  JComboBox jc = new JComboBox(pcm);
  jc.setRenderer(lcr);
  JButton jb = new JButton("Add Merchandise");
  jb.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
      pcm.addElement(partsList[
        (int)(Math.random()*partsList.length)]);
    }
  });
  Container c = f.getContentPane();
  c.add(jsp, BorderLayout.NORTH);
  c.add(jc, BorderLayout.CENTER);
  c.add(jb, BorderLayout.SOUTH);
  f.setSize(250, 250);
  f.show();
}
```

```
  public static void main(String args[]) {
    new ListIt();
  }
}
```

Tablas

En capitulos anteriores se vio lo sencillo que era crear un **JTable** de un array de elementos en dos dimensiones. En esta sección, veremos todas las estructuras subyacentes involucradas y como se puede personalizar practicamente todo. El modelo de datos para una tabla es `TableModel`. La tabla está formada por multiples columnas a partir de este modelo de datos. `TableColumnModel` controla el modelo de orientación correspondiente a las columnas, mientras que `ListSelectionModel` dirige la selección de celdas, filas y columnas con la ayuda de varios métodos **JTable**. `TableCellRenderer` define la forma en que se muestra cada celda de la tabla, y el `TableCellEditor` describe el mecanismo para editar celdas. Todas estas clase de **JTable** se encuentran en el paquete `javax.swing.table`.

La clase **JTable** está disponible en **javax.swing**. Su definición se muestra en la tabla 15.11.

Para **JTable**, se proporciona un modelo personalizado, un render que se utiliza para dibujar celdas y también un editor de celdas. Además se pueden configurar en él si se desea diferentes opciones utilizando para ello controles de tipo visual. Todos estos elementos se describen en las siguientes secciones. Además, veremos de nuevo el método `ListSelectionModel` ya que **JTable** lo emplea también para indicar selecciones.

Nota

Los diseñadores de **JTable** no pretendían que esta se utilizara como hoja de calculo. El modelo de datos subyacentes de una tabla suele ser muy denso, mientras que el de una hoja de calculo pretende estar mas sencillo.

Tabla **15.11.** Cornposicion de la clase JTable

Variables/Métodos	Descripción
AUTO—RESIZE—ALL—COLUMNS	Tipo de carnbio efectuado al modelo
AUTO—RESIZE—LAST—COLUMN	
AUTO—RESIZE_NEXT—COLUMN	
AUTO—RESIZE_OFF	

Variables/Métodos	Descripción
AUTO – RESIZE – SUBSEQUENT – COLUMNS	
autoCreateColumnsFromModel	Variables para almacenar el estado de la tabla
autoResizeMode	
cellEditor	
cellSelectionEnabled	
columnModel	
dataModel	
defaultEditorsByColumnClass	
defaultRenderersByColumnClass	
editingColumn	
editingRow	
editorComp	
gridColor	
preferredViewportSize	
rowHeight	
rowMargin	
rowSelectionAllowed	
selectionBackground	
selectionForeground	
selectionModel	
showHorizontalLines	
showVerticalLines	
tableHeader	
`JTable()`	Construye un objeto JTable
`addColumn()`	Aiiade una colurnna al modelo de columna
`addColumnSelectionInterval()`	Aiiade una colurnna a la selección
`addNotify()`	Ayuda a configurar el tablero de desplazarniento
configureEnclosingScrollPane()	
`removeNotify()`	
unconfigureEnclosingScrollPane()	

Variables/Métodos	Descripción
addRowSelectionInterval()	Añade una fila a la seleccion
changeSelection()	Actualiza el modelo de selección de la tabla
clearSelection()	Elimina el modelo de seleccion de la tabla
columnAdded()	Utilizados internamente para notificar los carnbios en el modelo de la colurnna
columnMarginChanged()	
columnMoved()	
columnRemoved()	
columnSelectionChanged()	
columnAtPoint()	Comprueba que celda se encuentra debajo de un punto determinado
rowAtPoint()	
convertColumnIndexToModel()	Envía una vista de colurnna a un modelo de tabla
convertColumnIndextoView()	Envía una colurnna modelo a vista
createDefaultColumnModel()	Ayuda a crear la configuración por defecto de la tabla
createDefaultColumnsFromModel()	
createDefaultDataModel()	
createDefaultEditors()	
createDefaultRenderers()	
createDefaultSelectionModel()	
createDefaultTableHeader()	
initializeLocalVars()	
doLayout()	Hace que la tabla organice los elementos
editCellAt()	Los pulsadores comienzan a editar en una celda determinada
editingCanceled()	Notifica a la tabla que se ha cancelado o detenido la edición
editingStopped()	
getAccesibleContext()	Devuelve el contexto accesible actual
getAutoCreateColumnsFromModel()	Notifica si el modelo de columna debe crearse automaticamente según el modelo de la tabla

Variables/Métodos	Descripción
`getAutoResizeMode()`	Devuelve el rnodo de carnbio de tamaño actual
`getCellEditor()`	Devuelve el editor de celda
`getCellRect()`	Devuelve el rectangulo que forrna la celda
`getCellRenderer()`	Devuelve el renderer de la celda
`getCellSelectionEnabled()`	Devuelve el mecanisrno de selección actual
`getColumnSelectionAllowe()`	
`getRowSelectionAllowed()`	
`getColumn()`	Devuelve una colurnna de la tabla
`getColumnClass()`	Devuelve la clase de las colurnnas
`getColumnCount()`	Devuelve el número de colurnnas
`getColumnModel()`	Devuelve el rnodelo de colurnna
`getColumnName()`	Devuelve el nombre de la colurnna
`getDefaultEditor()`	Devuelve el editor de celdas de la clase
`getDefaultrenderer()`	Devuelve el renderer de la clase
`getDragEnabled()`	Indica si la opción de arrastrar esta activada
`getEditingColumn()`	Devuelve la colurnna de la edicion actual
`getEditingRow()`	Devuelve la fila de la edicion actual
`getEditorComponent()`	Devuelve el cornponente que controla la edicion
`getGridColor()`	Devuelve el color de las líneas de division
`getIntercellSpacing()`	Devuelve el espacio entre celdas
`getRowMargin()`	
`getModel()`	Devuelve el rnodelo de la tabla
`getPreferredScrollableViewportSize()`	Devuelve el tamaño del puerto de visualizacion de la tabla
`getRowCount()`	Devuelve el número de filas del modelo de tabla
`getRowHeight()`	Devuelve la altura de las filas
`getScrollableBlockIncrement()`	Devuelve el incremento de desplazamiento
`getScrollableUnitIncrement()`	

Variables/Métodos	Descripción
`getScrollableBlockIncrement()`	Devuelve los atributos del puerto de visualizacion
`getScrollableUnitIncrement()`	
`getSelectedColumn()`	Devuelve la columna o columnas seleccionadas
`getSelectedColumns()`	
`getSelectedColumnCount()`	Devuelve el numero de columnas seleccionadas
`getSelectedRow()`	Devuelve la fila o filas seleccionadas
`getSelectedRows()`	
`getSelectedRowCount()`	Devuelve el numero de filas seleccionadas
`getSelectionBackground()`	Devuelve el color de fondo para la seleccion
`getSelectionForeground()`	Devuelve el color frontal para la seleccion
`getSelectionModel()`	Devuelve el modelo de la seleccion
`getShowHorizontalLines()`	Indica si las líneas de division deben mostrarse
`getShowVerticalLines()`	
`getSurrendersFocusOnKeyStroke()`	Indica si el editor debe obtener un foco
`getTableHeader()`	Devuelve el encabezamiento de la tabla
`getToolTipText()`	Devuelve el mensaje de ayuda de la posicion
`getUI()`	Devuelve el componente UI
`getUIClassID()`	Devuelve el nombre de clase de UI
`getValueAt()`	Devuelve el valor de la celda
`isCellEditable()`	Indica si la celda es editable
`isCellSelected()`	Indica si la celda esta seleccionada
`isColumnSelected()`	Indica si la columna esta seleccionada
`isEditing()`	Indica si la tabla esta en modo de edición
`isRowSelected()`	Indica si la fila esta seleccionada
`moveColumn()`	Mueve la columna
`paramString()`	Devuelve la representación en cadena del estado del componente

Variables/Métodos	Descripción
prepareEditor()	Prepara el editor para ser utilizado
prepareRenderer()	Prepara el renderer para ser utilizado
processKeyBinding()	Hace que la pulsación de teclas se procese
removeColumn()	Elimina la columna del modelo
removeColumnSelectionInterval()	Deselecciona una seleccion del modelo
removeRowSelectionInterval()	
removeEditor()	Elimina el editor
resizeAndRepaint()	Hace que la llamada cambie su valor y su aspecto
selectAll()	Añade toda la tabla al modelo de seleccion
setAutoCreateColumnsFromModel()	Cambia si el modelo de columna debe realizarse de manera automática segun el modelo
setAutoResizeMode()	Cambia el modo de cambiar el tamaño de la colurnna
setCellEditor()	Cambia el componente de editor de la celda
setCellSelectionEnabled()	Cambia el mecanismo de seleccion actual
setColumnSelectionAllowed()	
setRowSelectionAllowed()	
setColumnModel()	Cambia el modelo de columna
setColumnSelectionInterval()	Cambia el intervalo de la seleccion
setRowSelectionInterval()	
setDefaultEditor()	Cambia el editor de las clases
setDefaultRenderer()	Cambia el renderer de las clases
setDragEnabled()	Cambia el modo de arrastre actual
setEditingColumn()	Cambia la colurnna de la edicion actual
setEditingRow()	Cambia la fila de la edicion actual
setGridColor()	Cambia el color de las líneas de división
setIntercellSpacing()	Cambia el espaciado entre celdas
setRowMargin()	
setModel()	Cambia el modelo de la tabla

Variables/Métodos	Descripción
setPreferredScrollableViwPort-Size()	Cambia el tamaño preferente del puerto de visualizacion
setRowHeight()	Cambia la altura de las filas
setSelectionBackground()	Cambia el color de fondo de la seleccion
setSelectionForeground()	Cambia el color frontal de la seleccion
setSelectionMode()	Cambia el rnodo de seleccion
setSelectionModel()	Cambia el modelo de seleccion
setShowGrid()	Cambia si las líneas de division deben mostrarse
setShowHorizontalLines()	
setShowVerticalLines()	
setSurrendersFocusOnKeyStroke()	Cambia si el editor debe obtener el foco al comenzar
setTableHeader()	Cambia el encabezamiento de la tabla
setUI()	Cambia el cornponente UI
setValueAt()	Cambia el valor de la celda
tableChanged()	Utilizado internamente cuando el modelo de la tabla cambia
updateUI()	Actualiza el cornponente de visualizacion al cambiarse su apariencia y sensibilidad
valueChanged()	Utilizado internarnente cuando la seleccion de filas cambia

TableModel

La interfaz `TableModel` **define la estructura de datos subyacente** correspondiente a un `JTable`:

```
public interface TableModel {
public void addTableModelListener(TableModelListener l);
  public Class getColumnClass(int column);
  public int getColumnCount();
  public String getColumnName(int column);
  public int getRowCount();
  public Object getValueAt(int row, int column);
  public boolean isCellEditable(int row, int column);
  public void removeTableModelListener(TableModelListener l);
  public void setValueAt(Object obj, int row, int column);
}
```

Para crear un modelo de datos propio, se podrian implementar todos los métodos, como ocurre con todas las demás interfaces. Sin embargo, tal y como sucede con `ListModel` y `AbstractListModel`, el comportamiento para dirigir las listas de listener podría ser común para todos los modelos de datos. En este caso existe también una clase similar a `AbastracListModel`, llamada `AbstractTableModel`, que controla todo esto automáticamente. La clase `AbstractTableModel` se describe en la tabla 15.12.

Tabla 15.12. Cornposicion de la clase AbstractTableModel

Variables/Métodos	Descripción
listenerList	Constituye la lista de los listener de eventos
`AbstractTableModel()`	Construye un objeto AbstractTableModel
`addTableModelListener()`	Aiiade un listener al modelo
`findColumn()`	Devuelve una columna deterrninada por su nornbre
`fireTableCellUpdated()`	Notifica a los listener los carnbios producidos en el rnodelo
`fireTableChanged()`	
`fireTableDataChanged()`	
`fireTableRowsDeleted()`	
`fireTableRowsInserted()`	
`fireTableRowsUpdated()`	
`fireTableStructureChanged()`	
`getColumnClass()`	Devuelve la clase de la columna
`getColumnName()`	Devuelve el nombre de la columna
`isCellEditable()`	Devuelve el valor `false`, indicando que el modelo no es editable
`removeTableModelListener()`	Elimina el listener del modelo
`setValueAt()`	Elimina los carnbios del modelo

Si se compara `TableModel` y `AbstractTAbleModel`, se puede observar que no aparecen los siguientes métodos de `AbstractTableModel`: `getColumnCount()`, `getRowCount()` y `getValueAt()`. Se trata de métodos que un modelo de tabla especifico podría necesitar implementar. Para un modelo editable, se podría también necesitar implementar `setValueAt()` e invalidar `isCellEditable()`. Existe una clase de ayuda para manejar la estructura de datos: `DefaultTableModel`.

Además de ser editable, este método añade algunos métodos más al conjunto anterior, como se detallan en la tabla **15.13.**

Tabla **15.13.** Composición de la clase DefaultTableModel

Variables/Métodos	Descripción
columnIdentifiers	Hace referencia a los nombres de columna
dataVector	Vector de los datos de las celdas
DefaultTableModel()	Construye un objeto DefaultTableModel
addColumn()	Añade datos al modelo
addRow()	
insertRow()	
convertToVector()	Convierte un array en un vector
getColumnCount()	Devuelve el tamaño del modelo de tabla
getRowCount()	
getColumnName()	Devuelve el nombre de la columna
getDataVector()	Devuelve los datos del modelo de tabla
getValueAt()	Devuelve el valor de la celda
isCellEditable()	Devuelve el valor `true`, indicando que el modelo es editable
moveRow()	Cambia la posición de los datos dentro del modelo
newDataAvailable()	Notifica a los listener los cambios producidos en el modelo de tabla
newRowsAdded()	
rowsRemoved()	
removeRow()	Elimina datos del modelo
setColumnCount()	Cambia el tamaño del modelo
setRowCount()	
setColumnIdentifiers	Cambia los identificadores de las columnas
setDataVector()	Cambia el modelo de los datos
setValueAt()	Cambia los datos del modelo

Para demostrar cómo se trabaja con un modelo de tabla propio, crearemos uno con tres columnas: un identificador no editable, un nombre y un color favorito. Si se

utiliza este modelo, la última columna mostraria un resultado deficiente. Mds tarde, cuando se trate `TableCellRenderer`, se observard una mejor manera de mostrar colores. El siguiente codigo pertenece a un modelo de tres columnas:

```
class MyTableModel extends DefaultTableModel {
  Object data[][] = {
    {"Willie", Color.RED},
    {"Ned", Color.ORANGE},
    {"Homer", Color.YELLOW},
    {"Barney", Color.GREEN},
    {"Marge", Color.BLUE},
    {"Bart", Color.MAGENTA},
    {"Lisa", Color.CYAN},
    {"Maggie", Color.PINK},
    {"Troy", Color.BLACK},
    {"Moe", Color.GRAY}
  };
  MyTableModel() {
    setColumnIdentifiers(new String[] {"ID", "Name", "Color"});
    for (int i=0,n=data.length;i<n;i++)
      addRow(new Object[] {new Integer(i+1), data[i][0], data[i][1]});
  }
  public boolean isCellEditable(int row, int column) {
    return(column != 0);
```

Para poder observar modificaciones en el modelo de la tabla, se registra un `TableModelListener`, como se muestra a continuación. Una vez hecho, cuando el modelo cambia, se notifica con un **`TableModelEvent`** (tabla 15.14).

```
public interface TableModelListener extends EventListener {
  public abstract void tableChanged(TableModelEvent e);
```

Tabla 15.14. Cornposicion de la clase TableModelEvent

Variables/Métodos	Descripción
ALL—COLUMNS	Utilizado para indicar todas las columnas del evento
DELETE	Indica el tipo de acción
INSERT	
UPDATE	
HEADER – ROW	Utilizado para indicar la fila encabezarniento del evento
`TableModelEvent()`	Construye un objeto TableModelEvent
`getColumn()`	Devuelve la columna de la acción
`getFirstRow()`	Devuelve la prirnera fila afectada

Variables/Métodos	Descripción
`getLastRow()`	Devuelve la última fila afectada
`getType()`	Devuelve el tipo de acción

TableColumnModel

El modelo para cada columna de una tabla se define por medio de la interfaz `TableColumnModel`.

```
public interface TableColumnModel {
  public void addColumn(TableColumn column);
  public void addColumnModelListener(TableColumnModelListener l
  public TableColumn getColumn(int col);
  public int getColumnCount();
  public int getColumnIndex(Object obj);
  public int getColumnIndexAtX(int xPos);
  public int getColumnMargin();
  public Enumeration getColumns();
  public boolean getColumnSelectionAllowed();
  public int getSelectedColumnCount();
  public int[] getSelectedColumns();
  public ListSelectionModel getSelectionModel();
  public int getTotalColumnWidth();
  public void moveColumn(int fromColumn, int toColumn);
  public void removeColumn(TableColumn column);
  public void removeColumnModelListener(TableColumnModelListener l);
  public void setColumnMargin(int margin);
  public void setColumnSelectionAllowed(boolean b);
  public void setSelectionModel(ListSelectionModel model);
}
```

En la mayoría de los casos, la implementación de `DefaultColumnModel` es suficiente, y entre otras cosas, proporciona una ayuda para controlar la lista de objetos de `TableColumnModelListeners`, como se muestra en la tabla 15.15. Varios de estos métodos conducen directamente a los métodos `JTable`.

Tabla 15.15. Cornposicion de la clase DefaultTableColumnModel

Variables/Métodos	Descripción
changeEvent	Evento a enviar cuando el modelo cambia
columnMargin	Variables para alrnacenar el estado del componente
columnSelectionAllowed	
selectionModel	

Variables/Métodos	Descripción
tableColumns	
totalColumnWidth	
listenerList	Contiene la lista de los listener de un evento
DefaultTableColumnModel()	Construye un objeto DefaultTableColumnModel
addColumn()	Añade una colurnna al modelo de colurnna
addColumnModelListener()	Atiade un listener al modelo
createSelectionModel()	Crea el modelo de selección por defecto
fireColumnAdded()	Notifica los listener registrados
fireColumnMarginChanged()	
fireColumnMoved()	
fireColumnRemoved()	
fireColumnSelectionChanged()	
getColumn()	Devuelve una colurnna del rnodelo
getColumnCount()	Devuelve un numero de columnas
getColumnIndex()	Devuelve una colurnna para el identificador
getColumnIndexAtX()	Devuelve una colurnna para la posición
getColumnMargin()	Devuelve el rnargen de la colurnna
getColumnModelListeners()	Devuelve el conjunto actual de listeners
getListeners()	
getColumns()	Devuelve las columnas actuales
getColumnSelectionAllowed()	Devuelve el mecanismo de seleccion actual
getSelectedColumnCount()	Devuelve el número de colurnnas seleccionadas
getSelectedColumns()	Devuelve un conjunto de colurnnas seleccionadas
getSelectionModel()	Devuelve el modelo de seleccion
getTotalColumnWidth()	Devuelve la anchura necesaria para rnostrar todas las colurnnas
moveColumn()	Desplaza una columna
propertyChange()	Notifica al modelo los cambios en las propiedades
recalcWidthCache()	Ayuda a calcular el total de anchura las colurnnas
removeColumn()	Elimina la colurnna del rnodelo

Variables/Métodos	Descripción
removeColumnModelListener O	Elirnina el listener del modelo
setColumnMargin()	Cambia los márgenes de la colurnna
setColumnSelectionAllowed()	Cambia el rnecanisrno de selección actual
setSelectionModel()	Cambia el modelo de seleccion
valueChanged()	Utilizado internarnente cuando la seleccion de columna cambia

La interfaz `TableColumnModelListener` describe todos los eventos que pueden ocurrir para alterar el `TableColumnModel`. También se recibirá un `TableColurnnModelEvent` (tabla 15.16), un `ListSelectionEvent` (descrito anteriormente en la lista 15.10) o bien un `ChangeEvent` (tabla 15.17), dependiendo, una vez más, del evento actual que este sucediendo.

```
public interface TableColumnModelListener extends EventListener {
  public void columnAdded(TableColumnModelEvent e);
  public void columnMarginChanged(ChangeEvent e);
  public void columnMoved(TableColumnModelEvent e);
  public void columnRemoved(TableColumnModelEvent e);
  public void columnSelectionChanged(ListSelectionEvent e);
}
```

Tabla 15.16. Composición de la clase TableColurnnModelEvent

Variables/Métodos	Descripción
TableColurnnModelEvent()	Construye un objeto TableColurnnModelEvent
getFromIndex()	Devuelve el tipo de columnas afectadas
getToIndex()	

Tabla 15.17. Cornposicion de la clase ChangeEvent

Variables/Métodos	Descripción
ChangeEvent()	Construye un objeto ChangeEvent

ListSelectionModel

Como ya se ha descrito anteriormente, el `ListSelectionModel` define las posibilidades de selección. La parte que es seleccionable depende de la que se encuentre disponible.

- Para controlar la selección de columnas, la Vista (`JTable`) lo hace a través del método `setColumnSElectionAllowed(boolean)`.
- Para la selección de filas, el método es `setRowSelectionAllowed(boolean)`.
- Para una selección a nivel de celdas, el método utilizado es `setCellSelectionEnable(boolean)`. Cuando la selección a nivel de celdas se encuentre disponible, el estado de las otras dos no cambiará y hará caso omiso de ellas.

Al igual que para las listas y los cuadros de lista, las tres constantes correspondientes a `ListSelectionModel` son importantes para las tablas: `SINGLE SELECTION`, `SINGLE_INTERVAL_SELECTION` o `MULTIPLE_INTERVAL_SELECTION`.

Para demostrar los efectos de los diferentes modelos, el siguiente método avanzará a través de los diferentes modos y modelos de selección.

```
JButton selectionType = new JButton("Next Type");
selectionType.addActionListener[new ActionListener() {
  public void actionPerformed(ActionEvent e) {
    boolean rowSet = jt.getRowSelectionAllowed();
    boolean colSet = jt.getColumnSelectionAllowed();
    boolean cellSet = jt.getCellSelectionEnabled();
    // aritmética binaria
    boolean setRow = !rowSet;
    boolean setCol = rowSet ^ colSet;
    boolean setCell = rowSet & colSet;
    // Determina la celda primero
    jt.setCellSelectionEnabled(setCell);
    jt.setColumnSelectionAllowed(setCol);
    jt.setRowSelectionAllowed(setRow);
    System.out.println("Row Selection Allowed? " + setRow);
    System.out.println("Column Selection Allowed? " + setCol);
    System.out.println("Cell Selection Enabled? " + setCell);
    jt.repaint();
  }
});
JButton selectionMode = new JButton("Next Mode");
selectionMode.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent e) {
    ListSelectionModel lsm = jt.getSelectionModel();
    int mode = lsm.getSelectionMode();
    int nextMode;
    String nextModeString;
    if (mode == ListSelectionModel.SINGLE_SELECTION) {
      nextMode = ListSelectionModel.SINGLE_INTERVAL_SELECTION;
      nextModeString = "Single Interval Selection";
    } else if (mode == ListSelectionModel.SINGLE_INTERVAL_SELECTION) {
      nextMode = ListSelectionModel.MULTIPLE_INTERVAL_SELECTION;
      nextModeString = "Multiple Interval Selection";
    } else {
      nextMode = ListSelectionModel.SINGLE_SELECTION;
      nextModeString = "Single Selection";
    }
    lsm.setSelectionMode(nextMode);
    System.out.println("Selection Mode: " + nextModeString);
```

```
    jt.repaint();
  }
});
```

TableCellRenderer

La interfaz `TableCellRenderer` describe cómo personalizar la visualización de una celda particular.

```
public interface TableCellRenderer {
  public Component getTableCellRendererComponent(JTable table,
    Object value, boolean isSelected, boolean hasFocus, int row,
    int col);
}
```

La interpretación de las celdas de la tabla funciona de un modo parecido al `ListCellRenderer`. Cualquier cosa que el objeto haya enviado a una columna en particular del modelo de datos, será reenviado al parámetro del método `getTableCellRendererComponenet()`.

Hay que tener en cuenta que el modelo no tiene el concepto de dónde se mostrará fisicamente la columna en la Vista. La columna 1 del modelo podría ser la 3 en la Vista.

A continuación se creará el código para dibujar la tercera columna de una manera personalizada. Se podrán hacer todas las celdas de un solo color y cambiar el borde cuando éstas esten seleccionadas.

```
class MyTableCellRenderer extends JLabel implements TableCellRenderer {
  final Border blueBorder = BorderFactory.createLineBorder(Color.BLUE);
  MyTableCellRenderer() {
    setOpaque(true);
  }
  public Component getTableCellRendererComponent(JTable table,
    Object value, boolean isSelected, boolean hasFocus, int row, int col) {
    setBackground((Color)value);
    if (isSelected)
      setBorder(blueBorder);
    else
      setBorder(BorderFactory.createEmptyBorder());
    return this;
  }
}
```

Cuando se instala el código para dibujar las celdas (renderer) se especifica la columna en la que se utilizará éste mediante el siguiente código:

```
TableColumnModel tcm = jt.getColumnModel();
TableColumn column = tcm.getColumn (tcm.getColumnCount()-1);
TableCellRenderer renderer = new MyTableCellRenderer();
column.setCellRenderer (renderer);
```

La figura 15.17 muestra la apariencia final de la tabla, y en el listado 15.8 se detalla todo el código del programa.

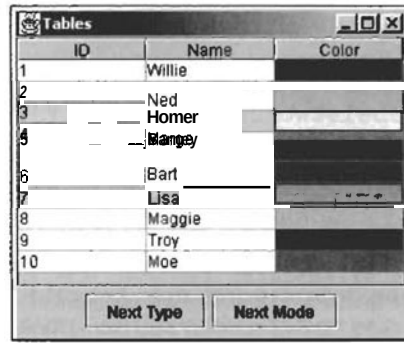


Figura 15.17. JTable con un renderer de celda personalizado

Listado 15.8. Personalización de la vista de un JTable

```
import java.awt.*;
import java.awt.event.*;
import java.util.*;
import javax.swing.*;
import javax.swing.border.*;
import javax.swing.event.*;
import javax.swing.table.*;
public class TableIt {
  class MyTableCellRenderer extends JLabel implements TableCellRenderer {
    final Border blueBorder = BorderFactory.createLineBorder(Color.BLUE);
    MyTableCellRenderer() {
      setOpaque(true);
    }
    public Component getTableCellRendererComponent(JTable table,
      Object value, boolean isSelected, boolean hasFocus, int row, int col)
!
      setBackground((Color)value);
      if (isSelected) {
        setBorder(blueBorder);
      } else {
        setBorder(BorderFactory.createEmptyBorder());

      return thi:;

  I
  class MyTableModel extends DefaultTableModel {
    Object data[][] = {
      {"Willie", Color.RED},
      {"Ned", Color.ORANGE},
      {"Homer", Color.YELLOW},
      {"Barney", Color.GREEN},
      {"Marge", Color.BLUE},
      {"Dart.", Color.MAGENTA},
      {"Lisa", Color.CYAN},
      {"Maggie", Color.PINK},
      {"Troy", Color.BLACK},
      {"Moe", Color.GRAY}
```

```
  };
  MyTableModel() {
    setColumnIdentifiers(new String[] {"ID", "Name", "Color"});
    for (int i=0,n=data.length;i<n;i++)
      addRow(new Object[] {new Integer(i+1), data[i][0], data[i][1]});

  public boolean isCellEditable(int row, int column) {
    return(column != 0);
  }
}
public TableIt() {
  JFrame f = new ExitableJFrame("Tables");
  TableModel tm = new MyTableModel();
  final JTable jt = new JTable(tm);
  TableColumnModel tcm = jt.getColumnModel();
  TableColumn column = tcm.getColumn(tcm.getColumnCount()-1);
  TableCellRenderer renderer = new MyTableCellRenderer();
  column.setCellRenderer(renderer);
  JButton selectionType = new JButton("Next Type");
  selectionType.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
      boolean rowSet = jt.getRowSelectionAllowed();
      boolean colSet = jt.getColumnSelectionAllowed();
      boolean cellSet = jt.getCellSelectionEnabled();
      // aritmética binaria
      boolean setRow = !rowSet;
      boolean setCol = rowSet ^ colSet;
      boolean setCell = rowSet & colSet;
      // Deterrnina la celda primero
      jt.setCellSelectionEnabled(setCell);
      jt.setColumnSelectionAllowed(setCol);
      jt.setRowSelectionAllowed(setRow);
      System.out.println("Row Selection Allowed? " + setRow);
      System.out.println("Column Selection Allowed? " + setCol);
      System.out.println("Cell Selection Enabled? " t setCell);
      jt.repaint();

  });
  JButton selectionMode = new JButton("Next Mode");
  selectionMode.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
      ListSelectionModel lsm = jt.getSelectionModel();
      int mode = lsm.getSelectionMode();
      int nextMode;
      String nextModeString;
      if (mode == ListSelectionModel.SINGLE_SELECTION) {
        nextMode = ListSelectionModel.SINGLE_INTERVAL_SELECTION;
        nextModeString = "Single Interval Selection";
      } else if (mode == ListSelectionModel.SINGLE_INTERVAL_SELECTION)
        nextMode = ListSelectionModel.MULTIPLE_INTERVAL_SELECTION;
        nextModeString = "Multiple Interval Selection";
      } else {
        nextMode = ListSelectionModel.SINGLE_SELECTION;
        nextModeString = "Single Selection";
      }
      lsm.setSelectionMode(nextMode);
```

```
        System.out.println("Selection Mode: " + nextModeString);
        jt.repaint();
      }
    });
    JPanel jp = new JPanel();
    jp.add(selectionType);
    jp.add(selectionMode);
    JScrollPane jsp = new JScrollPane(jt);
    Container c = f.getContentPane();
    c.add(jsp, BorderLayout.CENTER);
    c.add(jp, BorderLayout.SOUTH);
    f.setSize(300, 250);
    f.show();
  }
  public static void main(String args[]) {
    new TableIt();
  }
}
```

TableCellEditor

Si se esta ejecutando el programa del listado 15.8 y se hace doble clic en una celda que no se encuentra en la primera columna, se puede editar el contenido de esta celda. Esta acción también funciona con el nombre de la columna. Sin embargo, para la columna de color, falla y entonces es necesario instalar un `TableCellEditor` personalizado, definido de la manera siguiente:

```
public interface TableCellEditor extends CellEditor {
  public Component getTableCellEditorComponent(JTable table,
    Object value, boolean isSelected, int row, int column);
}
```

El editor de celda funciona de manera similar al renderer, aunque el editor se utiliza para modificar los contenidos. La mayor parte del trabajo de edición de celdas esta definido por la interfaz de `CellEditor`:

```
public interface CellEditor {
  public void addCellEditorListener(CellEditorListener l);
  public void cancelCellEditing();
  public Object getCellEditorValue();
  public boolean isCellEditable(EventObject e);
  public void removeCellEditorListener(CellEditorListener l);
  public boolean shouldSelectCell(EventObject e);
  public boolean stopCellEditing();
}
```

Por defecto, el editor para una tabla es el `DefaultCellEditor`, un campo de texto, que en el caso de un color no es apropiado, ya que no se puede convertir la cadena de texto en un objeto `color`. Debido a ello, debemos proporcionar nuestro propio editor. La clase `AbstractCellEditor` provee la estructura básica, implementando `CellEditor`. Esta clase gestiona la lista de `CellEditorListener`. Después, debemos implementar `TableCellEditor` para obtener el editor correcto. La composición del editor abstracto se muestra en la tabla 15.18.

Tabla **15.18.** Composición de la clase AbstractCellEditor

Variables/Métodos	Descripción
changeEvent	Evento a enviar cuando el editor cambia
listenerList	Contiene la lista de listener de eventos
`AbstractCellEditor()`	Construye un objeto AbstractCellEditor
`addCellEditorListener()`	Añade un listener al editor
`cancelCellEditing()`	Notifica al editor que la edición se ha cancelado o detenido
`stopCellEditing()`	
`fireEditingCanceled()`	Notifica los listener registrados
`fireEditingStopped()`	
`getCellEditorListeners()`	Devuelve el conjunto de listener actual
`isCellEditable()`	Comprueba si la celda es editable
`removeCellEditorListener()`	Elirnina el listener desde el editor
`shouldSelectCell()`	Hace que se seleccione la celda

Nota

Aunque `DefaultCellEditor` implementa la interfaz de `TableCellEditor`, solo funciona con un `JtextField` o un `JComboBox` como editor.

Según la definición de nuestro propio `TableCellEditor`, se puede hacer que un `JColorChooser` trabaje como nuestro editor de color. Gran parte de este trabajo se hace al conseguir que el editor se ejecute tras un doble clic. Debido a que la actual celda se cambia a un `JBotton`, el `JColorChooser` se crea para que aparezca fuera de la tabla. El listado 15.9 contiene la composición del editor de celda personalizado.

Listado 15.9. Visualización de un JColorChooser para un TableCellEditor

```
class MyColorChooser extends AbstractCellEditor
    implements TableCellEditor {
  JButton delegate = new JButton();
  final int CLICK_COUNT_TO_START = 2;
  public Object getCellEditorValue() {
    return delegate.getBackground();
  }
  public boolean isCellEditable(EventObject e) {
```

```
    boolean retValue = false;
    if (e instanceof MouseEvent) {
      if (((MouseEvent)e).getClickCount() >= CLICK-COUNT-TO-START) {
        delegate.requestFocus();
        retValue = true;


    return retValue;
  }
  public boolean shouldSelectCell(EventObject e) {
    boolean retValue = false;
    if (isCellEditable(e)) {
      if (e == null || ((MouseEvent)e).getClickCount()
          >= CLICK_COUNT_TO_START) {
        delegate.requestFocus();
        retValue = true;
      }
    }
    return retValue;
  }
  private void changeColor[Color c) {
    if (c != null) {
      delegate.setBackground(c);
    }
    delegate.setForeground((c == Color.BLACK) ?
      Color.WHITE : Color.BLACK);
  }
  public Component getTableCellEditorComponent(JTable table,
      Object value, boolean isSelected, int row, int column)
    changeColor((Color)value);
    delegate.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        Color color = JColorChooser.showDialog(delegate,
          "Color Chooser", delegate.getBackground());
        MyColorChooser.this.changeColor(color);
      }
    });
    return delegate;
  }
}
```

La instalación correspondiente a un `TableCellEditor` es muy parecida a la de un `TableCellRenderer`.

```
TableCellEditor editor = new MyColorChooser();
column.setCellEditor (editor);
```

Una vez instalado y al activar el editor se mostrard un `JColorChooser`, como el de la figura 15.11, que permitird al usuario cambiar colores.

Árboles

Swing proporciona un componente llamado `JTree` y que se utiliza para mostrar jerarquías. El modelo de datos para un drbol (tree) es el `TreeModel` y estd com-

puesto por muchos objetos `TreeNode`. El sentido de seleccionar nódulos dentro del árbol se define en `TreeSelectionModel`. `TreeCellRenderer` define cómo se muestra cada elemento del árbol. Todas las clases soporte de `JTree` se localizan en el paquete `javax.swing.tree`.

TreeModel

La interfaz `TreeModel` define la estructura de datos subyacente de un `JTree`.

```
public interface TreeModel {
  public void addTreeModelListener(TreeModelListener l);
  public Object getChild(Object parent, int index);
  public int getChildCount(Object parent);
  public int getIndexOfChild(Object, parent, Object child);
  public Object getRoot();
  public boolean isLeaf(Object node);
  public abstract void removeTreeModelListener(TreeModelListener l);
  public void valueForPathChanged(TreePath path, Object newValue);
}
```

Como ocurre con la mayoría de los otros componentes Swing, existe una implementación por defecto. En este caso es `DefaultTreeModel` que controla la notificación de listener cuando el modelo de datos cambia, como se muestra en la tabla 15.19.

Tabla **15.19.** Composición de la clase DefaultTreeModel

Variables/Métodos	Descripción
asksAllowsChildren	Indica si los nodulos hijo deben cuestionarse
listenerList	Contiene la lista de los listener de evento
root	Raiz del arbol
`DefaultTreeModel()`	Construye un objeto DefaultTreeModel
`addTreeModelListener()`	Añade un listener al arbol
`asksAllowsChildren()`	Indica como determinar los nodulos hijo
fireTreeNodesChanged()	Notifica a los listener de los carnbios en el rnodelo
fireTreeNodesInserted()	
fireTreeNodesRemoved()	
fireTreeStructureChanged()	
`getChild()`	Devuelve el nodulo hijo para el nodulo padre
`getChildCount()`	Devuelve el nurnero de hijos por nodulo
`getIndexOfChild()`	Devuelve la posición del hijo para el padre

Variables/Métodos	Descripción
`getListeners()`	Devuelve el conjunto de listener actual
`getTreeModelListeners()`	
`getPathToRoot()`	Devuelve la ruta desde la raiz al nodulo
`getRoot()`	Devuelve la raiz del arbol
`insertNodeInto()`	Aiiade un hijo al padre
`isLeaf()`	Comprueba si el nodulo tiene hijos
`nodeChanged()`	Hace que se notifique a los listener de cambios en el modelo
`nodesChanged()`	
`nodeStructureChanged()`	
`nodesWereInserted()`	
`nodesWereRemoved()`	
`reload()`	Hace que el modelo se cargue de nuevo
`removeNodeFromParent()`	Elimina el hijo del padre
`removeTreeModelListener()`	Elimina el listener del arbol
`setAsksAllowsChildren()`	Cambia la manera del arbol de buscar hijos
`setRoot()`	Cambia la raiz del arbol
`valueForPathChanged()`	Cambia el nodulo del árbol

Para conocer el momento en que cambia un modelo de árbol se registra un `TreeModelListener`, lo que permite obtener objetos `TreeModelEvent` (véase la tabla 15.20) cuando cambia la estructura del árbol.

```
public interface TreeModelListener extends EventListener {
  public void treeNodesChanged(TreeModelEvent e);
  public void treeNodesInserted(TreeModelEvent e);
  public void treeNodesRemoved(TreeModelEvent e);
  public void treeStructureChanged(TreeModelEvent e);
```

Tabla **15.20.** Composición de la clase TreeModelEvent

Variables/Métodos	Descripción
`childIndices`	Indica los hijos eliminados
`children`	Hijos eliminados

Variables/Métodos	Descripción
`path`	Ruta hasta la raiz del arbol
`TreeModelEvent()`	Construye un objeto TreeModelEvent
`getChildIndices()`	Devuelve un conjunto de indices de hijos
`getChildren()`	Devuelve un conjunto de nódulos de hijos de los indices
`getPath()`	
`getTreePath()`	Devuelve la ruta desde los padres de los objetos cambiados
`toString()`	Devuelve la representación en cadena del estado del componente

TreeNode

Las ~lasesmás utilizadas cuando se trabaja con `JTree` son los nódulos. Debido a la naturaleza jerárquica de los Brboles, se tienen que crear manualmente los nódulos y conectarlos entre sí antes de tener el Brbol completo. La base de los nódulos es la interfaz **`TreeNode`.**

```
public interface TreeNode {
  public Enumeration children();
  public boolean getAllowsChildren();
  public TreeNode getChildAt(int childIndex);
  public int getChildCount();
  public int getIndex(TreeNode node);
  public TreeNode getParent();
  public boolean isLeaf();
```

Se puede pensar en un **`TreeNode`** especifico como un Brbol de sólo lectura. No se puede modificar el nódulo utilizando tan sólo los métodos de esta interfaz. Para permitir a un nódulo de un Brbol tener hijos o para regenerarlos, se debe mover a **`MutableTreeNode`.** Esta interfaz incluye un soporte para añadir y eliminar hijos, asi como para cambiar los padres.

```
public interface MutableTreeNode extends TreeNode {
  public void insert(MutableTreeNode child, int index);
  public void remove(MutableTreeNode nude);
  public void remove(int index);
  public void removeFromParent();
  public void setParent(MutableTreeNode node);
  public void setUserObject(Qbject obj);
}
```

Puesto que tanto **`TreeNode`** como **`MutableTreeNode`** son interfaces, necesitamos una implementación concreta de éstas. Se trata de la clase `DefaultMuta-`

bleNode, que viene acompañada por un gran número de métodos, principalmente para árboles transversales. Estos métodos se detallan a continuación en la tabla 15.21.

Tabla 15.21. Composición de la clase DefaultMutableTreeNode

Variables/Métodos	Descripción
allowsChildren	Indica si los nodulos adrniten hijos
children	Hijos del nodulo
EMPTY—ENUMERATION	Conjunto vacio de hijos
parent	Padres del nodulo
userObject	Objeto de datos para el nodulo
DefaultMutableTreeNode()	Construye un objeto DefaultMutableTreeNode
add()	Añade un nodulo como hijo
insert()	
breadthFirstEnumeration()	Devuelve la enurneracion segun el orden de desplazarniento
depthFirstEnumeration()	
pathFromAncestorEnumeration()	
postorderEnumeration()	
preorderEnumeration()	
children()	Devuelve el conjunto de hijos por nodulo
clone()	Clona las propiedades del nódulo
getAllowsChildren()	Indica si el nódulo permite hijos
getChildAfter()	Devuelve el nodulo de los hijos
getChildAt()	
getChildBefore()	
getFirstChild()	
getLastChild()	
getChildCount()	Devuelve el número de hijos directos
getDepth()	Devuelve la cantidad de entramado por debajo del nodulo
getFirstLeaf()	Devuelve los nodulos hijo de ese nodulo
getLastLeaf()	

Variables/Métodos	Descripción
getNextLeaf()	
getPreviousLeaf()	
getIndex()	Devuelve la posición de un hijo concreto
getLeafCount()	Devuelve el nurnero de nodulos hijo por debajo
getLevel()	Devuelve la distancia hasta el nodulo raiz
getNextNode()	Devuelve el nodulo segun la orden previa de desplazamiento
getPreviousNode()	
getNextSibling()	Devuelve los hermanos del padre
getPreviousSibling()	
getParent()	Devuelve los padres
getPath()	Devuelve la ruta desde la raiz hasta ese nodulo
getPathToRoot()	Construye la ruta desde la raiz hasta ese nódulo
getRoot()	Devuelve la raiz del arbol
getSharedAncestor()	Devuelve el antepasado más cercano entre otro nódulo y este
getSiblingCount()	Devuelve le nurnero de herrnanos
getUserObject()	Devuelve el objeto de datos del nodulo
getUserObjectPath()	Devuelve la ruta de los objetos del usuario, desde la raiz hasta ese nodulo
isLeaf()	Cornprueba si los nódulos tienen hijos
isNodeAncestor()	Busca relaciones entre los nodulos
isNodeChild()	
isNodeDescendant()	
isNodeRelated()	
isNodeSibling()	
isRoot()	Cornprueba si el nodulo es la raiz del arbol
remove()	Elimina el hijo o hijos
removeAllChildren()	
removeFromParent()	Desconecta el nodulo de sus padres
setAllowsChildren()	Cambia si el nódulo adrnite hijos

Variables/Métodos	Descripción
`setParent()`	Cambia los padres
`setUserObject()`	Cambia el objeto de datos
`toString()`	Devuelve una representación en cadena del estado del componente

Para construir un Brbol se crean múltiples `DelfaultMutableTreeNode`. Por ejemplo, para construir un árbol de tipo calendario con los dias de la semana y los meses como hijos y donde cada uno espera tener más hijos, se tendrá que seguir un proceso como el que observa a continuación. Es una de las partes más pesadas de trabajar con `JTree`, sin embargo una vez que se ha construido el árbol ofrece muchas posibilidades.

```
DefaultMutableTreeNode root =
  new DefaultMutableTreeNode("Calendar");
DefaultMutableTreeNode months =
  new DefaultMutableTreeNode("Months");
root.add(months);
String monthLabels[] = {"January", "February", "March", "April", "May",
  "June", "July", "August", "September", "October", "November",
  "December"};
for (int i=0,n=monthLabels.length; i<n; i++)
  months.add(new DefaultMutableTreeNode(monthLabels[i]));
DefaultMutableTreeNode weeks =
  new DefaultMutableTreeNode("Weeks");
root.add(weeks);
String weekLabels[] = {"Monday", "Tuesday", "Wednesday", "Thursday",
  "Friday", "Saturday", "Sunday"};
for (int i=0,n=weekLabels.length; i<n; i++)
  weeks.add(new DefaultMutableTreeNode(weekLabels[i]));
JTree jt = new JTree(root);
JScrollPane jsp = new JScrollPane(jt);
```

TreeSelectionModel

El modelo de selección del árbol representa una de las tres constantes de `TreeSelectionModel` que son las siguientes: **`CONTINUOUS_TREE_SELECTION`**, **`SINGLE_TREE_SELECTION`** y **`DISCONTIGUOUS_TREE`**`_SELECTION`. El último es el valor por defecto.

La implementación real de `TreeSelectionModel` es `DefaultTreeSelectionModel`.

El `TreeSelectionModel` nos permite seguir la pista a los nódulos seleccionados cuando cambian. Se registra un **`TreeSelectionListener`** y se obtienen `TreeSelectionEvents`, como se detalla en la tabla 15.22.

Antes de registrar un listener de selección de árbol es necesario describir el objeto que se obtiene de los eventos de selección del Brbol: `TreePath`.

Tabla 15.22. Cornposicion de la clase TreeSelectionEvent

Variables/Métodos	Descripción
areNew	Devuelve los elernentos nuevos de la ruta
newLeadSelectionPath	Ruta de seleccion antes y despues del carnbio
oldLeadSelectionPath	
paths	Ruta actual del evento
`TreeSelectionEvent()`	Construye un objeto TreeSelectionEvent
`getNewLeadSelectionPath()`	Devuelve las rutas de seleccion nuevas y antiguas
`getOldLeadSelectionPath()`	
`getPath()`	Devuelve la ruta actual
`getPaths()`	Devuelve las rutas rnodificadas
`isAddedPath()`	Indica si se ha aiiadido una ruta

TreePath

Un `TreePath` representa un path o ruta entre nódulos. En el caso de un `TreeSelectionEvent`, es desde la raíz del árbol hasta el nódulo seleccionado. La tabla 15.23 muestra los posibles métodos de un `TreePath`.

Tabla 15.23. Cornposicion de la clase TreePath

Variables/Métodos	Descripción
`TreePath()`	Construye un objeto TreePath
`equals()`	Cornprueba si hay sernejanzas entre rutas
`getLastPathComponent()`	Devuelve el ultimo componente de la ruta
`getParentPath()`	Devuelve la ruta sin el nodulo actual
`getPath()`	Convierte la ruta en un array ordenado
`getPathComponent()`	Obtiene el cornponente en la posición dentro de la ruta
`getPathCount()`	Obtiene la extension de la ruta
`hashCode()`	Devuelve el codigo hash del arbol
`isDescendant()`	Cornprueba si hay otra ruta que surja de esta
`pathByAddingChild()`	Devuelve la ruta con el nodulo hijo añadido

Variables/Métodos	Descripción
`toString()`	Devuelve la representación en cadena del estado del componente

El siguiente `TreeSelectionListener` imprimirá la ruta desde la raiz hasta el nódulo seleccionado:

```
jt.addTreeSelectionListener(new TreeSelectionListener() {
  public void valueChanged(TreeSelectionEvent e) {
    TreePath path = e.getPath();
    System.out.println("Picked: " + path.getLastPathComponent());
    Object elements[] = path.getPath();
    for (int i=0, n=elements.length; i<n; i++) {
      System.out.print("->" + elements[i]);
    }
    System.out.println();

}});
```

Si se selecciona el martes (Thuesday) en el Brbol previamente creado, el resultado será el siguiente:

```
Picked: Tuesday
->Calendar->Weeks->Tuesday
```

Para expandir de una manera programada el drbol, sólo necesitamos transmitir el `TreePath` apropiado al método `setSelectionPath()` del `JTree`. El siguiente código, por ejemplo, abrirá el Brbol en Sunday (Domingo), que fue el último nódulo añadido:

```
DefaultMutableTreeNode lastLeaf = root.getLastLeaf();
TreePath path = new TreePath(lastLeaf.getPath());
jt.setSelectionPath(path);
```

TreeCellRenderer

Un `TreeCellRenderer` se usa para mostrar cada celda de un Brbol. Cuando se personaliza la visualización, normalmente se utiliza un `JLabel` y después se configura.

```
public interface TreeCellRenderer {
  public Component getTreeCellRendererComponent(JTree tree,
    Object value, boolean selected, boolean expanded, boolean leaf,
    int row, boolean hasFocus);
}
```

La figura 15.18 nos muestra un programa creado para dibujar celdas de Brbol personalizadas. El siguiente código es el correspondiente al renderer personalizado:

```
class MyCellRenderer extends JLabel implements TreeCellRenderer {
  MyCellRenderer() {
    setOpaque(true);
  }
  public Component getTreeCellRendererComponent(JTree tree,
      Object value, boolean selected, boolean expanded,
      boolean leaf, int row, boolean hasFocus) {
    StringBuffer text = new StringBuffer();
    if (expanded)
      text.append("E:");
    if (leaf)
      text.append("L:");
    if (hasFocus)
      text.append("H:");
    text.append(row + "->");
    text.append(value.toString());
    setBackground(selected ? Color.BLUE : Color.YELLOW);
    setForeground(selected ? Color.YELLOW : Color.BLUE);
    setText(text.toString());
    return this;
  }
}
```

Figura 15.18. Un JTree con un **TreeCellRenderer** personalizado

Para instalar el programa sólo hay que comunicarlo al `JTree`. Únicamente hay un programa para dibujar todo el árbol. Cualquier personalización se tendrá que hacer en el `TreeCellRenderer`.

```
jt.setCellRenderer(new MyCellRenderer());
```

El código completo para el ejemplo de `JTree` es el que se incluye en el listado 15.10:

Listado 15.10. Jerarquia de datos dentro de un JTree

```
import javax.swing.*;
import javax.swing.event.*;
import javax.swing.tree.*;
import java.awt.*;
```

```
import java.awt.event.*;
public class TreeIt {
  class MyCellRenderer extends JLabel implements TreeCellRenderer {
    MyCellRenderer() {
      setOpaque(true);
    }
    public Component getTreeCellRendererComponent(JTree tree,
        Object value, boolean selected, boolean expanded,
        boolean leaf, int row, boolean hasFocus) {
      StringBuffer text = new StringBuffer();
      if (expanded)
        text.append("E:");
      if (leaf)
        text.append("L:");
      if (hasFocus)
        text.append("H:");
      text.append(row + "->");
      text.append(value.toString());
      setBackground(selected ? Color.BLUE : Color.YELLOW);
      setForeground(selected ? Color.YELLOW : Color.BLUE);
      setText(text.toString());
      return this;
    }
  }
  public TreeIt() {
    JFrame f = new ExitableJFrame("Trees");
    DefaultMutableTreeNode root =
      new DefaultMutableTreeNode("Calendar");
    DefaultMutableTreeNode months =
      new DefaultMutableTreeNode("Months");
    root.add(months);
    String monthLabels[] = ["January", "February", "March", "April", "May",
      "June", "July", "August", "September", "October", "November",
       "December"];
    for (int i=0,n=monthLabels.length; i<n; i++)
      months.add(new DefaultMutableTreeNode(monthLabels[i]));
    DefaultMutableTreeNode weeks =
      new DefaultMutableTreeNode("Weeks");
    root.add(weeks);
    String weekLabels[] = {"Monday", "Tuesday", "Wednesday", "Thursday",
      "Friday", "Saturday", "Sunday"];
    for (int i=0,n=weekLabels.length; i<n; i++)
      weeks.add(new DefaultMutableTreeNode(weekLabels[i]));
    JTree jt = new JTree(root);
    jt.addTreeSelectionListener(new TreeSelectionListener() {
      public void valueChanged(TreeSelectionEvent e) {
        TreePath path = e.getPath();
        System.out.println("Picked: " + path.getLastPathComponent());
        Object elements[] = path.getPath();
        for (int i=0, n=elements.length; i<n; i++) {
          System.out.print("->" + elements[i]);
        }
        System.out.println();
      }
    });
    DefaultMutableTreeNode lastLeaf = root.getLastLeaf();
```

```
    TreePath path = new TreePath(lastLeaf.getPath());
    jt.setSelectionPath(path);
    jt.setCellRenderer(new MyCellRenderer());
    JScrollPane jsp = new JScrollPane(jt);
    Container c = f.getContentPane();
    c.add(jsp, BorderLayout.CENTER);
    f.setSize(250, 250);
    f.show();
  }
  public static void main(String args[]) {
    new TreeIt();
  }
}
```

Propiedades del cliente

El administrador de la interfaz Swing de usuario mantiene una lista de propiedades para los diferentes componentes, lo que permite alterar la apariencia de los componentes. Por ejemplo, cambiar las fuentes para la herramienta truco o mostrar líneas de conexión de nodulos en arboles sin subclasificar componentes. Para obtener la lista de propiedades solamente hay que preguntarle al `UIManager`:

```
impart java.util.*;
import javax.swing.*;
public class ListProps {
  public static void main(String args[]) {
    Hashtable defaultProps = UIManager.getDefaults();
    Enumeration enum = defaultProps.keys();
    while (enum.hasMoreElements()) {
      Object key = enum.nextElement();
      System.out.println("Property: " t key);
      System.out.println("Value: " + defaultProps.get(key) t '\n');

  }
```

El código anterior imprimira una lista bastante grande que se podrá examinar cuando se ejecute el programa. Para cambiar una propiedad especifica, se llamara al método `putClientProperty()` de un `JComponent` especifico. Por ejemplo para cambiar la linea de estilo de un `JTree` a angulado (angled), debemos invocar:

```
jt.putClientProperty("JTree.lineStyle", "Angled");
```

Este cambio resulta en la transformación de la figura 15.18 en la figura 15.19.

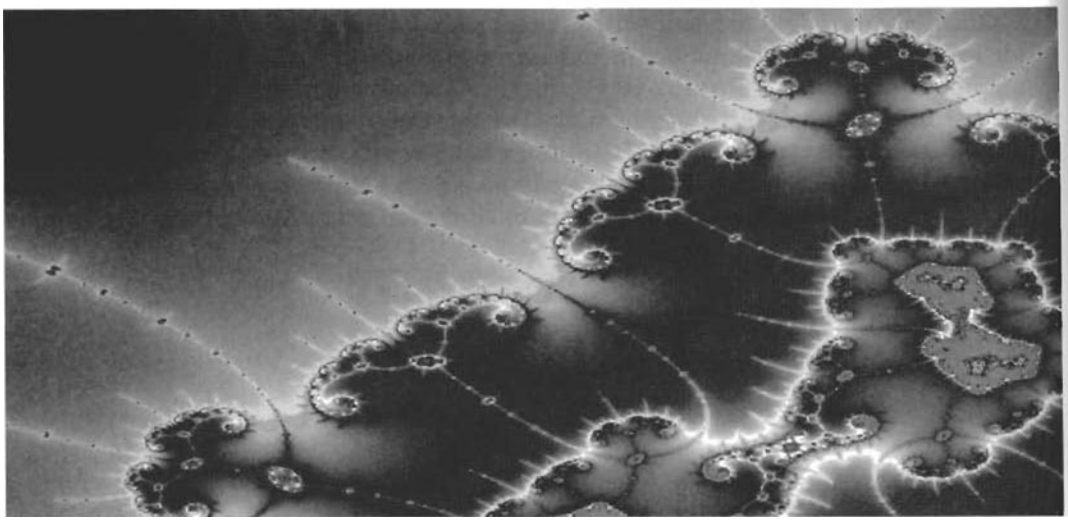
Resumen

En este capitulo hemos visto muchas partes de Swing que en realidad no tenemos que emplear. Se pueden utilizar los componentes de Swing simplemente como

componentes gráficos. Existen muchas maneras mejores, sin embargo, de utilizarlos y de conseguir que los programas puedan mantenerse mejor. El control de eventos se realiza mejor mediante Actions, y el control del teclado mediante Keystrokes, sin tener que controlar manualmente los datos que introduzca el usuario. Para no tener que crear cuadros de diálogo de manera manual, Swing proporciona muchos ejemplos de los cuadros de diálogo más comunes, tanto de mensajes, como los utilizados para introducir datos, colores o archivos.

Figura 15.19. JTree con el estilo de la linea modificado

Se suele pensar que Java es unicamente para el servidor, pero lo cierto es que se estd desarrollando también Java para el cliente. Las interfaces de implantación de cliente estdn ganando gran popularidad. Con el desarrollo de Java para el servidor, es conveniente conocer la arquitectura Modelo/Vista/Controlador ya que cada vez es mds importante poder separar el modelo de datos de la vista. Además las nuevas alternativas frontales, tales como los nuevos dispositivos inaldmbricos, requieren, cada vez mds, una completa separación.



16 Transferencia de datos

Una vez descritos los componentes de Swing, analizaremos en este capitulo otros aspectos del conjunto de paquetes de Java (JFC).Dos funciones adicionales de JFC, y además muy similares, sirven para transferir datos. La primera se encuentra en el paquete `java.awt.datatransfer` y ayuda a transferir datos al portapapeles. La segunda está en el paquete `java.awt.dnd` y ayuda a arrastrar y soltar datos tanto dentro de una aplicación como entre aplicaciones. Si incluimos estas funciones en nuestros programas, sobre todo para componentes Swing, la utilización de los mismos se facilitará enormemente.

Tanto `java.awt.datatransfer` como `java.awt.dnd` utilizan el mismo sistema para transferir datos (la interfaz `Transfearable`) que es la que se describirti en primer lugar, para después centrarnos en la manera de transferir texto mediante el portapapeles, cómo desplazar imtigenes y la técnica de arrastrar y soltar.

Objetos transferibles

Los objetos que son transferibles (desde o hacia el portapapeles de una aplicación) o que soportan la tarea de arrastrar y soltar, deben implementar la interfaz `java.awt.datatransfer.Transferable` que se muestra a continuación.

```
public interface Transferable {
  public Object getTransferData(DataFlavor flavor)
    throws UnsupportedFlavorException, IOException;
```

```
public DataFlavor[] getTransferDataFlavors();
public boolean isDataFlavorSupported(DataFlavor flavor);
}
```

Nota

El portapapeles es un area de la memoria utilizada para almacenar informacion temporalmente, que soporta las tareas de cortar, copiar y pegar. Por lo general, existen dos tipos de portapapeles: un portapapeles de sistema para compartir datos entre aplicaciones y un portapapeles privado para compartir datos dentro de una misma aplicacion. La tarea de arrastrar y soltar consiste en seleccionar un elemento determinado y, manteniendo el boton del ratón pulsado, arrastrarlo hasta otra posición dentro de la misma aplicacion o a una distinta.

Dataflavor

Cuando observamos la definición de la interfaz, la primera pregunta que puede surgir es ¿qué es un `DataFlavor`? Un flavor es la manera de representar datos cuando éstos son transferidos. Por medio de este método la clase que recibe el objeto transferible, el cual puede representarse de muchas maneras distintas, puede preguntar por un flavor conocido para así utilizar la forma más acorde con sus caracteristicas. Como se muestra en la tabla 16.1, existe una amplia gama de posibilidades para que un flavor describa la manera de representar los datos.

Tabla 16.1. Composición de la clase DataFlavor

Variables/Métodos	Descripción
`imageFlavor`	Flavor que representa un objeto `java.awt.Image`
`javaFileListFlavor`	Flavor que representa un objeto `java.util.List`
`javaJVMLocalObjectMimeType`	Tipo MIME que representa un objeto local
`javaRemoteObjectMimeType`	Tipo MIME que representa un objeto remote
`javaSerializedObjectMimeType`	Tipo MIME que representa un objeto serializado
`stringFlavor`	Flavor que representa un objeto `java.lang.String`

Variables/Métodos	Descripción
`DataFlavor()`	Construye un objeto `DataFlavor`
`clone()`	Crea una copia de la clase
`equals()`	Comprueba las semejanzas
`match()`	
`getDefaultRepresentationClass()`	Obtiene la clase por defecto
`getDefaultRepresentationClass-AsString()`	Obtiene la clase por defecto en forma de cadena
`getHumanPresentableName()`	Obtiene el nornbre que se va a mostrar al usuario
`getMimeType()`	Obtiene el tipo MIME
`getParameter()`	Obtiene el tipo MIME o el nombre legible para el usuario
`getPrimaryType()`	Obtiene el tipo MIME principal
`getReaderForText()`	Obtiene el interprete para leer el flavor como texto normal
`getRepresentationClass()`	Obtiene la clase por defecto
`getSubType()`	Obtiene el subtipo MIME
`getTextPlainUnicodeFlavor()`	Obtiene el flavor como texto normal de caracteres Unicode
`hashCode()`	Genera el codigo hash para la pulsación
`isFlavorJavaFileListType()`	Comprueba si el flavor es del tipo lista de archivos
`isFlavorRemoteObjectType()`	Cornprueba si el flavor es del tipo objetos rernotos
`isFlavorSerializedObjectType()`	Cornprueba si el flavor es del tipo objetos serializados
`isFlavorTextType()`	Comprueba si el flavor es un texto
`isMimeTypeEqual()`	Cornprueba si los dos tipos MIME son equivalentes
`isMimeTypeSerializedObject()`	Comprueba si el tipo MIME es para objetos serializados
`isRepresentationClassByteBuffer()`	Cornprueba si la clase es `java.nio.ByteBuffer`
`isRepresentationClassCharBuffer()`	Comprueba si la clase es `java.nio.CharBuffer`

Variables/Métodos	Descripción
`isRepresentationClassInputStream()`	Comprueba si la clase es `java.io.InputStream`
`isRepresentationClassReader()`	Comprueba si la clase es `java.io.Reader`
`isRepresentationClassRemote()`	Comprueba si la clase implementaj `ava.rmi.Remote`
`isRepresentationClassSerializable()`	Comprueba si la clase implementa `java.io.Serializable`
`readExternal()`	Restaura el flavor serializado
`selectBestTextFlavor()`	Elige el mejor flavor de texto
`setHumanPresentableName()`	Cambia el nombre de presentación
`toString()`	Convierte el flavor en un String
`tryToLoadClass()`	Intenta ejecutar la clase mediante diferentes cargadores
`writeExternal()`	Serializa el flavor

Nota

La clase por defecto es la manera por defecto en que se transfieren las clases, en forma de cadena de datos de entrada.

Afortunadamente `java.awt.datatransfer.DataFlavor` es una clase y no una interfaz, que proporciona acceso a los siguientes flavor y tipos MIME:

```
getTextPlainUnicodeFlavor()
stringFlavor
javaFileListFlavor
javaJVMLocalObjectMimeType,
javaRemoteObjectMimeType
javaSerializedObjectMimeType
```

Cuando se quiere que alguna información sea transferible, debemos decidir en qué flavor se van a hacer disponibles los datos. Estos flavor se gestionan internamente como tipos MIME (un estándar de Internet muy común) que se ayudan a su vez de la clase `MimeType`. Sin embargo, rara vez es necesario pensar en términos MIME, ya que se trata únicamente del nombre que Java utiliza para identificar un flavor. Se puede mostrar `DataFlavor` con un nombre legible mediante la función `getHumanPresetableName()`, aunque sería la única ocasión en que se podría ver algo como `application/x-java-serialized-object`, queresultaserel nombre legible de `javaSerializedObjectMimeType`.

Nota

Si no se esta familiarizado con los tipos MIME, estos estan definidos por los RFC 2045 y 2046, disponibles en http://www.imc.org/rfc2045 y http://www.imc.org/rfc2046. MIME corresponde a una extension de correo Internet multiproposito, sin embargo, su tecnologia va mas allá del ambito de los mensajes electronicos.

Un flavor funciona de la siguiente forma: cuando se tiene alguna informacion para transferir, se identifican los formatos en los que el objeto final puede recibir los datos. Por ejemplo, si la fuente de una transferencia es el contenido de un `JTextPane`, (que soporta texto con múltiples atributos) se deben utilizar tres flavor:

- Un flavor basado en `javaSerializedObjectMimeType` para preservar los atributos, para que de esta forma puedan ser añadidos a otro `JTextPane`.
- Un `stringFlavor`, para cadenas JAVA sin información de formato
- El flavor de texto normal de `getTExtPlainUnicodeFlavor()`, para cadenas Unicode, sin información de formato

Estas opciones permiten al objeto destino de la transferencia de datos preguntar al objeto `Transferable` si soporta un `DataFlavor` que él comprenda, ya sea por medio de `getTransferDataFlavors()` para obtener una lista de todos los flavor o `isDataFlavorSupported(DataFlavor)` para preguntar por uno especifico. El objeto destino de la transferencia comenzará con el modo más complejo que conozca, esperando retener tanta información como le sea posible. Si el objeto final fuera un `JtextArea`, el cual no soporta texto con múltiples atributos, preguntaría por un `stringFlavor`.

La clase StringSelection

Como el concepto de la transferencia de objetos `string` de Java es muy común, las bibliotecas de Java proporcionan como ayuda la clase `StringSelection`. Esta clase incluye un objeto `Transferable` que puede obtener el objeto final en forma de `stringFlavor`. Además de implementar la interfaz `Transferable`, la clase implementa también la interfaz `ClipboardOwner`, que se describe mas adelante.

Transferencia de datos mediante el portapapeles

Como se dijo antes, el portapapeles es una área de la memoria para almacenar temporalmente información y que soporta las tareas de cortar, copiar y pegar. Se

puede tener acceso al portapapeles del sistema preguntando al objeto `ToolKit` por medio de `getSystemCliboard()`.

También se pueden crear portapapeles privados, inaccesibles mds allá del ámbito del programa. La tabla 16.2 muestra la clase `Clipboard`, de gran sencillez, y define el alcance de sus posibilidades.

Tabla 16.2. Composición de la clase Clipboard

Variables/Métodos	Descripción
contents	El contenido transferible que se encuentra en el portapapeles
owner	El propietario del portapapeles
`Clipboard()`	Construye un objeto Clipboard
`getContents()`	Obtiene el contenido del portapapeles
`getName()`	Obtiene el nombre del portapapeles
`setContents()`	Cambia el contenido del portapapeles

Advertencia

Por defecto, el acceso al portapapeles del sisterna esta protegido por el adrninistrador de seguridad y no se permite éste desde applet.

Se puede nombrar al portapapeles desde el constructor `public Clipboard (String name)`, configurarlo u obtener su contenido. Cuando se coloca algo en el portapapeles no se copia hasta que alguien intenta obtenerlo, asi que la fuente debe estar segura de que el objeto estd disponible mientras este permanezca en el portapapeles.

Para saber cuándo un objeto situado en el portapapeles ha sido reemplazado, necesitamos la ayuda de la interfaz `ClipboardOwner`.

```
public interface ClipboardOwner {
  public abstract void lostOwnership(Clipboard clip, Transferable t);
}
```

La clase `StringSelection`, a la que ya se hizo mención anteriormente, realiza esta operación automdticamente cuando se están transfiriendo objetos `string`. Esta clase mantiene una referencia al `String` que se va a transferir.

Para otros tipos de objetos transferibles, habria que hacer el mismo seguimiento de manera manual.

En caso de que los datos copiados originalmente no se encuentren ya disponibles, entonces será necesario utilizar una `IOException`.

Transferencia de texto

Cuando se trabaja con los componentes Swing de texto, las operaciones de cortar, copiar y pegar se controlan automáticamente mediante el `EditorKid`. De este modo, normalmente no hay que preocuparse por crear código para acceder al portapapeles. Para demostrar las posibilidades de transferencia de texto, vamos a crear un programa que manualmente copia y pega texto mediante un `JTextArea`. La figura 16.1 muestra la pantalla de este programa ejemplo.

El listado 16.1 incluye el código de la aplicación de la figura 16.1. Tras ejecutar las operaciones internamente, pruebe a pegar texto desde aplicaciones externas, tales como Microsoft Word y no únicamente desde dentro del mismo programa.

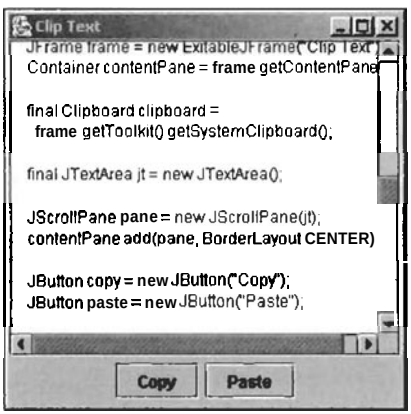


Figura 16.1. Programa de acceso al portapapeles

Listado 16.1. Comunicacion con el portapapeles del sistema

```
import java.awt.*;
import java.awt.event.*;
import java.awt.datatransfer.*;
import javax.swing.*;
import java.io.IOException;
public class ClipText {
  public static void main(String args[]) {
    JFrame frame = new ExitableJFrame("Clip Text");
    Container contentPane = frame.getContentPane();
    final Clipboard clipboard =
      frame.getToolkit().getSystemClipboard();
    final JTextArea jt = new JTextArea();
    JScrollPane pane = new JScrollPane(jt);
    contentPane.add(pane, BorderLayout.CENTER);
    JButton copy = new JButton("Copy");
    copy.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        String selection = jt.getSelectedText();
        StringSelection data = new StringSelection(selection);
```

```
        clipboard.setContents(data, data);

    });
    JButton paste = new JButton("Paste");
    paste.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent actionEvent) {
        Transferable clipData = clipboard.getContents(clipboard);
        if (clipData != null) {
          try {
            if (clipData.isDataFlavorSupported(DataFlavor.stringFlavor)) {
              String s = (String)(clipData.getTransferData(
                DataFlavor.stringFlavor));
              jt.replaceSelection(s);
            }
          } catch (UnsupportedFlavorException ufe) {
            System.err.println("Unsupported flavor: " + ufe);
          } catch (IOException ufe) {
            System.err.println("Unable to get data: " + ufe);
          }
        }

    });
    JPanel p = new JPanel();
    p.add(copy);
    p.add(paste);
    contentPane.add(p, BorderLayout.SOUTH);
    frame.setSize(300, 300);
    frame.show();
```

El código mas importante de este ejemplo es el de los métodos `actionPerformed()` de cada boton, que aparece en negrita.

- Para almacenar datos en el portapapeles, invocamos el método `setContests()` de `clipboard`, con el objeto `Transferable` y el `ClipboardOwner`, que en este caso, al utilizarse `StringSelection`, son el mismo.
- Para extraer datos del portapapeles, debemos extraer primero el objeto `Transferable` con `getContents()`, y utilizar después `DataFlavorSupported()` para asegurarnos de que el flavor deseado está disponible. Asumiendo que es asi, procederemos a obtener los datos de la interfaz `Transferable` con `TransferData()`, pasando el flavor elegido.

El EditorKit y las acciones de texto

Aunque el programa `ClipText` del listado anterior demuestra la manera de acceder al portapapeles, los métodos personalizados `ActionListener` no eran necesarios. Podemos sustituir las dos llamadas `addActionListener()` por algo mucho más sencillo. Cada componente Swing de texto utiliza un `javax.swing.text.EditorKit` para definir las acciones que pueden llevarse a cabo con el componente. La clase `javax.swing.text.DefaultEditorKit` se utiliza por

defecto para definir el comportamiento de las funciones de copiar y pegar, y los movimientos del cursor, entre muchas otras.

Las operaciones copiar y pegar son objetos especiales de `Action` llamados `TextAction`. Su funcionamiento se basa en encontrar el ultimo componente de texto con foco y en ejecutar en él la acción designada.

Para utilizar una `Action` asociada con un componente de texto, debemos primero obtener la lista de acciones desde el componente de texto y encontrar la acción concreta basada en pulsaciones de teclado proporcionadas por el `EditorKit`. Con ello, conseguimos un `ActionListener` que podremos asociar con un componente `Jbutton`, como ocurre en el ejemplo. Al activarse el componente, se procesa automáticamente el último componente de texto con foco.

En el siguiente ejemplo, vamos a sustituir los controladores de acción del programs `ClipText` por controladores integrados. El código que se muestra a continuación servirá para obtener los objetos `ActionListener` concretos. Esencialmente, se trata de crear una tabla con todos los controladores para después buscar los dos en los que estamos interesados.

```
import javax.swing.text.*;
...
// Obtiene la tabla de comandos
HashMap commands = new HashMap();
Action actions[] = jt.getActions();
for (int i = 0,n=actions.length; i < n; i++) {
  Action a = actions[i];
  commands.put(a.getValue(Action.NAME), a);
}
// Encuentra los comandos concretos
Action copyAction = (Action)commands.get(DefaultEditorKit.copyAction);
Action pasteAction = (Action)commands.get(DefaultEditorKit.pasteAction);
```

Una vez tengamos los controladores, sólo nos resta asignarlos al botón. Para ello, no es necesario acceder al portapapeles dentro del programa.

```
copy.addActionListener(copyAction);
paste.addActionListener(pasteAction);
```

La tabla 16.3 detalla las constantes de la clase `DefaultEditorKit` que proporcionan los nombres de algunas de las acciones existentes. Otras acciones estan disponibles a través de `StyledEditorKit`, y `JTextField`, que cuenta con una unica acción (`notifyAction`).

Tabla **16.3.** Constantes de la clase **DefaultEditor** para Actions

Variables	Descripción
backwardAction	Desplaza el cursor hacia atras
beepAction	Hace que el sistema emita un pitido

Variables	Descripción
beginLineAction	Desplaza el cursor al comienzo de la línea
beginParagraphAction	Desplaza el cursor al comienzo del parrafo
beginWordAction	Desplaza el cursor al cornienzo de la palabra
copyAction	Copia al portapapeles el contenido seleccionado
cutAction	Elimina el contenido seleccionado y lo almacena en el portapapeles
defaultKeyTypedAction	Comportamiento por defecto cuando no se ha definido ninguna otra acción rnediante una tecla
deleteNextCharAction	Borra el carácter siguiente
deletePrevCharAction	Elimina el caracter anterior
downAction	Desplaza el cursor hacia abajo
endAction	Desplaza el cursor al final del documento
endLineAction	Desplaza el cursor al final de la linea
endParagraphAction	Desplaza el cursor al final del parrafo
endWordAction	Desplaza el cursor al final de la palabra
forwardAction	Desplaza el cursor hacia adelante
insertBreakAction	Inserta un salto de linea
insertContentAction	Sustituye el contenido seleccionado por otro contenido
insertTabAction	Inserta un tabulador
nextWordAction	Desplaza el cursor a la siguiente palabra
pageDownAction	Desplaza el cursor una página hacia adelante
pageUpAction	Desplaza el cursor una pagina hacia atras
pasteAction	Sustituye el contenido seleccionado por el contenido alrnacenado en el portapapeles del sistema
previousWordAction	Desplaza el cursor a la palabra anterior
readOnlyAction	Cambia el area de trabajo a modo de solo lectura
selectAllAction	Seleccionado todo el contenido
selectionBackwardAction	Selecciona el elemento anterior

Variables	Descripción
selectionBeginAction	Selecciona el contenido al comienzo del documento
selectionBeginLineAction	Selecciona el contenido al comienzo de linea
selectionBeginParagraphAction	Selecciona el contenido al comienzo de parrafo
selectionBeginWordAction	Selecciona el contenido al comienzo de palabra
selectionDownAction	Selecciona el contenido desplazando el cursor hacia abajo
selectionEndAction	Selecciona el contenido al final del documento
selectionEndLineAction	Selecciona el contenido al final de linea
selectionEndParagraphAction	Selecciona el contenido al final de parrafo
selectionEndWordAction	Selecciona el contenido al final de palabra
selectionForwardAction	Selecciona el elemento siguiente
selectionNextWordAction	Selecciona el contenido al final de la palabra siguiente
selectionPreviousWordAction	Selecciona el contenido al principio de la palabra anterior
selectionUpAction	Selecciona el contenido desplazando el cursor hacia arriba
selectLineAction	Selecciona la linea actual
selectParagraphAction	Selecciona el parrafo actual
selectWordAction	Selecciona la palabra actual
upAction	Desplaza el cursor una linea hacia arriba
writableAction	Cambia el Area de trabajo a modo de lectura y escritura

Transferencia de imágenes

Una de las nuevas funciones introducidas en Java 1.4 es la capacidad de transferir imágenes a través del portapapeles. En realidad, esta tarea era posible antes, pero no con la facilidad y sencillez con la que se puede realizar ahora. Anteriormente, era necesario conocer el formato de la imagen que se iba a transferir y no era posible almacenar una imagen en el portapapeles para que otros usuarios la utilizaran, como se puede hacer ahora.

Para demostrar esta nueva función, vamos a modificar el programa `ClipText`, para que incluya un `JLabel` en lugar de un `TextArea`. En la etiqueta incluiremos un `ImageIcon`. Aunque la etiqueta se limita en principio a una imagen, es posible copiar y pegar imágenes del programa de Java a otra aplicación que soporte la tarea de copiar y pegar imágenes, y viceversa. En la figura 16.2, podemos ver la pantalla del programa `ClipImage`. Como imagen inicial, hemos utilizado el perro del capítulo 11.

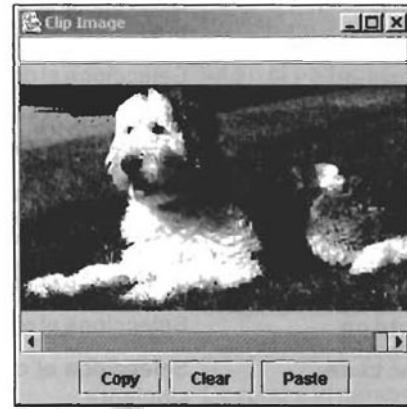


Figura 16.2. Programa de acceso al portapapeles

En lugar de incluir todo el codigo, examinemos algunos de sus componentes. Puesto que en esta ocasion la clase `ImageSelection` no corresponde a la clase `StringSelection`, debemos crear el objeto `Transferable` nosotros mismos. La interfaz `Transferable` es muy sencilla, ya que consta unicamente de tres métodos. La parte más dificil es trabajar con el portapapeles. La biblioteca Swing incluye una clase llamada `TransferHandler` que nos ayudará en el proceso. La clase `ImageSelection` se obtiene al implementar de manera estratégica los métodos de la subclase así como la interfaz `Transferable`. La tabla 16.4 describe los componentes de la clase `javax.swing.TransferHandler`.

Tabla **16.4.** Composición de la clase TransferHandler

Variables/Métodos	Descripción
COPY	Transfiere acciones
COPY—OR—MOVE	
MOVE	
NONE	
`TransferHandler()`	Construye un objeto TransferHandler

Variables/Métodos	Descripción
`canImport()`	Averigua si un componente aceptara un elemento importado de los flavor de datos
`createTransferable()`	Crea la fuente de transferencia de los datos
`exportAsDrag()`	Inicia el soporte de la funcion de arrastrar dentro de un componente Swing
`exportDone()`	Indica cuando ha finalizado el proceso de exportacion
`exportToClipboard()`	Transfiere el componente al portapapeles
`getCopyAction()`	Devuelve la acción de una operacion
`getCutAction()`	
`getPasteAction()`	
`getSourceActions()`	Obtiene el conjunto de acciones soportadas
`getVisualRepresentation()`	Obtiene una representación en forma de icono del objeto transferible
`importData()`	Inicia la transferencia de datos desde el portapapeles o mediante la operacion arrastrar y soltar

Para crear la clase `ImageSelection` del listado 16.2 debemos personalizar cuatro comportamientos de la clase `Transferhandler`, que se describen a continuación:

- `getSourceActions()`: Este método indica las acciones soportadas. Para el ejemplo del listado 16.3, la acción de copiar es la más adecuada, ya que de ese modo no se elimina la imagen de la ventana si alguien la pega en otro lugar.

```
public int getSourceActions(JComponent c) {
  return TransferHandler.COPY;
}
```

- `canImport()`: Este método se utiliza con un `JComponent` y una cadena de objetos `DataFlavor`. Para este programa en concreto, el componente debe ser un `JLabel`, y la cadena flavor debe compararse con el conjunto de los flavor soportados.

```
private static final DataFlavor flavors[] = {DataFlavor.imageFlavor};
...
public boolean canImport(JComponent comp, DataFlavor flavor[]) {
  if (!(comp instanceof JLabel)) {
    return false;
  }
  for (int i=0, n=flavor.length; i<n; i++) {
    for (int j=0, m=flavors.length; j<m; j++) {
```

```
        if (flavor[i].equals(flavors[j])) {
          return true;

      }
    }
    return false;
  }
```

- `createTransferable()`: Cuando utilizamos `TransferHandler` no estamos almacenando un elemento en el portapapeles, sino que definimos `createTransferable()` para que devuelva un objeto `Transferable` que mantenga una referencia del objeto que deseamos transferir al portapapeles o para poder aplicar la tarea arrastrar y soltar. Lo único que necesitamos hacer es extraer la imagen de la etiqueta y devolver `this`, para mantener la referencia de la imagen. El siguiente código corresponde a la operación de copiar.

```
private JLabel source;
private Image image;
...
public Transferable createTransferable(JComponent comp) {
  // Borra
  source = null;
  image = null;
  if (comp instanceof JLabel) {
    JLabel label = (JLabel)comp;
    Icon icon = label.getIcon();
    if (icon instanceof ImageIcon) {
      image = ((ImageIcon)icon).getImage();
      source = label;
      return this;
    }
  }
  return null;
}
```

- `importData()`: Este método, que corresponde a la operación de pegar, tiene dos parámetros: `Jcomponent` en el que se almacena la información del portapapeles. Debemos asegurarnos de que obtenemos el componente adecuado y que el flavor sea compatible con la interfaz `Transferable`. El siguiente paso consiste en obtener la imagen y asignarla a la etiqueta, asumiendo que dicha imagen tiene un formato que Java entiende, tal como GIF, JPEG, PNG, o XBM.

```
public boolean importData(JComponent cornp, Transferable t) {
  if (cornp instanceof JLabel) {
    JLabel label = (JLabel)comp;
    if (t.isDataFlavorSupported(flavors[0])) {
      try {
        image = (Image)t.getTransferData(flavors[0]);
        ImageIcon icon = new ImageIcon(image);
        label.setIcon(icon);
        return true;
      } catch (UnsupportedFlavorException ignored) {
```

```
            } catch (IOException ignored) {
            }
          }
        }
        return false;
      }
```

La clase ImageSelection

El codigo del listado **16.2 corresponde a la clase** `ImageSelection`, **a la cual se ha implementado el** método `Transferable`. **La** interfaz sólo **tiene que devolver el objeto** `Image` **obtenido mediante el** método `createTransferable()`.

Listado 16.2. Definición de la clase ImageSelection

```
import java.awt.*;
import java.awt.image.*;
import java.awt.datatransfer.*;
import java.io.*;
import javax.swing.*;
import javax.imageio.*;
import javax.imageio.stream.*;
public class ImageSelection extends TransferHandler
    implements Transferable {
  private static final DataFlavor flavors[] = {DataFlavor.imageFlavor};
  private JLabel source;
  private Image image;
  public int getSourceActions(JComponent c) {
    return TransferHandler.COPY;
  }
  public boolean canImport(JComponent comp, DataFlavor flavor[]) {
    if (!(comp instanceof JLabel)) {
      return false;
    }
    for (int i=0, n=flavor.length; i<n; it+) {
      for (int j=0, m=flavors.length; j<m; j++) {
        if (flavor[i].equals(flavors[j])) {
          return true;
        }
      }
    }
    return false;
  }
  public Transferable createTransferable(JComponent comp) {
    // Borra
    source = null;
    image = null;
    if (comp instanceof JLabel) {
      JLabel label = (JLabel)comp;
      Icon icon = label.getIcon();
      if (icon instanceof ImageIcon) {
        image = ((ImageIcon)icon).getImage();
        source = label;
        return this;
      }
```

```
    }
    return null;

  public boolean importData(JComponent comp, Transferable t) {
    if (comp instanceof JLabel) {
      JLabel label = (JLabel)comp;
      if (t.isDataFlavorSupported(flavors[0])) {
        try {
          image = (Image)t.getTransferData(flavors[0]);
          ImageIcon icon = new ImageIcon(image);
          label.setIcon(icon);
          return true;
        } catch (UnsupportedFlavorException ignored) {
        } catch (IOException ignored) {



    return false;

  // Transferible
  public Object getTransferData(DataFlavor flavor; {
    if (isDataFlavorSupported(flavor)) {
      return image;
    }
    return null;
  }
  public DataFlavor[] getTransferDataFlavors() {
    return flavors;
  }
  public boolean isDataFlavorSupported(DataFlavor flavor) {
    return flavor.equals(DataFlavor.imageFlavor);
  }
}
```

Una vez definida la clase `ImageSelection`, el programa principal (representado en el listado 16.3) es casi idéntico al programa `ClipText`, que hemos detallado anteriormente. La única diferencia es que los botones para copiar y pegar no aparecen en este caso.

Además, si el componente que ejecuta la acción es el mismo que el que se realiza, sólo tendriamos que pedir al `TransferHandler` las acciones de copiar y pegar.

Listado 16.3. Envio de imagenes mediante el portapapeles

```
import java.awt.*;
import java.awt.event.*;
import java.awt.datatransfer.*;
import javax.swing.*;
import java.io.IOException;
public class ClipImage {
  public static void main(String args[]) {
    JFrame frame = new ExitableJFrame("Clip Image");
    Container contentPane = frame.getContentPane();
    final Clipboard clipboard =
      frame.getToolkit().getSystemClipboard();
```

```
    Icon icon = new ImageIcon("jaeger.jpg");
    final JLabel label = new JLabel(icon);
    label.setTransferHandler(new ImageSelection());
    JScrollPane pane = new JScrollPane(label);
    contentPane.add(pane, BorderLayout.CENTER);
    JButton copy = new JButton("Copy");
    copy.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        TransferHandler handler = label.getTransferHandler();
        handler.exportToClipboard(label, clipboard, TransferHandler.COPY);
      }
    });
    JButton clear = new JButton("Clear");
    clear.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent actionEvent) {
        label.setIcon(null);
      }
    });
    JButton paste = new JButton("Paste");
    paste.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent actionEvent) {
        Transferable clipData = clipboard.getContents(clipboard);
        if (clipData != null) {
          if (clipData.isDataFlavorSupported(DataFlavor.imageFlavor)) {
            TransferHandler handler = label.getTransferHandler();
            handler.importData(label, clipData);
          }
        }
      }
    });
    JPanel p = new JPanel();
    p.add(copy);
    p.add(clear);
    p.add(paste);
    contentPane.add(p, BorderLayout.SOUTH);
    frame.setSize(300, 300);
    frame.show();
  }
}
```

Arrastrar y soltar

La tarea de arrastrar y soltar dentro de Java es similar a la de acceder al portapapeles, excepto por el proceso de mover los datos, que es un poco más complicado. Los datos que se quieren transferir siguen necesitando implementar la interfaz `Transferable`. Sin embargo, el comportamiento de los componentes Swing vienen determinados por defecto, por lo que ya disponemos de los más importantes y los que no tenemos, pueden añadirse de una manera muy sencilla.

Los siguientes componentes soportan la operación de arrastrar mediante el método `setDragEnabled(true)`: `JColorChooser`, `JFileChooser`, `JList`, `JTree`, `JTable`, así como todas las subclases de `JTextComponent`.

Para los demás componentes, debemos definir manualmente la manera de iniciar la tarea de arrastrar. Con la opción `MouseListener`, el controlador `ImageSelection` exporta la imagen mediante una operación de arrastre con un clic del ratón. Para soltar la imagen en una aplicación compatible con imágenes, tal como Microsoft Excel, debemos primero aiiadir el siguiente código después de `ImageSelection`:

```
MouseListener mouseListener = new MouseAdapter() {
  public void mousePressed(MouseEvent e) {
    JComponent comp = (JComponent)e.getSource();
    TransferHandler handler = comp.getTransferHandler();
    handler.exportAsDrag(comp, e, TransferHandler.COPY);
  }
};
label.addMouseListener(mouseListener);
```

Aunque este ejemplo utiliza el controlador personalizado `ImageSelection` creado anteriormente, no siempre es necesario crear uno. Para arrastrar una propiedad del componente que acompaiia `TransferHandler`, sólo debemos pasar el nombre de la propiedad al constructor `TransferHandler`. En el siguiente ejemplo, la etiqueta de texto del botón `Copiar` permitirá la acción de arrastre, con lo que el botón sólo podrá ser seleccionado desde el teclado. Además, dicho ejemplo le mostrará cómo crear un controlador de transferencia para la propiedad de un componente.

```
clear.setTransferHandler(new TransferHandler("text"));
mouseListener = new MouseAdapter() {
  public void mousePressed(MouseEvent e) {
    JComponent comp = (JComponent)e.getSource();
    TransferHandler handler = comp.getTransferHandler();
    handler.exportAsDrag(comp, e, TransferHandler.COPY);

};
clear.addMouseListener(mouseListener);
```

El listado 16.4 es una actualización del programa de `ClipImage` visto anteriormente, al que se le ha aiiadido el comportamiento de arrastre de imágenes, asi como un `JTextField` en la parte superior de la pantalla, que puede arrastrarse.

Listado 16.4. Adicion del cornportamiento de arrastre de irnagenes

```
import java.awt.*;
import java.awt.event.*;
import java.awt.datatransfer.*;
import javax.swing.*;
import java.io.IOException;
public class DragImage {
  public static void main(String args[]) {
    JFrame frame = new ExitableJFrame("Clip Image");
    Container contentPane = frame.getContentPane();
    final Clipboard clipboard =
```

```
    frame.getToolkit().getSystemClipboard();
Icon icon = new ImageIcon("jaeger.jpg");
final JLabel label = new JLabel(icon);
label.setTransferHandler(new ImageSelection());
MouseListener mouseListener = new MouseAdapter() {
  public void mousePressed(MouseEvent e) {
    JComponent comp = (JComponent)e.getSource();
    TransferHandler handler = comp.getTransferHandler();
    handler.exportAsDrag(comp, e, TransferHandler.COPY);
  }
};
label.addMouseListener(mouseListener);
JScrollPane pane = new JScrollPane(label);
contentPane.add(pane, BorderLayout.CENTER);
JButton copy = new JButton("Copy");
copy.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent e) {
    TransferHandler handler = label.getTransferHandler();
    handler.exportToClipboard(label, clipboard, TransferHandler.COPY);
  }
});
JButton clear = new JButton("Clear");
clear.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent actionEvent) {
    label.setIcon(null);
  }
});
clear.setTransferHandler(new TransferHandler("text"));
mouseListener = new MouseAdapter() {
  public void mousePressed(MouseEvent e) {
    JComponent comp = (JComponent)e.getSource();
    TransferHandler handler = comp.getTransferHandler();
    handler.exportAsDrag(comp, e, TransferHandler.COPY);
  }
};
clear.addMouseListener(mouseListener);
JButton paste = new JButton("Paste");
paste.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent actionEvent) {
    Transferable clipData = clipboard.getContents(clipboard);
    if (clipData != null) {
      if (clipData.isDataFlavorSupported(DataFlavor.imageFlavor)) {
        TransferHandler handler = label.getTransferHandler();
        handler.importData(label, clipData);
      }
    }
  }
});
JPanel p = new JPanel();
p.add(copy);
p.add(clear);
p.add(paste);
contentPane.add(p, BorderLayout.SOUTH);
JTextField tf = new JTextField();
tf.setDragEnabled(true);
contentPane.add(tf, BorderLayout.NORTH);
```

```
frame.setSize(300, 300);
frame.show();
```

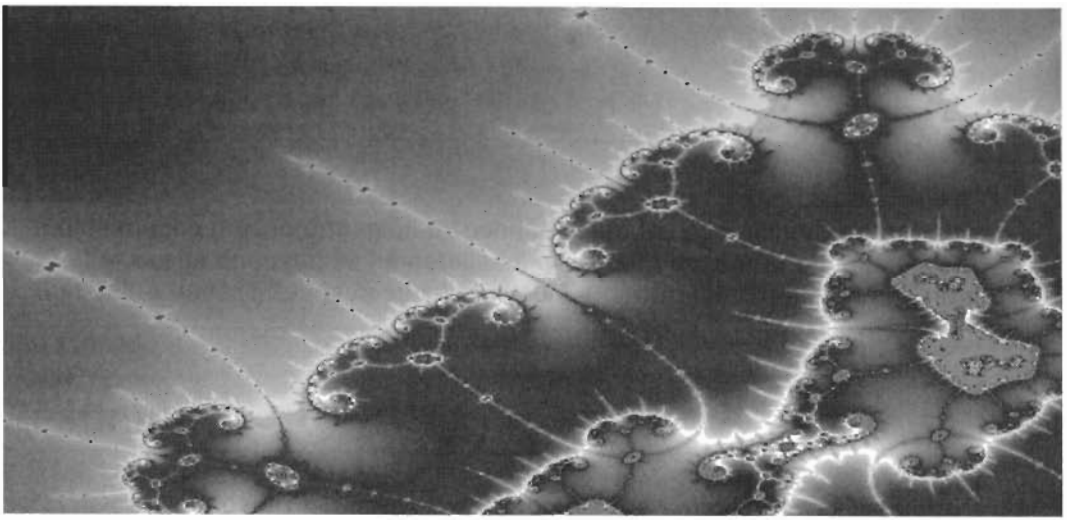
Nota

Existe mucha mas información disponible sobre las operaciones de arrastrar y soltar que la que aqui se ha detallado, que podra encontrar en la pagina de Sun sobre Java: http://java.sun.com/j2se/l.4/docs/guide/swing/1.4/dnd.html

Resumen

El mecanismo de arrastrar y soltar ha cambiado totalmente en la version 1.4 de Java 2, por lo que en lugar de actualizar la versión 1.3 es mejor utilizar la nueva version, al menos en cuanto a éste mecanismo se trata, ya que el nuevo `TransferHandler` simplifica enormemente el trabajo.

Esta version aun incluye el paquete `java.awt.dnd` para trabajar con `DragSource` y `DropTarget`, aunque ya no es necesario para operaciones sencillas de arrastrar y soltar. Pruebe a sustituir código antiguo por las nuevas clases `DragSourceAdapter` y `DropTargetAdapter`. La primera sirve de implementación de las interfaces `DraSourceListener` y `DragSourceMotionListener`, mientras que la segunda proporciona la estructura para `DropTargetListener`, y ambas hacen que el trabajo de programación sea más sencillo.



17
Utilidades de Java

Para convertirse en un buen programador de Java es necesario conocer todas las clases disponibles en la plataforma de Java 2. En este capitulo, repasaremos gran parte de los componentes del paquete `java.util` que nos ayudaran a incrementar nuestra producción y mejorar la eficacia de nuestros programas.

Comenzaremos describiendo las colecciones de Java y el soporte de estructura de datos que proporciona, para continuar con el soporte matemático disponible en las bibliotecas de Java, desde la trigonometria más básica y las funciones matemáticas de la clase `Math` (dentro del paquete `java.lang`), hasta el soporte aritmético de precision arbitrario de las clases `BigInteger` y `BigDecimal` (ambas, en el paquete `java.math`).

Tambien exploraremos el soporte de Java para trabajar con fechas y calendarios, mediante las clases `Date` y `Calendar`, respectivamente, ambas en el paquete `java.util`.

Otra de las opciones descritas en este capitulo es la capacidad de internacionalización y cambio de formato que necesitamos para mostrar adecuadamente fechas y números de diferentes paises. Por ejemplo, un usuario americano considerard la fecha 06/03/00 como el tres de Junio, mientras que algunos usuarios europeos la interpretará como el 6 de Marzo. Mediante la ayuda de la clase `java.text.DateFormat`, nos aseguraremos de que las fechas se muestran de la manera acorde al país destino. La clase `NumberFormat`, por su parte, nos ayudará en el trabajo con números, ya que nos permite mostrar simbolos de monedas locales, o formatear un número para que sólo tenga dos decimales. Desafortunadamente, depende de nosotros averiguar y calcular los cambios de moneda vigentes en el mercado.

El marco de las Colecciones de Java 2

Desarrollar en Java, requiere a menudo trabajar con una cantidad desconocida de objetos. Incluso cuando se trabaja con un numero fijo de objetos, los arrays no son siempre las mejores estructuras de almacenamiento. Como ayuda para trabajar con grupos de objetos, Java proporciona un juego completo de clases, de las cuales, unas 25, que se incluyen en el paquete `java.util`, constituyen el marco API de las colecciones de Java. En este capitulo repasaremos los antiguos contenedores API, asi como el nuevo marco de las colecciones. A pesar de que los nuevos programas evitan las clases antiguas, especialmente **`Hashtable`** y **`Vector`**, algunos continúan utilizándolas. Conocer su funcionamiento y sus limitaciones, nos ayudará a entender por qué se añadieron estas nuevas colecciones.

Nota

Si le gusta trabajar con el marco de colecciones de Java 2, puede utilizar algunas de ellas con Java 1.1. Puede descargar las bibliotecas de la pagina http://java.sun.com/products/javabeans/infobus/#collections, y acceder a un archivo con mas información en http://java.sun.com/products/javabeans/infobus/collectionsreadme.html.

Vectores, Stack y enumeración

Las clases de colecciones históricas son aquellas que existen desde el comienzo de la programacidn en Java. A pesar de que técnicamente no existe nada erróneo en utilizarlas, este uso deberia limitarse al mantenimiento de programas de Java 1.0 y 1.1. Dentro de estas clases, **`Vector`** y **`Stack`** son las dos principales que proporcionan un acceso ordenado. Por su parte, la interfaz **`Enumeration`** ofrece los medios para desplazarse entre sus contenidos.

Trabajando con vectores

Mientras que para los array es condición imprescindible que tengan un tamaño fijo y un tipo de datos homogéneos, la clase **`Vector`** engloba listas heterogéneas con un hibrido de array.

- Los vectores son heterogéneos porque no insisten en que cada elemento sea de un determinado tipo (se pueden mezclar dentro de un vector muchos tipos de objetos).
- Los vectores son un hibrido de array porque pueden crecer dinámicamente cuando se les añaden elementos.

La tabla 17.1 incluye las variables y métodos de la clase **`Vector`**.

Tabla **17.1.** Cornposicion de la clase Vector

Variables/Métodos	Descripción
Vector()	Construye un vector vacio
capacityIncrement	Incrernento del tamaño
elementCount	Elernentos dentro del vector
elementData	Bufer interno para los elementos
modCount	Comprueba las rnodificaciones que coincidan durante la repeticion (desde AbstractList)
add()	Añade un elernento al vector
addAll()	Añade una coleccion de elernentos al vector
addElement()	Añade un elemento al final del vector
capacity()	Devuelve la capacidad del bufer para el vector
clear()	Elimina todos los elernentos del vector
removeAllElements()	
clone()	Crea un clon del vector
contains()	Cornprueba si el vector contiene algún elernento
containsAll()	Comprueba si el vector contiene alguna coleccion
copyinto()	Copia los elernentos de un vector dentro de un array
elementAt()	Devuelve el elernento en la posición especificada
get()	
elements()	Devuelve un objeto que permite visitar cada elemento del vector
iterator()	
listIterator()	
ensureCapacity()	Asegura que la capacidad del buffer interno sea de un tamaño concreto
equals()	Busca elernentos iguales
firstElement()	Devuelve el primer elernento del vector
hashCode()	Devuelve el codigo hash cornputado
indexOf()	Busca elernentos desde el principio del vector
insertElementAt()	Inserta un elemento dentro del vector
isEmpty()	Cornprueba si el vector esta vacio

Variables/Métodos	Descripción
`lastElement()`	Devuelve el ultimo elernento del vector
`lastIndexOf()`	Busca un elernento desde el final del vector
`remove()`	Elirnina un elernento concreto del vector
`removeElement()`	
`removeAll()`	Elirnina una coleccion de elernentos del vector
`removeElementAt()`	Elirnina el elernento situado en la posición especificada
`removeRange()`	Elirnina un grupo de elernentos del vector
`retainAll()`	Elirnina del vector todos los elementos que no esten en otra coleccion
`set()`	Cambia el elernento situado en una posicion concreta del vector
`setElementAt()`	
`setSize()`	Cambia el tamaño del búfer interno
`size()`	Devuelve el número de elernentos del vector
`subList()`	Devuelve una parte del vector
`toArray()`	Devuelve los elementos del vector en forma de array
`toString()`	Convierte le contenido del vector en un String
`trimToSize()`	Reduce la capacidad del bufer interno al tamaño actual

Nota

Si se utilizan colecciones de clases anteriores a Java 1.2, se debe tener en cuenta que algunas han sido reinsertadas en las nuevas colecciones.

Debido a que los vectores pueden almacenar cualquier tipo de datos, todas las operaciones de vectores funcionan con la clase **`Object`**, lo que significa que se puede perder la seguridad de los tipos de datos que se tenia con un array. Cuando se accede a cualquier dato desde un vector, se tiene que convertir a su tipo original. Esto, junto con el coste asociado al crecimiento dindmico, son los inconvenientes de utilizar vectores.

Cuando trabajamos con arrays, se puede acceder a su tamaño mediante la variable **`length`** y asignar valores a los elementos con el operador de asignación (=). Para un vector, la longitud se obtiene con el método **`size()`** y se asigna por medio de **`set(int position, Object obj)`** o con el método `setElementAt(object obj, int position)`. Ni `set()` ni `setElemtnAt()` soportan crecimiento diná-

mico. Para que un vector pueda crecer sin limites, se utiliza tanto `add(Object obj)` como `addElement(Object obj)`, que añaden el objeto al final. La recuperación de elementos se realiza por medio de los métodos `get(int position)` o `getElement(int position)`.

Truco

Si se quiere almacenar un tipo de datos primitivo dentro de un vector o de cualquier otra coleccion que no sea un array, se necesitaran unas clases que se pueden encontrar en el paquete `java.lang`. Por ejemplo, para almacenar un `int`, se envuelve dentro de un `Integer`: `Integer in = new Integer(5)`. Posteriormente se puede añadir el objeto `Integer` dentro de la coleccion.

Para demostrar la clase `vector`, el programa del listado 17.1 usa algunas de sus operaciones básicas. Almacena los nombres de los planetas en un vector e imprime cada uno de ellos con una llamada implicita a `Object.toString()` hecha por `println()`.

Listado 17.1. Almacenaje y recuperación de elementos de un vector

```
import java.util.*;
public class ThePlanets {
  static class Planet {
    private String name;
    Planet(String s) {
      name = s;
    }
    public String toString() {
      return getClass().getName() + "[" t name + "]";

  }
  public static void main(String args[]) {
    String names[] = ("Mercury", "Venus", "Earth",
      "Mars", "Jupiter", "Saturn", "Uranus",
      "Neptune", "Pluto"};
    int namesLen = names.length;
    Vector planets = new Vector(namesLen);
    fur (int i=0; i < namesLen; i++) {
      planets.add(new Planet(names[i]));
    }
    int planetsLen = planets.size();
    for (int i=0; i < planetsLen; i++) {
      System.out.println(planets.get(i));
    }
  }
}
```

Por lo general, necesitaremos otras opciones además de las operaciones de vectores básicas de este listado. Seguramente necesitaremos hacer algo más con

cada elemento recibido que una simple operación heredada de un `Object`. El listado 17.2 extiende el programa anterior, añadiendo el número de lunas a la definición de cada planeta. En lugar de utilizar el método `toString()` para imprimir los datos de cada planeta, se utilizan los mktodos especificos de `Planet` para obtener información sobre los elementos del vector:

Listado 17.2. Asignacion de elernentos del vector

```
import java.util.*;
public class ThePlanetsAndMoons {
  static class Planet {
    private String name;
    private int moonCount;
    Planet(String s, int moons) {
      name = s;
      moonCount = moons;
    }
    public String toString() {
      return getClass().getName() t "[" t name t "-" + moonCount + "]";
    }
    public final String getName() {
      return name;
    }
    public final int getMoonCount() {
      return moonCount;

  }
  public static void main(String args[]) {
    String names[] = {"Mercury", "Venus", "Earth",
      "Mars", "Jupiter", "Saturn", "Uranus",
      "Neptune", "Pluto"};
    int moons[] = {0, 0, 1, 2, 16, 18, 21, 8, 1};
    int namesLen = names.length;
    Vector planets = new Vector(namesLen);
    for (int i=0; i < namesLen; itt) {
      planets.add(new Planet(names[i], moons[i]));

    Planet p;
    for (int i=0, n=planets.size(); i < n; i++) {
      p = (Planet)(planets.get(i));
      System.out.println(p.getName() t " : " t p.getMoonCount(i);
```

Para demostrar los problemas potenciales derivados de trabajar con vectores, el siguiente ejemplo define una clase nueva, `Comet`, y además añade un elemento al vector de los planetas utilizados anteriormente. Debido a que `vector` permite añadir cualquier elemento al vector `addElement()`, éste no provoca ningún problema.

Sin embargo, al asignar el valor recibido con `get()`, a un `Planet`, entonces se emite una `ClassCastException`:

```
static class Comet {
  private String name;
  Comet (String s) {
    name = s;
  }
  public String toString() {
    return getClass().getName() + "[" + name + "]";
  }
  public final String getName() {
    return name;

}
...
  planets.add(new Comet ("Hale-Bopp"));
...
  // Emisión de excepción para el cometa
  (Planet)(planets.get(i));
```

Este ejernplo demuestra que siempre se debe saber qué se va a obtener de una colección cuando se utiliza cualquiera de sus elernentos. Por supuesto, siemyre podemos emplear el operador `instanceof` para comprobar el tipo de datos del elemento recuperado.

Nota

Para hacer un recorrido multimedia online por el sistema solar puede visitar http://www.nineplanets.org/.

Tipos parametrizados

En C++, la posibilidad de especificar el uso de un tipo de datos concretos (para algo en tiempo de compilacion) se controla directamente en el lenguaje por medio de plantillas. Este concepto se llama *tipos parametrizados* (parametized types). Java no suministra ni tipos parametrizados ni colecciones de un tipo de datos especifico. Visite la pagina http://www.jcp.org/jsr/detail/14.jsp, para mas información sobre este tema.

Pushing Stack

La clase `stack` implementa una estructura de datos antigua pero a la vez útil y sencilla: la stack last-in-first-out (LIFO). En Java, la clase `Stack` es actualmente una subclase de `Vector`. Además de todas las operaciones de vectores, tarnbien podemos forzar objetos dentro del stack, extraerlos, observar el que está en la parte superior del stack, buscar uno determinado, así corno comprobar si el stack está vacio. La definicidn de la clase `Stack` de la tabla 17.2 rnuestra esta funcionalidad:

Tabla 17.2. Composición de la clase Stack

Variables/Métodos	Descripción
`Stack()`	Construye un stack vacio
`empty()`	Deterrnina si el stack esta vacio
`peek()`	Obtiene el elernento situado en la parte superior del stack
`pop()`	Elirnina el elernento situado en la parte superior del stack
`push()`	Añade un elernento en la parte superior del stack
`search()`	Cornprueba si hay algun elernento en el stack

El listado 17.3 representa una demostración simple de la clase **`Stack`**. Debe tenerse en cuenta que los métodos `Vector` se encuentran disponibles debido a que el stack es una subclase; sin embargo, se deberán evitar para conseguir una mayor claridad:

Listado 17.3. Forzar y extraer objetos del stack

```
import java.util.Stack;
public class JupiterMoons {
  public static void main(String args[]) {
    String names[] = {"Metis", "Adrastea", "Amalthea", "Thebe",
      "Io", "Europa", "Ganymede", "Callisto", "Leda", "Himalia",
      "Lysithea", "Elara", "Ananke", "Carme", "Pasiphae", "Sinope"
    };
    int namesLen = names.length;
    Stack moons = new Stack();
    for (int i=0; i < namesLen-1; i++) {
      moons.push(names[i]);
    }
    // Los métodos del vector todavía funcionan
    moons.add(names[namesLen-1]);
    while (!moons.empty()) {
      System.out.print(moons.pop() + " ");

    System.out.println();
  }
```

Al ejecutar el programa anterior, se genera el siguiente resultado:

```
Sinope Pasiphae Carme Ananke Elara Lysithea Himalia Leda Callisto Ganymede
Europa Io Thebe Amalthea Adrastea Metis
```

Desplazamiento por las enumeraciones

A pesar de que las clases `Vector` y `stack`, tienen un orden natural, hay veces en que realmente no nos preocupa el orden real de una colección, sino que todos los

nódulos se visiten. Tambikn puede ser que sólo importe la colección de objetos que tenemos y no la estructura de datos subyacente, lo que permite pensar en un nivel superior durante el disefio de nuestra aplicación.

Por ejemplo, puede que llegue el momento en que un vector sea ineficiente debido a que necesitemos afiadir elementos frecuentemente dentro de una colección. Si esto llegase a ocurrir, no deberiamos tener que cambiarlo todo unicamente por querer utilizar un `LinkedList` (quese encuentra en el marco de colecciones de Java) en lugar de un `Vector`. La interfaz antigua que proporciona este nivel de abstracción es la `Enumeration`, que se define a continuación:

```
public interface Enumeration {
  public boolean hasMoreElements();
  public Object nextElement();
```

La interfaz `Enumeration` requiere cualquier clase que contenga una enumeración para implementar dos mktodos: `hasMoreElements()` y `nextElement()`. Estos dos métodos permiten que cualquier objeto que soporte la enumeración de sus componentes pueda ser explorado utilizando el bucle `while`.

```
Enumeration enum = someContainer.methodReturningEnumeration();
while (enum.hasMoreElements()) {
  Object obj = enum.nextElement();
  // procesa este objeto
}
```

Cuando se trabaja con un `Vector`, podemos obtener un conjunto de elementos mediante el mktodo `elements()`. Debido a que `Stack` es una subclase de `Vector`, tambikn se utiliza aqui `elements()`. En lugar del código del ejemplo anterior

```
Planet p;
for (int i=0, n=planets.size(); i < n; i++) {
  p = (Planet)(planets.get(i));
  System.out.println (p.getName() + " : " + p.getMoonCount());
}
```

el siguiente código nos permitirá desplazarnos por la enumeración de los planetas:

```
Enumeration enum = planets.elements();
Planet p;
while (enum.hasMoreElements()) {
  p = (Planet)(enum.nextElement());
  System.out.println (p.getName() t " : " + p.getMoonCount());
}
```

Cuando deseemos crear tipos de colecciones propios necesitamos implementar dos mktodos de la interfaz `Enumeration`. El programa del listado 17.4 lo demuestra convirtiendo cualquier array en una `Enumeration`.

Para ello sólo debemos pasar el array a `ArrayEnumerationFactory.makeEnumeration()`.

Listado 17.4. Tratarniento de un array corno una enurneracion

```
import java.lang.reflect.Array;
import java.util.Enumeration;
final public class ArrayEnumerationFactory {
  static public Enumeration makeEnumeration(final Object obj) {
    Class type = obj.getClass();
    if (!type.isArray()) {
      throw new IllegalArgumentException(obj.getClass().toString());
    } else {
      return (new Enumeration() {
        int size = Array.getLength(obj);
        int cursor;
        public boolean hasMoreElements() {
          return (cursor<size);
        }
        public Object nextElement() {
          return Array.get(obj, cursor++);
        }
      });
    }
  }
  public static void main(String args[]) {
    Enumeration enum = makeEnumeration(args);
    while (enum.hasMoreElements()) {
      System.out.println(enum.nextElement());
    }
    enum = makeEnumeration(new int[] {1, 3, 3, 5});
    while (enum.hasMoreElements()) {
      System.out.println(enum.nextElement());
    }
    try {
      enum = makeEnumeration(new Double[Math.PI));
    } catch (IllegalArgumentException e) {
      System.err.println("Can't enumerate that: " t e.getMessage());
    }
  }
}
```

La clase incluye un ejemplo de un test ejecutado en el método `main()`. Si ejecutamos el programa con la siguiente entrada:

```
java ArrayEnumerationFactory one two three
```

provoca el siguiente resultado:

```
one
two
three
1
3
4
5
Can't enumerate that: class java.lang.Double
```

Diccionarios, tablas Hash y propiedades

Debido a que los arrays, vectores y stacks suministran un acceso ordenado a los contenidos de las colecciones manteniendo el orden, este proceso suele ser muy costoso. Para proporcionar una metodologia alternativa en lugares donde el orden o el acceso secuencial son innecesarios, Java facilita unas colecciones nuevas. En lugar de consultar los elementos mediante una posición integer, se suministra una clave para cada valor que se pudiera colocar en la colección (como por ejemplo, un mapa de consulta). La figura 17.1 muestra un ejemplo de este esquema de consulta.

La clase abstracta que soporta este concepto en los grupos de colecciones históricas es `Dictionary`. `Hasthtable` y `Properties` facilitan además dos implementaciones especificas de estas capacidades.

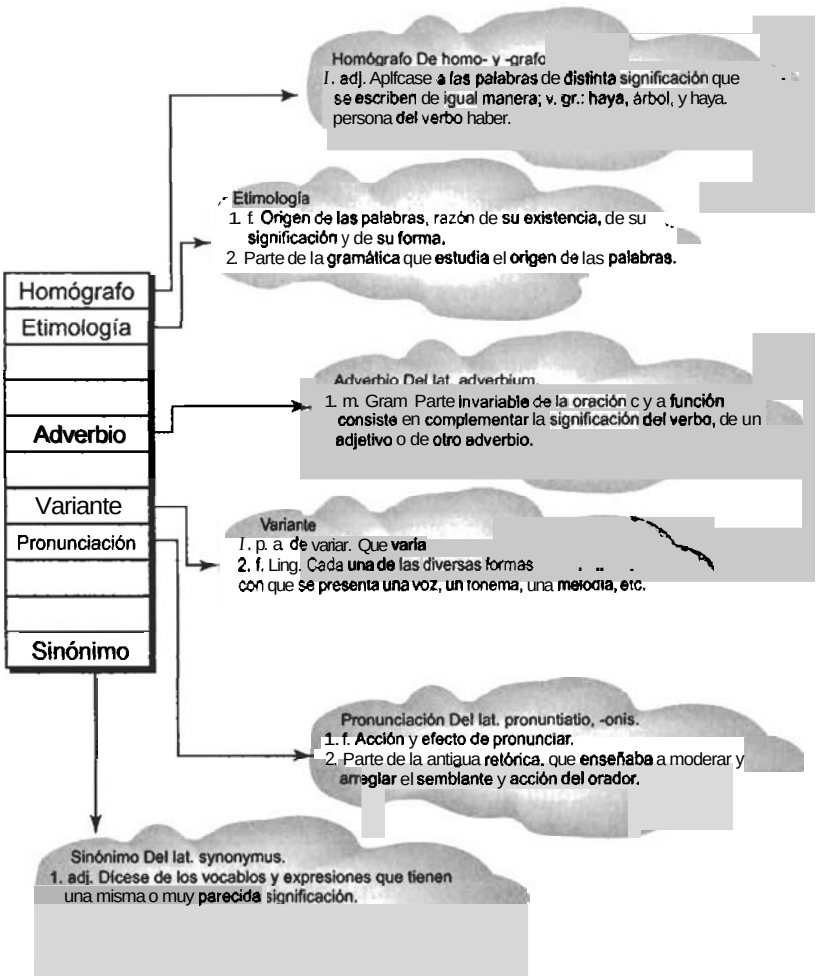


Figura 17.1. Ejemplo de consulta en el diccionario

Hojeando diccionarios

La clase abstracta **Dictionary** describe la interfaz para trabajar con mapas de valores clave. La manera con que trabaja un diccionario es la siguiente: Si se proporciona un valor como clave, el diccionario devuelve al menos, un objeto. Si la clave no esta en el mapa, no se obtiene nada, o **null** en términos de Java. El mismo objeto puede ser el valor para claves múltiples. Sin embargo, cada clave sólo puede corresponder con un valor. La manera de almacenar las correspondencias entre valores y claves depende de la implementación empleada. La tabla 17.3 incluye la definición correspondiente a la clase **Dictionary.**

Tabla **17.3.** Composición de la clase Dictionary

Variables/Métodos	Descripción
Dictionary()	Construye un diccionario vacio
elements()	Devuelve un objeto que permite visitar todos los valores del diccionario
get()	Devuelve el elemento situado en la posición especificada
isEmpty()	Determina si el diccionario esta vacio
keys()	Devuelve un objeto que permite visitar todas las claves del diccionario
put()	Introduce una pareja formada por un valor y una clave
remove()	Elimina una pareja formada por un valor y su clave del diccionario
size()	Devuelve el número de parejas en el diccionario

Nota

Tecnicamente hablando, Dictionary deberia ser una interfaz ya que todos los metodos son abstractos. Aparentemente, Dictionary precede al concepto de interfaces en Java, así que es tan solo una clase abstracta. Ya que Dictionary forma parte de las colecciones historicas, se deberia utilizar en su lugar la interfaz Map, del marco de las colecciones de Java, cuando se trabajaba con colecciones de valores clave.

Tablas Hash

La clase **Hashtable** representa una implementación concreta de la arquitectura de **Dictionary.** La tabla **17.4** muestra los métodos de esta clase.

Tabla **17.4.** Composición de la clase Hashtable

Variables/Métodos	Descripción
`Hashtable()`	Construye una tabla hash vacia
`clear()`	Elimina todas las entradas de una tabla hash
`clone()`	Crea un clon de la tabla
`contains()`	Determina si el objeto es un valor de la tabla hash
`containsValue()`	
`containsKey()`	Determina si el objeto es una clave de la tabla hash
`elements()`	Devuelve un objeto que permite visitar todos los valores de la tabla
`values()`	
`entrySet()`	Devuelve un objeto que permite visitar todas las entradas de la tabla
`equals()`	Busca elementos iguales
`get()`	Recupera el valor de una clave
`hashCode()`	Devuelve el codigo hash
`isEmpty()`	Comprueba si la tabla esta vacia
`keys()`	Devuelve un objeto que permite visitar todas las claves de la tabla
`keySet()`	
`put()`	Introduce una pareja formada por un valor y una clave
`putAll()`	Introduce una coleccion de parejas de valores y claves
`rehash()`	Se utiliza para incrementar la capacidad interna
`remove()`	Elimina un par de valores de la tabla hash
`size()`	Devuelve el numero de elementos de la tabla
`toString()`	Convierte el contenido de la tabla en un String

El nombre de esta clase se basa en el almacenamiento de claves. Cuando se le proporciona a la tabla una clave para un valor, la estructura de datos genera un código hash y lo utiliza como medio para consultar el valor cuando es necesario. Para generar el código hash de una clave necesitamos emplear el método `hashCode()`, tanto el correspondiente al `Object` como una versión anulada de una subclase. La tabla hash representa un medio eficaz de almacenar parejas de valores y claves ya que el algoritmo que genera el código hash asigna siempre el

mismo valor a la misma clave (dos claves son equivalentes si `equals()`, lo indica) y distribuye el código tan uniformemente como es posible.

Nota

Dos entradas no son necesariamente iguales porque `hashCode()` devuelva el mismo valor. De hecho, cuando muchas entradas tienen el mismo valor de codigo hash, es el resultado de un algoritmo de hash inapropiado. Si se ha anulado el método `hashCode()`, y se producen múltiples colisiones ,en las que muchas entradas tienen los mismos valores de codigo hash, se deberá rediseñar el mecanismo hash para la clase. Por ejemplo, la longitud de un `sting` es un algoritmo inapropiado para hash, ya que tendra como resultado en muchas cadenas con el mismo codigo. Además, al anular `equals()` para definir la igualdad de dos elementos, si ambos elementos son iguales deben generar el mismo codigo hash para que funcionen adecuadamente.

Las operaciones que se pueden realizar con una `HashTable` son heredadas en gran parte de `Dictionary` (o son parte de la nueva interfaz `Map`).

- `put(Object key, Object value)`: Para añadir un par de valores claves
- `get(Object key)`: Para obtener un valor para una clave
- `remove (Object key)`: Para eliminar un elemento
- `size()`: Para comprobar el tamaño
- `empty()`: Para comprobar si el valor es 0
- `keys()` o `element()`: Para obtener un conjunto de todas las claves o valores. Ambos devuelven una `Enumeration`.
- `keySet()` o `entrySet()`: Para obtener un valor de todas las claves o valores para utilizar con las nuevas colecciones API. Ambos devuelven un `Set`.

Adicionalmente, `Hashtable` permite vaciar la colección mediante `clear()`, comprobar si una clave especifica está presente mediante `containsKey(Object key)`, e incluso si un valor se encuentra presente utilizando para ello `containsValue(Object value)`. La última operación, aunque es posible, es muy costosa ya la tabla hash se utiliza al revés.

Truco

Si necesita descubrir con frecuencia si un valor concreto es una tabla hash, pruebe a mantener un mapa distinto con el orden inverso, aunque esto solo funciona si cada valor esta asociado a una unica clave, por lo que deberá conocer la clave correspondiente.

El listado 17.5, en el que se han empleado los nombres de los planetas como claves y su diámetro (en kilómetros) como los valores, demuestra la manera de utilizar Hashtable.

Listado 7.5. Parejas de valores y claves en una tabla Hash

```
import java.util.Enumeration;
import java.util.Hashtable;
public class PlanetDiameters {
  public static void main(String args[]) {
    String names[] = {"Mercury", "Venus", "Earth",
      "Mars", "Jupiter", "Saturn", "Uranus",
      "Neptune", "Pluto");
    float diameters[] = {4800f, 12103.6f, 12756.3f,
      6794f, 142984f, 120536f, 51118f, 49532f, 2274f};
    Hashtable hash = new Hashtable();
    for (int i=0, n=names.length; i < n; i++) {
      hash.put(names[i], new Float(diameters[i]));
    }
    Enumeration enum = hash.keys();
    Object obj;
    while (enum.hasMoreElements()) {
      obj = enum.nextElement();
      System.out.println(obj + ": " + hash.get(obj));
    }
  }
}
```

Al ejecutar el programa se producen los siguientes resultados:

```
Neptune: 49532.0
Pluto: 2274.0
Uranus: 51118.0
Jupiter: 142984.0
Mars: 6794.0
Venus: 12103.6
Earth: 12756.3
Mercury: 4800.0
Saturn: 120536.0
```

Se debe tener en cuenta que el orden resultante no tiene absolutamente ninguna correlación con el orden en que fueron añadidos los elementos. Al igual que ocurría con `Vector`, la rutina `get()` devuelve un `Object`. Si se necesita llamar al método de una clase especifica aiiadida a la tabla, se tendrá clue asignar el valor devuelto a la clase apropiada.

Nota

Se puede utilizar la clase `LinkedHashMap` para extraer los elementos de la tabla en el orden en el que se introdujeron. La sección sobre mapas que encontrara en este mismo capitulo incluye mas informacidn sobre esta clase.

Examen de propiedades

Cuando las claves y los valores almacenados en la tabla hash son string, se debe considerar utilizar la clase **Properties**. En lugar de tener que convertir los valores devueltos a la clase **String**, la clase **Properties** lo realiza automáticamente gracias a sus métodos de ayuda. La tabla 17.5 describe métodos adicionales para ejecutar y guardar las propiedades de un disco rfipidamente.

Tabla **17.5.** Composición de la clase Properties

Variables/Métodos	Descripción
Properties()	Construye una lista de propiedades vacia
getProperty()	Recupera el valor de una clave en la lista de propiedades
list()	Hace un listado de las propiedades y sus valores
load()	Abre la lista de propiedades del stream
propertyNames()	Devuelve un objeto que permite visitar todas las claves de la lista de propiedades
setProperty()	Añade una pareja formada por un valor y una clave a la lista de propiedades
store()	Guarda la lista de propiedades en el stream
defaults	Un grupo predefinido de valores de propiedades

Debido a que **Properties** es una subclase de **Dictionary**, podemos continuar añadiendo entradas con **put**() y devolviéndolas con **get**(). Sin embargo, estos métodos funcionan con valores **Object**. El modomfis natural para trabajar con la clase **Properties** es el siguiente:

- Aiiadir una pareja de valores y claves String mediante **set**Poperty(St**ring key, String value)**.
- Obtener el valor de una clave mediante getProperty(**String key**) o getProperty(**String key, String** defaultValue) cuando la clave no esté configurada.
- Obtener el conjunto de todas las claves mediante propertyNames() en lugar de **keys**(). **Properties** puede tener valores por defecto. Utilizando propertyNames() se crea una **Enumeration** que es la unión de las **keys**() y los valores por defecto.

También se pueden obtener propiedades desde un stream de entrada, tal como un archivo, o guardarlas, mediante **load**() y **store**() respectivamente. (El método **save**() fue rechazado ya que no devolvia una IOException cuando se produ-

cian errores al guardar.) La manera en que se especifican las propiedades en un archivo de texto es mediante un signo igual (=). Por ejemplo, si el archivo **planet.properties** contiene los siguientes valores:

```
Mercury=god of commerce, travel, and thievery
Venus=goddess of love and beauty
Earth=not derived from Greek/Roman mythology
Mars=god of War
Jupiter=King of the gods
Saturn=god of agriculture
Uranus=deity of the Heavens
Neptune=god of the Sea
Pluto=god of the underworld
```

se tendrian nueve propiedades para cada nombre de planeta. El siguiente programa abriria el archivo, añadiría una nueva propiedad para un décimo planeta ficticio, guardaria las propiedades en un archivo nuevo, haría una lista con todas las propiedades y valores en el dispositivo de salida estándar y finalmente ubicaria una propiedad especifica:

```
import java.io.*;
import java.util.Properties;
public class PlanetProperties {
  public static void main(String args[]) throws IOException {
    String header = "A header";
    Properties props = new Properties();
    props.load(new FileInputStream("planet.properties"));
    props.setProperty("Planet X", "god of dreams");
    props.store(new FileOutputStream ("save.the.planets"), header);
    props.list(System.out);
    System.out.println(props.getProperty ("Pluto"));
  }
}
```

La utilización más comun de **Properties** es a través de una lista de propiedades de sistema. Si se pregunta a la clase `System` por su lista de propiedades (mediante `getProperties()`), se puede encontrar información como la versión de Java utilizada, la persona que está utilizando el programa que se está ejecutando, el URL de dónde enviar informes sobre posibles problemas de una versión especifica de Java, e incluso en qué directorio se almacenan los archivos temporales. Para comprobar todas estas opciones, sólo tiene que añadir la siguiente línea de código al programa anterior.

```
System.getProperties().list(System.out);
```

Truco

Cuando trabajamos con un applet, y debido a restricciones de seguridad, no es posible obtener la lista de todas las propiedades; únicamente el valor de

propiedades concretas. En el archivo java.policy que se encuentra en el directorio jre\lib\security del directorio de instalacion SDK se detallan las propiedades a las que todo el mundo tiene acceso.

Trabajo con Preferencias

El concepto de las preferencias API, introducidas en Java 1.4 y parte del paquete `java.util.prefs`, es muy similar al de la clase `Properties`. Estas preferencias proporcionan mecanismos para almacenar y recuperar las preferencias del sistema y del usuario. Podemos almacenar información sobre configuración mediante las nuevas API, en lugar de utilizar `Properties`, como hasta ahora. Una de las mayores ventajas de trabajar con preferencias API es que no tenemos que preocuparnos sobre donde almacenar el archivo `Properties`.

Las preferencias se guardan dentro de un mecanismo especifico de implementacicin. Se almacenan en el Registro del sistema de Windows y dentro de un archivo, en el caso del sistema Unix.

Las preferencias API nos permiten realizar una gran variedad de operaciones, pero las dos fundamentales consisten en encontrar el ncidulo raiz con el que se va a trabajar y, en leer o escribir su valor. Existen dos jerarquias distintas de nódulos, una al nivel del usuario y otra para las preferencias del sistema. Los valores almacenados para cada nódulo pueden ser cadenas, valores booleanos, arrays de bytes o números (`float`, `long`, `int`, o `double`).

Truco

Podemos emplear el array de bytes para guardar el estado de los objetos Java mediante serialización, tema que se describe en el capitulo 19.

La clase `Preferences` es en realidad una serie de mktodos abstractos y estáticos. Una vez que obtenemos el nódulo raiz con un método estático, recibimos también una implementación concreta de la que podemos invocar los métodos, que han dejado de ser abstractos, para despuks obtener o establecer los valores para varios nodulos.

Existen cuatro maneras distintas de obtener un nódulo, dos para los nódulos del sistema y dos para los del usuario:

- Si estamos trabajando en un método no estático, podemos pasar una muestra del objeto del paquete en el que deseemos guardar las preferencias a los mktodos `userNodeForPackage()` y `systemNodeForPackage()`.

```
Preferences userPrefs = Preferences.userNodeForPackage(this);
Preferences sysPrefs = Preferences.systemNodeForPackage(this);
```

- Si por el contrario, estamos trabajando en un método estático como `main()`, debemos obtener primero el nodulo raiz y pasar despues el paquete en forma de cadena para desplazarse al nódulo apropiado.

```
Preferences userPrefs = Preferences.userRoot().node("/com/sybex");
Preferences sysPrefs = Preferences.systemRoot().node("/com/sybex");
```

  En este caso, y asumiendo que nuestra clase esté en el paquete `com.sybex`, las llamadas del nódulo del usuario devolverian un nódulo con el que trabajar, mientras que las llamadas del nódulo del sistema devolveria un nódulo separado. Utilizariamos la versión del sistema para los ajustes que deseáramos establecer para todos los usuarios de la aplicación en el equipo de un unico cliente.

Los nombres de los nodulos estdn limitados a la maxima longitud de `Preferences.MAX_NAME_LENGTH`, que es de 80 caracteres. Una vez tenemos el nódulo, podemos almacenar preferencias en su interior con el método `put()`. Existen además versiones especializadas del método para convertir tipos de datos en un String para almacenarlos en el nodulo `preferences`, y que se detallan a continuación. La clave de la preferencia está limitada a la longitud de `Preferences.MAX_KEY_LENGTH`, que es 80 caracteres. Los valores están limitados a `Preferences.MAX_VALUE_LENGTH`, es decir a 8.192 caracteres.

- `put(String key, String value)`
- `putBoolean(String key, boolean value)`
- `putByteArray(String key, byte value[])`
- `putDouble(String key, double value)`
- `putFloat(String key, float value)`
- `putInt(String key, int value)`
- `putLong(String key, long value)`

Para recuperar preferencias necesitamos el método `get()`, y como en el caso anteior, también contamos con métodos especializados para los diferentes tipos de datos soportados. Dichos métodos recuperan el valor de la clave desde el nódulo de preferencias. La unica diferencia entre las operaciones `get()` y `put()` es que la primera utiliza un valor por defecto en caso de que no se encuentre el nódulo o no esté disponible.

- `get(String key, String default)`
- `getBoolean(String key, boolean default)`
- `getByteArray(String key, byte default[])`
- `getDouble(String key, double default)`
- `getFloat(String key, float default)`

- `getInt(String key, int default)`
- `getLong(String key, long default)`

Ademds de obtener o establecer las preferencias, podemos realizar muchas otras tareas, tales como detectar cambios en las preferencias mediante los `listeners`: `addNOdeChangeListener()` o `addPreferenceChangeListener()`. También podemos encontrar una lista de claves dentro de un nódulo mediante `keys()`, o eliminar nódulos y valores mediante `clear()`, `remove()`, y `removeNode()`.

El **listado 17.6** demuestra las preferencias API al guardar el conjunto de planetas y sus lunas a un nódulo.

Listado 17.6. Guardando datos con las preferencias API

```
package MasteringJava.Chap17;
import java.util.prefs.*;
public class PlanetPrefs {
  public static void main(String args[]) {
    String names[] = !"Mercury", "Venus", "Earth",
      "Mars", "Jupiter", "Saturn", "Uranus",
      "Neptune", "Pluto"};
    int moons[] = {0, 0, 1, 2, 16, 18, 21, 8, 1};
    Preferences prefs =
      Preferences.userRoot().node("/MasteringJava/Chap17");
    for (int i=0, n=names.length; i < n; i++) {
      prefs.putInt(names[i], moons[i]);
    }
    try {
      String keys[] = prefs.keys();
      for (int i=0, n=keys.length; i < n; i++) {
        System.out.println(keys[i] + ": " + prefs.getInt(keys[i], 0));
      }
    } catch (BackingStoreException e) {
      System.err.println("Unable to read backing store: " + e);
    }
  }
}
```

Nota

Este programa está en un paquete. Asegúrese de compilarlo con la opción `-d`, como en **`javac -d . PlanetPrefs.java`**. Tambien puede mover el archivo `.class` al subdirectorio apropiado (`MasteringJava\Chap17`) antes de ejecutarlo. Despues, ejecute el programa con el comando **`java MasteringJava.Chap17.PlanetPrefs`**.

Ademds de añadir o buscar preferencias, podemos utilizar un `NodeChangeListener` o un `PreferenceChangeListener` a un nódulo para detectar cambios en los datos. El `NodeChangeListener` nos notificard cudndo se añaden o

eliminan nódulos y el PreferenceChangeListener cuándo se producen cambios en los valores. El NodeChangeListener proporciona dos métodos de implementación para detectar cambios. Su variable, NodeChangeEvent, nos permite averiguar el nódulo y el padre de el punto concreto de interés, mediante getChild() y getParent():

```
public interface NodeChangeListener extends EventListener {
  public void childAdded(NodeChangeEvent evt);
  public void childRemoved(NodeChangeEvent evt);
```

El PreferenceChangeListener, por su parte, sólo dispone de un método de detección, aunque el PreferenceChangeEvent nos permite obtener la clave del evento (getKey()), el nuevo valor (getNewValue()), o el nódulo (getNode()).

```
public interface PreferenceChangeListener extends EventListener {
  public void preferenceChange(PreferenceChangeEvent evt);
}
```

Nota

En las plataformas de Windows, las entradas del usuario estan almacenadas en el directorio HKEY_CURRENT_USER\Software\JavaSoft\Prefs del Registro y las preferencias del sistema en HKEY_LOCAL_MACHINE.

BitSet

La clase BitSet (véase la tabla 17.6) implementa un conjunto de bits o bien un vector de valores booleanos. A diferencia de muchas otras clases de bits o de caracteristicas de otros lenguajes, esta clase no tiene limites de tamaño. Se puede ir por tanto más all6 de los limites de 32 6 256 bits impuestos por otras implementaciones.

Nota

Aunque la clase BitSet forma parte de las colecciones historicas de Java, el nuevo marco de colecciones no cuenta con otra clase que pueda reemplazarla.

Tabla 17.6. Cornposicion de la clase BitSet

Variables/Métodos	Descripción
BitSet()	Construye un BitSet con todos los bits elirninados
and()	Cornbina éste y otro grupo con la operación lógica AND

Variables/Métodos	Descripción
andNot()	Realiza la operacion lógica NOT de otro bitset, y despues combina este y otro grupo de bits mediante AND
cardinality()	Cuenta el numero de bits
clear()	Elimina un bit especifico del bitset
clone()	Crea un clon del bitset
equals()	Cornprueba la igualdad entre dos objetos
flip O	Conrnuta un bit
get()	Obtiene los ajustes de un bit especifico del bitset
hashCode()	Devuelve el codigo hash cornputado
intersects()	Comprueba las semejanzas entre unos bits y los de otro bitset
isEmpty()	Comprueba si hay bits en el bitset
length()	Devuelve el tamaño lógico del bitset
nextClearBit()	Devuelve el índice del siguiente bit que no pertenezca a un bitset
nextSetBit()	Devuelve el índice del siguiente bit en un bitset
or()	Combina este conjunto de bits con otro mediante la operacion OR
set()	Ajusta un bit concreto del bitset
size()	Devuelve el espacio interno utilizado para representar los bits
toString()	Convierte el contenido del grupo de bits en String
xor()	Cornbina este grupo de bits con otro rnediante la operacion lógica XOR

Las operaciones de `BitSet` tienen las siguientes funciones:

- Dar valores, eliminar u obtener bits individuales.
- Operaciones de AND, NOT, OR y XOR con bits.
- Comparación de bits.

El programa representado en el listado 17.7 demuestra estas operaciones, colocando un `BitSet` que indica si el planeta tiene o no un número determinado de lunas. Normalmente, se almacenará un grupo mayor de bits en un `BitSet` para ahorrar de este modo espacio.

Listado 17.7. Manipulando el BitSet

```
import java.util.*;
public class TwoBitPlanets {
  public static void main (String args[]) {
    String names[] = {"Mercury", "Venus", "Earth",
      "Mars", "Jupiter", "Saturn", "Uranus",
      "Neptune", "Pluto"};
    int moons[] = {0, 0, 1, 2, 16, 18, 17, 8, 1};
    int namesLen = names.length;
    BitSet bits = new BitSet(namesLen);
    for (int i=0; i < namesLen; i++) {
      if ((moons[i] % 2) == 0) {
        bits.set (i);
      }
    }
    for (int i=0; i < namesLen; i++) {
      System.out.println (names[i] + " Even # Moons ("
        + moons[i]+ ")? " + bits.get(i));     }
  }
```

Truco

El método `size()` de `BitSet` devuelve el tamaño de la estructura interna utilizada, aunque no necesariamente el numero maximo de bits utilizado. Debido a que los valores de bit son almacenados con la ayuda de un `double`, `size()` salta a 64 cuando es necesario. Normalmente se utilizará el método `length()` que devuelve la posición de los bits mas elevados.

Collections e Iterators

Todas las clases de Java se revisaron completamente en la plataforma de Java 2. Introducidas inicialmente por la versión 1.2, el marco de colecciones de Java ha cambiado con cada nueva versión. Dicho marco se compone ahora de unas 25 clases e interfaces que se encuentran en el paquete `java.util` y que ofrecen una compleja arquitectura para manipular grupos de datos. El marco básico se basa en una serie de interfaces que describen las capacidades soportadas de las implementaciones. La jerarquia de dichas interfaces se muestra en la figura 17.2, y la de las implementaciones de las interfaces en la figura 17.3. Como se puede ver en la figura 17.2, las cuatro interfaces `Collection`, `List`, Map y `Set` describen los diferentes tipos de colecciones. Para cada tipo de colección, existen implementaciones especificas dependiendo del tipo de estructura de datos usada para almacenar la colección. Dichas implementaciones son las clases `ArrayList`, `TreeSet` y `HashMap`. El resto de las clases e interfaces del marco son de ayuda. Por ejemplo, la interfaz `Iterator` sustituye a la interfaz `Enumeration` que se utilizaba en las clases históricas, y Map. `Entry` representa una entrada en un Map.

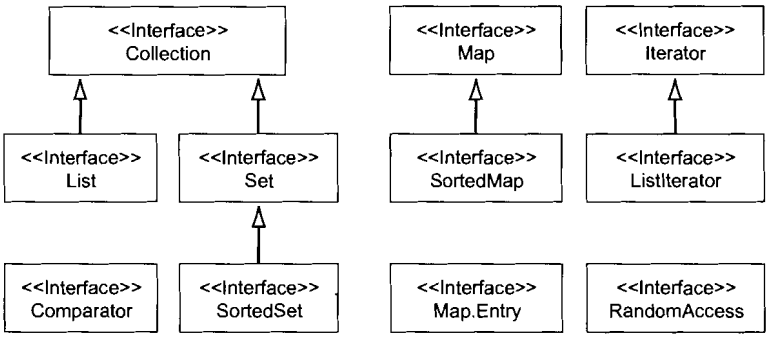


Figura 17.2. Jerarquia de las interfaces del marco de colecciones

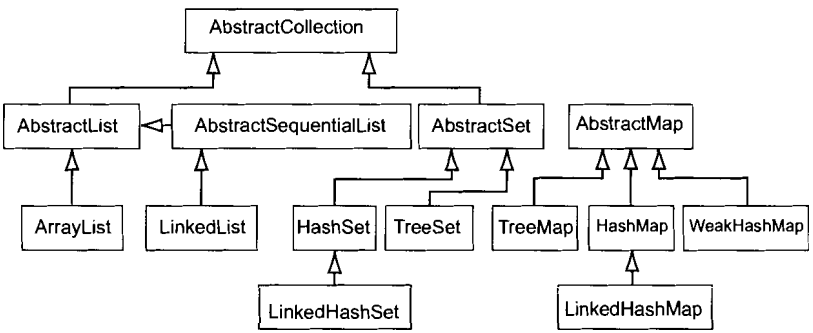


Figura 17.3. Jerarquia de las clases del marco de colecciones

Cuando se trabaja con las diferentes clases, se tienen que crear siempre implementaciones especificas de las estructuras de los datos, debido a que el sistema necesita conocer la manera de almacenar los objetos dentro de la colección. Sin embargo, cuando se accede a la colección actual se deberan utilizar los metodos de la interfaz, ya que con ello, se podrá cambiar la estructura de datos, por ejemplo, desde una colección basada en arrays a una basada en una lista conectada, si las circunstancias lo requieren. Posteriormente, debido a que la clase `collection` continua implementando la misma interfaz `collection`, no se necesita cambiar nada más del código. Por ejemplo, el siguiente código muestra como crear un `HashMarp`:

```
Map phoneBook = new HashMap();
```

Si posteriormente se decide mantener un listín de teléfonos ordenado se puede cambiar la estructura de los datos de un `HashMap` a un `TreeMap`:

```
Map phoneBook = new TreeMap();
```

Como Gnicamente se accede a la variable `phoneBook` a través de la interfaz `Map`, el resto del código permanecerá sin cambios, y el listín de teléfonos estará siempre organizado.

De las cuatro interfaces primarias, `collection` y `Map` son las bbsicas. Una `collection` es cualquier grupo de objetos, mientras que `Map` es similar a `Dictionary` y funciona con parejas de valores y claves. Dos interfaces extienden la interfaz `Collection`: `Set` y `List`. `Set` se comporta como su definición matembtica, no duplica; mientras que `List` mantiene una secuencia u orden similar al comportamiento de `Vector`.

Las interfaces `SortedMap` y`SortedSet` ofrecen un desplazamiento ordenado en los elementos de `Map` y `Set`, respectivamente.

Para comenzar con la nueva API, observemos la definición de la interfaz `Collection`:

```
public interface Collection {
  public boolean add(Object obj);
  public boolean addAll(Collection c);
  public void clear();
  public boolean contains(Object obj);
  public boolean containsAll(Collection c);
  public boolean equals(Object obj);
  public int hashCode();
  public boolean isEmpty();
  public Iterator iterator();
  public boolean remove(Object obj);
  public boolean removeAll(Collection c);
  public boolean retainAll(Collection c);
  public int size();
  public Object[] toArray();
  public Object[] toArray(Object obj[]);
}
```

Las operaciones de `Collection` consisten en lo siguiente:

- Añadir elementos.
- Eliminar elementos.
- Comprobar contenidos.
- Trabajar con los elementos.

La mayoría de las operaciones se explican por si solas. Sin embargo, hay un par de ellas que requieren una pequeña explicación.

- `retainAll()`: El método `retainAll()` combina dos colecciones al realizar la operación de intersección. La colección original se modifica para incluir cualquier elemento en el parbmetro `collection` que no es todavia miembro de ésta.
- `toArray()`: El método `toArray()` permite convertir una `Collection` en un array, lo que puede ser necesario cuando se necesite enviar un conjunto de datos a un programa que no trabaje todavia con las clases `collection`.

El programa del listado 17.8 nos demuestra la capacidad con que cuenta la interfaz `Collection` utilizando la clase `ArrayList` como estructura de datos:

Listado 17.8. Convirtiendo una coleccion a un Array

```
import java.util.*;
public class PlanetSet {
  public static void main(String args[]) {
    String names[] = {"Mercury", "Venus", "Earth",
      "Mars", "Jupiter", "Saturn", "Uranus",
      "Neptune", "Pluto"};
    Collection planets = new ArrayList();
    for (int i=0, n=names.length; i < n; i++) {
      planets.add(names[i]);

    String s[] = (String[])planets.toArray(new String[0]);
    for (int i=0, n=s.length; i < n; i++) {
      System.out.println(s[i]);

    planets.remove(names[3]);
    System.out.println(names[1] + " " + planets.contains(names[1]));
    System.out.println(names[3] + " " + planets.contains(names[3]));
```

El método **to Array**() se ha utilizado, en este programa, para tener acceso a cada elemento de la coleccion. Aunque funciona correctamente, el modo mds adecuado para hacerlo es mediante la interfaz **Iterator.**

La interfaz definida aqui, reemplaza a la interfaz **Enumeration** utilizada anteriormente.

```
public interface Iterator {
  public boolean hasNext();
  public Object next();
  public void remove();
```

Igual que con **Enumeration,** se puede tener un bucle **while** similar para obtener todos los elementos. Afortunadamente, los nombres de los metodos son un poco mds claros y se ha añadido el método **remove**(). Este nuevo método es sin embargo un método opcional. Cuando una coleccion no permite que se eliminen entradas, este método arrojara la nueva excepción `UnsupportedOperationException`.

Advertencia

Pero esta no es la única situación en la que nos podemos encontrar un `UnsupportedOperationException`. Algunos otros metodos de modificacion de las clases de la coleccion son opcionales y lo provocaran tambien. Aunque es un `RunTimeException` que intenta de manera programada realizar una operación no permitida, es en realidad un error de programacion que deberia identificarse y corregirse antes de que el programa llegue hasta usuarios reales.

En el listado anterior, se puede imprimir el contenido de cada elemento de la colección con el siguiente código:

```
Iterator it = planets.iterator();
while (it.hasNext()) {
  System.out.println(it.next());
}
```

Una vez que descrito el nivel mds elevado del marco de `collection`, pasemos a detallar algunas implementaciones.

La librería Generic Collection

Recursion Software, Inc., proporciona un paquete Java denominado Librerias JGL.
Para cualquierdesarrollador acostumbrado a C++, JGL proporciona un conjunto de clases collection que concuerda con la librería estandar de plantillas (STL) de C++. Anteriormente a la version 1.2 del JDK, STL era probablemente el mejor paquete orientado a colecciones disponible.
Sin embargo, a pesar de que las similitudes entre la STL y JGL eran grandes, la primera requeria un mayor marco de soporte. Las colecciones API de Java introducidas en el JDK 1.2 son mucho mas pequeñas en tamaño y parecen tambien mas sencillas de entender.
Para obtener mas información sobre el paquete JGL, visite la pagina http://www.recursionsw.com/products/jgl/.

Set

Como anteriormente se mencionó, un set es una colección sin duplicados. Debido a que esto no afiade ningun comportamiento nuevo, sino estipulaciones al existente, la interfaz `Set` es idéntica a `Collection`. Para evitar duplicaciones cuando se afiaden elementos a un set, se utiliza el método `equals()` del elemento que se ha afiadido para establecer la singularidad del objeto.

Existen tres tipos de sets con los que se puede trabajar:

- La estructura de datos recomendada para un `Set` de propósito general es `HashSet`. Un `HashSet` es una colección `Set` que se basa en una tabla hash. Como la clase `Hashtable`, los elementos afiadidos necesitan implementar el método `hashCode()`.
- Un `LinkedHashSet` es similar a un `HashSet`, aunque al desplazarse por el set, devuelve los elementos en el orden en que fueron afiadidos.
- La estructura `TreeSet` es adecuada bajo circunstancias especificas. Como su nombre implica, se basa en una estructura de drbol. Si el orden es importante,

TreeSet es la elección adecuada. La clase TreeSet estd basada en los metodos de la interfaz SortedSet, que se muestran a continuacibn:

```
public interface SortedSet extends Set {
  public Comparator comparator();
  public Object first();
  public SortedSet headSet(Object toElement);
  public Object last();
  public SortedSet subSet(Object fromElement, Object toElement);
  public SortedSet tailSet(Object fromElement);
}
```

El listado 17.9 demuestra el uso de la interfaz Set. El principal objetivo es mostrar que no se afiaden duplicados al set y que los elementos se devuelven en orden alfabético.

Listado 17.9. Trabajando con HashSet y TreeSet

```
import java.util.*;
public class MoonSet {
  public static void main(String args[]) {
    String names[] = {"Metis", "Adrastea", "Amalthea", "Thebe",
      "Io", "Europa", "Ganymede", "Callisto", "Leda", "Himalia",
      "Lysithea", "Elara", "Ananke", "Carme", "Pasiphae", "Sinope"
    };
    Set moons = new HashSet();
    int namesLen = names.length;
    int index;
    for (int i=0; i < 100; i++) {
      index = (int)(Math.random()*namesLen);
      moons.add(names[index]);
    }
    Iterator it = moons.iterator();
    while (it.hasNext()) {
      System.out.println(it.next());

    System.out.println("---");
    Set orderedMoons = new TreeSet(moons);
    it = orderedMoons.iterator();
    while (it.hasNext()) {
      System.out.println(it.next());

  }
}
```

List y ListIterator

La interfaz List es la colección bdsica para posicionamiento. Cuando un elemento se añade a una lista, se puede aiiadir en un posicibn especifica o al final de ésta. A continuación, se muestra la definición de la interfaz:

```
public interface List extends Collection {
  public void add(int index, Object element);
```

```
  public boolean add(Object obj);
  public boolean addAll(int index, Collection c);
  public boolean addAll(Collection c);
  public void clear();
  public boolean contains(Object obj);
  public boolean containsAll(Collection c);
  public boolean equals(Object obj);
  public Object get(int index);
  public int hashCode();
  public int indexOf(Object obj);
  public boolean isEmpty();
  public Iterator iterator();
  public int lastIndexOf(Object obj);
  public ListIterator listIterator();
  public ListIterator listIterator(int startPosition);
  public Object remove(int);
  public boolean remove(Object obj);
  public boolean removeAll(Collection c);
  public boolean retainAll(Collection c);
  public Object set(int index, Object element);
  public int size();
  public subList(int fromIndex, int toIndex);
  public Object[] toArray();
}
```

Hay dos implementaciones especificas de **List** dentro de las **Collections: ArrayList** y **LinkedList**. Un **ArrayList** es similar, en cuanto a funcionalidad se refiere, a un **Vector** (ambos mantienen la coleccion en un array redimensionable). La diferencia fundamental es que el acceso a los elementos de **ArrayList** no se hace de una manera sincronizada, como es el caso de **Vector**. Para mantener las colecciones en forma de lista, se usa por lo general un **ArrayList**, aunque tiene un inconveniente: cuando se hacen frecuentes adiciones y eliminaciones dentro de la lista el resultado puede no ser el adecuado. Es en este caso precisamente donde actúa **LinkedList**. **LinkedList** es una lista con referencias a los elementos anteriores y posteriores de cada nódulo. Gracias a este enlace, el acceso secuencial es satisfactorio, aunque bastante lento en relacidn con un **ArrayList**.

La tabla 17.7 contiene los métodos adicionales de **ArrayList**. Además, **ArrayList** implementa las interfaces **Cloneable** y **RandomAccess**:

Tabla **17.7.** Composición parcial de la clase ArrayList

Variables/Métodos	Descripción
`ArrayList()`	Construye una lista basada en un array
`clone()`	Crea un clon de una lista de array
`ensureCapacity()`	Asegura que la capacidad del bufer interno sea la adecuada
`trimToSize()`	Reduce el bufer interno al tamaño actual

La clase **LinkedList** además de implementar la interfaz **Cloneable**, como hace **ArrayList**, tiene su propio conjunto de constructores y métodos, que se detallan en la tabla 17.8.

Tabla 17.8. Composición parcial de la clase LinkedList

Variables/Métodos	Descripción
`LinkedList()`	Construye una lista basada en enlaces
`addFirst()`	Añade un elemento al principio de la lista
`addLast()`	Añade un elemento al final de la lista
`clone()`	Crea un clone de la lista array
`getFirst()`	Devuelve el primer elemento de la lista
`getLast()`	Devuelve el ultimo elemento de la lista
`removeFirst()`	Elimina el primer elemento de la lista
`removeLast()`	Elimina el ultimo elemento de la lista

Si se desea una estructura de datos stack, se puede usar **LinkedList** para añadir elementos con `addFirst()` y eliminarlos con `removeFirst()`. Para añadir elementos al final de la lista y eliminarlos del principio se utilizan `addLast()` y `removeFirst()`, respectivamente. También se puede hacer lo contrario mediante `addFirst()` y `removeLast()`, aunque no es lo habitual.

El listado 17.10 vuelve a utilizar el ejemplo de los planetas y sus lunas, utilizado anteriormente en el ejemplo de **Vector**, para listar los planetas en orden inverso.

Listado 17.10. Obtencion de elementos de un ArrayList

```
import java.util.*;
public class PlanetsAndMoonsList {
  static class Planet {
    private String name;
    private int moonCount;
    Planet(String s, int moons) {
      name = s;
      moonCount = moons;
    }
    public String toString() {
      return getClass().getName() + "[" + name + "-" + moonCount + "]";
    }
    public final String getName() {
      return name;
    }
    public final int getMoonCount() {
      return moonCount;
    }
```

```
  public static void main(String args[]) {
    String names[] = {"Mercury", "Venus", "Earth",
      "Mars", "Jupiter", "Saturn", "Uranus",
      "Neptune", "Pluto"};
    int moons[] = {0, 0, 1, 2, 16, 18, 21, 8, 1};
    int namesLen = names.length;
    List planets = new ArrayIist(namesLen);
    for (int i=0; i < namesLen; i++) {
      planets.add (new Planet(names[i], moons[i]));
    }
    for (int i=0, n=planets.size(); i < n; it+) {
      Planet p = (Planet)(planets.get(i));
      System.out.println (p.getName() + " : " + p.getMoonCount());
    }
  }
}
```

ListIterator

Mientras que la interfaz `Iterator` está disponible en todas las colecciones, aquellas que implementan la interfaz `List` tienen disponible, además, la interfaz `ListIterator`. `ListIterator` es bidireccional y ofrece la posibilidad de eliminar, recolocar y aiiadir elementos. Su definición es la siguiente:

```
public interface ListIterator Iterator {
  public void add(Object element);
  public boolean hasNext();
  public boolean hasPrevious();
  public Objtct next();
  public int nextIndex();
  public Object previous();
  public int previousIndex();
  public void remove();
  public void set(Object element);
}
```

El programa del listado 17.11 demuestra la interfaz por medio de un reordenamiento del sistema solar utilizado.

Listado 17.11. Desplazarniento bidireccional con ListIterator

```
import java.util.*;
public class MovingPlanets {
  public static void main(String args[]) {
    String names[] = ["Mercury", "Venus", "Earth",
      "Mars", "Jupiter", "Saturn", "Uranus",
      "Neptune", "Pluto"};
    List planets = new ArrayList();
    for (int i=0, n=names.length; i < n; i++) {
      planets.add(names[i]);
    }
    ListIterator lit = planets.listIterator();
    String s;
    lit.next();
```

```
    lit.next();
    s = (String)lit.next();
    lit.remove();
    lit.next();
    lit.next();
    lit.next();
    lit.add(s);
    lit.next(); // Devuelve el que se acaba de añadir
    lit.previous();
    lit.previous();
    s = (String)lit.previous();
    lit.remove();
    lit.next();
    lit.next();
    lit.add(s);
    Iterator it = planets.iterator();
    while (it.hasNext()) {
      System.out.println(it.next());
    }
  }
```

El resultado de ejecutar este programa es el siguiente:

```
Mercury
Venus
Mars
Saturn
Earth
Jupiter
Uranus
Neptune
Pluto
```

El `ListIterator` utiliza un cursor para posicionar el repetidor. Dicho cursor no apunta a ningun elemento en concreto sino que se situa entre dos elementos, o al final de la lista. Mientras que los métodos **`next()`** y **`previous()`** devuelven valores, no todos los métodos lo hacen. El método `add()` siempre inserta el elemento antes del cursor. Los métodos **`remove()`** y **`set()`** afectan al último elemento devuelto por **`next()`** o **`previous()`**.

Truco

Si **`next()`** o **`previous()`** pasan el final de la lista, se arroja una excepción del tipo `NoSuchElementException`.

Map

Un **`Map`** se utiliza para mantener colecciones de parejas de valores y claves, de la misma forma en que lo haría un **`Dictionary`**. Cada entrada de la colección se

añade suministrando un valor y la clave por la que se buscará. En el ejemplo de un listín telefónico, si se pregunta por el numero de alguien llamado Joe se buscará en él. La clave en este caso es "Joe" y el valor es su número de teléfono. La definición actual de la interfaz Map es la siguiente:

```
public void clear();
  public boolean containsKey(Object key);
  public boolean containsValue(Object value);
  public Set entrySet();
  public boolean equals(Object obj);
  public Object get(Object key);
  public int hashCode();
  public boolean isEmpty();
  public Set keySet();
  public Object put(Object key, Object value);
  public void putAll(Map m);
  public Object remove(Object key);
  public int size();
  public Collection values();
}
```

Además de añadir entradas y buscar valores, se puede trabajar con tres colecciones de Map: el conjunto de claves (`keySet()`), la colección de valores (`values()`) y un conjunto de ambos (`entrySet()`). Si se pregunta por ambos, cada uno de los elementos de la colección es una muestra de la interfaz Map. `Entry`, que se define a continuación:

```
public interface Map.Entry {
  public boolean equals(Object obj);
  public Object getKey();
  public Object getValue();
  public int hashCode();
  public Object setValue(Object value);
}
```

Con las clases `collection` se suministran cuatro implementaciones de Map:

- `HashMap` puede servir como mapa general y se basa en una tabla hash.
- `LinkedHashMap` es una extension de HashMap que mantiene el orden de inserción (o el orden de acceso más reciente).
- `WeakHashMap` se basa en una tabla hash con referencias débiles. Si no se mantiene una referencia a una clave fuera del mapa, el liberador de espacio puede reclamar la memoria utilizada por la clave.

Nota

Para una mayor información sobre referencias débiles, examine la clase `java.lang.ref.WeakReference` y la documentación sobre objetos de referencia en la pagina http:lljava.sun.comlj2sell.4/docslguidelrefobsl.

- `TreeMap` es el mapa basado en una estructura de arbol. Al desplazarnos por las claves del mapa, éstas se ordenan. Si necesitamos que los elementos estén ordenados pero no queremos aiiadir un nuevo nódulo, debido a los problemas que esto pueda ocasionar, podemos utilizar un `HashMap` una vez aiiadidos todos los nodulos.

```
Map map = new HashMap()
// añade entradas
map = new TreeMap(map);
```

El listado 17.12 convierte el ejemplo `PlanetDiameters` del listado 17.5 para utilizar un `TreeMap`:

Listado 17.12. Clasificacion de elernentos en un TreeMap

```
import java.util.*;
public class DiameterMap {
  public static void main(String args[]) {
    String names[] = {"Mercury", "Venus", "Earth",
      "Mars", "Jupiter", "Saturn", "Uranus",
      "Neptune", "Pluto"};
    float diameters[] = {4800f, 12103.6f, 12756.3f,
      6794f, 142984f, 120536f, 51118f, 49532f, 2274f};
    Map map = new TreeMap();
    for (int i=0, n=names.length; i < n; itt) {
      map.put (names[i], new Float(diameters[i]));
    }
    Iterator it = map.keySet().iterator();
    Object obj;
    while (it.hasNext()) {
      obj = it.next();
      System.out.println (obj t ": " t map.get(obj));
    }
  }
}
```

El resultado que se obtiene aparecera ahora ordenado alfabkticamente por el nombre del planeta (clave):

```
Earth: 12756.3
Jupiter: 142984.0
Mars: 6794.0
Mercury: 4800.0
Neptune: 49532.0
Pluto: 2274.0
Saturn: 120536.0
Uranus: 51118.0
Venus: 12103.6
```

`TreeMap` es el único `map` que permite obtener un submapa:

- `subMap(Object fromKey, Object toKey)` devuelve un mapa de elementos que estkn dentro de un margen especificado.

- `headMap(Object tokey)` devuelve un map de todo menos la clave.
- `tailMap(Object fromKey)` devuelve un map de todo aquello que sea mayor que la clave, mientras que `subMap(Object fromKey, Object toKey)` devuelve un mapa de la clave y todo lo que sea mayor que ella.

Todos ellos son parte de la interfaz `SortMap`, que se describe a continuación:

```
public interface SortedMap extends Map {
  public Comparator comparator();
  public Object firstKey();
  public SortedMap headMap(Object toKey);
  public Object lastKey();
  public SortedMap subMap(Object fromKey, Object toKey);
  public SortedMap tailMap(Object toKey);
}
```

Sincronización y legibilidad

Si ya ha utilizado las clases históricas de Java, sabrá que uno de sus mayores problemas es su pobre representación, lo que no quiere decir que fueran diseñadas para ser lentas, sino para ser siempre seguras con los thread. Esto significa que si los thread múltiples que se estén ejecutando acceden a una estructura de datos sencilla, todos los accesos a ésta son seguros y están sincronizados de manera adecuada. Sin embargo, cuando no se requieren comportamientos sincronizados, entonces no hay modo de apagarlo. Todas las utilizaciones de clases tales como `Hashtable` y `Vector` se encuentran sincronizadas, y, por lo tanto, son más lentas.

Las clases del marco de colecciones, sin embargo, funcionan de otro modo, ya que no sincronizan nada. Si se necesita un comportamiento sincronizado, una vez que se tenga un colección con la que trabajar, se sincroniza mediante la clase `Collections`. La clase `Collections` proporciona éstos y otros comportamientos que serán descritos brevemente. Como todos los métodos de `collections` son estáticos y el constructor es privado, no se necesita crear ninguna mencidn a la clase.

Tabla **17.9.** Cornposicion de la clase Collections

Métodos	Descripción
EMPTY – LIST	Representa una listavacia inmutable
EMPTY – MAP	Representa un mapa vacio inmutable
EMPTY – SET	Representa un set vacio inmutable
`binarySearch()`	Realiza una busqueda binaria de un elernento en la lista ordenada
`copy()`	Copia elementos de dos listas distintas

Métodos	Descripción
`enumeration()`	Convierte la coleccion en una enurneracion
`fill()`	Llena la lista con copias multiples de un unico elemento
`indexOfSubList()`	Devuelve el índice de una lista dentro de otra lista
`lastIndexOfSubList()`	Devuelve el índice de una lista dentro de otra lista, cornenzando la busqueda por el final
`list()`	Convierte una enurneracion en una lista array inmutable
`max()`	Devuelve el valor rnaxirno de la coleccion
`min()`	Devuelve el valor minirno de la coleccion
`nCopies()`	Crea una lista inrnutable con copias multiples de un elemento
`replaceAll()`	Sustituye todas las copias de un elernento por otro elernento
`reverse()`	Invierte el orden de los elernentos de la lista
`reverseOrder()`	Devuelve el comparador para invertir el orden de los elernentos cornparables
`rotate()`	Rota los elernentos de la lista una distancia concreta
`shuffle()`	Reordena los elernentos de la lista de una rnanera aleatoria
`singleton()`	Devuelve el elemento individual de un set inrnutable
`singletonList()`	Devuelve el elemento individual de un lista inrnutable
`singletonMap()`	Devuelve la pareja valor-clave de un rnapa inmutable
`sort()`	Reordena los elernentos de la lista
`swap()`	Intercambia dos elernentos de la lista
`synchronizedCollection()`	Crea una coleccion segura para thread
`synchronizedList()`	Crea una lista segura para thread
`synchronizedMap()`	Crea un rnapa seguro para thread
`synchronizedSet()`	Crea un set seguro para thread
`synchronizedSortedMap()`	Crea un rnapa ordenado seguro para thread
`synchronizedSortedSet()`	Crea un set ordenado seguro para thread
`unmodifiableCollection()`	Crea una coleccion solo de lectura
`unmodifiableList()`	Crea una lista solo de lectura

Métodos	Descripción
`unmodifiableMap()`	Crea un mapa sólo de lectura
`unmodifiableSet()`	Crea un set solo de lectura
`unmodifiableSortedMap()`	Crea un mapa ordenado solo de lectura
`unmodifiableSortedSet()`	Crea un set ordenado solo de lectura

Por ejemplo, para sincronizar el acceso a un `Map`, deberemos hacer lo siguiente:

```
Map map = new HashMap()
map = Collections.synchronizedMap(map);
// accede al mapa desde threads múltiples
```

Truco

Cuando se sincroniza una coleccion es conveniente no almacenar la coleccion nueva en una variable nueva para que así se pueda acceder a la coleccion sin sincronizar desde la variable antigua.

Además de la sincronización, la clase `Collections` también ofrece la posibilidad de convertir una colección específica a un modo de sólo-lectura. Ello provoca que cualquier método que pueda alterar la coleccion devuelva una `UnsupportedOperationException`. De esta manera, para conseguir que una colección sea de sólo-lectura, se crea, se rellena y posteriormente se hace inmodificable. Si se intenta hacer de sólo-lectura antes de rellenarla, se obtendrá una `UnsupportedOperationException`:

```
Map map = new HashMap()
// añade entradas
map = Collections.unmodifiableMap (map);
```

Algoritmos y organización

El marco de colecciones incluye muchas capacidades algoritmicas. Al describir anteriormente las colecciones basadas en estructuras de árbol, no se hizo mención a la manera de organizar dicho esquema. A continuación se va a describir este comportamiento y otras capacidades similares del marco. Para ordenaciones aleatorias o para cambiar el orden de los elementos de una lista, basta con usar `Collection.shuffle()`.

Comparable y Comparator

Existen dos interfaces que soportan organizaciones. La interfaz `Comparable`, que se utiliza cuando una clase tiene un orden natural. Dados muchos objetos del

mismo tipo, la interfaz permite a uno de ellos ordenar todos los demás. La definición de la interfaz Comparable es:

```
public interface Comparable {
  public int compareTo(Object obj);
}
```

Si el valor devuelto es negativo, indica que la llamada a la interfaz que implementa va antes del parámetro. Cero significa que tanto el implementador como el parámetro son iguales, y un valor positivo indica que el pardmetro va antes.

Si una clase implementa a la interfaz Comparable, los objetos de la clase pueden usarse como la clave para una colección basada en un esquema de árbol. Si un objeto no implementa Comparable, pero queremos que sea una clave, se puede implementar la interfaz Comparator y se hardn las comparaciones automdticamente.

```
public interface Comparator {
  public abstract int compare (Object obj1, Object obj2);
}
```

Nota

Las clases pertenecientes a Java 1.4 que implementan la interfaz Comparable son: BigDecimal, BigInteger, Byte, Character, CollationKey, Date, Double, File, Float, Integer, Long, ObjectStreamField, Short, String y URI.

Como demostracibn, vamos a comparar los resultados de ordenar la lista con la comparación String por defecto y un comparador que no distingue entre caracteres en mayúsculas y en minusculas. Como ayuda, necesitamos la clase Arrays, cuyos cinco métodos se describen en la tabla 17.10.

Tabla **17.10.** Composición de la clase Arrays

Métodos	Descripción
asList()	Convierte array en lista
binarySearch()	Busca array para el elemento
equals()	Comprueba si el contenido de dos arrays es el mismo
fill()	Llena el array con copias múltiples de un elemento
sort()	Organiza los elementos del array

La clase Arrays es como Collections, todos los métodos son estdticos y no tienen constructor, solamente una colección de utilidades para trabajar con arrays.

Sin embargo, a diferencia de las colecciones, la clase `Arrays` está repleta de métodos anulados para cada tipo primitivo de datos (uno o dos para cada mktodo de la clase).

El mktodo `asList()` es el Gnico mktodo de la clase con una única implementacidn.

La clase `Arrays` proporciona un mktodo `short()` para organizar cualquier array. Una vez que un array ha sido organizado, entonces se puede realizar una búsqueda rápida binaria para encontrar un elemento en el array. En el listado 17.13, el programa organiza una lista de nombres utilizando para ello la interfaz `String Comparable`, y posteriormente lo hace tambikn con `Comparator` que no diferencia entre mayúsculas y minúsculas.

Listado 17.13. Organización de Strings

```
import java.util.*;
public class SortingPlanets {
  static class InsensitiveComp implements Comparator {
    public int compare (Object al, Object a21 {
      String s1 = a1.toString().toLowerCase();
      String s2 = a2.toString().toLowerCase();
      return s1.compareTo (s2)
    }
  !
  public static void main(String args[]) {
    String names[] = {"Mercury", "Venus", "Earth",
      "Mars", "Jupiter", "Saturn", "Uranus",
      "Neptune", "Pluto",
      "mercury", "venus", "earth",
      "mars", "jupiter", "saturn", "uranus",
      "neptune", "pluto"
    };
    Arrays.sort(names);
    for (int i=0, n=names.length; i<n; i++) {
      System.out.print(names[i] + " ");
    }
    System.out.println();
    Arrays.sort(names, new InsensitiveComp());
    for (int i=0, n=names.length; i<n; i++) {
      System.out.print(names[i] t " ");
    }
    System.out.println();
  }
```

Al ejecutar el programa se produce el siguiente resultado. (Observe cdmo la segunda ejecucidn ignora las mayúsculas):

```
Earth Jupiter Mars Mercury Neptune Pluto Saturn Uranus Venus earth
    jupiter mars mercury neptune pluto saturn uranus venus
Earth earth Jupiter jupiter Mars mars Mercury mercury neptune Neptune
    pluto Pluto Saturn saturn Uranus uranus Venus venus
```

Advertencia

Si se realiza una `binarySearch` en un array, recuerde incluir un `short()` antes. Si no se hace así, se sucederán todo tipo de comportamientos extraños, incluyendo la posibilidad de un bucle infinito.

Truco

Si se quiere organizar un objeto array en orden inverso, se puede utilizar `Arrays.sort(anObjectArray, Collections.reverseOrder())`.

Utilidades variadas

Además de organizar y buscar con **`Arrays`** y **`Collections`** se pueden usar **`Collections`** para encontrar valores minimos y mdximos. Al contrario de cuando se usaba **`binarySearch()`** no es necesario incluir un **`short()`** antes, sino que basta con usar los métodos **`min()`** y **`max()`** de **`collections`**, que tienen un **`Comparator`** opcional.

Ir de un estilo de colección antiguo a uno nuevo es también relativamente fácil.

- Para crear una **`Enumeration`** desde cualquier **`collection`** debemos emplear **`collections.enumeration()`**.
- Para ir desde una **`Enumeration`** a una **`Collection`** (un `ArrayList`, mds concretamente), utilizaremos **`Collections.list()`**.
- Para ir desde un array a una **`List`**, usaremos **`Arrays.asList()`**.

```
List list = Arrays.asList(args);
```

Nota

La listas devuelta por **`Arrays.asList()`** es inmutable. Si necesitamos añadir o eliminar algun elemento de la lista, debemos pasarlos mediante el constructor de otra lista, del siguiente modo: **`List list = new ArrayList (Arrays.asList(args))`**.

Biblioteca de matemáticas de Java

Las bibliotecas estdndar de Java proporcionan implementación para una gran variedad de necesidades matemdticas. La clase **`Math`** del paquete **`java.lang`** ofre-

ce un grupo común de métodos numéricos y trigonométricos. Para trabajos de precisión infinita, el paquete `java.lang` ofrece las clases `BigInteger` y `BigDecimal`. Para mostrar de manera adecuada los datos numéricos contamos con la clase `NumberFormat` del paquete `java.text`.

La clase java.lang.Math

La clase `Math` proporciona gran cantidad de funciones numéricas comunes utilizadas en los programas de Java. La mayoría de estas funciones estaban en las declaraciones `C` `math.h`. La clase Math de Java es final y todos sus métodos son estiticos, lo que significa que no es necesario crear una copia de la clase ni se pueden crear subclases. Además de los metodos, existen dos constantes, para `pi` y `e`, frecuentemente utilizadas. La tabla 17.11 muestra los 22 métodos de esta clase y las dos constantes mencionadas.

Algunos de los métodos de la tabla 17.11 tienen variaciones que dependen del tipo de argumento. Por ejemplo, si necesitamos calcular el valor absoluto de un numero, podemos emplear las cuatro versiones de `abs()` (para `int`, `long`, `float` o `double`). De este modo, el valor devuelto es del mismo tipo que el que se ha pasado.

Si pasamos un `byte`, `short` o `char`, se actualizard primero a `int`, para después devolver un `int`.

Tabla **17.11.** Composición de la clase Math

Variables/Métodos	Descripción
E	Base de logaritmos naturales, e
PI	Ratio de la circunferencia y el diarnetro de un círculo, pi
abs()	Devuelve el valor absoluto de un número
acos()	Devuelve el coseno del arco de un angulo
asin()	Devuelve el seno del arco de un angulo
atan()	Devuelve la tangente de un angulo
atan2()	Convierte coordinadas rectangulares en un diagrama polar
ceil()	Devuelve la integral mas pequeña pero no menor que el argumento
cos()	Devuelve el coseno del angulo
exp()	Devuelve e elevado a la potencia del argurnento
floor()	Devuelve la integral mayor pero no superior al argumento
IEEEremainder()	Devuelve el resto de la division con coma flotante (similar a %)

Variables/Métodos	Descripción
`log()`	Devuelve el logaritmo natural
`max()`	Devuelve el maximo de dos valores
`min()`	Devuelve el minimo de dos valores
`pow()`	Devuelve el primer argumento elevado a la potencia del segundo
`random()`	Devuelve el numero mayor o igual que 0 pero menor que 1
`rint()`	Redondea el argumento segun el valor de la integral mas cercana
`round()`	
`sin()`	Devuelve el seno del angulo
`sqrt()`	Devuelve la raiz cuadrada de un valor
`tan`	Devuelve la tangente de un angulo
`toDegrees()`	Convierte un angulo de radianes a grados
`toRadians()`	Convierte un angulo de grados a radianes

Nota

Al trabajar con funciones trigonometricas de Math, debemos pasar el angulo medido en radianes y no en grados. Para hacer la conversion, podemos utilizar el método `toRadians()`.

Para demostrar la clase Math, el programa del listado 17.4 computa el valor de varias funciones trigonométricas de algunos ángulos semejantes.

Listado 17.14. Metodos trigonometricos de Math

```
public class MathTest {
  public static void main(String args[]) {
    int angles[] = {0, 30, 45, 60, 90, 180};
    for (int i=0, n=angles.length; i < n; i++) {
      double rad = Math.toRadians(angles[i]);
      System.out.println("Angle: " + angles[i]);
      System.out.println("  Sine   : " + Math.sin(rad));
      System.out.println("  Cosine : " + Math.cos(rad));
      System.out.println("  Tangent: " + Math.tan(rad));
    }
```

Nota

Mas adelante describiremos la manera de formatear numeros como **0,49999999999999994**En este ejemplo, el seno de **30** no aparece como 0,5 porque el angulo al estarexpresado en radianes (pi * 1/6) no puede almacenarse internamente en forma binaria.

El paquete java.math

Si trabajamos en finanzas y los pequefios errores al redondear no están permitidos, es preferible utilizar las clases `BigInteger` y `BigDecimal` del paquete `java.math`.

- Si trabajamos con numeros con mds digitos de los que pueden representarse mediante `long`, y necesitamos mantener la precision infinita en lugar de pasar a un `double` (o `float`), la clase `BigInteger` nos permite mantener los numeros con tantos digitos como se precisen.
- Por otro lado, la clase `BigDecimal` funciona `comodouble` o `float`, es decir con numeros con coma flotante. En este caso, la clase nunca redondea el numero ya que solo un número limitado de digitos pueden almacenarse en el tipo de datos.

Cuando trabajemos con `BigInteger` y `BigDecimal` no podemos utilizar los operadores matemáticos estándar de la clase `Math`. Si necesitamos realizar una operación sencilla como sumar o hacer que el numero sea negativo, debemos invocar un método de la clase correspondiente, lo que trae a colacion el segundo problema de trabajar con estas dos clases, y es que las muestras creadas son inmutables, como ocurre tambikn con las clases `Integer` y `Float`. Casi todas las operaciones devuelven una nueva muestra tras ejecutar la operación deseada.

BigInteger

La clase `BigInteger` representa un `Number` para almacenar grandes integrales. Esta clase estd dirigida al cdlculo de números donde el resultado supera los 64 bits (el tamaño de `long`). El siguiente programa demuestra la necesidad de utilizar `BigInteger`, al intentar multiplicar el valor mdximo de `long`.

```
import java.math.*;
public class BigInt {
  public static void main(String args[]) {
    long bigl = Long.MAX_VALUE;
    System.out.println(bigl * bigl);
    Long maxLong = new Long(Long.MAX_VALUE);
    BigInteger bigIntl = new BigInteger(maxLong.toString());
    BigInteger bigInt2 = new BigInteger(maxLong.toString());
```

```
        System.out.println(bigInt1.multiply(bigInt2));
    }
}
```

Este programa muestra la necesidad de utilizar la clase `BigInteger`, siempre que necesitemos trabajar con números tan largos, ya que el resultado de esta operación requiere 38 dígitos.

```
85070591730234615847396907784232501249
```

Además de para manipular valores de integrales con muchos dígitos, podemos trabajar con `BigInteger` de una manera similar a como lo hacemos con `BitSet`, ya que dispone de métodos que realizan operaciones relacionadas con bits, tales como las operaciones lógicas de AND, o la utilización de bits de prueba. Sin embargo, como los objetos son inmutables, debemos crear una nueva `BigInteger` cada vez que modifiquemos la configuración del bit. Como consecuencia, la ejecución se ralentiza en comparación con `BigSet`.

BigDecimal

La clase `BigDecimal` se utiliza con más frecuencia que la clase `BigInteger`. El mundo de las finanzas requiere un mayor control sobre la precisión de los dígitos al realizar computaciones numéricas.

Con `BigDecimal`, se especifica una escala con lo que el número de dígitos detrás de la coma decimal de todas las computaciones numéricas corresponden con dicha escala. Para redondear debemos especificar una constante (tabla 17.12) para que el sistema sepa la manera de eliminar el último dígito.

Tabla 17.12. Constantes de redondeo de BigDecimal

Variables	Descripción
`ROUND_CEILING`	Redondea hacia arriba (hacia el infinito positivo)
`ROUND_DOWN`	Redondea a cero
`ROUND_FLOOR`	Redondea hacia el infinito negativo
`ROUND_HALF_DOWN`	Redondea hacia el valor del dígito conexo, o a cero si ambos valores coinciden
`ROUND_HALF_EVEN`	Redondea hacia el valor del dígito conexo, o a un número par si ambos valores coinciden
`ROUND_HALF_UP`	Redondea hacia el valor del dígito conexo, o alejándose de 0 si ambos valores coinciden
`ROUND_UNNECESSARY`	El valor puede representarse sin necesidad de redondear
`ROUND_UP`	Redondea alejándose de 0

Se puede especificar la escala de `BigDecimal` de una de las tres formas siguientes:

- Al llamar al constructor de `BigDecimal` con un `String`, el numero de digitos decimales determina la escala.
- Al llamar al constructor, pasando un `BigInteger`, también se transmite la escala.
- Se puede cambiar la escala posteriormente mediante `setScale()`.

Para demostrar cómo utilizar `BigDecimal`, el listado 17.15 añade un interés mensual del 3,25 por ciento anual a una cuenta bancaria con un balance inicial de 10.000 dólares.

Listado 17.15: BigDecimal para precision

```
import java.math.*;
public class BigDec {
  public.static void main(String arqs[]) {
    BigDecimal rate = new BigDecimal(".03250000");
    BigDecimal months = new BigDecirnal("12");
    BigDecirnal monthlyRate = rate.divide(months,
      BigDecimal.ROUND_HALF_DOWN);
    System.out.println("Annual rate : " + rate);
    System.out.println("Monthly rate: " + monthlyRate);
    BigDecirnal balance = new BigDecimal("10000.0000");
    for (int i=0; i<12; itt) {
      BigDecimal interest = balance.multiply(monthlyRate);
      balance = balance.add(interest);
      System.out.println("Balance: " t balance);
    }

}
```

Al ejecutarse el programa, genera el siguiente resultado (con cada paso se incrementa en ocho el numero de digitos):

```
Annual rate : 0.03250000
Monthly rate: 0.00270833
Balance : 10027.083300000000
Balance : 10054.23995051388900000000
Balance : 10081.470150199064280995370000000
Balance : 10108.774098250952912759518190432100000000
Balance : 10136.151994404468916061732176332792969393000000000
Balance : 10163.604038935474371361169647437920362575796143690000000
Balance : 101~1.130432662244484665358244029165905431371049659839937
   700000000
Balance : 10218.731376946936621270511973722217416328027994814765171
   9847104100000000
Balance : 10246.407073697063318129997539416008509423191682873961844
   956749660074471530000000
Balance : 10274.157725366969285626388555651935067749517795604439882
   040301373881341023478844900000000
```

```
Balance : 10301.983534959312409551729072568863873051555847135809254706027583301265075812963360008017000000000
Balance : 10329.884706026548764129890306967974304144857567383282580984825558985947391055739883064827512681610000000
```

Formato del resultado numérico

El resultado de este programa no esdemasiado atractivo, y para mejorarlo entra en juego la clase `NumberFormat` del paquete `java.text`. Aunque es posible cambiar la escala antes de la visualización, esta acción crea muchos objetos `BigDecimal` extras que deben eliminarse.

La clase `NumberFormat` ayuda a crear un formato una unica vez y reutilizarlo cada vez que necesitemos imprimir un numero.

Para demostrarlo, aiiadamos al programa `BigDec` la siguiente linea fuera del bucle:

```
NumberFormat nf = NumberFormat.getCurrencyInstance();
```

y cambiemos el contenido de `println()`

```
System.out.println("Balance : " + nf.format(balance.doubleValue()));
```

Con estos cambios, la apariencia del resultado mejora considerablemente:

```
Annual rate : 0.03250000
Monthly rate: 0.00270833
Balance : $10,027.08
Balance : $10,054.24
Balance : $10,001.47
Balance : $10,108.77
Balance : $10,136.15
Balance : $10,163.60
Balance : $10,191.13
Balance : $10,218.73
Balance : $10,246.41
Balance : $10,274.16
Balance : $10,301.98
Balance : $10,329.83
```

Tambien podemos mostrar los decimales como porcentajes mediante el método `getPercentInstance()`. Sin embargo, este método no admite porcentajes fraccionales tal como 3,25 por ciento, por lo que debemos crear nuestro propio formato. En este caso, nos serviremos de la clase

`DecimalFormat` para conseguirlo. Podemos pasar a su constructor un formato adecuado, tal como `#,##0.00000%` donde `#` representa un digito opcional; para este ejemplo se utiliza la convención estadounidense para la coma, el punto y el simbolo del tanto por ciento.

```
NumberFormat pf = new DecimalFormat("#,##0.00000%");
System.out.println("Annual rate : " + pf.format(rate.doubleValue()));
```

```
System.out.println("Monthly rate: " +
   pf.format(monthlyRate.doubleValue()));
```

Esta modificación produce el siguiente resultado:

```
Annual rate : 3.25000%
Monthly rate: 0.27083%
```

Aunque en este caso se ha utilizado el formato numérico de los EE.UU., se puede elegir en cada caso el formato adecuado segun las necesidades del mercado internacional.

Las bibliotecas no realizan el cambio de moneda, pero son capaces de producir el formato adecuado segun el país. La clase **`Locale`** de la biblioteca `java.util` cuenta con el mecanismo para especificar una localidad concreta y asi poder utilizar el formato adecuado. Por ejemplo, muchos paises europeos, incluido España, cambia la colocación del punto y la coma con respecto a EE.UU. Si el país de destino fuera Italia, el resultado seria el que se muestra en la figura 17.4.

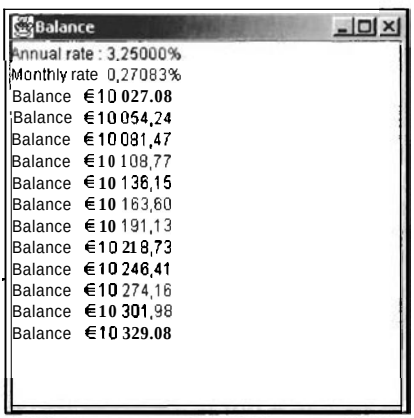


Figura 17.4. Resultado con formato numérico adecuado para Italia

Nota

En esta figura, el resultado se hasituado en un `JTextArea` para poder mostrar el símbolo del euro.

El listado 17.16 corresponde al programa `BigDec` actualizado.

Listado 17.16. Numeros formateados segun el pais

```
import java.math.*;
import java.text.*;
import java.util.*;
import javax.swing.*;
```

```
public class BigDecFormat {
  public static void main(String args[]) {
    StringBuffer buffer = new StringBuffer();
    Locale italian = new Locale("it", "IT", "EURO");
    Locale.setDefault(italian);
    BigDecirnal rate = new BigDecimal(".03250000");
    BigDecirnal months = new BigDecimal("12");
    BigDecimal monthlyRate = rate.divide(months,
BigDecirnal.ROUND_HALF_DOWN);
    NumberFormat pf = new DecimalFormat("#,##0.00000%");
    buffer.append("Annual rate : " + pf.format(rate.doubleValue()) + "\n");
    buffer.append("Monthly rate: " + pf.format(
      monthlyRate.doubleValue()) + "\n");
    BigDecimal balance = new BigDecimal("10000.0000");
    NumberFormat nf = NumberFormat.getCurrencyInstance();
    for (int i=0; i<12; i++) {
      BigDecimal interest = balance.multiply(monthlyRate);
      balance = balance.add(interest);
      buffer.append("Balance : " + nf.format(balance.doubleValue()) +
"\n");
    }
    JFrame frame = new JFrame("Balance");
    JTextArea text = new JTextArea(buffer.toString());
    JScrollPane pane = new JScrollPane(text);
    frame.getContentPane().add(pane);
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setSize(300, 300);
    frame.show();
  }
```

Fechas y calendarios

Antes de trabajar con fechas y horas, es necesario entender las siguientes clases del paquete `java.util`: Date, Calendar, y `GregorianCalendar`. Al igual que para mostrar numeros, existe una clase adicional para cambiar el formato (`DateFormat`).

La clase Date

La clase Date representa una muestra en el tiempo, a partir de 1970. Además del dia y la fecha, esta clase también representa la hora, incluyendo los milisegundos.

Por si solo, el objeto Date no se utiliza mucho. En realidad se encarga de almacenar una configuración de Date. Se puede mostrar dicha configuración con `toString()`, y compararla con otra fecha mediante `before()`, `after()`, `compareTo()`, o `equals()`.

Al constructor de Date se le puede pasar el número de milisegundos desde el uno de Enero de 1970, a las 0:00:00 segun el meridiano de Greenwich, o, en el caso

de no incluirse argumentos, se utilizard la hora actual (`System.currentTimeMillis()`).

La clase Calendar

La clase `Calendar` representa el valor de `Date` segun los atributos de un calendario, es decir, dia de la semana, mes del aiio, y semana del aiio.

La única implemenación disponible en las bibliotecas estdndar para esta clase es `GregorianCalendar`, que representa el calendario estdndar de 365 dias, incluyendo los años bisiestos, utilizado en la gran mayoría de los paises.

Para obtener un calendario con el que trabajar, necesitamos `getInstance()`. También es posible crear `GregorianCalendar` de manera manual, aunque no resulta necesario. Es posible cambiar la fecha mediante diferentes variables del método `set()`, u obtener la configuración de la fecha actual con `getTime()`. La fecha actual no se corresponde obligatoriamente con la real sino con la resultante de modificar la fecha.

La operación mas sencilla que se puede realizar con un calendario es preguntar el valor de un campo concreto. El método `get()` acepta muchas de las constantes de consulta de la tabla 17.13, devolviendo la configuración actual del calendario.

Tabla 17.13. Constantes de consulta de Calendar

Variables	Descripción
`AM_PM`	Carnpo A.M. y P.M.
`DAY_OF_MONTH`	Carnpo del dia del rnes
`DAY_OF_WEEK`	Carnpo del dia de la sernana
`DAY_OF_WEEK_IN-MONTH`	Carnpo del dia de la sernana en el mes
`DAY_OF_YEAR`	Carnpo del dia del aiio
`DST_OFFSET`	Carnpo del carnbio horario
`ERA`	Carnpo de la era (a.c. o d.c.)
`HOUR`	Carnpo de un reloj de doce horas
`HOUR-OF-DAY`	Carnpo de un reloj de 24 horas
`MILLISECOND`	Carnpo de los rnilisegundos
`MINUTE`	Carnpo de los rninutos
`MONTH`	Carnpo de los rneses
`SECOND`	Carnpo de los segundos
`WEEK-OF-MONTH`	Carnpo de la sernana del mes

Variables	Descripción
WEEK_OF_YEAR	Campo de la semana del año
YEAR	Campo del año
ZONE—OFFSET	Desajuste de la zona horaria

Mucho de los campos de consulta devuelven constantes propias. Por ejemplo, `get(Calendar.MONTH)` devuelve una constante de cada uno de los meses de año, aunque no los numera del uno al doce, ya que Enero corresponde al numero 0 y no al numero 1. Existe incluso un mes artificial adicional (UNDECIMBER) por si se requiere una implementación de `LunarCalendar`. Para los dias de la semana, las constantes van del domingo al sábado, comenzando por uno y no por 0, como en el caso de los meses.

También es posible ejecutar operaciones aritméticas con métodos como `roll()`, que es el más sencillo de utilizar.

Dado un campo concreto y desplazándolo una unidad hacia adelante o hacia atrás, este método cambiara el año al traspasar el umbral de Diciembre.

```
Calendar cal = Calendar.getInstance();
cal.set(2000, Calendar.DECEMBER, 25);
cal.roll(Calendar.MONTH, true); // roll month forward
System.out.println(cal);
```

Truco

Es muy importante recordar que los meses se cuentan desde 0, por lo que `cal.set(2000, 11, 25)` es lo mismo que `cal.set(2000, Calendar.DECEMBER, 25)`.

Con bastante frecuencia, desearemos desplazarnos en el tiempo más rápido que de una en una unidad, para lo cual emplearemos el método `add()`. El primer argumento es el campo a modificar (de la tabla **17.13)** y el segundo es el numero de unidades.

El Dr. Naegele determino que la duración media de la gestacion humana es de 266 dias, por lo que si deseamos conocer la fecha en la que se espera que nazca un niño añadiremos 266 dias al calendario de la manera:

```
cal.add(Calendar.DAY_OF_YEAR, 266);
```

Tambien se puede retroceder desde la fecha de nacimiento para determinar la fecha de gestacion mediante:

```
cal.add(Calendar.DAY_OF_YEAR, -266);
```

Calcular la diferencia entre ambas fechas es un poco mds complicado, ya que no existe una llamada estdndar para lograrlo. Es necesario calcular manualmente el valor y ajustarlo según el cambio de hora, o conformarnos con una aproximación. De manea general, se calcula la diferencia entre dos fechas representadas mediante `long`, y se divide por 24 horas en un dia, 60 minutos en una hora, 60 segundos en un minuto y 1.000 milisegundos en un segundo para obtener el número de dias entre las dos fechas:

```
Calendar cal1 = …; // fecha anterior
Calendar cal2 = …; // fecha posterior
double days = (cal2.getTimeInMillis() - cal1.getTimeInMillis())
  /1000.0/60.0/60.0/24.0;
```

Nota

Existe tambien la clase `TimeZone` que permite ajustar la hora según la franja horaria. lass

Formato de las *fechas* de entrada y salida

El paquete `java.text` incluye las clases `DateFormat` y `SimpleDateFormat`, que sirven para gestionar la entrada y salida de fechas. Estas clases nos ayudan a definir el formato, añadiendo gran flexibilidad a los programas. Los formatos pueden ser estdndar o modificables según el país.

La clase `DateFormat` dispone de cuatro formatos estándar predeterminados. Siempre tenemos la opción de elegir uno de ellos o crear nuestro propio formato.

- `DateFormat.SHORT`: Es un formato numérico del tipo 12/25/00
- DateFormat .MEDIUM: Se basa en nombres de meses con tres caracteres, del tipo Dec 25,2000.
- DateFormat. LONG: Se basa en nombres completos de meses, del tipo December 25, 2000.
- DateFormat. FULL: Incluye el dia de la semana (e incluso la hora), como en Monday, December 25,2000.

Al igual que con `NumberFormat`, el resultado de DateFormat puede localizarse. Para España, por ejemplo, las cuatro variedades de formatos se mostrarian de la manera siguiente:

- SHORT: 25/12/00.
- MEDIUM: 25-dic-00.
- LONG: 25 diciembre 2000.
- FULL: lunes, 25 dicembre 2000.

Para obtener el formato adecuado, el método básico es `getDateInstance()`. Si no se especifica ningún argumento, produce el formato MEDIUM del país. También es posible pasar un estilo o país determinado. Para mostrar la hora, utilizaremos `getDateTimeInstance()`.

Una vez definido el formato, el funcionamiento es como el utilizado para mostrar números. Sólo tenemos que emplear el método `format()`, pasando Date:

```
DateFormat df = DateFormat.getDateInstance();
System.out.println(df.format(cal.getTime()));
```

Para convertir un string en un objeto Date, debemos llamar al método `parse()` en lugar de `format()`. El método `parse()` analizard sólo el formato `DateFormat` que se esté utilizando.

Si el estilo a analizar no es uno de los cuatro estdndar, será necesario crearlo nosotros mismos mediante la clase `SimpleDateFormat`.El siguiente fragmento demuestra cómo leer una fecha en la que aparecen dos digitos para el mes, el dia y el año.

```
String date = "12-25-00";
DateFormat input = new SimpleDateFormat("MM-dd-yy");
try {
  Date when = input.parse(date);
  System.out.println(df.format(when));
} catch (ParseException e) {
  e.printStackTrace();
}
```

Truco

El `javadoc` de la clase `SimpleDateFormat` contiene un listado con la sintaxis de los formatos de hora. La tabla 17.14 enumera los distintos estilos.

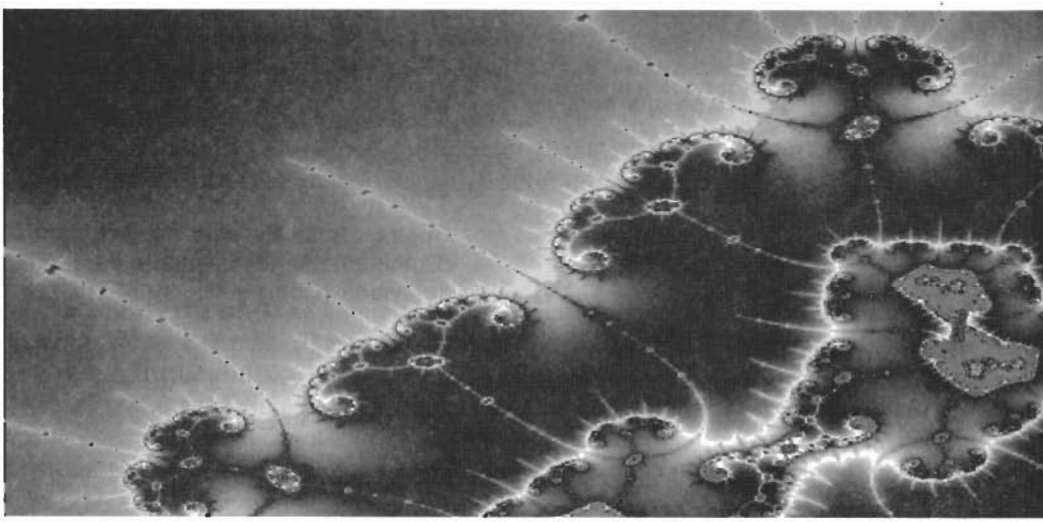
Tabla 17.14. Estilos de SimpleDateFormat

Simbolo	Significado	Resultado
G	Designador de Era	BC (siglas en ingles de a.c.)
Y	Año	2001
M	Mes	Tres Ernes para formato abreviado (Ene); cuatro o mas para formato completo (Enero); y rnenos de tres para números (01)
d	Dia del mes	01
h	Hora en AM/PM (1-12)	11
H	Hora del dia (0-23)	23

Símbolo	Significado	Resultado
m	Minutos en una hora	59
s	Segundos en un minuto	13
S	Milisegundos	314
E	Dia de la semana	Cuatro o mas Es para nombre completo (Domingo); menos de cuatro para formato abreviado (Dom)
D	Dia en el año	14
F	Dia de la semana en el mes	2
W	Semana en el año	22
W	Semana en el mes	2
a	Marcador AMIPM	PM
k	Hora en el dia (1-24)	13
K	Hora en AMIPM (0-11)	0
z	Zona horaria	Cuatro o mas zetas devuelve el nombre completo (Eastern Standard Time); menos de cuatro, devuelve el nombre abreviado (EST)
''	Comilla sencilla	
	Escape del texto	' o' clock'

Resumen

Para aprender el lenguaje de programación de Java es esencial conocer las bibliotecas esthndar. En lugar de tener que recrear bibliotecas comunes, es mucho mas sencillo utilizar las clases disponibles en las bibliotecas de Java que se han descrito en este capitulo. Para trabajar con estructuras de datos de Java disponemos de muchas opciones. El marco de colecciones de Java puede complementarse con las clases de las colecciones históricas disponible aún en muchos códigos. El código de Java 1.0 y 1.1 resultarh mucho más sencillo si conocemos las clases `vector`, `Dictionary`, `Hashtable`, `Properties`, `BitSet`, y sus correspondientes implementaciones. Siempre que sea posible, es conveniente utilizar unicamente las clases y arrays mas recientes. Las bibliotecas de las clases `Math` y `Date` constituyen también complementos de gran utilidad. Podemos incluso utilizar las clases de formato de texto para aprender nuevos lenguajes, ya que la clase `DateFormat` conoce los nombres de los meses y los dias de la semana en unos 30 6 40 idiomas distintos.



18 Programación avanzada de Applet

Hasta ahora todos los applet vistos en este libro han sido demostraciones sencillas de clases o métodos de JAVA. Pero en la vida real, los applet son mucho más funcionales. Al escribir applet llevara sus habilidades creativas y de programación hasta lo más alto. Pero antes de empezar a soñar con tener su applet listado en la JARS (Servicio de Clasificación de applet de Java, http://www.jars.com) es necesario trabajar un poco más en los principios.

Este capitulo comienza con ejemplos de applet completamente orientados a GUI, para explicar despues la mejor manera de trabajar con el más reciente entorno de ejecucion de Java (JRE), con sus periféricos y con Java Web Start. También se incluyen trucos para reducir el tiempo de arranque de programas Java y del sistema.

Interfaces GUI exclusivas para Applet

Usando las técnicas vistas hasta aqui, se pueden escribir algunos applet muy funcionales. Tan pronto como se confíe exclusivamente en una interfaz GUI para controlar la funcionalidad de los applet, se tendrá el suficiente conocimiento para diseiiar e implementar programas como simples editores (programas gráficos, editores de texto, etc.) e incluso pequefios juegos. Para demostrar lo que es posible realizar usando unos cuantos botones y un pequeiio área display, se va a desarrollar ahora una calculadora cientifica. En este ejemplo, observaremos que es necesa-

rio crear e implantar un applet mucho mds complejo. Si sólo le interesa la tarea de implantación, encontrará una sección referente a este tema mds adelante en este mismo capitulo.

Ejemplo: Calculadora basada en stack

Comenzaremos con un primer análisis. ¿Cuál es la estructura más elemental de cualquier calculadora? Respuesta: un gran numero de botones y un display. Estos dos aspectos pueden formar un nivel superior de división para el applet que se va a escribir. A pesar de que las pantallas de las calculadoras modernas pueden implantar gráficos, e incluso imprimir texto, en este ejemplo nos limitamos a números simples en notación cientifica.

La primera caracteristica del applet de este ejemplo son los botones de la calculadora, utilizados para introducir números y seleccionar la función a realizar. En este caso, se proporcionan botones para los números del 0 al 9, ademds de las funciones matemáticas m6s comunes, tales como suma, resta, funciones trigonometricas (incluyendo seno y coseno) y almacenamiento de datos en la memoria.

Como sabemos, la notación cientifica utiliza una notación compacta para números muy pequeiios y muy grandes, expresando los números como un numero base (la mantisa) elevado a alguna potencia de diez. Como la calculadora tendrá esta apariencia cientifica, serd conveniente diseñarla como una calculadora basada en stack (pila de almacenamiento). La elección de la arquitectura es apropiada por su simplicidad comparada con calculadoras normales. Los stack son muy importantes en Java: la máquina virtual de Java (Java VM) es una arquitectura basada en stack con muchos parecidos con la calculadora que se presenta aquí.

La segunda caracteristica del applet de calculadora que se va a escribir es su display, o más concretamente, los digitos que muestran. Java tiene un número de fuentes que proporciona un par de estilos para digitos, pero ninguno se asemeja a los digitos de una calculadora real. Las calculadoras reales se han basado (hasta hace poco) en elementos de siete segmentos para construir los digitos y números (primero usando LED, diodos emisores de luz y posteriormente tecnologia LCD, de cristal liquido). Para dar a la calculadora una apariencia real se diseñarán caracteres propios de siete segmentos. La figura 18.1 muestra la convincente apariencia de la calculadora.

¿Qué es una calculadora de Stack?

Otro nombre para las calculadoras de stack es calculadoras de Notación Polaca Inversa (RPN, Reverse Polish Notation). Estas calculadoras se utilizaron en las misiones Apolo a la luna debido a que el sistema RPN permite una entrada rapida de calculos complejos. En una calculadora convencional se suman 10 y 20 introduciendo primero el número 10, pulsando la tecla de sumar,

introduciendo **20** y por último pulsando la tecla igual para calcular y mostrar el resultado. Con una calculadora RPN esta misma suma se realiza introduciendo **10,** pulsando la tecla **Intro,** introduciendo **20** y finalmente pulsando la tecla de suma. Cuando se pulsa esta última tecla, se muestra el resultado sin necesidad de pulsar la tecla igual. De hecho, en una calculadora RPN no existe la tecla igual (lo cual puede confundir y hasta frustrar a personas que no esten acostumbradas a usarlas).

La ventaja de una calculadora de stack es facil de entender cuando se trabaja con expresiones complejas que suponen priorizar subcalculos utilizando parentesis. En calculadoras RPN (que no tienen parentesis) el stack de la maquina proporciona automaticamente un mecanismo para priorizar y recordar. Es este simple mecanismo de ahorro de tiempo (al no tener que pulsar parentesis) lo que condujo a la NASA a utilizar calculadoras RPN en lugar de las convencionales.

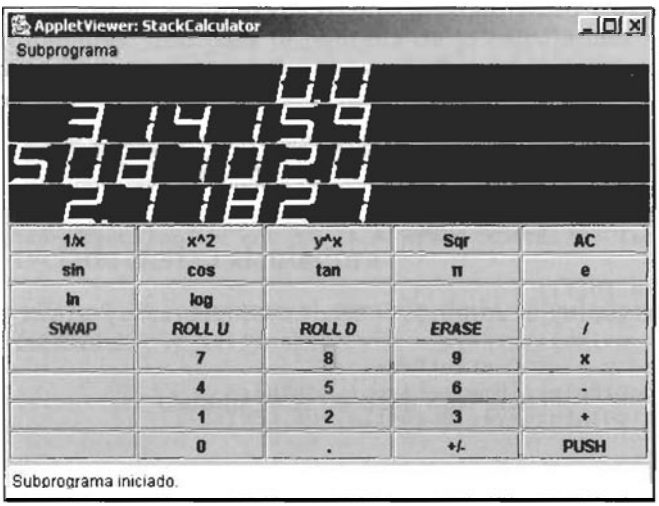


Figura 18.1. Una calculadora cientifica

El diseño de los applet refleja la funcionalidad y la división estructural de cualquier calculadora: Existe un display y un panel de botones. El programa de la Calculadora de Stack se subdivide en cinco clases:

- `StackCalculator`: Elmarcodelapplet.
- `KeyPanel`: El componente personalizado que exporta todas las teclas de la calculadora y sus funciones asociadas.
- `LCDDigitDisplay`: El componente personalizado que cambia con el número mostrado.

- `CalcStack`: Una clase que encapsula la arquitectura de la calculadora de stack.
- `SevenSegmentDigits`: Una clase que encapsula un conjunto de caracteres de los digitos de la calculadora.

Lo anterior no es mas que una aplicacion del principio de "divide y vencerás". Estas cinco clases subdividen claramente el programa y por tanto simplifican el problema de diseñar y escribir un applet de calculadora basada en stack. A pesar que este applet se puede considerar bastante trivial, en tkrminos de dificultad de programacion o complejidad, es ya en realidad demasiado grande para implementarlo como una unica clase. Como ya se ha visto antes, el dividir los problemas en pequeños subproblemas que se solucionan individualmente, implementando una clase para cada uno, es una buena práctica, asi como un modo de prevenir que las cosas puedan ir mal.

Resumen de los bloques de una calculadora de stack

El listado 18.1 presenta cada uno de los bloques empezando por el nivel superior.

Listado 18.1. StackCalculator

```
import java.awt.*;
import javax.swing.*;
public class StackCalculator extends JApplet {
  static final int STACK-DEPTH = 4; // Una calculadora snack con un número
                                    // N de entradar
  public void init() {
    LCDDigitDisplay lcdDisplay = new LCDDigitDisplay(STACK-DEPTH);
    KeyPanel calculatorKeys = new KeyPanel(8, 5, lcdDisplay);
    Container c = getContentPane();
    c.add(lcdDisplay, BorderLayout.NORTH);
    c.add(calculatorKeys, BorderLayout.CENTER);
  }
  public String getAppletInfo() {
    StringBuffer s = new StringBuffer();
    s.append("Stack Calculator\n");
    s.append((char)169); // Símbolo de Copyright
    s.append("1996-2001 SYBEX Ins., All Rights Reserved\n");
    return s.toString();
  }
  public String[][] getParameterInfo() {
    return new String[][] {};
  }
```

El codigo del applet que se encuentra en el nivel superior no es mucho mas sencillo que kste. Carece de casi cualquier código relevante y se concentra simplemente en ser un applet. Como tal, anula sólo tres mktodos `JApplet` inherentes al applet: `init()`, `getAppletInfo()` y `getParamenterInfo()`.

Los métodos `getAppletInfo()` y `getParameterInfo()` son opcionales, pero sirven para añadir un toque de perfeccionamiento al programa. Se utilizan cuando los usuarios seleccionan los buscadores Acerca de Applet o las opciones de menu de Información de Applet (los titulos de los menus son distintos en los distintos buscadores).

- El método `getAppletInfo()`: Normalmente devuelve un string que contiene información como el autor, la versión y el copyright (más lo que se quiera introducir).
- El método `getParameterInfo()`: Devuelve un array de `string` triplicado que describe todos los parámetros que el applet acepta por medio de los parámetros HTML `<PARAM NAME="parameterName" VALUE="parameterValue">`. Cada uno de los tres está constituido por el nombre del parámetro, su tipo y una breve descripción explicando la función del parámetro. Como en este ejemplo no hay parámetros, `getParameterInfo()` devuelve un array vacio.

El punto principal de este applet está en su método `init()`. Este método construye dos componentes: uno para la pantalla de la calculadora (`LCDDisplay`) y otro para el panel de teclas (`calculatorKeys`). Éstos se colocan luego en el applet como en cualquier calculadora real con las teclas debajo de la pantalla. Y esto es básicamente todo lo concerniente al applet, ya que éste no se encarga de controlar la lógica de la calculadora, cambiar la pantalla o las pulsaciones de botones. En definitiva, los objetos creados por el applet tienen autonomia propia.

El panel del teclado de la calculadora

La clase `KeyPanel` es realmente el corazón de la calculadora. El listado 18.2 muestra como funciona.

Listado 18.2. La clase KeyPanel

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
/**************************************************************************
 * La clase KeyPanel contiene todos los botones "panel" de la calculadora y
 * todas las operaciones matemáticas ejecutables desde los botones.
 **************************************************************************/
class KeyPanel extends JPanel implements ActionListener {
  int rowSize;
  LCDDigitDisplay display; // el display del stack
  CalcStack stack; // el stack dentro del display
  // Los botones de la calculadora se organizan en ocho filas de cinco
  // botones cada una.
  private String[] keyLabels = {
     "1/x", "x^2", "y^x", "Sqr", "AC", // fila 0
     "sin", "cos", "tan", "\u03c0", "e", // fila 1
     "ln", "log", " "  " ", " ", // fila 2
```

```
    display.redrawStack(true);

  private void cosine0 {
    stack.setAccumulator(Math.cos(stack.getAccumulator()));
    display.redrawStack(true);
  }
  private void tan() {
    stack.setAccumulator(Math.tan(stack.getAccumulator()));
    display.redrawStack(true);

  private void log() {
    if (stack.getAccumulator() > 1.0) {
      stack.setAccumulator(Math.log(stack.getAccumulator(I) /
Math.log(10.0));
      display.redrawStack(true);
    I

  private void ln() {
    if (stack.getAccumulator() > 1.0) {
      stack.setAccumulator(Math.log(stack.getAccumulator()));
      display.redrawStack(true);
    \
  }
  // Eleva el elemento siguiente en stack (NOS) a la potencia del elemento
en la parte superior del stack (TOS)
  private void power() {
    double pow = Math.pow(stack.getStackElement(1),
stack.getAccumulator());
    stack.drop();
    stack.setAccumulator(pow);
    display.redrawStack(true);
  }
  private void constantE() {
    stack.setAccumulator(Math.E);
    display.redrawStack(true);
  }
  private void constantPI() {
    stack.setAccumulator(Math.PI);
    display.redrawStack(true);

  private void inverse() {
    if (stack.getAccumulator() != 0.0) {
      stack.setAccumulator(1.0/ stack.getAccumulator());
      display.redrawStack(true);
    }
  }
  private void decimal() {
    display.decimal();
  }
  private void rollUp() {
    stack.rollUp(1);
    display.redrawStack true);
  }
  private void rollDown() [
    stack.rollDown(1);
    display.redrawStack(true);
```

```
  }
// Intercambia TOS y NOS
  private void swap() {
    stack.swap();
    display.redrawStack(true);
  }
  private void allClear() {
    stack.clearStack();
    display.redrawStack(true);
  }
  private void add() {
    double sum = stack.getAccumulator() t stack.getStackElement(1);
    stack.drop();
    stack.setAccumulator(sum);
    display.redrawStack(true);
  }
  private void times0 {
    double prod = stack.getAccumulator() * stack.getStackElement(1);
    stack.drop();
    stack.setAccumulator(prod);
    display.redrawStack(true);
  }
  private void subtract() {
    double diff = stack.getStackElement(1) - stack.getAccumulator();
    stack.drop();
    stack.setAccumulator(diff);
    display.redrawStack(true);
  }
  private void divide0 {
    double div = stack.getStackElement(1) / stack.getAccumulator();
    stack.drop();
    stack.setAccumulator(div);
    display.redrawStack(true);

  public String keyLabel(int row, int column) {
    return keyLabels[row*rowSize t column];
  }
```

Antes de centrarnos en lo que hace esta clase en la aplicacion, es conveniente ver lo que hace la clase `KeyPanel` a nivel de Java y AWT. Para empezar, `KeyPanel` es una subclase de `JPanel`. Esta es la razon por la que necesitamos que los botones de la calculadora se coloquen usando un administrador de distribución distinto al que utiliza el propio applet. Para especificar una distribucion diferente para un area, lo primero que debemos hacer es crear un nuevo `container`. Debido a que las teclas de la calculadora están universalmente distribuidas en filas y columnas, una elección natural para un administrador de distribucion es `GridLayout`. El constructor para la clase `KeyPanel` toma una referencia (read:link) al componente del display numérico que el applet ha creado. La clase `KeyPanel` necesita tener este enlace de modo que pueda comunicarse con el display para realizar diversas funciones. El diseño de esta calculadora indica que el panel de teclado (key panel) es la entidad activa que controla las cinco clases.

Lo primero que hace la clase KeyPanel, con la referencia al objeto del display, es pedirle a Cste que le envíe una referencia al stack de cálculo numérico que se encuentra incluido dentro del objeto del display. Como se puede ver, la parte principal de la máquina calculadora (el stack) pertenece en realidad al display. Esta decisión de diseño es muy arbitraria, el stack de la calculadora podría haber sido creado por el objeto del panel del teclado, o incluso por el propio applet. En cualquier opción que se elija, los objetos del panel, del display y del stack se necesitan comunicar entre si. El constructor KeyPanel representa posteriormente su función principal, que no es otra que crear los botones del panel de la calculadora a partir de un array constante de etiquetas de botones.

El siguiente método se encarga del proceso de los eventos del applet: el método `actionPerformed()` del KeyPanel. Cada botón tiene registrado un KeyPanel como un listener de acción. Esto significa que al pulsar cualquier botón, se llama al mismo método `actionPerformed()`. El método consiste en una gran selección de constructores `if` que intentan determinar el origen del evento del botón pulsado. El método `keyLabel` se usa para mejorar la legibilidad del programa. Éste toma dos coordenadas en dos dimensiones (filas y columnas) y devuelve la etiqueta para el botón localizado en esa posición. Esto resulta m6s sencillo que llamar al array unidimensional `keyLabels`.

Si el método captura una pulsación de un botón en una de las teclas de digitos (del 0 al 9), convierte la etiqueta `String` del digito en una integral equivalente y le indica al display que lo pase por medio del método `enterDigit()`. (El panel del teclado no tiene que ver con los tecnicismos de entrada y edición de datos, sino que se trata de un elemento totalmente independiente.) Asimismo, `actionPerformed()` relaciona cada uno de los botones con un método que encarna su función. Por ejemplo, cuando se pulsa el botón situado en la posición fila=0, columna=4 la función AC (borrar todo) se ejecuta.

Entre los métodos llamados para responder a las pulsaciones de botón está el método `cosine()`. Viendo el funcionamiento de este método se entenderdn mejor los demás, por su gran similitud. Las funciones matemáticas de la calculadora se implementan a partir de la clase Math. La implementación concreta del stack se oculta del KeyPanel mediante la clase `CalcStack` que incluye todos los detalles. La función de coseno muestra cómo las funciones y comportamientos se reunen de nuevo.

El objeto `stack` proporciona los métodos `getAccumulator()` y `setAccumulator()` para recuperar y establecer los valores actuales para el elemento TOS (en la parte superior del stack). Mediante estos dos métodos, se hace innecesario almacenar el coseno del elemento TOS en el stack.

Nota

La etiqueta para la tecla pi, va especificada corno \u03c0 que es el caracter Unicode para el sirnbolo griego pi.

La pantalla de la calculadora

El siguiente elemento de la calculadora que se va a describir es el `LCDDigitDisplay`, cuya definición aparece en el listado 18.3.

Listado 18.3. La clase LCDDigitDisplay

```
import java.awt.*;
import java.util.*;
import javax.swing.*;
class LCDDigitDisplay extends JComponent {
  CalcStack  stack; // el stack de nfimeros
  int stackDepth; // el número de elementos que contiene el stack
  Dimension displaySize, oldSize; // dimensión del Area
  Dimension digitSize;
  SevenSegmentDigits  lcd = null; // ref a los dígitos LCD de 7 segmentos
  boolean freshNumber = true;
  int decimalPosition = 0;
  LinkedList undoStack; // Sistema de deshacer para eliminar los dígitos
  private static final int LINE_GAP = 2;
  public LCDDigitDisplay(int depth) {
    stackDepth = depth;
    stack = new CalcStack(stackDepth);
    newNumber(); // reset all entry modes
    setPreferredSize(new Dimension(100, 120));
  }
/** Reestablece es estado de edición y construcción de números **/
  private void newNumber() {
    freshNumber = true;
    decimalPosition = 0;
    undoStack = new LinkedList();
  }
/** Se introduce un digit  extra. Dependiendo del modo actual, se añadirá
  * este digito delante o detrás de la coma decimal. **/
  public void addDigit(int digit) {
    int changed = 1; // número de líneas del stack que deben dibujarse de
                     // nuevo
    rememberUndo(); // Recuerda el valor actual por si se borran digitos
    if (freshNumber) { // nuevo número
      stack.rollUp(1);
      stack.setAccumulator(0.0); // Elimina A
      freshNumber = false; // Se acumulan los digitos a partir de ahora
      changed = stackDepth; // todo el stack necesita redisefiarse
    }
    // Si aún no hemos intrnducido decimales, podemos desplazar el número
    // un dígito hacia arriba multiplicándolo por 10 y añadiendo el digito
    double acu = stack.getAccumulator();
    if (decimalPosition == 0) {
      stack.setAccumulator(acu * 10.0 + digit);
    } else {
      // Para añadir un decimal debemos añadir una fracción en el lugar del
      // decimal que hayamos alcanzado
      stack.setAccumulator(acu + ((float)digit)/decimalPosition);
      decimalPosition *= 10;
    }
```

```
    redrawStack(false);
  }
  public void removeDigit() {
    undo(); // reestablece el número y el estado al valor anterior
    redrawStack(false);
  }
/**********************************************************
 * Estarnos empleando un stack que contiene valores previos y estados de
 * edición para poder regresar fácilmente al nhrnero anterior si el usuario
 * borra un dígito.*
 **********************************************************/
  public void rememberUndo() {
    undoStack.addFirst(new Double(stack.getAccumulator()));
    undoStack.addFirst(new Boolean(freshNumber));
    undoStack.addFirst(new Integer(decimalPosition));
  }
  public void undo() {
    if (!undoStack.isEmpty()) {
      decimalPosition = ((Integer) undoStack.removeFirst()).intValue();
      freshNumber = ((Boolean) undoStack.removeFirst()).booleanValue();
      stack.setAccumulator((Double)
undoStack.removeFirst()).doubleValue());
    } else {
      newNumber();
      stack.setAccumulator(0.0);
    }
  }
  /** El usuario pulsó "." por lo que a partir de ahora, se añaden
decimales.*/
  public void decimal() {
    if (decimalPosition == 0) {
      decimalPosition = 10; // decenas, centenas, etc.
    }
  }
  public CalcStack getStack() {
    return stack;
  }
  public void redrawStack(boolean newNumber) {
    if (newNumber) {
      newNumber();
    }
    repaint();
  }
  public void paintComponent(Graphics g) {
    displaySize = getSize();
    //Si no hemos iniciado la clase de digitos LCD o si el tamaño de la
    //pantalla ha carnbiado
    // inicializa digitos LCD
    if ((lcd == null) || (displaySize.width != oldSize.width) ||
        (displaySize.height != oldSize.height)) {
      oldSize = displaySize; // recuerda el nuevo tamaño
      digitSize = new Dimension(displaySize.width/12,
        (displaySize.height/stackDepth) - LINE-GAP);
      // Crea digitos escalados LCD
      lcd = new SevenSegmentDigits(digitSize);
    }
```

```
    for(int i=0, y; i<stackDepth; i++) {
      y = (stackDepth-i-1) * (digitSize.height + LINE-GAP);
      g.fillRect(0,y,displaySize.width, digitSize.height);
      lcd.drawNumber(stack.getStackElement(i), g, 0, y);
    }
  }
```

La clase `LCDDigitDisplay` es la responsable de la apariencia de los digitos de la calculadora. Es también una subclase de `JComponent`, de modo que se puede dibujar dentro de su area. Todo lo que hace el constructor de la clase es crear el stack numérico y reestablecer las variables del modo de entrada de datos y determinar el tamaiio de los componentes. El constructor no es, en este caso, donde se oculta toda la acción, sino que ésta se extiende sobre las dos partes de la clase: una parte está constituida por los métodos `addDigit()` y `removeDigit()`, y la segunda parte está formada por el método `redrawstack()` y otros relacionados con el redibujado (incluido el método anulado `JComponent paintComponent)`.

Consideremos por un momento la manera en que una calculadora sencilla funciona. No consiste unicamente en pulsar la tecla del digito y ver el numero en la pantalla. Si se utiliza un acumulador para mantener el valor actual, ¿cómo se realiza la formación del numero a partir de los digitos introducidos? Digamos que en la pantalla aparece el numero 12. Al pulsar el 7, tendremos el numero 127, ya que el primero y el segundo numero se combinan para formar uno solo, multiplicando 12 por 10 y afiadiendo el 7. Pero si en la pantalla tenemos el numero 3,14 e introducimos el 1, el numero 3,14 se multiplica por 10, lo que hace 31,4, y se aiiade el 1, lo que resulta en 32,4. Para que esto no suceda, debemos ajustar el algoritmo.

Cuando se pulsa la tecla de punto decimal, el algoritmo de entrada de datos deberia introducir un nuevo modo: modo de entrada decimal. Cualquier digito que se introduzca antes, sera dividido por alguna potencia de diez, y el resultado se añadirá al acumulador actual sin multiplicarlo de antemano por diez. La potencia de diez se determina por la posición correspondiente al siguiente decimal. Este algoritmo está implementado por el método `addDigit()`. (El test para `freshNumber` no es parte de este algoritmo. `freshNumber` permite comenzar desde 0.0 cada vez que se introduzca un primer digito).

Las calculadoras reales no sólo nos permiten introducir numeros, como ya se ha comentado, sino que además son capaces de ignorar los pequeiios errores que se produzcan. Estas calculadoras tienen una tecla de borrado para eliminar el ultimo digito introducido. El applet de la calculadora de stack también posee una tecla para este propósito. Una manera posible para implementarlo podría ser deshacer el ultimo digito aiiadido por cualquier modo inverso, utilizando los mismos datos de entrada que el método `addDigit()`: el valor del acumulador y la variable decimal de entrada (`decimalPos`).

A pesar de que esto es posible, es extremadamente complejo e innecesario. En su lugar, se le puede engafiar recordando el valor del acumulador (y su estado de entrada decimal asociado) antes de cambiarlo, y de esta manera restaurarle a su estado original si le llega un "digito de borrado".

Este mecanismo se basa en otro stack (`java.util.stack`) para almacenar cada paso que se pueda deshacer. Para mas detalles sobre `rememberUndo()` y `undo()` revise de nuevo el listado 18.3.

Tanto `addDigit()` como `removeDigit()`, necesitan actualizar el display para reflejar los cambios producidos en el numero que se esta calculando o editando. Esto se realiza mediante `refreshStack()`, que le indicara a `paint()` que actualice el display que, por su parte, llamard al método `paintComponent()`.

Como podemos observar, el método consiste en un bucle que simplemente redibuja los numeros suministrados por el Stack. Para renderizar los digitos de los valores, se podría haber utilizado `drawstring()` y alguna fuente estándar de Java. No obstante, los requerimientos del proyecto inicial eran que se utilizarian caracteres de siete segmentos al estilo LCD. Por ello, `drawstring()` se ha reemplazado por `lcd.drawNumber()`. El objeto `lcd` es parte de la clase `SevenSegmentDigit`. Esta clase tiene la responsabilidad de generar (en tiempo de ejecución) un conjunto de caracteres de siete segmentos. Puesto que resultaria muy ineficaz generar estas imágenes de los digitos sobre la marcha (es decir, como `drawNumber()` los requiere) el objeto `lcd` los crea y almacena para usarlos posteriormente como parte de su propia inicializacion. Éste es el motivo por el que la construcción de el objeto `lcd` es una condicional de `refreshStack()`. La declaración `if` comprueba si el objeto `lcd` ya ha sido creado o si el Area de `JComponent` ha cambiado sus dimensiones. En ambos casos, el conjunto de caracteres necesita regenerarse (para ajustarse a las dimensiones del `JComponent`).

Posteriormente se verd qu6 ocurre cuando se invoca al constructor del objeto `lcd`. Pero antes, regresemos al código para continuar viendo aspectos más fundamentales sobre el funcionamiento de la calculadora.

El Stack de la calculadora

La clase `ClacStack`, descrita en el listado 18.4, se utiliza como font-end de la estructura de datos del stack numérico que utiliza la calculadora.

Listado 18.4. La clase CalcStack

```
/*****************************************************************************
 * La clase CalcStack incluye el stack numérico y todas sus manipulaciones
 * no matemáticas (sólo las manipulaciones del stack)
 * No se han utilizado ni java.util.Stack ni LinkedList ya que el stack
 * requerido no puede tener una capacidad infinita y porque además sólo
 * necesitarnos dobles y no objetos completos.
 *****************************************************************************/
public class CalcStack {
  double stack[]; // el stack en el corazón de la calculadora
  int stackDepth;
// Constructor: Construye un stack de la capacidad indicada
  public CalcStack(int depth) {
    stackDepth = depth;
    stack = new double[stackDepth];
    clearStack();
```

```
  }
  public void pushValue(double x) {
    rollUp(1);
    setAccumulator(x);
  }
  public void rollUp(int times) {
    double lastVal;
    for (int r=0; r < times; r++) {
      lastVal = stack[stackDepth-1];
      for (int i = stackDepth-2; i >= 0; i--) {
        stack[i+1] = stack[i];
      }
      stack[0] = lastVal;

  }
  public void rollDown(int times) {
    rollUp(stackDepth-times);

  public void swap() {
    double temp;
    temp = stack[0];
    stack[0] = stack[1];
    stack[1] = temp;
  }
  public void clearStack() {
    for (int i=0; i < stackDepth; i++) {
      stack[i] = 0.0;
    }
  }
  public void drop() {
    rollDown(1);
    stack[stackDepth-1] = stack[stackDepth-2];

  public void setAccumulator(double x) {
    stack[0] = x;

  public double getAccumulator() {
    return stack[0];

  public double getStackElement(int n) {
    return stack[n];
```

La declaración `double stack[]` en la sección de variables, al principio de la clase `CalcStack`, es la base de la estructura de datos para todo el programa. El applet de la calculadora de Satck construye toda la funcionalidad alrededor de este array de valores `double`. El applet se apoya indirectamente en `double stack[]` ya que la clase `CalcStack` interviene en todos los accesos a la estructura de este stack. Esta mediación se hace a través de métodos suministrados por la clase:

```
class CalcStack {
  public CalcStack(int stackDepth)
  public void clearStack()
```

```
public void drop()
public double getAccumulator()
public double getStackElement(int n)
public void pushValue(double x)
public void rollDown(int times)
public void rollUp(int times)
public void setAccumulator(double x)
public void swap()
```

Esta claro, por la ausencia del método `pop()`, que este stack tiene propiedades que no son propias de un stack. Esto se debe a que las caracteristicas de un stack se han modificado para que sean más funcionales en una aplicación de cálculo:

- Un `CalcStack` tiene siempre un tamaño fijo. Dispone de un número de espacios N que se completan con numeros. Los stack reales, por el contrario, pueden estar vacio, medio vacios o llenos.

Nota

En este ejemplo se emplean cuatro espacios para numeros, aunque es posible modificar el tamaiio cambiando la constante `STACK_DEPTH` del applet.

- Un `CalcStack` permite que su contenido se desplace hacia arriba o hacia abajo. La mejor manera de comprobarlo, es introducir un número en el applet de la calculadora y presionar las teclas **ROLL** U y **ROLL** D.
- Al eliminar el valor TOS, mediante un `getAccumulator()` y `drop()`, el ultimo espacio para numero mantiene su valor original y no se cambia a 0.0.

Se podría también utilizar un `pop()`, que combinara un `getAccumulator()` y un `drop()`, pero este otro diseño es más eficaz. Por ejemplo, para sumar dos números, una solución seria utilizar el siguiente algoritmo:

```
a   pop()
b = pop()
sum = a + b
push(sum)
```

Esta solución requiere tres stack completos: dos para los métodos `pop()`, y uno para el método `push()`. Sin embargo, el método `add()` requiere un único stack. Aunque esto no parezca muy importante con cuatro elementos, cobra mucha importancia cuando tenemos 20 6 50.

El subsistema de dígitos de siete segmentos

La única clase que queda por discutir es `SevenSegmentDigits`, que se describe en el listado 18.5.

Listado 18.5. La clase SevenSegmentDigits

```
/**************************************************************************
 * Class SevenSegmentDigits contiene los digitos escalables que pueden
 * renderizarse de manera individual o como valores dobles completos
**************************************************************************/
import java.awt.+;
import java.awt.image.*;
import java.util.StringTokenizer;
public class SevenSegmentDigits {
  protected Image digits[]; // Contiene las imágenes cornputadas para los
digitos 0 al 9
  protected Dimension scale;
  static final int NUM_DIGITS = 10 t 1; // un espacio extra para el símbolo
"-"
  static final int on = 207;
  static final int oy  = 53;
  static final int segW = 230;
  static final int segH = 227;
  int originalWidth;
  static final double italicPercent = 0.15; // anchura necesaria para
utilizar cursivas
  int italicRange;
  int decimalW, decimalH, decimalXoff, decimalYoff;
   // Lns vertices que forman la celda para mostrar los siete segmentos
  static final int points[] = {207,53, 385,53, 207,280, 395,280, 240,75,
360,75,
    240,258, 360,258, 207,159, 385,159, 240,144, 360,144, 240,176,
360,176);
    // Cada segmento es un polígono compuesto de vertices
  static final int segDefs[][] = {{0,1,5,4}, {0,4,10,8}, {1,5,11,9},
    {8,10,11,9,13,12}, {8,12,6,2}, {9,13,7,3}, {6,7,3,2}};
  // Cada dígito es una colección de segmentos "ON"
  static final String[] digitDefs= {"ABCEFG", "CF", "ACDEG", "ACDFG",
"BCDF", "ABDFG",
    "RDEFG", "ACF", "ABCDEFG", "ABCDF", "I?"}; // el signo menos es "digit
10"
/**************************************************************************
 * Constructor: percibe el tamaño al que han sido escalados los digitos y
   los renderiza dinámicamente
 **************************************************************************/
  public SevenSegmentDigits(Dimension size) {
    scale = size;
    originalWidth = scale.width;
    scale.width = (int)(scale.width * (1.0 - italicPercent));
    italicRange = originalWidth - scale.width;
    digits = new Image[NUM_DIGITS];
    for(int digit=0; digit < NUM_DIGITS; digit++) {
      digits[digit] = renderDigit(digit);
    ]
    // La imagen del puntn decimal se calcula a partir del tamaño del
    // dígito
    decimalW = scale.width/6;
    decimalH = scale.height/10;
    decimalXoff = scale.width - decimalW;
```

```
    decimalYoff = scale.height - decimalH;
  }
/*************************************************************************
 * Renderiza un dígito de 7 segmentos desde la definición de la lista de
 * segrnentos.
 * Lo devuelve como una imagen a almacenar.
 *************************************************************************/
  protected Image renderDigit(int number) {
    Image digitImage;
    Graphics g;
    String segments;
    digitImage = new BufferedImage(originalWidth, scale.height,
      BufferedImage.TYPE_INT RGB);
    g = digitImage.getGraphics();
    segments = digitDefs[number];
    for(int seg=0; seg < segments.length(); seg++) {
      int segIndex = segments.charAt(seg)-"A";
      renderSegment(g, segIndex);
    }
    return digitImage;
  }
/*************************************************************************
 * Renderiza el polígono que forma el segmento en el contexto de gráficos
 * dado.
   Se normalizan los estados del segmento entre 0.0 y 1.0
   Se escalan al tamaño requerido
 * Se renderiza el seqmento como un polígono
 *************************************************************************/
  protected void renderSegment(Graphics g, int segment) {
    int segDef[] = segDefs[segment];
    Polygon p = new Polygon();
    for(int vertex=0; vertex < segDef.length; vertex++) {
      int v = segDef[vertex];
      int x = points[v * 2 + 0] - ox; // traslada al origen
      int y = points[v * 2 t 1] - oy;
      double normX = ((double)x) / segW; // lo normaliza a entre 0.0 y 1.0
      double normY = ((double)y) / segH;
      int polyX = (int)(normX * scale.width); // lo escala al tamaño
                                              // requerido
      int polyY = (int)(normY * scale.height);
      polyX += (1.0 - normY) * italicRange; // dígitos en cursiva
      p.addPoint(polyX, polyY);
    }
    g.fillPolygon(p);
  }
/*************************************************************************
 * digitImage() da a los clientes acceso a los dígitos renderizados
 *************************************************************************/
  public Image digitImage(int digit) {
    return digits[digit];
  }
/*************************************************************************
 * drawNumber() renderiza un valor doble en el contexto de gfx (x,y)
 *************************************************************************/
  public void drawNumber(double number, Graphics g, int x, int y) {
    String doubleStr = clean(Double.toString(number), 5);
```

```
    boolean isNegative = (doubleStr.charAt(0) == "-");
    int exponentPos = doubleStr.indexOf("E");
    int decimalPos = doubleStr.indexOf(".");
    int significantDigits = 6; // Necesitarnos toString() para generar 6
                               // digitos significativos
    int digitIndex = 0;
    int digitSlot;
    int digit;
    int digitX = 0 ;
    Color savedColor = g.getColor();
    g.setColor(Color.white);
    significantDigits = doubleStr.length();
    // Si el número es negativo, renderiza un menos en el espacio mas hacia
    // la izquierda
    if (isNegative) {
      g.drawImage(digitImage(10), x, y, null); // dibuja un menos
      significantDigits--;
      digitIndex = 1;
    }
    if (exponentPos != -1) {
      significantDigits -= (doubleStr.length() - exponentPos);
     // posiciones del exponente ("E99")

    if (decimalPos != -1) {
      significantDigits--; // un punto decimal no es tampoco un dígito
    }
    // Calcula el espacio del primer digito para que todos los números se
    // alineen
    digitSlot = 7 - significantDigits;  // espacio 1 para los dígitos más
                                        // siqnificativos
/* Código depurador. Se puede habilitar al ampliar este código
   System.out.println("significantDigits " + significantDigits);
   System.out.println("digitIndex " + digitIndex);
   System.out.println("digitSlot " + digitSlot);
   System.out.println("isNeg " + isNegative);
   System.out.println("Exp " + exponentPos);
*/
    // El bucle renderizador de mantisa: renderiza todos los digitos
    // significativos
    while (significantDigits != 0) {
      if ((digit = doubleStr.charAt(digitIndex++)) != ".") {
        digit -= "0";
        digitX = x + digitSlot*originalWidth;
        g.drawImage(digitImage(digit), digitX, y, null);
        significantDigits--;
        digitSlot++;
      } else { // renderiza el punto decimal en el mismo espacio que el
               // último digito
        g.fillRect(digitX + decimalXoff, y + decimalYoff, decimalW,
decimalH);

    }
    // Si el número contiene un exponente, lo renderiza también
    if (exponentPos != -1) {
      if (doubleStr.charAt(exponentPos + 1) == "-") { // dibuja un menos
        g.drawImage(digitImage(10), x + 8*originalWidth, y, null);
```

```
        exponentPos++; // Ignora el menos

      digitIndex = exponentPos + 1; // ignora "E" o "E"
      digitSlot = 9;
      while (digitIndex < doubleStr.length()) { // renderiza "ENN"
        digit = doubleStr.charAt(digitIndex++) - "0";
        digitX = x + digitSlot*originalWidth;
        g.drawImage(digitImage(digit), digitX, y, null);
        digitSlot++;
      }
    }
    g.setColor(savedColor);
  }
  private String clean(String src, int nDecimals) {
    String clean = "";
    StringTokenizer st = new StringTokenizer(src, "E.", true);
    boolean atDecimal = false;
    while(st.hasMoreTokens()) {
      String token = st.nextToken();
      if (atDecimal) {
        if (token.length()
> 5 && Character.isDigit(token.charAt(0))) {
          token = token.substring(0, 5);
        }
        atDecimal = false;
      } else if (token.equals(".")) {
        atDecimal = true;
      }
      clean += taken;
      // depura la línea System.out.println(clean);
    }
    return clean;

}
```

Aunque la función fundamental de esta clase es proporcionar una apariencia mds realista (a la hora de dibujar los numeros) usando una fuente estdndar AWT y `drawstring`, la clase `SevenSegmentDigits` contiene mucho código interesante.

Definición de los caracteres vectoriales

Debido a que el objetivo de este subsistema es obtener digitos escalables (de modo que el tamaño del display pueda variar de acuerdo a las dimensiones del applet), es mejor rechazar el método de almacenar cada digito como un GIF externo, o cualquier otro tipo de archivos de imdgenes. En este caso, una fuente basada en bitmap no es flexible ni se puede escalar fdcilmente. Es mds, cuando se fuerza, el resultado del redimensionado es de muy poca calidad. Es su lugar se utilizan definiciones de vectores para los caracteres. Las fuentes escalables pueden definirse en espacios muy compactos, almacenando algunas definiciones matemdticas (vectores) de sus líneas exteriores.

La figura 18.2 muestra la estructura de un display de siete segmentos que se utiliza en casi todas las calculadoras.

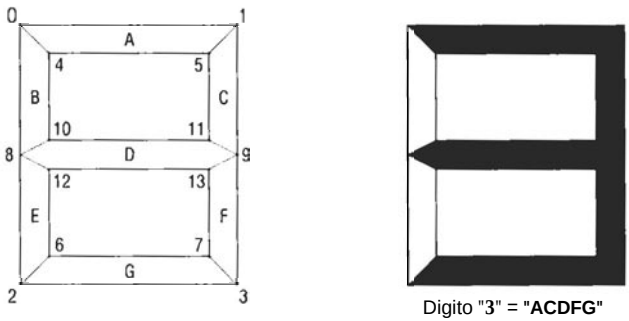


Figura 18.2. Display de siete segmentos y una configuración para el digito 3

Si se describe cada digito como una colección de segmentos en "on" (es decir, segmentos iluminados que se necesitan para representarlo) y cada segmento como la lista de vértices que forma el contorno de un polígono para un segmento, se tendrd una definición compacta y flexible para el conjunto de todos los caracteres. De este modo, la union de los arrays `points[]`, `segDef[]` y `digitDefs[]` definen el conjunto de caracteres de la calculadora.

El array `points` mantiene el par (x,y) de los nódulos (o vertices) a partir de los cuales se construye el display. El valor exacto de estas coordenadas se consigue utilizando un programa simple de pintar, y como se verd más adelante, estos valores son los mds irrelevantes para el resto del código. El array `segDefs` define cada segmento como una lista de vértices (un vértice es codificado como un índice dentro del array `points[]`). De la misma manera, el array `DigitDefs[]` define los digitos como una lista de segmentos. Pero, para que sea legible, se utilizan letras en lugar de utilizar indices de los arrays de los segmentos. (Mds tarde, se simplificará el código convirtiendo las letras en indices numéricos.) Los 3 digitos para el ejemplo, están codificados como la lista de segmentos A-B-C-D-F-G.

> **Nota**
>
> Las definiciones de los digitos estan en fuente normal (roman) y no en cursiva. Al ejecutar el applet `StackCalculator`, observaremos que los digitos aparecen en cursiva, lo que se consigue algoritmicamente durante la etapa de renderizacion.

Creación de imágenes a caché

El constructor de la clase `SevenSegmentDigits` admite el argumento `Dimension`, que se emplea para redimensionar los digitos hasta el tamaño deseado. El mecanismo que se utiliza consiste en dibujar cada digito en un objeto `Image` separado fuera de la pantalla. Estos objetos `Image` pueden ser almacenados por la clase, y de este modo, el método `drawNumber()` puede volver a llamarlos y dibu-

jarlos en pantalla sin necesidad de realizar ninguna operación posterior. (Es la misma técnica utilizada por todos los subsistemas modernos de dibujo de fuentes en las impresoras láser y sistemas operativos).

Tal y como se vio en capitulos anteriores, para crear una imagen fuera de pantalla, necesitamos primero crear un `BufferedImage`. Si se opta por tener un **`Component`** manual, se puede usar su método `createImage()`. Sin embargo, como la clase `SevenSegmentDigits` no es un **`Component`**, no se podrá usar "`createImage()`" en sí misma. En su lugar, se utilizará la clase `BufferedImage` para crear una estructura **`Image`** que se pueda utilizar.

Una vez que el constructor ha tomado nota de los argumentos que ha recibido (haciendo copias locales), realiza un calculo que ajusta las dimensiones de los digitos que debe utilizar el resto del código. El ca'lculo debe hacerse con el algoritmo de cambiar el formato de los digitos a cursiva. En la figura 18.3 se muestra por qué se requiere un ajuste.

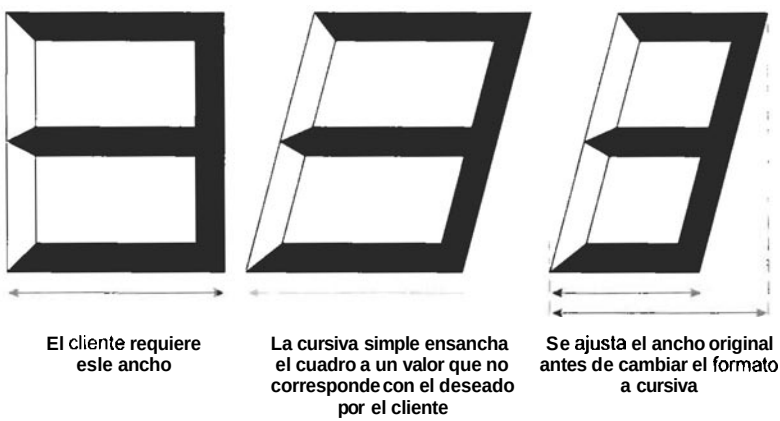


Figura 18.3. Ajuste de caracteres necesario para utilizar cursivas

El constructor ajusta el ancho requerido a un -15 por ciento para dejar de esta manera espacio para los digitos en cursiva, que deberán encajar perfectamente dentro de los limites del cuadro de las dimensiones deseadas.

La diferencia de anchura que hay entre el digito original y el digito nuevo es el margen posible de desviación que se puede variar en funcidn de la altura que tenga el cara'cter.

Una vez que este paso se ha realizado, el objeto `SevenSegmentDigits` esta preparado para renderizar cada digito y almacenarlo en un array de objetos **`Image`** llamado **`digits`** en el programa del ejemplo. Hay que tener en cuenta que se dibujan 11 digitos. El signo menos (segmento D) se renderiza también, como el digito undécimo.

El punto decimal requiere otro procedimiento. Puesto que el punto decimal nunca ocupa un espacio de digito sino que se coloca dentro del espacio del digito de las unidades, necesitamos renderizar un punto decimal mediante **`fillRect()`**.

Renderiracion de los dígitos

El método `renderDigit()` llamado por el constructor es muy sencillo. Crea un nuevo objeto Image en el que dibujar el digito, obtiene el contexto `Graphics` de Image, y dibuja cada segmento del digito requerido. El método `renderSegment()` en el que se basa `renderDigit()`, es el método con el código más interesante.

El método `renderSegment()` esencialmente construye un poligono y lo renderiza mediante `fillPolygon()`. El cálculo de las coordenadas del poligono es la base del algoritmo que escala pone en cursiva el digito vectorial. Las coordenadas de los vertices tienen que normalizarse para que se ajusten a un cuadro imaginario de 1,0 de ancho por 1,0 de alto. Este paso facilita la tarea de escalar los digitos a cualquier dimensión requerida por el usuario. Las variables ox, oy, `segW`, y `segH` utilizadas en este proceso se definen como constantes al comienzo de la clase. Sus valores se obtienen empiricamente a partir del sistema arbitrario de coordenadas utilizado para definir los vértices de los digitos.

Nota

No tiene importancia que el punto 0 este situado en 207,53. Lo importantes es que este punto sirva de origen para todos los demas puntos, por lo que debemos aplicar esas coordenadas a todos los puntos antes de aplicar el escalado.

El código `polyX += (1.0 - normY) * italicRange;` esdondeseaplicael atributo de cursivas. Este atributo inclina los puntos horizontalmente, afectando sus coordenadas x, como una funcion de la altura del digito. Una vez añadidas todas las coordenadas calculadas al objeto `Polygon` utilizado para guardarlas, el método `renderDigit()` emplea `fillPolygon()` para dibujar el segmento escalado en cursiva.

Por último, el método `drawNumber()` dibuja los distintos digitos para representar un `double` pasado como un argumento.

Como habrá podido observar, se puede escribir gran cantidad de código applet basándonos meramente en GUI para activar las diferentes partes del programa. Sin embargo, llegará el momento en que escriba un applet que no siga esta regla y que arrojará muchos fallos y comportamientos inesperados, que tendrá que solucionar investigando con más profundidad.

Implantación de Applets

Una vez finalizado el desarrollo del programa, hay que implantar `StackCalculator`, basado en la clase `javax.swing.JApplet` que es distinto de los applet explicados anteriormente y puede causar complicaciones al intentar ejecutarlo en el

buscador. Esto se debe al requisito de la clase `JApplet` de que las clases Swing este'n presentes. Los ambientes de tiempo de ejecución por defecto de la mayoría de los buscadores (desde Internet Explorer 4.x hasta 6.x y Netscape Navigator/ Communicator hasta la versión 4.7x) se basan en los ambientes de Java 1.1, por lo que el applet no funcionarán en ellos debido a que Swing no se estandarizó hasta Java 1.2. Estos fallos no se limitan a los applets basados en Swing, sino que ocurrirán tambie'n cada vez que abramos un applet que utilice un elemento de la biblioteca de Java que sea más reciente que el JRE 1.1.

Solucionar este problema no es tan dificil como puede parecer. Pero antes, debemos considerar si es conveniente introducir esta complicación. En un programa tipico basado en Internet, donde muchos usuarios se conectan a la Red con velocidades muy lentas, es conveniente utilizar unicamente los elementos de la plataforma de tiempo de ejecución integrada de Java 1.1. Por otro lado, en un ambiente de intranet o en una situación en la que tengamos un mayor control sobre la base de datos del usuario, podemos utilizar las plataformas mtis recientes. Además de utilizar elementos de la versión más reciente de tiempo de ejecución, los procesos de pruebas se simplificarán, ya que sólo será necesario comprobar el tiempo de ejecución esttindar más reciente.

La ayuda para utilizar la plataforma de tiempo de ejecución mtis reciente viene en dos formas soportadas por Sun: el periférico de Java, que mantiene la ejecución del applet e un buscador, y Java Web Start, que ejecuta los programas fuera del buscador, en un ambiente local.

El periférico de Java

El periferico de Java fue la respuesta inicial de Sun para proporcionar un JRE actualizado dentro de los buscadores de usuarios. Este periferico comenzo llamándose `Activator`, y los desarrolladores tenian que implantar sus applets con una etiqueta distinta de `<APPLET>`. Las páginas web se ejecutan mediante una herramienta llamada `HTMLConverter`, que convierte las etiquetas `<APPLET>` en etiquetas `<OBJECT>` y `<EMBED>`, para reemplazar todos los accesos de applet por el acceso a un control ActiveX en Internet Explorer y a un control periférico de Netscape en Communicator.

Con la versión 1.3.1_01a del periférico, dejó de utilizarse el `HTMLConverter`, ya que las últimas versiones del perife'rico pueden reemplazar directamente el tiempo de ejecución de Java dentro del buscador para Internet Explorer 4.x hasta 6.x, y para el buscador de Netscape 6.0 (aunque no para las versiones 4.x). Esto significa que despue's de instalar el periférico dentro del buscador, los usuarios pueden ejecutar applets basados en versiones de Java mtis recientes que la ayuda integrada y los applets se especifican con la etiqueta `<APPLET>`.

Obtención e instalación del periférico de Java

Los desarrolladores, a diferencia de los usuarios, tienen la opción de instalar el periférico de Java junto con el SDK. Si queremos utilizar elementos de versiones

más recientes que la versión 1.1, debemos proporcionar un enlace adecuado en la página web.

Una vez instalado el periferico, debemos ejecutar el applet que requiere los elementos rnis recientes o el que aún emplea los anteriores.

En un ambiente corporativo, un departamento de tecnologia de la información instalaria el periferico en la imagen de instalación estándar de la corporación, para que los usuarios pudiesen trabajar con él. En muchos casos, esto es suficiente para habilitar un applet basado en Swing. Puesto que el periferico ocupa rnis de 5MB para cualquier versión de Microsoft Windows e incluso más para las plataformas Unix, es preferible no obligar a los usuarios a descargarlo. En la página http://java.sun.com/ products/ plugin/ encontrará más información para instalar el periférico de esta manera, incluido una opción de instalación silenciosa para descargas masivas.

Si, por el contrario, desea proporcionar ayuda en su página web para instalar el periferico, dirija a los usuarios a la pigina http://java.sun.com/getjava/ que incluye instrucciones de instalación más orientadas hacia el usuario.

Asumiremos que se ha instalado el periférico con el SDK de Java, por lo que no se incluyen instrucciones concretas para su instalación. Si aun no lo ha instalado, visite la página http://java.sun.com /getjava.

Si es usuario de Windows trabajando con Internet Explorer, asegúrese que a la hora de instalar el periferico selecciona el buscador adecuado, como se ve en la figura 18.4.

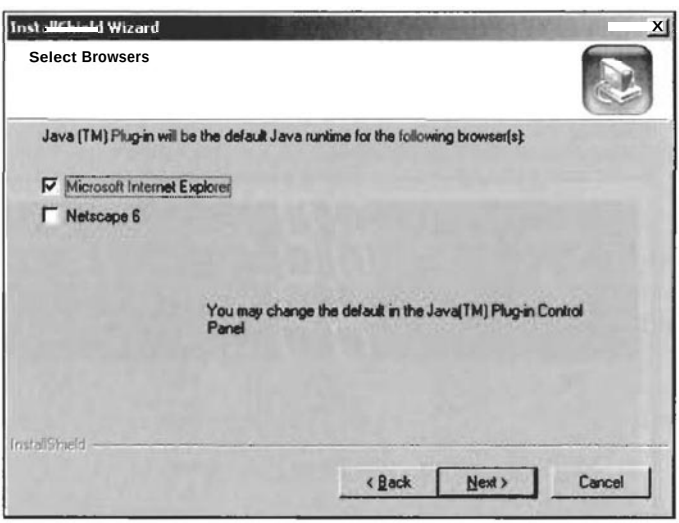


Figura 18.4. Selección del buscador del periferico de Java

Puede que desconozca si el periferico se instal6 con el SDK, en cuyo caso, (y si trabaja con Internet Explorer) seleccione Herramientas>Opciones de Internet, y busque en la ficha Avanzado un ajuste"Java(Sun)", como se ve en la figura 18.5.

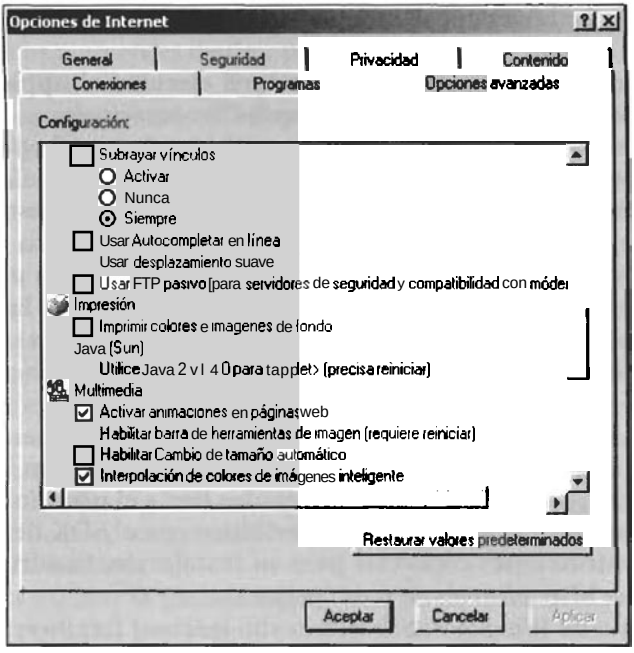


Figura 18.5. Verifique la instalacion del periferico en la ficha de Opciones de Internet de Internet Explorer

Una vez instalado el periférico, sólo nos resta abrir el applet en el buscador para que funcione (figura 18.6.)

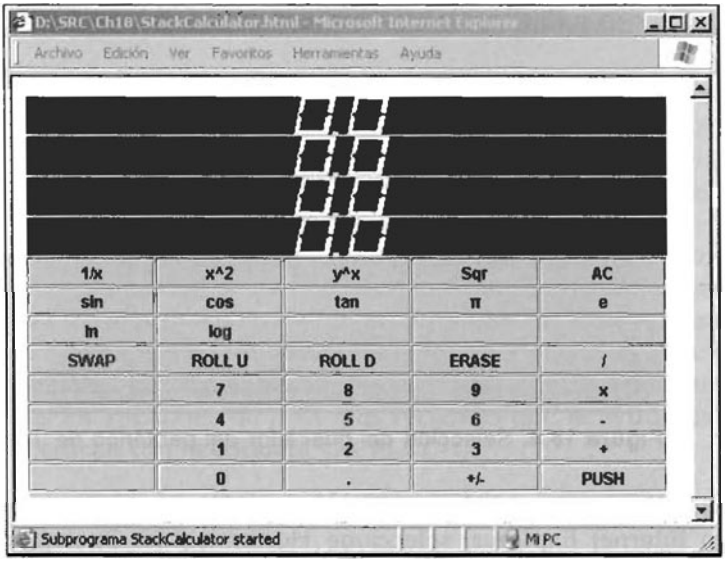


Figura 18.6. El StackCalculator en Internet Explorer con el periferico de Java

Nota

Utilizar el periferico una vez instalado es bastante sencillo, aunque debemos considerar cuidadosamente si obligar al usuario a que lo tenga. Si no podemos garantizar que los usuarios ya disponen del periferico, debemos recordar que la descarga de un archivo de más de 5MB puede causar que el usuario abandone la pagina, sin descargarlo.

Java Web Start

Este programa es casi del mismo tamaño que el periférico, aunque Java Web Start proporciona un ambiente totalmente distinto para ejecutar programas. En lugar de ejecutarlos dentro de un buscador, los applets se instalan mediante el buscador pero se ejecutan fuera. Este mecanismo se basa en la tecnología llamada JNPL (Java Network Launching Protocol). Java Web Start es sólo una implementación de este protocolo, y DeployDirector de Sitraka, es otro.

Java Web Start requiere algunos ajustes en el servidor de la red. Para mostrar las características de Web Start, demostraremos algunos de los programas de muestra disponibles a través de dicha implementación (en lugar de ejecutar `StackCalculator` dentro de el).

Visite la página http://java.sun.com/products/javawebstart/ y haga clic en el boton **Get Java Web Start Now!** Siga las opciones de instalacion. Puesto que ya tiene un JRE más reciente que la versión 1.2.2 en su sistema, sólo le resta instalar el paquete de actualizaciones de Windows, que ocupa menos de 1MB. Las opciones de instalacion de Solaris y Linux no disponen de un tiempo de ejecucion. Compruebe que el buscador está cerrado antes de comenzar la instalacion. Puede que sea necesario reiniciar el equipo para completar el procedimiento.

Ejecución de las aplicaciones demo

Después de instalar Java Web Start, ábralo y verá el gestor de aplicaciones que se muestra en la figura **18.7.** Las cuatro aplicaciones demo preconfiguradas incluyen un programa básico de diseño (Draw 4), un juego (MilitaryGame), un procesador básico de texto (Notepad), y un demo de los componentes de Swing (SwingSet2). Los programas reales no están instalados en su sistema, sólo los enlaces a su sitio en la Red.

Nota

Otros demo de Sun estan disponibles en http://java.sun.com/products/javawebstart/demos.html, o puede ver lo que otros usuarios estan haciendo en http://java.sun.com/products/javawebstartlpartners. html.

Figura 18.7. El Gestor de aplicaciones Java Web Start

Al hacer doble clic en cualquiera de los demo se inicia el Gestor de aplicaciones (figura 18.8) para buscar actualizaciones del programa (figura 18.9) y descargarlas, si fuera necesario (figura 18.10) antes de ejecutar el demo (figura 18.11).

Figura 18.8. El Gestor de aplicaciones iniciando los comprobantes del demo

Figura 18.9. El Gestor de aplicaciones buscando actualizaciones

La aplicación seleccionada se instala localmente. La descarga de este programa no es sólo para una única ejecución, como ocurre con los applet. La próxima vez que haga doble clic en una demo ya instalada, la ejecuta directamente. En realidad,

la segunda vez que se ejecuta un programa, le aparecerán pantallas para crear accesos directos (figura 18.12), para evitar utilizar el Gestor de nuevo para ejecutar el demo.

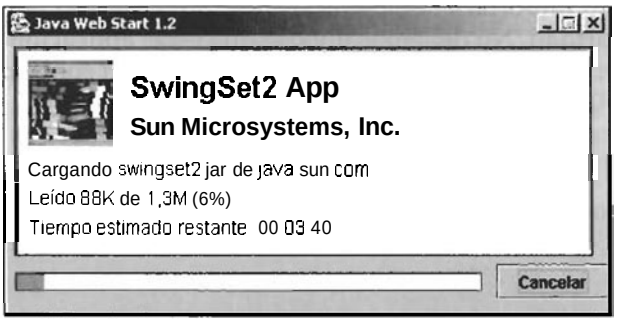


Figura 18.10. El Gestor de aplicaciones descargando el demo

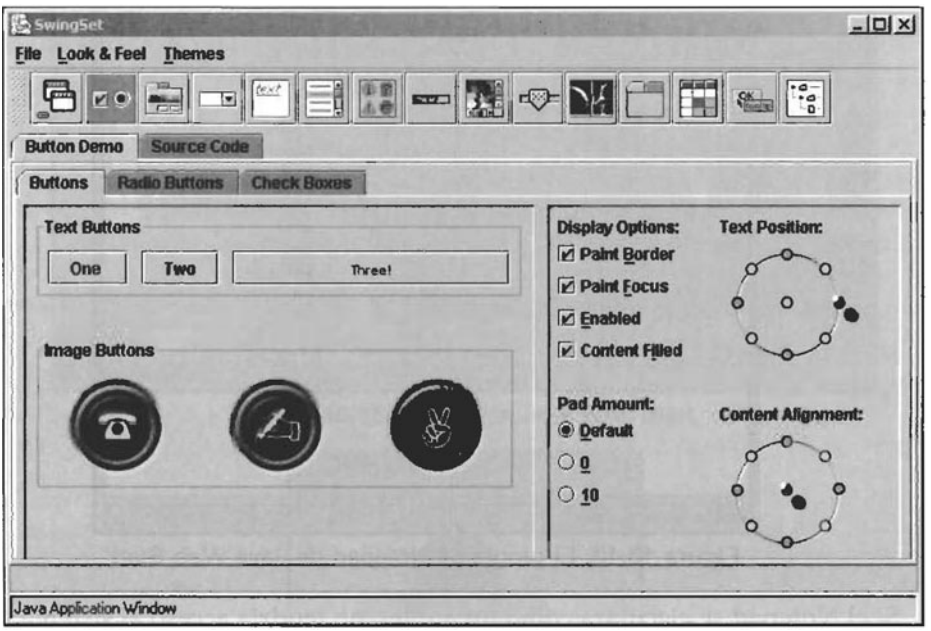


Figura 18.11. El demo SwingSet2

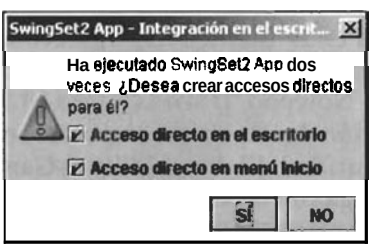


Figura 18.12. Java Web Start le permitira crear accesos directos

Java Web Start soporta las operaciones desconectadas, por lo que una vez instalado el programa, no es necesario conectarse a Internet para ejecutarlo (siempre y cuando el programa no necesite comunicarse con el servidor).

Otro aspecto importante de ejecutar programas dentro del ambiente Java Web Start es la seguridad. No debemos olvidar que estas aplicaciones no tienen libertad absoluta al ejecutarse en el ambiente del usuario. Por el contrario, se ejecutan en un ambiente con limitaciones mds flexibles que los applets.

Para experimentar la seguridad de Java Web Start, abra la aplicación Notepad (figura 18.13)

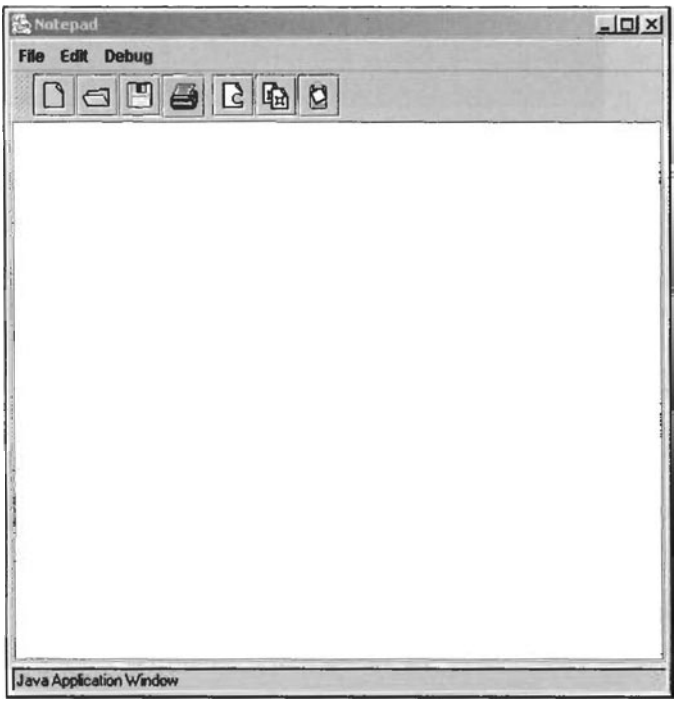


Figura 18.13. El programa Notepad de Java Web Start

Si el Notepad se ejecutara como un applet, no tendría acceso al sistema local de archivos. A través de Java Web Start, sin embargo, si lo tiene, aunque con ciertas restricciones. Siempre que necesite sobrepasar el área de seguridad, aparecerd un aviso. El que se muestra en la figura 18.14 aparece al intentar leer o escribir en el disco duro. Incluso el acceso al escritorio del sistema está protegido de esta manera.

Al igual que el demo Notepad, Draw4 (figura 18.15) permite acceder al sistema local de archivos. La opción de imprimir (disponible en Notepad y Draw4) hace que aparezca el aviso de seguridad. El demo MilitaryGame se basa en bloquear al oponente (figura 18.16) si juega con el color negro, o escapar, si lo hace con el rojo. Jugar no demostrará ninguna caracteristica especial de Java Web Start, pero si consigue averiguar la estrategia, ganará el juego (figura 18.17).

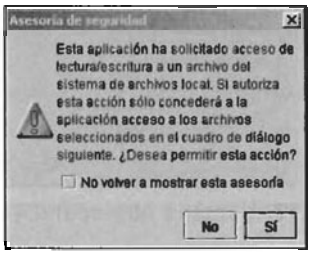


Figura 18.14. Aviso de seguridad de Java Web Start

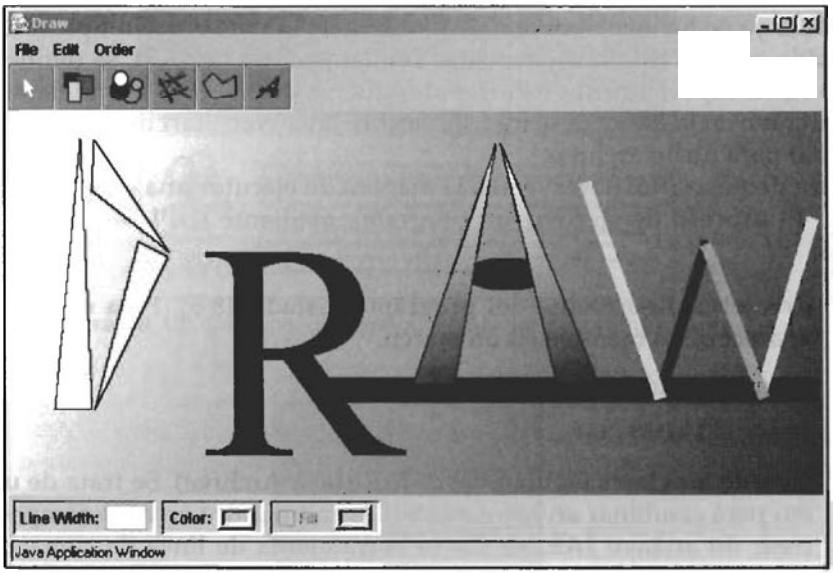


Figura 18.15. Draw4 de Java Web Start

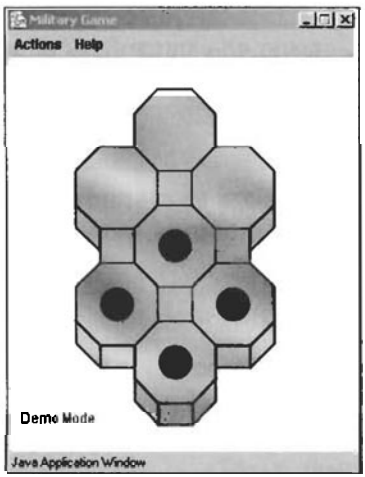


Figura 18.16. MilitaryGame de Java Web Start

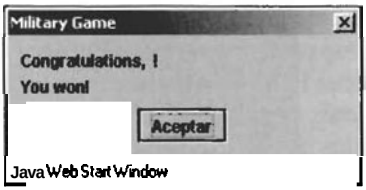


Figura 18.17. Ventana que aparece al ganar el juego

Funcionamiento de Web Start

Como ya se ha mencionado, Java Web Start funciona con un protocolo especial conocido como JNPL. Los elementos fundamentales de JNPL se definen en un archive XML especialmente formateado, que se describirá más adelante. Al poner este archivo en la Red y descargarlo, se abre Java Web Start debido a un tipo MIME especial para dicho archivo.

Para demostrarlo, observemos la manera de ejecutar una clase local a través de JNPL. El proceso de ejecutar un programa mediante JNPL se compone de cinco pasos:

1. Cree el código fuente del programa (listado 18.6). Para este ejemplo, sólo aparecerá un mensaje en un marco.
2. Compile el código. Escriba:

```
javac JNLPTest.java
```

3. Guarde las clases en un archivo JAR (Java Archive). Se trata de un mecanismo para combinar archivos múltiples en un único archivo comprimido. Para crear un archivo JAR, se usa la herramienta de linea de comando `jar` del modo siguiente:

```
jar cf JNLPTest.jar JNLPTest.class
```

4. Cree un archivo de definición JNLP para la aplicacidn como un documento XML. Es necesario proporcionar informacidn sobre lo que se está cargando, sus fuentes, y dónde está situado.

```
<?xml version="1.0" encoding="UTF-8"?>
<jnlp spec="1.0+"
  codebase="file:///c:/sybex/code/Ch18/"
  href="start.jnlp">
<information>
  <title>JNLP Test</title>
  <vendor>JZ Ventures, Inc.</vendor>
  <homepage href="http://www.jzventures.com" />
  <description>Demonstration of JNLP</description>
</information>
<resources>
  <j2se version="1.2+" />
  <jar href="JNLPTest.jar"/>
</resources>
```

```
<application-desc main-class="JNLPTest" />
</jnlp>
```

5. Haga doble clic en el archivo JNPL. Asumiendo que todo se ha configurado correctamente, aparecerá la ventana de configuración (figura 18.18) antes de que lo haga el programa (figura 18.19)

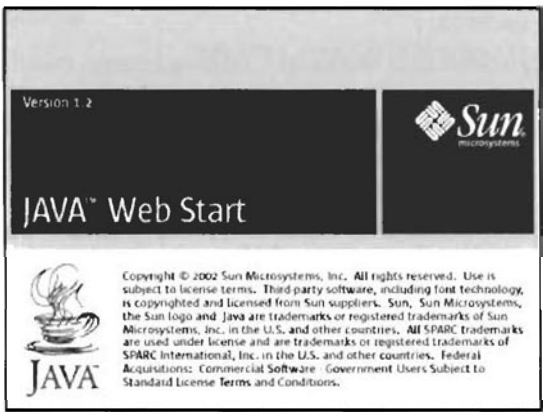


Figura 18.18. Ventana de configuración del programa de prueba

Figura 18.19. Programa de prueba JNLP

Nota

Para mas información sobre el formato de archivos JNLP, visite la guia sobre Java Web Starter en http://java.sun.com/products/javawebstart/docs/developersguide.html.

Listado 18.6. La clase JNLPTest

```
import java.awt.*;
import javax.swing.*;
public class JNLPTest {
```

```
  public static void main(String args[]) {
    JFrame frame = new JFrame("JNLP Rocks");
    Container content = frame.getContentPane();
    JLabel label = new JLabel("Hello, JNLP");
    content.add(label, BorderLayout.CENTER);
    frame.setSize(200, 200);
    frame.show();
    try {
      Thread.sleep(5000);
    } catch (InterruptedException ignored) {
    } finally {
      System.exit(0);
    }
  }
}
```

Se pueden hacer muchas más cosas con JNLP y Java Web Start de lo que aqui se ha incluido, tal como instalar iconos y ventanas splash personalizadas. Este programa de prueba sólo sirve de introducción a todas las demás posibilidades. Si desea probar este ejemplo en la Web, debe configurar el servidor para que devuelva `application/x-java-jnlp-file` como el tipo MIME para los archivos JNLP. El procedimiento exacto dependerá del tipo de servidor.

Optimización del tiempo de inicio

Como pudimos observar en el ejemplo de `StackCalculator`, los applets reales y funcionales no son pequeños. A más funcionalidad, mayor tiempo de espera. A medida que los applets se hacen cada vez más comunes, se utiliza más banda ancha para transferir applets desde el servidor hasta el cliente, ejecutando nuestro buscador favorito. Esto significa una mayor presión y una velocidad más lenta de descarga. Es muy importante minimizar la cantidad de recursos de Internet utilizados por el applet y reducir los tiempos de espera para mantener una apariencia de interactividad. Esto se puede lograr de la siguiente manera:

- Minimizar el tamaño de los ejecutables del applet.
- Minimizar el numero de clases empleadas.
- Utilizar archivos JAR.
- Reducir el tiempo de iniciación del applet.
- Utilizar signos para representar código.

Mantener ejecutables pequenos

Los grandes archivos tardan más en trasmitirse por Internet que los archivos rnis pequeños, y con los applet pasa lo mismo. El tamaño de los archivos `*.class`

generados por el compilador es asombrosamente pequeiio. Comparado con el código compilado C o C++, los ejecutables de Java son muy pequeiios, debido a dos razones:

- El código de Java está muy compactado.
- No todas las clases estándar importadas por un programa Java son una parte fisica del programa.

Dicho esto, unos bytes menos pueden significar una descarga de 20 segundos, en lugar de 12 (y para el usuario, la espera siempre parece una eternidad). La diferencia de dos segundos es el resultado de la segmentación de paquetes, una propiedad de la transferencia de los archivos de Internet. Probablemente lo haya experimentado personalmente al navegar en la Red: El indicador de descarga se va llenando de segmento en segmento, y justo antes de completar la descarga, se detiene, esperando la llegada de unas pocas docenas de bytes para finalizar. Es en este momento cuando desearia que el applet tuviera un par de docenas menos de bytes.

Esta parada justo antes de finalizar la descarga se debe a la segmentación del archivo trasmitido. Aunque el protocolo TCP de Internet acepta cadenas de datos de longitud arbitraria, otros dispositivos responsables de llevar los datos del punto A al punto B tienen normalmente limites estrictos en cuanto al tamaiio de los paquetes (256, 512, 1.024, o 2,048 bytes, generalmente). Al trasmitir un applet, se segmenta en un numero determinado de paquetes de un tamaiio establecido, más un paquete final que contiene los bytes restantes. Todos estos paquetes se trasmiten independientemente y deben reagruparse en la cadena original ejecutable al llegar al usuario. Al retrasarse el ultimo paquete, deberemos esperar varios segundos esperando unos pocos bytes. Una manera de solucionarlo es intentar que el tamaiio de las clases del applet sean multiplos de 512, pero, por desgracia, esto no es tan sencillo. Internet contiene una inmensa diversidad de colecciones de hardware, lo que significa que la unidad de transmisión máxima resultante (MTU) puede ser cualquiera. (MTU es el tamafio mayor del paquete utilizado por la ruta seleccionada.) Lo mejor es intentar mantener las clases lo más pequeñas posible.

Las siguientes técnicas ayudan a limitar el tamafio de las clases sin perder demasiado tiempo en el proceso:

- Eliminar código de depuración.
- Reducir el tamaiio de las explicaciones dentro del código.
- Compilar con la opción `java-O` (Optimizada)activada.
- Eliminar los métodos o variables que no se utilizan.

Minimizar el numero de clases

Pocos applets consisten en una unica clase que se base exclusivamente en las clases estandar de Java para todos sus bloques de construcción. La norma es que los

applets se basen en clases creadas por nosotros mismos. Todas estas clases deben cargarse por separado mediante el VM de Java, lo que constituye un problema. Por ejemplo, cargar 20 bloques pequefios de datos por separado tarda más que cargar un elemento 20 veces mayor que los pequefios bloques. Esto es consecuencia de las conexiones que se crean entre cada bloque (read: class), por lo que es un buen consejo limitar el numero de clases extra en que se basa el applet.

Un área en que un proyecto puede generar muchas clases es el código GUI. Se puede tratar la tarea de procesamiento de eventos del GUI mediante la creación de clases internas anónimas o implementando las interfaces. Cuantas más clases anónimas utilicemos, más archivos de clases creamos. Es una buena idea tener en cuenta todo esto a la hora de disefiar el GUI, lo cual no quiere decir que debemos guardar todo en una enorme clase, ya que eso significaria ignorar todos los principios de programación del libro. Hay que encontrar una solución intermedia.

Utilizar archivos JAR

Como se demostró en el ejemplo JNLP, Java soporta archivos JAR para agregar archivos como tar o zip. Si agrupamos todos los archivos `.class` de un applet en un unico archivo, un buscador puede leer ese archivo desde un servidor en una unica operación. Esto reduce enormemente el consumo del applet de archivos multiples. La clase `loader` sabe cómo extraer clases de los archivos, de manera automática, por lo que esta técnica no requiere desarrollo extra por parte del cliente. Por parte del servidor, será necesario expandir la linea `<applet>` del archivo HTML. Por ejemplo, si el archivo JAR se llama `myclasses.jar`, deberemos afiadir lo siguiente a la etiqueta `applet`:

```
<applet code=… archive="myClasses.jar">
```

Para enumerar multiples archivos `.jar`, necesitamos separar cada nombre de archivo con una coma. Recuerde que no es necesario incluir todos los archivos del applet dentro del archivo JAR, sino el 80 por ciento, es decir, los más utilizados. Las demás clases se cargarán sólo cuando se las necesite.

Reducir el tiempo de inicio

Todo el mundo odia tener que esperar. Pero además, una vez que la espera ha concluido, se necesita más tiempo para procesar los cambios ocurridos, lo que se traduce en que los applets deberian comenzar a hacer algo tan pronto como sea posible. Deberian saludar al usuario, preguntarle el nombre, mostrar un indicador de progreso, en resumen, algo para mantener al usuario entretenido durante la espera. Mientras tanto, el applet puede continuar la iniciación. Esto requiere algunos threads extra que interactuen con el usuario mientras el proceso de iniciación se ejecuta en el thread principal. De este modo un applet irritantemente lento puede convertirse en un applet de rápida ejecución que entretiene al usuario.

Recuerde que el tiempo pasa volando cuando uno lo estd pasando bien. La velocidad variard enormemente dependiendo de la percepción. Manteniendo al usuario entretenido, alteramos su percepción del tiempo.

Nota

Si quiere que el applet se ejecute rnediante el periferico de Java, se puede carnbiar la apariencia del cuadro gris que aparece mientras se esta cargando. La etiqueta `<APPLET>` soporta los atributos `BOXMESSAGE`, `PROGRESSCOLOR`, `BOXFGCOLOR`, Y `BOXCOLOR` para cambiar la apariencia. Estos atributos solo afectan en el periferico y no modificara los applets que usan el JRE por defecto de la mayoría de los buscadores.

Ofuscar código

Esta última tarea requiere algunas herramientas extras, no disponibles en SDK. El termino inglés *obfuscate* significa hacer algo mds confuso u opaco para que sea mds dificil de entender. A menudo este tdrmino se refiere al acto de hacer que el código sea mds dificil de separar. Otra consecuencia de este método es reducir la longitud del código. Al ejecutar un `obfuscator` en un bloque de código, cambia nombres por simbolos aparentemente sin sentido. Por ejemplo, todas las referencias a mdtodos como `makeIceCream` se convertiria en â (beta). Puesto que `makeIceCream` tiene 12 caracteres y beta sólo uno, esta técnica supone una reduccion de 22 bytes, ya que cada cardcter se compone de dos bytes. Esta reduccidn que ahora parece insignificante, puede ser muy significativa con códigos de miles de líneas. Algunas de las herramientas `obfuscation` disponibles son comerciales. Algunas son gratuitas. Muchos IDE (incluido JBuilder) contienen ya herramientas de este tipo. Sólo hay que activar la opción que viene desactivada por defecto. La figura 18.20 muestra la ventana donde se encuentra la opción Obfuscate en JBuilder 6: una de las casillas de verificación de la sección Other Compiler Options.

A continuacidn se detallan algunas herramientas que pueden ser de utilidad:

- Jshrink http://www.e-t.com/jshrink.html.
- JODE http://jode.sourceforge.net/.
- DashO http://preemptive.com/tools/
- Condensity http://www.condensity.com/.
- Zelix KlassMaster http://www.zelix.com/klassmaster/
- CodeShield http://www.codingart.com/codeshield.html.
- WingGuard http://www.wingsoft.com/wingguard.html.
- 2LKit http://www.2lkit.com.

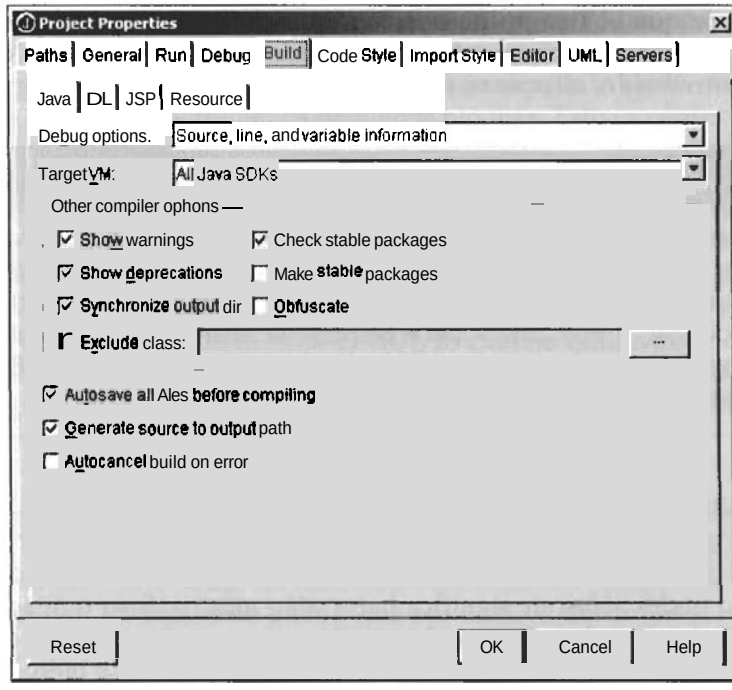


Figura 18.20. JBuilder 6 tiene una opción Obfuscate entre las opciones de cornpiladores

Resumen

Los decompiladores se encargan de tomar archivos `.class` y convertirlos en `.java`. Si los archivos de la clase no han pasado por el `obfuscator`, los archivos `.java` generados tendriin la misma apariencia que el código fuente. Los nombres de los pariimteros se pierden normalmente, pero aparte de eso, la mayoría del resto de la estructura y código se pueden recuperar. Una razón más para utilizar un `obfuscator`. Como ejemplo, vamos a utilizar el código del listado 18.4 para un caso simple de muestra. Compilando con el compilador regular `javac` produce un archivo `.class` de 1.082 bytes. El listado 18.7 es el código generado de ejecutar un decompilador en `CalcStack.class`.

Listado 18.7. La clase CalcStack

```
public class CalcStack
{
    public CalcStack(int i)
    {
        stackDepth = i;
        stack = new double[stackDepth];
```

```
        clearStack();
    }
    public void pushValue(double d)

        rollUp(1);
        setAccumulator(d);

    public void rollUp(int i)
    {
        for(int j = 0; j < i; j++)
        {
            double d = stack[stackDepth - 1];
            for(int k = stackDepth - 2; k >= 0; k--)
                stack[k + 1] = stack[k];
            stack[0] = d;


    public void rollDown(int i)
    {
        rollUp(stackDepth - i);
    }
    public void swap()
    {
        double d = stack[0];
        stack[0] = stack[1];
        stack[1] = d;
    }
    public void clearStack()
    {
        for(int i = 0; i < stackDepth; i++)
            stack[i] = 0.0D;
    }
    public void drop()
    {
        rollDown(1);
        stack[stackDepth - 1] = stack[stackDepth - 2];
    }
    public void setAccumulator(double d)
    {
        stack[0] = d;
    }
    public double getAccumulator()
    {
        return stack[0];
    }
    public double getStackElement(int i)
    {
        return stack[i];
    }
    double stack[];
    int stackDepth;
```

Básicamente **hemos obtenido el mismo** código. **Si empleamos la** técnica `obfuscate` **en el** archivo **de la clase al compilar** `CalClass.java`. **El compilador**

de JBuilder 6 crea un archivo de clases de 814 bytes. Al ejecutar el decompilador produce el mismo código. Aunque el archivo compilado era más pequeiio, el `obfuscator` no pudo reducir ninguna clase ni ningun nombre de variables porque ninguno era privado. Todos los miembros de la clase deben ser visibles para que no puedan cambiarse.

Nota

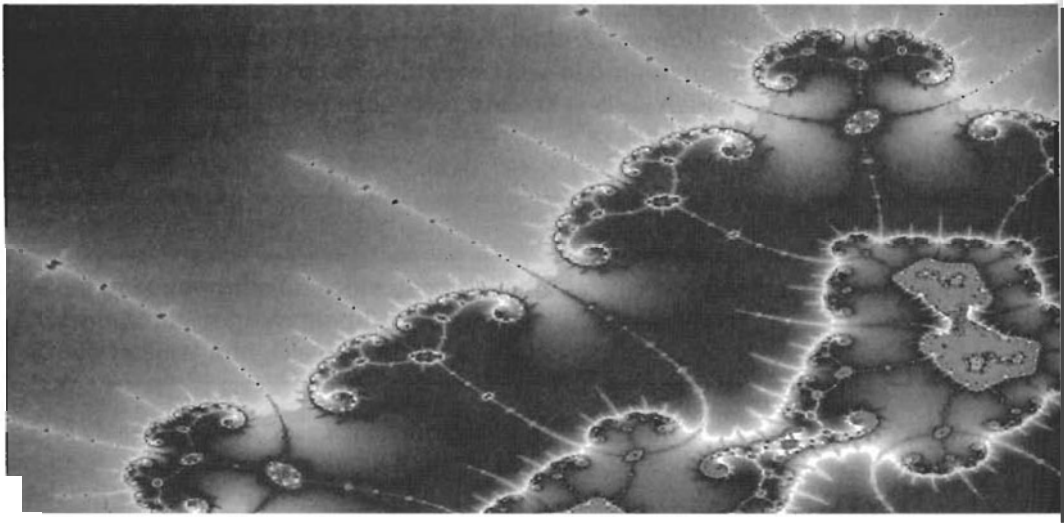
Para mas inforrnacion sobre decompiladores, visite la pagina http://drnoz.org/Computers/Programming/Languages/Java/Development_Tools/Translators/Decompilers_and_Disassemblers/.

La clase `keyPanel` es un ejemplo mejor para demostrar el resultado de aplicar un `obiuscator`. El listado 18.8 muestra parte del código regenerado despuds de descompilar.

Listado 18.8. Parte del codigo de keyPanel generado de un archivo de clase ofuscado

```
private void _$243()

        double d = stack.getStackElement(1) / stack.getAccumulator();
        stack.drop();
        stack.setAccumulator(d);
        display.redrawStack(true);
    }
    public String keyLabel(int i, int j)
    {
        return _$208[i * rowSize + j];
    }
    private String _$208[] = {
        "1/x", "x^2", "y^x", "Sqr", "AC", "sin", "cos", "tan", "\u03C0",
"e",
        "ln", "log", "   ", "    ", "  ", "SWAP", "ROLL U", "ROLL D",
"ERASE",
        "/", "  ", "7", "8", "9", "x", "  ", "4", "5", "6", "-",
        "  ", "1", "2", "3", "+", "  ", "0", ".", "+/-", "PUSH"
    };
```

El código resultante es casi imposible de entender. La variable `keyLabels` se ha convertido en `_$208`, y el método `divide()` en `_$243`. Aunque aún se puede ver el código asociado con cada método, han desaparecido los nombres significativos de métodos y variables.



19

Streams y programación E/S

El modelo más sencillo de un ordenador consiste en tres conexiones: datos de entrada, procesamiento y datos de salida. Entrada y salida (E/S) constituyen un aspecto fundamental de la computación. Un ordenador no serviria de nada si no pudiera aceptar datos exteriores y presentar los resultados computados. Como soporte de esta actividad fundamental, los ordenadores están siempre acompañados de interfaces E/S integradas o periféricas llamadas puertos, tales como puertos de serie, paralelos, de teclado, de audio, de video, y SCSI, a traves de los cuales podemos conectar todo tipo de aparatos E/S, como módems, impresoras láser, teclados, sistemas de sonido, monitores y discos duros, respectivamente.

Este capitulo describe cómo Java, a través de `java.io.java.nio`, y otros paquetes relacionados, proporciona clases independientes para manipular archivos y streams. Un stream es un concepto abstracto usado frecuentemente en programación E/S. Representan un flujo lineal y secuencial de datos de entrada o de salida. Los stream pueden ser de entrada, si fluyen hacia nuestro equipo, o de salida, si proceden de nuestro equipo. Esencialmente, leemos un stream de entrada y escribimos un stream de salida.

El punto fundamental de trabajar con streams es que sirven de escudo de los dispositivos E/S con los que nos comunicamos en realidad. Al trabajar con streams abstractos, podemos alternar dispositivos fisicos de entrada y salida sin modificar el código de la aplicación.

Antes de explicar las diferentes clases de stream, examinaremos primero el dispositivo físico E/S más básico, que es el archivo. Después veremos la manera en que Java lee y escribe bytes, caracteres, y objetos de archivos y otros stream. Explo-

raremos también el soporte de compresión de Java y las bibliotecas para analizar expresiones regulares, antes de describir con detalle las nuevas bibliotecas E/S en las que los stream se basan en canales, terminando con la nueva facilidad de registro para mensajes.

Java 1.4 aiiade muchas capacidades nuevas para realizar tareas E/S. En la mayoria de los casos, los métodos antiguos no se han sustituido, sino que se han complementado con nuevas funciones.

Métodos de gestión de archivos de Java

El sistema de archivos del sistema operativo es el servicio más básico de una aplicacidn. Fue ademds uno de los primeros servicios en ser desarrollados. Durante los años 50, la habilidad de un equipo para localizar y grabar un programa en su memoria no era una tarea que se diera por descontado. En la actualidad, todos los sistemas de archivos permiten organizar archivos de dimensiones arbitrarias, mediante una estructura jerarquica de directorios de diversa complejidad (la capacidad limite más común de un archivo es de 4GB).

Los nombres de archivo sirven para identificar los distintos archivos. Dependiendo de la plataforma del usuario, estos nombres pueden contener de 32 a 256 caracteres. Aunque todos los sistemas de archivos son idénticos en cuanto a los servicios bdsicos que ofrecen, las implementaciones de los diferentes sistemas hacen que por lo general sean incompatibles.

La clase File de Java

Para evitar este obstáculo producido por la incompatibilidad de nombres de archivos, la clase **`File`** define unos métodos independientes que permiten manipular un archivo, mantenido por un sistema de archivos nativo. La tabla 19.1 muestra la definicidn de la clase `File`.

Tabla 19.1. Cornposicion de la clase File

Variables/Métodos	Descripción
`pathSeparator`	Soporta un caracter separador para plataforrnas concretas utilizado para las rutas
`pathSeparatorChar`	
`separator`	Soporta un caracter separador para plataformas concretas utilizado entre nornbres de archivos y directorios
`separatorChar`	

Variables/Métodos	Descripción
File()	Construye un objeto File
canRead()	Devuelve la indicacion de si un archivo es de escritura o solo de lectura
canWrite()	
compareTo()	Cornprueba la conexión entre rutas y archivos
createNewFile()	Crea un archivo vacio si no existe aun
createTempFile()	Crea un archivo vacio en un directorio concreto
delete()	Elirnina un archivo del sisterna de archivos
deleteOnExit()	Hace que el archivo se elirnine del sisterna de archivos cuando el prograrna terrnina sin errores
equals()	Comprueba si existen objetos de archivo iguales (no en el contenido)
exists()	Devuelve la indicacion de si el archivo existe
getAbsoluteFile()	Devuelve la ruta cornpleta al archivo
getAbsolutePath()	
getCanonicalFile()	Devuelve la ruta completa al archivo, sin indicadores relativos
getCanonicalPath()	
getName()	Devuelve el nornbre del archivo
getParent()	
getParentFile()	Devuelve el directorio raiz del archivo
getPath()	Devuelve el archivo como una ruta relativa
hashCode()	Devuelve el codigo hash del archivo
isAbsolute()	Devuelve el estado del tipo de archivo
isDirectory()	
isFile()	
isHidden()	
lastModified()	Devuelve el ultimo marcador de tiempo modificado
length()	Devuelve el tamaño del archivo
list()	Devuelve la lista de archivos y directorios de un archivo
listFiles()	
listRoots()	Devuelve la lista de los sistemas de archivos raiz

Variables/Métodos	Descripción
`mkdir()`	Crea un directorio nombrado por un archivo
`mkdirs()`	Crea un directorio y una ruta cornpleta nornbrada por el archivo
`renameTo()`	Cambia el nornbre del archivo
`setLastModified()`	Cambia el último rnarcador de tiernpo modificado
`setReadOnly()`	Cambia el archivo para que sea solo de lectura
`toString()`	Devuelve la representación string del estado del archivo
`toURI()`	Construye un URI/URL para el archivo
`toURL()`	

Como podemos observar en la tabla anterior, la clase `File` no permite acceder al contenido del archivo, ya que no dispone de métodos `read()` o `write()`. Esta clase se encarga de nombrar archivos, cuestionar atributos, y manipular directorios o archivos temporales, independientemente del modo de gestionar la información del sistema. Las siguientes secciones describirdn lo que podemos hacer con archivos sin tener que abrirlos.

Asignación de nombres de archivo

Los archivos pueden nombrarse de dos maneras distintas. Podemos proporcionar tanto el directorio como el archivo dentro del directorio o ambos combinados en un unico valor.

De cualquier manera, debemos proporcionar el nombre completo de un archivo al que se desea acceder. La ventaja de proporcionar los nombres del directorio y del archivo por separado es que no nos tenemos que preocupar de la manera de combinarlos. Por ejemplo, para plataformas basadas en Microsoft Windows, el carácter de separación es la barra inclinada "\", mientras que para las plataformas Unix, se emplea la barra inversa "/". Aunque es posible preguntar a la clase `File` cuál es el cardcter separador, mediante la variable `separator`, es una operación innecesaria ya que el constructor de la clase `File` puede combinar dos términos de manera automdtica.

Para demostrarlo, el siguiente código intenta crear tres objetos `File`. Cada línea lo consigue, aunque si intentamos utilizar el objeto `File`, sólo al segundo o tercer intento lo logrard en todas las plataformas ya que el código no dispone del valor separador de archivos.

```
File f1 = new File("tmp/foo");
File f2 = new File("tmp", "foo");
File f3 = new File(new File("tmp"), "foo");
```

Cuestionamiento de atributos de archivos

La clase File proporciona una gran variedad de métodos para cuestionar un pequefio grupo de atributos de archivos. Con ellos, podemos descubrir lo siguiente sobre el archivo:

- Si existe.
- Si estd protegido contra lectura.
- Si estd protegido contra escritura.
- Si se trata de un directorio.
- Si estd oculto.
- Si estd especificado como una ubicación absoluta.

La clase File no permite descubrir otros atributos comunes de archivo, tales como si se trata de un archivo del sistema o un archivo "archivado". Como con las clases AWT de Java, los disefiadores de las clases E/S han utilizado el menor denominador comun de todos los sistemas de archivos para el modelo E/S. Si Java incluyera funciones que no fueran soportadas por todos los sistemas, se pondría en peligro la compatibilidad universal de Java con todas las plataformas.

El programa del listado 19.1 muestra cómo utilizar un File para cuestionar los atributos de un archivo. El archivo se especifica como un pardmetro de una linea de comandos.

Listado 19.1. Cuestionando inforrnacion sobre el archivo

```
import java.io.*;
public class Attr {
  public static void main (String arqs[]) {
    File path = new File(args[0]);  // obtiene un argumentos de línea de
                                    // comandos
    String exists   = getYesNo(path.exists());
    String canRead  = getYesNo(path.canRead());
    String canWrite = getYesNo(path.canWrite());
    String isFile   = getYesNo(path.isFile());
    String isHid    = getYesNo(path.isHidden());
    String isDir    = getYesNo(path.isDirectory());
    String isAbs    = getYesNo(path.isAbsolute());
    System.out.println("File attributes for `" t args[0] + "`");
    System.out.println("Exists        : " + exists);
    if (path.exists()) {
      System.out.println("Readable      : " t canRead);
      System.out.println("Writable      : " t canWrite);
      System.out.println("Is directory  : " t isDir);
      System.out.println("Is file       : " t isFile);
      System.out.println("Is hidden     : " + isHid);
      System.out.println("Absolute path : " t isAbs);
    }
```

```
private static String getYesNo(boolean b) {
  return (b ? "Yes" : "No");
}
```

Nota

Podemos experirnentar con el programa de este listado, pasando varios nombresde archivos y directorios, relativos o absolutos. Un nombre de archivo absoluto es aquel que comienza en el nivel raíz del sistema de archivos, tal como C: \. Aunque C: \ es absoluto, no es en realidad un archivo. La ruta del directorio a ese nivel seria C: \., donde "."especificaría el archivo del directorio.

Manipulación de directorios

Más útil que el programa `Attr` del listado 19.1 seria uno que permitiera enumerar el contenido de un directorio, tal como los comandos `dir` o `ls` en la mayoría de los sistemas operativos. Afortunadamente, la clase `File` soporta la producción de la lista de directorios, mediante los métodos `list()` y `listFiles()`.El listado 19.2 contiene otro programa que de manera recursiva enumera los directorios y sus contenidos.

Listado 19.2. Listado del contenido del directorio

```
import java.io.*;
import java.util.*;
public class Dir {
  static int indentLevel = -1;
  static void listPath(File path) {
    File files[];  // lista de archivos en un directorio
    indentLevel++; // hacia abajo
    // Crea una lista de archivos en este directorio
    files = path.listFiles();
    // Organiza con ayuda de las colecciones API
    Arrays.sort(files);
    for (int i=0, n=files.length; i < n; i++) {
      for (int indent=0; indent < indentLevel; indent++) {
        System.out.print("    ");
      }
      System.out.println(files[i].toString());
      if (files[i].isDirectory()) {
        // Desciende por el árbol de directorios
        listPath(files[i]);

      }
    indentLevel--; // hacia arriba
  }
  public static void main (String args[]) {
```

```
    listPath(new File(args[0]));
  }
}
```

El programa **Dir** se basa en un par de conceptos muy interesantes. En primer lugar, se llama asi mismo en la línea de código **listPath(files[i]).** Esta repetición en la invocación del método **listPath()** hace que **listPath()** se inicie de nuevo con una nueva ruta, para que contenga el directorio mds profundo.

La lista de directorios se organiza con ayuda de la clase **Arrays** del Marco de colecciones, descrita en el capitulo 17.

Para destacar el nivel actual del directorio, el contenido del directorio aparece sangrado de acuerdo con su nivel anidado en la jerarquia de archivos. La profundidad del nivel de recurrencia determina el nivel de anidamiento. Otro interesante detalle de **Dir** es que, cuando un método existe, las variables de la clase no se destruyen como las variables de métodos simples. El programa se basa en el comportamiento de variables estáticas para obtener el nivel de recurrencia a través de invocaciones multiples de **listPath().** Las variables estáticas cuentan con otro atributo de gran interés, y es que no se distribuyen cada vez que se crea una nueva clase. Esta es la razón por la que las variables estdticas reciben también el nombre de variables de clase.

Manipulación de archivos temporales

Otra caracteristica de la clase **File** es que soporta los archivos temporales. Gracias a los métodos estdticos `createTempFile()` y la propiedad del sistema `java.io.tmpdir`, podemos garantizar que el objeto **File** creado por el método no ha existido anteriormente. Además, mediante `deleteOnExit()`, nos aseguraremos de que el archivo se va a borrar si el ambiente de Java termina de una manera natural. El marco generado para utilizar `createTempFile()` es el siguiente:

```
File temp = File.createTempFile("zuk", ".tmp"); // prefijo y sufijo
temp.deleteOnExit();
// Se utiliza como cualquier otro archivo
```

Nota

La propiedad del sistema `java.io.tmpdir` contiene la ubicacion temporal del archivo dependiente del sistema, que, por lo general, sera `C:\Temp` para las plataformas de Microsoft Windows y `/temp` para los sistemas de Unix.

La clase RandomAccessFile de Java

La clase **File** no nos permite leer o modificar archivos, pero otras clases nos proporcionan esta capacidad. Podemos hacerlo de dos maneras diferentes: mediante las clases de stream de gran potencia, o mediante la clase `RandomAccessFile`.

Es muy sencillo trabajar con esta última clase, aunque tiene importantes limitaciones a la hora de modificar aplicaciones flexibles o complejas, ya que permite la entrada y salida de datos únicamente de archivos. Por otro lado, las clases de stream E/S no tienen esta limitación. Estas clases se detallarán más adelante en este capítulo, pero antes veamos la definición de la clase **`RandomAccessFile`** en la tabla 19.2, que resume su funcionalidad.

Tabla **19.2.** Cornposicion de la clase RandomAccessFile

Variables/Métodos	Descripción
`RandomAccessFile()`	Construye un objeto **`RandomAccessFile`**
`close()`	Cierra el archivo
`getChannel()`	Devuelve el `FileChannel` asociado
`getFD()`	Devuelve el `FileDescriptor` asociado
`getFilePointer()`	Devuelve el offset del archivo
`length()`	Devuelve el tamaño del archivo
`read()`	Lee los datos del archivo
`readBoolean()`	
`readByte()`	
`readChar()`	
`readDouble()`	
`readFloat()`	
`readFully()`	
`readInt()`	
`readLine()`	
`readLong()`	
`readShort()`	
`readUnsignedByte()`	
`readUnsignedShort()`	
`readUTF()`	
`seek()`	Cambia la ubicacion del offset del archivo
`setLength()`	Cambia el tamaño del archivo
`skipBytes()`	Ignora datos del archivo
`write()`	Escribe datos dentro del archivo

Variables/Métodos	Descripción
writeBoolean()	
writeByte()	
writeBytes()	
writeChar()	
writeChars()	
writeDouble()	
writeFloat()	
writeInt()	
writeLong()	
writeShort()	
writeUTF()	

En términos de organización, la clase `RandomAccessFile` puede verse como una implementación de las dos interfaces `DataInput` y `DataOutput`, ya que no añade nuevas funciones. La definicidn de la interfaz `DataOutput` es la siguiente:

```
public interface DataOutput extends Object {
  public abstract void write(byte b[]) throws IOException;
  public abstract void write(byte b[], int off, int len) throws
IOException;
  public abstract void write(int b) throws IOException;
  public abstract void writeBoolean(boolean v) throws IOException;
  public abstract void writeByte(int v) throws IOException;
  public abstract void writeBytes(String s) throws IOException;
  public abstract void writeChar(int v) throws IOException;
  public abstract void writeChars(String s) throws IOException;
  public abstract void writeDouble(double v) throws IOException;
  public abstract void writeFloat(float v) throws IOException;
  public abstract void writeInt(int v) throws IOException;
  public abstract void writeLong(long v) throws IOException;
  public abstract void writeShort(int v) throws IOException;
  public abstract void writeUTF(String str) throws IOException;
```

La interfaz `DataOutput` especifica una lista de mktodos de escritura que nos permiten crear cualquiera de los tipos sencillos de Java. Esta interfaz requiere todas las clases de implementación capaces de escribir o bloquear bytes. La tarea de emitir objetos (más all6 de String) se soporta mediante la interfaz `ObjectOutput`, que es más avanzada, y que se describe mas adelante. No tenemos que preocuparnos por el stream de bits exacto producido por estos mktodos ya que una interfaz simktrica (`DataInput`) se encarga de especificar los métodos equivalentes de lectura que nos permitirán leer cualquier tipo de datos.

La definición de la interfaz DataInput es la siguiente:

```
public interface DataInput extends Object {
  public abstract boolean readBoolean() throws IOException;
  public abstract byte readByte() throws IOException;
  public abstract char readChar() throws IOException;
  public abstract double readDouble() throws IOException;
  public abstract float readFloat() throws IOException;
  public abstract void readFully(byte b[]) throws IOException;
  public abstract void readFully(byte b[], int off, int len) throws IOException;
  public abstract int readInt() throws IOException;
  public abstract String readLine() throws IOException;
  public abstract long readLong() throws IOException;
  public abstract short readShort() throws IOException;
  public abstract int readUnsignedByte() throws IOException;
  public abstract int readUnsignedShort() throws IOException;
  public abstract String readUTF() throws IOException;
  public abstract int skipBytes(int n) throws IOException;
```

Las interfaces DataInput y DataOutput no son imágenes exactas, sino que existen las siguientes diferencias asimétricas:

- El método skipBytes() de la interfaz DataInput.
- Los métodos de lectura de numeros readUnsignedByte() y readUnsignedShort() de la interfaz DataInput.
- El método readLine() de DataInput, en lugar de los métodos writeBytes()/Chars() de la interfaz DataOutput.

En realidad, ambas interfaces son opuestas, y la clase RandomAccessFile es la implementación de las dos. Los principales métodos adicionales de RandomAccessFile son los siguientes:

- void seek(long pos): Arroja una excepcidn E/S.
- long getFilePointer(): Arroja una excepción E/S.
- long length(): Arroja una excepcidn E/S.
- void close(): Arroja una excepcidn E/S.
- void setLength(long newLength): Arroja una excepcidn E/S.

Esta clase no tiene un método open() ya que al construir un objeto RandomAccessFile, el archivo se abre automáticamente.

El método seek() refleja las funciones de lectura y escritura de acceso a la clase. Mediante este método, podemos colocar el puntero en cualquier lugar dentro del archivo para escribir o leer, sin estar limitados a lecturas o escrituras secuenciales (aunque sean el modo por defecto de estos métodos).

El método getFilePointer() nos permite descubrir el siguiente lugar donde se va a producir una lectura o una escritura. Para un archivo recién abierto, la

posición será siempre 0, es decir, al principio del archivo. El método **setLenght()** sirve para reducir el tamaño de un archivo cuando newLenght sea menor que **lenght()**. Si ocurre al contrario, el nuevo contenido no se define.

El tipo de excepción universal **IOException**, utilizada por la clase RandomAccessFile y por las interfaces DataInput y DataOutput, indica un fallo en cualquiera de los métodos de la clase.

El código "throws IOException" indica que es necesario seguir uno de los siguientes procedimientos:

- Aiiadir un código explícito para gestionar el error a los métodos de RandomAccess**File**, mediante parejas de **try-catch**.
- Dejar que la excepción aisle al llamador del método, haciendo que los métodos arrojen un **IOException**.

El listado 19.3 muestra esta clase y la manera de gestionar las posibles excepciones o errores. Este programa utiliza dos objetos RandomAccess**File** para comparar dos archivos.

Listado 19.3. Trabajando con RandomAccessFile

```
import java.io.*;
public class Diff {
//--------------------------------------------------------------------
// programa main()
//
// Comprueba argumentos de la línea de comandos (dos nombres de archivo)
// Abre y carga archivos
// Procesa archivos
// Cierra archivos
//--------------------------------------------------------------------
  public static void main(String args[]) {
    RandomAccessFile fh1 = null;
    RandomAccessFile fh2 = null;
    int  bufsize; // tamaño del archivo más pequeño
    long filesize1 = -1;
    long filesize2 = -1;
    byte buffer1[];      // los dos cachés de archivo
    byte buffer2[];
    // Cornprueba argumentos de la línea de comandos
    if (args.length == 0 || args[0].equals("?") ) {
      System.err.println("File Diff v1.2 (c) 06/98 L. Vanhelsuwe");
      System.err.println("-----");
      System.err.println("USAGE: java Diff <file1> <file2> | ?");
      System.err.println();
      System.exit(0);
    }
    // Abre el archivo ONE para lectura
    try {
      fh1 = new RandomAccessFile(args[0], "r");
      filesize1 = fh1.length();
    } catch (IOException ioErr) {
```

```
      System.err.println("Could not find " t args[0]);
      System.err.println(ioErr);
      System.exit(100);
    }
    // Abre el archivo TWO para lectura
    try {
      fh2 = new RandomAccessFile(args[1], "r");
      filesize2 = fh2.length();
    } catch (IOException ioErr) {
      System.err.println("Could not find " t args[1]);
      System.err.println(ioErr);
      System.exit(100);
    }
    if (filesize1 != filesize2) {
      System.out.println("Files differ in size !");
      System.out.println("`" t args[0] + "` is " + filesize1 + " bytes");
      System.out.println("`" t args[1] t "` is " t filesize2 t " bytes");
    }
    // Encuentra dos búfers donde incluir los archivos completos
    bufsize = (int) Math.min(filesize1, filesize2);
    buffer1 = new byte[bufsize];
    buffer2 = new byte[bufsize];
    try {
      fh1.readFully(buffer1, 0, bufsize);
      fh2.readFully(buffer2, 0, bufsize);

      // el corazón del prograrna..

      for (int i = 0; i < bufsize; itt) {
        if (buffer1[i] != buffer2[i]) {
          System.out.println("Files differ at offset " t i);
          break;
        }
      }
      catch (IOException ioErr) {
      System.err.println("ERROR: An exception occurred while processing the
         files");
      System.err.println(ioErr);
      finally {
      try 1
        fh1.close();
        fh2.close();
      } catch (IOException ignored) {}

}
```

El programa `Diff` utiliza un truco para mejorar el rendimiento que es de gran utilidad al procesar archivos.

Almacena todo el contenido en la memoria, leyendo todos los archivos de una sola vez, mediante `readFully()`. Esta técnica puede ser más rápida que leer un archivo, bit por bit, en un bucle.

Una vez ambos archivos estén almacenados en memoria, pueden compararse muy rápidamente mediante accesos sencillos de array.

Nota

Existe un segundo método mas directo de almacenar un archivo en memoria, mediante `FileChannel`, que se explicara mas adelante.

Advertencia

Recuerde que las invocaciones a los metodos E/S deben contar de `try-catch`. Debido a que Java requiere que los programas detecten eventos excepcionales o fallos, debemos añadir codigo para gestionar esas situaciones. La otra alternativa consiste en declarar el método `main()` para que sea capaz de arrojar un `IOException`. A pesar de que con ello los programas serian mas sencillos de leer, no es en realidad una practica de programacion recomendable. El método `main()` llega directamente al usuario por lo que cualquier excepción que encontrara la arrojaria directamente al usuario, o pasaría inadvertida. Este método no es aceptable en programacion ya que deberiamos optar siempre por detectar los errores explicitos y traducirlos en mensajes de error legibles para el usuario.

Las excepciones (que suelen tratarse de declaraciones `goto`) pueden dañar el flujo de control programado de un algoritmo, por lo que debemos comprobar que ambos archivos están cerrados bajo cualquier circunstancia. Esto lo hacemos en la declaración `finally` al final del método `main()`. Independientemente de si se produce alguna excepción, el código dentro de la declaración `finally` se ejecutara siempre, por lo que es el mejor lugar en el que colocar las llamadas `close()` al archivo.

Métodos de manipulación de Stream de E/S

La creación de los stream se produjo en 1984, cuando Dennis Ritchie (que junto con Brian Kernighan creó el lenguaje C) implement6 el primer sistema stream de E/S para el sistema operativo Unix de la compañía de telecomunicaciones AT&T. Los streams no son más que flujos lineales de información. Los usuarios pueden leer secuencialmente los elementos que llegan hasta ellos, desde uno de estos flujos, o también pueden añadir nuevos elementos con la seguridad de que llegarán al destino conocido al final del stream.

Cada stream dispone de una fuente de información en el caso de streams de entrada, o se de un destino, llamado sink, en el caso de streams de salida. Ambos tipos deben estar conectados a algun equipo antes de realizar cualquier tipo de tarea. Los streams de entrada deben conectarse a algún dispositivo que produzca datos y un stream de salida, a uno que los acepte.

El paquete java.io se compone básicamente de clases stream de E/S, de lectura y de escritura. Las clases stream se encargan de manipular bytes, y las de escritura y lectura, de manipular caracteres (que en Java contienen dos bytes). El paquete java.io define cuatro clases raíces (InputStream, OutPutStream, Reader y Writer) de las que se deriven la mayoría de las subclases de stream del paquete. Todas las clases stream se basan en la funcionalidad fundamental de estas superclases.

InputStream y Reader

Todos los streams de entrada pueden leer bytes provenientes de algún tipo de fuente de datos. Su principal función consiste en leer datos en forma de bytes y de dos maneras distintas: byte a byte o bloques de tamaño arbitrario. Todas las demas funciones de InputStream se derivan de su función principal. La tabla **19.3** incluye la definición de la clase InputStream.

Tabla 19.3. Cornposicion de la clase InputStream

Variables/Métodos	Descripción
InputStream()	Construye un objeto InputStream
available()	Devuelve el numero de bytes que pueden leerse inmediatarnente sin necesidad de agruparlos
close()	Cierra el stream
mark()	Marca la posicion dentro del stream
markSupported()	Devuelve el estado del indicador para soportar operaciones de rnarcado
read()	Lee datos del stream
reset()	Reestablece la posicion segun la marca
skip O	Ignora ciertos bytes

Hay tres variedades del método read(). Se puede leer un sólo carácter o un array de bytes. Las dos formas de array devuelven el número de bytes a leer en caso de que no existan suficientes para completar el array:

```
public int read()
public int read(byte b[])
public int read(byte b[], int off, int len;
```

Los tres métodos relacionados con marcas (mark(), reset() y markSupported()) se encargan de soportar un stream de entrada opcional para cancelar métodos read() Todos los streams de entrada deberian implementar markSupported(),

aunque tienen la libertad de devolver `true` o `false`. La mayoría devuelve `false`, lo que quiere decir que ni `mark()` ni `reset()` realizan ninguna función. La clase abstracta `InputStream` implementa `markSupported()` y devuelve `false`, por lo que ninguna de las clases derivadas de ésta, y que no la cancelan, soportan marcas.

Cuando sí se soportan marcas, el sistema funciona de la siguiente manera: Marcamos cualquier posición dentro de un stream de entrada a la que regresar mas tarde para eliminar la lectura de datos que se realizó desde el lugar donde colocamos la marca. El argumento `readlimit` del método `mark()` determina el alcance permitido de lectura en el que aún podamos reestablecer la posición de lectura a la posición marcada. El limite de lectura se determina en bytes y se utiliza para localizar un bufer interno para todos los datos cuya lectura pueda ser eliminada. Si leemos mas allá de este limite y reestablecemos mediante `reset()`, los resultados son indeterminados.

La clase `InputStream` incluye toda una colección de clases, cuya jerarquia se muestra en la figura 19.1. Todas ellas comparten una misma caracteristica, además de ser subclases de la misma clase, y es que todos sus constructores permiten especificar una fuente de datos concreta.

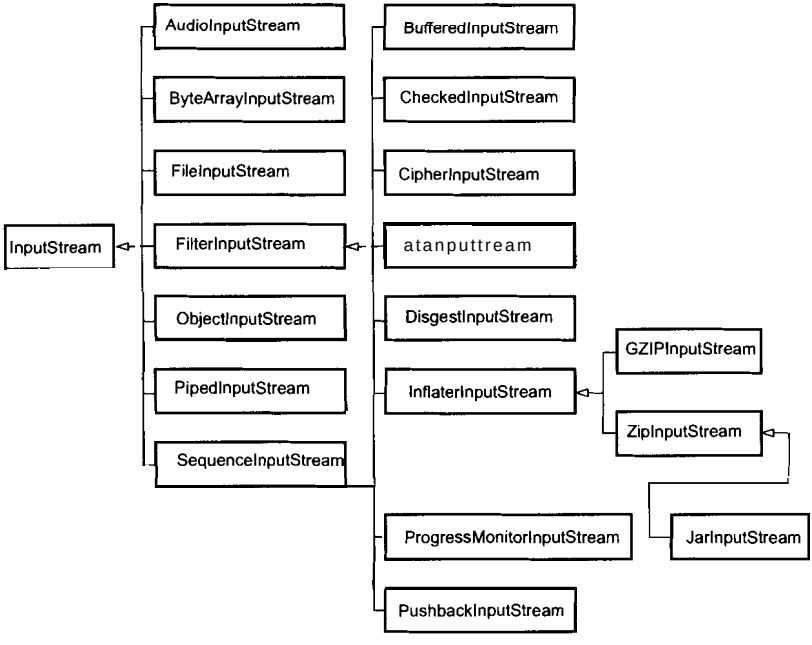


Figura 19.1. Jerarquia de las subclases de InputStream

En el caso de las clases `FileInputStream`, `ByteArrayInputStream`, `SequenceInputStream`, `ObjectInputStream`, y `PipedInputStream`, la fuente de datos es el atributo que las diferencia de otras clases de la lista. Un

FileInputStream, por ejemplo, recibe ese nombre porque el stream de datos que emite procede de un archivo que forma parte del sistema de archivos del equipo. Del mismo modo, el flujo que emite ByteArrayInputStream procede de un pipe, que es un mecanismo que permite dos procesos diferentes para comunicarse.

Todos los derivados de FilterInputStream tienen funciones muy distintas. Sus constructores también nos permiten especificar la fuente del stream. Pero si esto es lo único que podría que hacer, seria de poco valor, ya que se limitará a pasar los datos sin modificarlos. El propósito real de un FilterInputStream no consiste en conectarse a una fuente de datos concreta, como era el caso de las clases anteriores, sino en mejorar los stream, alterando los datos que contiene, (comprimiéndolos, por ejemplo) o añadiendo caracteristicas nuevas a la escasa funcionalidad de la clase InputStream.

La clase abstracta **Reader**, definida en la tabla 19.4, es muy similar a InputStream.

Tabla 19.4. Composición de la clase Reader

Variables/Métodos	Descripción
lock	Lock interno de sincronizacion
Reader()	Construye un objeto Reader
close()	Cierra el stream
mark()	Marca la posición dentro del stream
markSupported()	Devuelve el estado del indicador para soportar la operación de mark()
read()	Lee los datos del stream
reset()	Reestablece la posición según la operación de mark()
skip O	Ignora ciertos bytes

La diferencia entre un **Reader** y los stream de entrada es que el primero gestiona caracteres, mientras que el segundo gestiona bytes. La jerarquia de la clase **Reader** se muestra en la figura 19.2.

Readers, al igual que los stream de entrada, pueden tener distintas fuentes de datos, tales como archivos y strings. Las subclases BufferedReader y FilterReader admiten otros lectores como fuentes de datos y modifican los caracteres. La mayoría de las clases stream de entrada cuentan con sus correspondientes clases **Reader**.

FileInputStream y FileReader

La clase FileInputStream es un stream de entrada cuyos datos provienen de un archivo normal. Como tal, la podemos considerar casi como la clase

RandomAccessFile, que nos permite leer y modificar archivos. Podemos utilizar dos objetos FileInputStream para implementar nuestro programa Diff, con la misma facilidad.

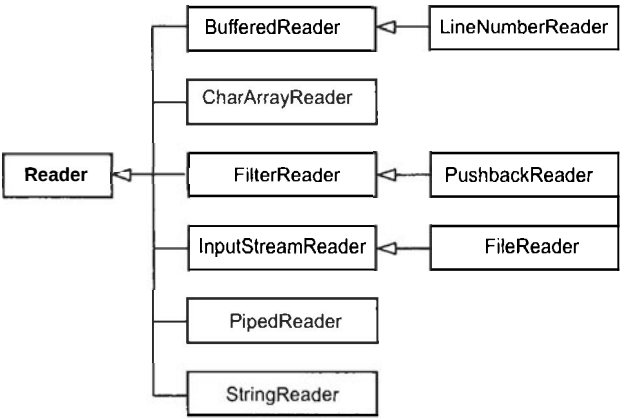


Figura 19.2. Jerarquia de las subclases de Reader

Los constructores de la clase FileInputStream constituyen su mayor contribucion a la clase abstracta InputStream:

```
public FileInputStream(File file) throws FileNotFoundException
public FileInputStream(FileDescriptor fd)
public FileInputStream(String name) throws FileNotFoundException
```

Estos constructores crean un objeto FileInputStream y, al mismo tiempo, abren el archivo, dejándolo listo para su lectura.

Nota

El tercer constructor admite un nombre de archivo en forma de string, aunque esto deberia evitarse si el nombre de archivo tiene algun caracter dependiente de la plataforma. Por ejemplo, el nombre de archivo de `abc\def` **funcionaría correctamente en una plataforma Windows, pero fallaría en un equipo Unix.**

El primer constructor es el que se utiliza, por lo general, ya que admite un objeto de archivo independiente de la plataforma y describe el archivo que va a accederse.

Como ya se dijo anteriormente, es aconsejable no trabajar con un objeto FileDescriptor.

Los constructores de FileReader son idénticos a los de FileInputStream:

```
public FileReader(File file) throws FileNotFoundException
public FileReader(FileDescriptor fd) throws FileNotFoundException
public FileReader(String name) throws FileNotFoundException
```

Nota

Los stream de archivosde entrada deberian usarse unicamente para leerdatos de bytes. Para los archivos de texto, utilizaremos los lectores de archivos ya que las clases `Reader` soportan caracteres completos de 16 bits.

El listado 19.4 representa un programa que utiliza la clase `FileReader` para leer un archivo de texto y calcular la frecuencia de palabras.

Listado 19.4. Cálculo de la frecuencia de palabras con FileReader

```
import java.io.*;
import java.util.*;
public class WC {
  private static final Integer ONE = new Integer(1);
  public static void main(String args[]) throws IOException {
    if (args.length != 1) {
      System.err.println("Usage: java WC <textfile>");
      System.exit(-1);
    }
    Map map = new TreeMap();
    FileReader fr = new FileReader(args[0]);
    StreamTokenizer wordStream = new StreamTokenizer(fr);
    // Trata cualquier signo de puntuación como delimitador de una palabra
    wordStream.whitespaceChars ('!','@');
    int tok;
    String word;
    while ((tok = wordStream.nextToken()) !=
        StreamTokenizer.TT_EOF) {
      if (tok == StreamTokenizer.TT_WORD) {
        word = wordStream.sval;
        count(map, word);

    }
    Iterator it = map.keySet().iterator();
    while (it.hasNext()) {
      word = (String)it.next();
      System.out.println(frequency(map, word) + " " + word);
    }
  }
  /**
   * Comprueba si se encontró la palabra
   * De no ser así, crea un contador nuevo
   * Si es así, incrementa el contador
   */
  private static void count(Map map, String word) {
    Integer counter;
    counter = (Integer)map.get(word);
    if (counter == null) {
      counter = ONE;
    } else {
      counter = new Integer(counter.intValue() + 1);
```

```
    }
    map.put(word, counter);
  }
  /**
   * Indica las veces que se encontró la palabra
   */
  private static int frequency(Map map, String word) {
    Integer counter = (Integer)map.get(word);
    if (counter == null) {
      return 0;
    } else {
      return counter.intValue();
```

La clase principal abre el archivo especificado como un argumento de linea de comandos al crear un objeto `FileReader`. Después, envía este objeto directamente a otra clase **`java.io`**`:StreamTokenizer`.

La clase **`StreamTokenizer`** envía un **`reader`** en forma de argumento a su constructor y después nos permite dividir el stream en las palabras que lo constituyen. El método `nextToken()` de **`StreamTokenizer`** divide el stream del archivo en palabras mediante la variable `sval` (String value) de **`StreamTokenizer`**. Después, las palabras extraidas del stream se envian al objeto `TreeMap`, que hace un seguimiento de las palabras que ya se han encontrado y de las veces. Puesto que **`TreeMap`** mantiene una lista ordenada de palabras, para imprimir los resultados sólo tenemos que desplazarnos por el *iterator* para encontrar el conjunto de claves.

Nota

La última sección del nuevo E/S incluye el paquete **`java.util.`**`regex`, que contiene un analizador de expresiones regulares que nos ofrece una manera alternativa de dividir líneas en palabras.

ByteArrayInputStream

La clase `ByteArrayInputStream` crea un stream de entrada a partir de un conjunto de bytes, más concretamente de un array de bytes. El principal constructor de esta clase es:

```
public ByteArrayInputStream(byte buf[])
```

Nota

Puesto que los string de Java se componen de caracteres Unicode de 16 bits, debemos utilizar un `StringReader` si vamos a utilizar un string como elemento de entrada. La clase `StringBufferInputStream` deberia evitarse.

La clase ByteArrayInputStream no copia sus datos respectivos de entrada en un búfer interno, por lo que la aplicación del cliente puede modificar el array fuente en cualquier moniento en que se esté utilizando el stream. Debido a que este tipo de conflicto puede causar errores muy dificiles de detectar, es aconsejable clonar el array antes de utilizarlo como la fuente de ByteArrayInputStream:

```
byte buf[]   ...;
byte buf2[] = (byte[])buf.clone();
ByteArrayInputStream bais = new ByteArrayInputStream(buf2);
```

CharArrayReader y StringReader

Las clases CharArrayReader y StringReader son la contrapartida de ByteArrayInputStream, pero orientadas a caracteres. Los constructores principales de estas dos clases son los siguientes:

```
public CharArrayReader(char chars[]);
public StringReader(String s)
```

Advertencia

Como ocurre con los correspondientes stream de entrada, puede darse un problema en cuanto a la integridad de los datos. Si se modifica el array de un char, el comportamiento del reader es imprevisible. Esto no constituye un problema con un stringreader ya que la fuente de datos esta constituida por un string inmutable.

SequenceInputStream

La clase SequenceInputStream nos permite unir dos o más streams de entrada. (No cuenta con la correspondiente clase reader. stream.) Cada vez que leamos este "superstream" y nos encontremos un marcador de final de archivo (EOF), la clase SequenceInputStream pasa al siguiente stream de la lista, sin que el EOF llegue hasta nosotros. Sólo al llegar al último stream de entrada, se hará visible el marcador EOF (un -1 devuelto por read()). Los constructores ofrecen dos modos de especificar la secuencia de streams. La primera manera es aconsejable sólo cuando necesitemos unir dos streams de entrada:

```
public SequenceInputStream(InputStream s1, InputStream s2)
```

La segunda forma es más general en su capacidad de admitir una lista abierta de streams de entrada:

```
public SequenceInputStream(Enumeration e)
```

El tipo de argumento requerido no es el esperado. La manera ideal de especificar una lista ordenada de Inputstream seria mediante un array seguro de

InputStream (tal como una lista InputStream[]), sin embargo, se utiliza un objeto Enumeration, mucho menos preciso.

Cualquier objeto Enumeration debe implementar los metodos hasMoreElements() y nextElement(), lo que nos puede hacer pensar que necesitamos crear una nueva clase que implemente la interfaz Enumeration sólo para especificar una lista de streams de entrada a este constructor. Pero existe una manera mds sencilla que consiste en declarar un simple vector en el que añadir la secuencia de los objetos InputStream, mediante add(), y utilizar después los elements() del vector para obtener un objeto Enumeration. Esto se hace del siguiente modo:

```
InputStream streaml = ...;
InputStream stream2 = ...;
InputStream stream3 = ...;
Vector v = new Vector(3);
v.add(stream1);
v.add(stream2);
v.add(stream3);
SequenceInputStream sis = new SequenceInputStream(v.elements());
```

PipedInputStream y PipedReader

El concepto de Pipes, muy similar al de stream, es otra de las invenciones de Unix. Un pipe (o conector) se utiliza especialmente cuando dos procesos distintos (tal como tasks o threads) necesitan transmitirse grandes cantidades de datos de una manera sincronizada. Uno de los aspectos mds complejos de utilizar multiples threads es precisamente la sincronización. Pocos sistemas multitareas pueden evitar la necesidad de que algunos threads se encuentren por algún motivo.

Si sólo se trata de intercambiar datos, los pipes pueden solucionar el problema de una manera muy sencilla. Un ejemplo menos comun es la utilización de barras verticales (|) para encadenar programas en los mensajes de Unix o DOS. Por ejemplo el siguiente comando:

```
C:\> DIR | SORT | MORE
```

crea dos pipes. El primero conecta los elementos de salida del comando DIR con los de entrada del comando SORT. El segundo pipe conecta los elementos de salida de SORT con los de entrada de MORE. El resultado es una lista de directorios organizados mediante SORT antes de que se muestren pdgina por pdgina.

La clase PipedInputStream debe conectarse con PipedOutputStream. Estas dos clases tienen que utilizarse juntas y su utilidad es nula por separado. Sus constructores respectivos son:

```
public PipedInputStream(PipedOutputStream src) throws IOException
public PipedOutputStream(PipedInputStream snk) throws IOException
```

Los constructores reader y write son:

```
public PipedReader(PipedWriter src) throws IOException
public PipedWriter(PipedReader snk) throws IOException
```

Si estudiamos estos constructores cuidadosamente, nos daremos cuentas de que falta algo. ¿Cómo podemos crear una conexión entre dos pipes si necesitamos transmitir al constructor el que ya existe en forma de argumento? Sólo puede conseguirse mediante un bypass realizado por uno de los otros tipos de constructores:

```
public PipedInputStream()
public PipedReader()
public PipedOutputStream()
public PipedWriter()
```

Estos cuatro constructores nos permiten crear un objeto `pipe` que no esté aún conectado. Para completar la conexión, utilizaremos el método respectivo `connect()`.

La refinería de petróleo

El siguiente programa demuestra cómo usar las clases de stream conectadas en el contexto de la comunicación entre threads. Utiliza la metáfora de un buque cisterna que se conecta mediante tuberias a la refineria de petróleo para transferir el crudo y así ser procesado. La refineria devuelve el crudo al buque en forma de una substancia refinada indeterminada. El listado 19.5 es el programa de este ejemplo:

Listado 19.5. Stream conectadas en la refineria de petroleo

```
import java.io.*;
public class PipeTest {
  public static void main(String args[]) {
    new OilRefinery();
    try { // retrasa la llegada
      Thread.currentThread().sleep(500);
    } catch (InterruptedException e) {
    }
    new SuperTanker();
  }
  /**
   * Esta clase consiste en un Thread que puede aceptar un pipe
   * Los clientes tienen que encontrar el nombre del Thread
   * "ThePipeTerminal"
   */
  static class OilRefinery extends Thread {
    static final int EOF = -1;
    boolean alone = true;
    // No puede conectarse
    PipedReader inPipe;
    PipedWriter outPipe;
    public OilRefinery() {
      start(); // Cornienza el thread
    }
    public synchronized void run() {
      int ch;
      // Comienza a funcionar
      setName("ThePipeTerminal");
      System.out.println("Processing plant operational and on-line");
```

```
    while (alone) {
      try {
        wait(); // No hay espera para la conexión
      } catch (InterruptedException ignored) {

    }
    System.out.println("Client arrived");
    // El cliente se ha conectado y ha comenzado a procesar el flujo de
     // "crudo"
    try {
      while ((ch = inPipe.read()) != EOF) {
        // añade algún valor a datos de entrada iniciales...
       outPipe.write(Character.toUpperCase((char)ch));
      }
    } catch (IOException pipeMalfunction) {
    }
    try {
      outPipe.close();  // Indica que se ha terrninado el proceso
    } catch (IOException ignored) {
    }
    alone = true;
    System.out.println("Processing plant shutting down.");
  }
  /**
   * Con este método el cliente realiza la llamada para conectarse a la
   * planta de procesarniento.
   */
  public synchronized boolean clickClunk(PipedWriter clientOutputPipe,
                                         PipedReader clientInputPipe) {
    System.out.println("Client arrives to hook-up its pipes");
    try {
      inPipe = new PipedReader(clientOutputPipe);
      outPipe = new PipedWriter(clientInputPipe);
    } catch (IOException connectionFailed) {
      System.err.println("Hook up failed");
      return false;
    }
    System.out.println("Hook-up successful");
    alone = false;
    notifyAll();
    return true;
  }
}
/**
 * Esta clase irnplernenta un cliente de la planta de procesamiento, en
 * este caso, el buque que descarga el crudo y se carqa con petrhleo
 * refinado.
 */
static class SuperTanker {
  OilRefinery pipeTerminal = null;
  PipedReader returnPipe = new PipedReader();
  PipedWriter crudePipe = new PipedWriter();
  public SuperTanker() {
    pipeTerminal = (OilRefinery)findThread("ThePipeTerminal");
    if (pipeTerminal == null) {
      System.err.println("Snow blizzards prevent rendezvous");
```

```
      System.exit(-1);
    } else {
      if (pipeTerminal.clickClunk(crudePipe, returnPipe)) {
        haveOilProcessed();
      } else {
        System.err.println("Failed to connect to processing plant");

      try {
        crudePipe.close();
      } catch (IOException brokenValves) {
        System.err.println("Couldn't  close valves!");
      }


  /**
   * Envía "el petróleo" a la planta de procesamiento, que lo refina y lo
   * devuelve mediante el segundo pipe
   */
  public void haveOilProcessed() {
    String oilToBeRefined = "Crude Oil";
    try {
      crudePipe.write(oilToBeRefined);
      crudePipe.close();
      // Recupera el petróleo refinado
      int ch;
      while ((ch = returnPipe.read()) != -1) {
        System.out.print((char)ch);
      }
      System.out.println();
    } catch (IOException oilFlowFailure) {
      System.err.println("Pipe malfunction");
    }

  /**
   * Este método genérico encuentra el thread
   * Observe que los threads pueden comenzar o terminarse mientras se
   * comprueban
   */
  public Thread findThread(String target) {
    int SAFETY-MARGIN = 10;
    // Encuentra el ThreadGroup mister
    ThreadGroup rootGroup = Thread.currentThread().getThreadGroup();
    while (rootGroup.getParent() != null) {
      rootGroup = rootGroup.getParent();
    }
    Thread threadList[] =
      new Thread [rootGroup.activeGroupCount() + SAFETY-MARGIN];
    int count = rootGroup.enumerate(threadList);
    Thread aThread;
    for (int i=0; i<count; i++) {
      aThread = threadList[i];
      if (aThread == null)
        continue;
      if (aThread.getName().equals(target)) {
        return aThread;
```

```
        return null;
    }
}
```

Este programa no es interesante sólo por su utilizacidn de `PipedReader` y `PipedWriter`. Como podemos observar, el programa principal crea dos muestras de las clases fundamentales (`OilRefinery` y `SuperTanker`) que son totalmente independientes en principio.

El objeto `OilRefinery` se ejecuta como un thread independiente al ser construido, lo cual es posible por tratarse de una subclase de `Thread`. Como consecuencia, el constructor llama a `start()`, y el método `run()` comienza a ejecutarse.

El cuerpo del thread `OilRefinery` se nombra asi mismo para que otras entidades de la misma máquina virtual de Java (JVM)se comuniquen con él. Después se inmersa en un estado de semi-inactividad esperando un cliente (un buque `SuperTanker`) que solicite su servicio.

Si analizamos al constructor del objeto `SuperTanker` observaremos que su primera tarea consiste en encontrar el thread `OilRefinery`, escaneando todos los threads de la raiz `ThreadGroup`.Una vez encontrado el objeto `OilRefinery`, `SuperTanker` le transmite los dos objetos pipe: un `PipedWriter` para el crudo, y un `PipedReader` para el petróleo procesado que va a devolverse al buque mediante `OilRefinery`. El `OilRefinery` recibe esos dos conectores (mediante el método `clickClunk()`) y los conecta a sus propios pipes. De este modo, el pipe de salida del buque es el pipe de entrada de la refineria, y a la inversa. La figura 19.3 es una representación grafica de las conexiones.

Como parte del método `clickClunk()`, el `OilRefinery` activa su thread principal al cambiar el estado de un valor booleano y al invocar `notifyAll()`. Mientras, el objeto `SuperTanker` comienza la transferencia principal en el método `haveOilProcessed()`. El código fundamental es la invocación a `write()` en el `PipedWriter` de salida. Estos datos llegan hasta el `read()` del `PipedReader`. Para simbolizar este proceso, los datos recibidos (el string `data` (el string de `chars` que lee el `Crude Oil`) se cambian a mayusculas y estos bytes procesados se envian a la refineria, para regresar de nuevo al buque mediante el envio del método `write()` al `PipedWriter` de la `OilRefinery`. El `SuperTanker` recibe estos caracteres y los imprime.

Un detalle importante de esta interacción es que cualquier entidad de escritura tiene que indicar a los otros elementos que no hay más datos, mediante el cierre del stream de salida. Esto es precisamente lo que hace el objeto `SuperTanker`, justo después de enviarwrite `()` a su `PipedWriter`. De manera similar, el `OilRefinery` envía un `close()`, tras detectar que se ha llegado al final de los datos del stream de entrada. Ésta es la unica manera en que el final puede recibir una indicacidn EOF que permita detener el bucle de lectura. Es posible también utilizar otros medios, tal como un indicador de fin de mensaje, pero para ello, debemos utilizar `flush()` para asegurar el tránsito de datos escritos previamente.

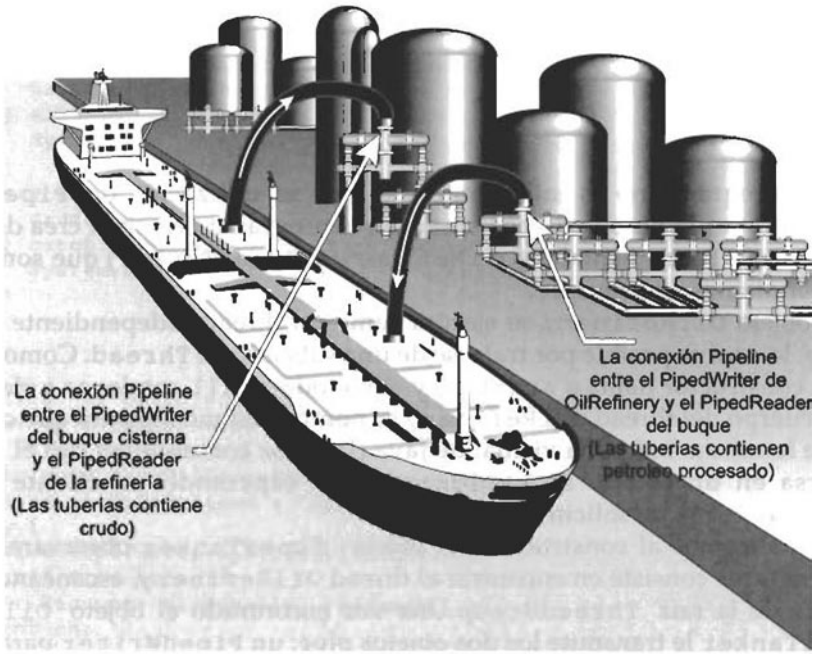


Figura 19.3. Los objetos PipedReader y PipedWriter en un escenario real

Una vez explicadas las dos clases stream de pipe, pasaremos a examinar cómo el objeto `SuperTanker` es capaz de comunicarse con el thread de `OilRefinery`. Cualquier thread que se ejecute en la JVM forma parte de una colección jerarquizada de objetos `ThreadGroup`. Para encontrar un thread dado, debemos primero encontrar la parte más importante en la jerarquia de `ThreadGroup`, para así poder buscar en este árbol dicho elemento. Esto es precisamente lo que hace el método `findThread()` en la clase `SuperTanker`, basándose en dos otros métodos fundamentales: `getParent()`, para encontrar el elemento pricipal en la jerarquia, y `enumerate()`, para mostrar todas las muestras de `Thread` en el grupo.

ObjectInputStream

Tanto los objetos stream de entrada como los de salida soportan la serialización de objetos. Mediante este proceso, lo que se lee o se escribe no es ni un byte, ni un array de bytes sino un objeto en si mismo, junto con todas sus referencias. Esto resulta bastante más complicado que escribir otro tipo de datos, y puede causar problemas en cuanto a la seguridad, la privacidad de los datos, y a la gestión de las diferentes versiones de clases.

Gracias a este proceso, un objeto puede almacenarse fuera de la JVM que lo creó, para ser restablecido más adelante. Por ejemplo, un programa gráfico de dibujo puede representar su estado como una muestra de una clase llamada `Picture`. Cuando el usuario quiera guardar el dibujo, el programa abrirá un stream de salida para archivo y enviarlo al constructor de un objeto stream de salida. Como conse-

cuencia, el programa podrá guardar toda la clase Picture mediante un sencillo método. La clase Picture puede entonces escribirse en el stream de salida, a través del stream de salida de archivo y después en un disco. Más tarde, en una invocación distinta del programa de dibujo, un stream de entrada del objeto y del archivo puede reestablecer el objeto Picture para que el usuario pueda continuar con su trabajo.

Nota

Si la fuente de ObjectInputStream es un PipedInputStream, el correspondiente ObjectOutputStream debe conectarse al PipedOutputStream antes de que su thread continue. De lo contrario, el thread estara permanentemente bloqueado en el punto de creación.

InputStreamReader

El InputstreamReader se diseñó para convertir tareas de bytes en tareas de caracteres. El reader de un stream de entrada lee bytes de un stream de entrada y los convierte en caracteres, de acuerdo con el algoritmo de mapeo. La aplicacidn de mapas por defecto depende del cddigo que se utilice también por defecto. Por lo general, se tratará de una conversión de bytes en forma de caracteres ASCII comunes en caracteres Unicode. El constructor de la clase InputStreamReader utiliza un stream de entrada como su fuente de datos:

```
public InputStreamReader(InputStream src)
```

Otros constructores permiten conversiones, basándose en conjuntos de caracteres o en nombres de conjuntos de caracteres:

```
public InputStreamReader(InputStream src, CharSet cs)
public InputStreamReader(InputStream src, CharsetDecoder dec)
public InputStreamReader(InputStream src, String charsetName)
```

FilterInputStream and FilterReader

Las ~lasesFilterInputStream y FilterReader son las superclases de muchas de las clases de java.io. FilterInputStream es una clase pseudo-abstracta, mientras que FilterReader es una clase genuinamente abstracta. Pero lo importante es que ambas sirven para subclasificar elementos y no para crear muestras. Para crear nuestro propio stream personalizado, es muy probable que empleemos una de las dos.

BufferedInputStream y BufferedReader

La clase BufferedInputStream mejora la funcionalidad de InputStream, al aiiadirle un búfer de bytes, lo cual suele mejorar la lectura de manera significativa.

Del mismo modo, BufferedReader añade un bufer de caracteres al Reader. Para demostrar el incremento en la velocidad de lectura, el listado 19.6 representa un programa que utiliza un FileReader para leer un archivo completo, byte por byte:

Listado 19.6. Cronometraje de reads sin almacenar en un bufer

```
import java.io.*;
import java.util.*;
public class Unbuffered {
  public static void main(String args[]) {
    Reader reader;
    int ch;
    System.out.println("Start! " + new Date());
    try {
      reader = new FileReader(args[0]);
      while ((ch=reader.read()) != -1) {
        // Lee todo el archivo
      }
    } catch (IOException io) {
      System.err.println(io);
      System.exit(-1);
    }
    System.out.println("Stop  " + new Date());
  }
```

Al ejecutar el programa de un archivo de gran tamaiio, tal como el VBA6.DLL de Windows 2000 de 1.694.992 bytes, tarda unos 23 segundos en leer los bytes. Pero si cambiamos la linea de codigo de Reader por las siguientes dos líneas:

```
FileReader fr = new FileReader(args[0]);
reader = new BufferedReader(fr);
```

el tiempo de lectura se reduce a sólo 1 segundo. La mejora en el resultado depende del tamaiio del bufer utilizado por el objeto BufferedReader.

Aunque en el ejemplo se utiliza el constructor por defecto, lo que resulta en un bufer de un tamaño también por defecto, pero indeterminado, también es posible definir explicitamente el tamaño del bufer, utilizando constructores de la siguiente manera:

```
public BufferedInputStream(InputStream in, int size)
public BufferedReader(Reader in, int size)
```

Nota

Las clases BufferedInputStream y BufferedReader son de las pocas clases estándar de java.io que implementan el mecanismo de marcado y de reestablecimiento, descrito anteriormente.

DataInputStream

Otra clase que implementa RandomAccessFile, además de DataInput, es DataInputStrearn.

Esta relación entre RandomAccessFile y DataInputStream existe también en las variantes de Output de las clases java.io. La clase DataOutputStream implementa la interfaz DataOutput, que se implementaba a su vez por RandomAccessFile. Como resultado, hay compatibilidad de formatos entre los datos escritos por RandomAccessFile y los que DataInputStream lee. Del mismo modo, los datos escritos por DataOutputStream pueden leerse mediante RandomAccessFile. En la práctica, sin embargo, esta comunicación cruzada no ocurre muy a menudo ya que los datos se escriben y se leen mediante las clases correspondientes, o, lo que es más frecuente, mediante DataOutputStream y DataInputStrearn.

La tabla 19.5 representa la definicidn de la clase DataInputStream. La interfaz DataInput especifica de manera implicita las clases que deben gestionar un **EOF,** referido al final de un stream y no sólo de un archivo. Para ello no devuelve el valor de "-1", sino que devuelve un objeto EOFException. (Debido a que la clase EOFException es una subclase de IOException; la definicidn no lo incluye.) Como consecuencia, puede utilizarse una plantilla de código distinta para leer streams mediante un DataInputStream. Podemos simplemente implementar un bucle continuado que no busque ni detecte **EOF,** y confiar en el código de detección de errores para que gestione la condición **EOF** de manera adecuada, lo que se demuestra en el listado 19.7.

Tabla **19.5.** Cornposicion de la clase DataInputStrearn

Variables/Métodos	Descripción
DataInputStream()	Construye un objeto DataInputStream
read()	Lee datos del archivo
readBoolean()	
readByte()	
readChar()	
readDouble()	
readfloat()	
readFully()	
readInt()	
readLong()	
readShort()	

Variables/Métodos	Descripción
readUnsignedByte()	
readUnsignedShort()	
readUTF()	
skipBytes()	Se salta datos del archivo

Listado 19.7. Lectura de los bytes de DataInputStream

```
import java.io.*;
public class EOF {
  public static void main(String args[]) {
    DataInputStream is = null;
    byte ch;
    try {
      is = new DataInputStream(new FileInputStream("EOF.java"));
      while [true) ( //La excepción se encarga de EOF
        ch = is.readByte();
        System.out.print([char)ch);
        System.out.flush();
      }
    } catch (EOFException eof) {
      System.out.println (" >> Normal program termination.");
    } catch (FileNotFoundException noFile) {
      System.err.println("File not found! " + noFile);
    } catch (IOException io) {
      System.err.println("I/O error occurred: " t io);
    } catch (Throwable anything) {
      System.err.println("Abnormal exception caught !: " + anything);
    } finally {
      if (is != null) {
        try {
          is.close();
        } catch (IOException ignored) {

      }
    }
  }
}
```

LineNumberReader

La clase **LineNumberReader** permite leer los numeros de línea de los streams de entrada de texto. Los dos métodos siguientes se encargan de controlar esta función.

```
public int getLineNumber()
public void setLineNumber(int lineNumber)
```

El listado 19.8 contiene un programa que utiliza la clase **`LineNumberReader`** para imprimir cualquier archivo de texto.

Listado 19.8. Visualización del número de linea para cada lectura de linea

```
import java.io.*;
public class Lineno {
  public static void main (String args[]) {
    FileReader       fr = null;
    BufferedReader   br = null;
    LineNurnberReader lnr = null;
    String           line;
    try {
      fr = new FileReader(args[0]);
      br = new BufferedReader(fr);
      lnr = new LineNurnberReader(br);
      while ((line = lnr.readLine()) != null) {
        int lineNo = lnr.getLineNumber();
        System.out.println(lineNo + " " + line);
      }
    ) catch (IOException ioErr) {
      System.err.println (ioErr);
      System.exit(-l);
    } finally {
      if (lnr != null) {
        try {
          lnr.close();
        } catch (IOException ignored) {
        }
      }
      if (br != null) {
        try {
          br.close();
        } catch (IOException ignored) {
        }
      \
      if (fr != null) {
        try {
          fr.close();
        } catch (IOException ignored) {
        }
      }
    }
  }
}
```

El programa Lineno utiliza un pipe de entrada de tres pasos:

```
args[0] file -> FileReader -> BufferedReader -> LineNurnberReader
```

Los datos leidos por `LineNumberReader` se desplaza por el stream, originados en el archivo especificado en la linea de comandos y pasando por **`FileReader`** y **`BufferedReader`**. Este ultimo, carga todo el programa y el **`LineNumberReader`** proporciona los numeros de linea. El orden de los diferentes tipos de stream puede

ser importante. En este ejemplo no tendría sentido situar el objeto `BufferedReader` al final de la cadena. Los stream de entrada almacenados y los lectores deben ser el segundo eslabón de la cadena para que todos los estados inferiores puedan beneficiarse del almacenamiento en buferes. Del mismo modo, los objetos `BufferedOutputStream` y `BufferedWriter` debenserlospenultimosenlacadenade salida.

Nota

En este programa, podriamos haber rastreado el numero de línea mediante una integral incrementada para cada repetición del bucle, pero estariamos poco menos que reinventando la rueda, que no es precisamente en lo que consiste la programacion. En lugar de tener que añadir el codigo de manera manual, podemos cortar y pegar el desarrollo del programa y el tiempo de prueba, reutilizando de esta manera clases ya existentes, y libres de errores.

PushbackInputStream y PushbackReader

Las clases `PushbackInputStream` y `PushbackReader` añaden la función "Deshacer" a un stream de entrada o a un lector, de la cual se encargan los siguientes métodos:

```
public void unread(int ch) throws IOException
public void unread(byte/char ch[]) throws IOException
public void unread(byte/char ch[], int offset, int len) throws IOException
```

InflaterInputStream and Children

Dentro de las biblioteca de Java encontramos métodos que posibilitan la compresión y descompresión. Los stream de entrada permiten la descompresión, mientras que los de salida permiten la compresión.

La clase básica para la compresión es `InflaterInputStream`, la cual se compone de las subclases `GZIPInputStream`, `ZipInputStream`, y `JarInputStream` para trabajar con archivos GZIP, ZIP, y JAR, respectivamente. Estas clases funcionan como streams de filtro normales, que sólo necesitan incluirse en otro stream para que sus caracteristicas se añadan a las del conjunto de streams. Con GZIP, el stream se comprime en masa. Con ZIP y JAR, sin embargo, cada stream contiene entradas individuales.

Las clases `Zip` se encuentran en el paquete `java.util.zip`. La clase `ZipFile` representa un archivo ZIP. Podemos construir una muestra de esta clase mediante cualquiera de estos constructores:

```
public ZipFile(String name) throws IOException
public ZipFile(File file) throws ZipException, IOException
public ZipFile(File file, int mode) throws ZipException, IOException
```

Esencialmente, este proceso se basa en conectarnos a un archivo zip mediante un `String` que contiene el nombre del archivo o un `File`. Para el tercer constructor, el modo es OPEN_READ (para archivos ZIP de sólo lectura), o OPEN–READ | OPEN_DELETE, (cuando el archivo ZIP puede leerse pero se borrará tras terminar la operación). La tabla 19.6 muestra lo que podemos hacer con un objeto `ZipFile`.

Tabla 19.6. Composición de la clase ZipFile

Variables/Métodos	Descripción
`OPEN_READ`	Modos de acceso
`OPEN_DELETE`	
`ZipFile()`	Construye un objeto ZipFile
`close()`	Cierra el archivo ZIP
`entries()`	Devuelve una entrada como enumeración
`finalize()`	Comprueba que el archivo esta cerrado
`getEntry()`	Devuelve la referencia a una entrada concreta
`getInputStream()`	Devuelve el stream de una entrada concreta
`getName` O	Devuelve la ruta de un objeto ZipFile
`size()`	Devuelve el numero de entradas en el objeto ZipFile

Como se ve en esta tabla, para extraer entradas del ZipFile, utilizamos el método `entries()`, que las devuelve en forma de enumeracibn.

```
public Enumeration entries()
```

Cada elemento de `Enumeration` es un `ZipEntry`. En la tabla 19.7 podemos ver que un `ZipEntry` es como un objeto `File`, aunque contiene sólo información sobre la entrada y no los datos en si. Para acceder el `InputStream` de un `ZipEntry` debemos utilizar `ZipFile`. Una vez hayamos accedido a una de las entradas a través de `Enumeration` de `ZipFile`, el siguiente paso consiste en pedir a `ZipFile` el `InputStream` de `ZipEntry` mediante `getInputStream()`:

```
Enumeration entries = zf.entries();
while (entrien.hasMoreElements()) {
  ZipEntry ze = (ZipEntry)entries.nextElement();
  InputStream is = zf.getInputStream(ze);
}
```

Para demostrar la lectura de archivos ZIP, el listado 19.9 acepta un nombre de archivo zip desde la linea de comandos, para, a continuacibn, mostrar el nombre de cada entrada y preguntar si deseamos ver el contenido. Si es asi, debemos introducir la respuesta en inglés yes, en mayúsculas o en minúsculas.

Tabla **19.7.** Cornposicion de la clase ZipEntry

Variables/Métodos	Descripción
DEFLATED	Método de cornpresion
STORED	
ZipEntry()	Construye un objeto ZipEntry
clone()	Crea una copia de ZipEntry
getComment()	Devuelve cornentarios de ZipEntry
getCompressedSize()	Devuelve el tarnatio de cornpresion de ZipEntry
getCrc()	Devuelve el valor de la verificación de la redundancia ciclica (CRC)
getExtra()	Devuelve los datos designados corno datos extra
getMethod()	Devuelve el rnetodo de cornpresion
getName()	Devuelve el nornbre
getSize()	Devuelve el tarnatio real
getTime()	Devuelve la rnarca de tiernpo
hashCode()	Devuelve el codigo hash de ZipEntry
isDirectory()	Indica si ZipEntry es un directorio
setComment()	Cambia los cornentarios de ZipEntry
setCompressedSize()	Cambia el tamaño de cornpresion de ZipEntry
setCrc()	Cambia el valor CRC
setExtra()	Cambia los datos designados corno datos extra
setMethod()	Cambia el rnetodo de cornpresion
setSize()	Cambia el tarnatio real
setTime()	Cambia la rnarca de tiernpo
toString()	Devuelve la representación en cadena del estado de ZipEntry

Listado 19.9. Lectura de los contenidos de un archivo ZIP

```
import java.util.zip.*;
import java.util.*;
import java.in.*;
public class ReadZip {
  public static void main(String args[]) {
    try {
```

```
ZipFile zf = new ZipFile(args[0]);
Enumeration entries = zf.entries();
// System.in es el elemento de entrada de la linea de comandos
BufferedReader input = new BufferedReader(
  new InputStreamReader(System.in));
while (entries.hasMoreElements()) {
  ZipEntry ze = (ZipEntry)entries.nextElement();
  System.out.println("Read " + ze.getName() + "?");
  String inputLine = input.readLine();
  if (inputLine.equalsIgnoreCase("yes")) {
    long size = ze.getSize();
    if (size > 0) {
      System.out.println("Length is " + size);
      BufferedReader br = new BufferedReader(
        new InputStreamReader(zf.getInputStream(ze)));
      String line;
      while ((line = br.readLine()) != null) {
        System.out.println(line);
      }
      br.close();
    }

  }
} catch(IOException el {
  e.printStackTrace();
}
```

Nota

Mas adelante verernos corno leer streams GZIP, **así** corno **los** rnetodos de salida de streams comprimidos.

Tokenizing Input Streams

El **término inglés** Tokenizing aplicado a datos de entrada significa reducirlos a **cadenas más sencillas** de marcadores que representan bloques de datos recurrentes en el stream. Cualquier compilador de Java comprueba de **manera** automdtica la **gramática** de **los** programas, examinando la secuencia de tokens o marcadores que representan strings de **términos** tales corno `class`, `import`, `public`, o `void`. Esta **técnica** ofrece dos **ventajas:**

- Reduce la complejidad **del código.**
- **Permite** cambios **rápidos** en la sintaxis.

Si **los** diseiiadores de Java hubiesen querido en **algún momento** cambiar el nombre de `extends` por `subclasses`, podrian haberlo hecho de una **manera** muy

sencilla sin que el compilador se viera afectado. El compilador transmitiria el mismo marcador `TOKEN_EXTENDS` para su comprobación gramatical, aunque el código no incluyera la palabra `subclasses`.

StreamTokenizer

La clase `StreamTokenizer` puede utilizarse para convertir cualquier stream de entrada en un stream de marcadores. Un stream puede contener tres tipos distintos de elementos:

- Palabras (o marcadores multicaracteres).
- Marcadores de un único carácter.
- Espacios en blanco (incluyendo comentarios para C/C++/Java)

Antes de comenzar a procesar un stream en tokens, debemos designar qué caracteres deberian tratarse como uno de estos tres tipos, lo que consiste en definir la tabla sintagmática del stream. Una vez terminado este proceso, podremos comenzar a extraer los marcadores reales.

> **Nota**
>
> En la actualidad solo necesitamos esta clase para tareas de mantenimiento. El paquete de expresiones regulares incluido al final de este capítulo proporciona maneras mucho mas efectivas de analizar archivos.

Output Streams y Writers

Otra parte fundamental del paquete `java.io` son las clases `OutputStream` y `Writer`. La tabla **19.8** contiene la definición completa de la primera.

`OutputStream` incluye tres variedades del método `write()`:

```
public abstract void write(int b) throws IOException
public void write(byte b[]) throws IOException
public void write(byte b[], int off, int len) throws IOException
```

Tabla 19.8. Composición de la clase OutputStream

Variables/Métodos	Descripción
`OutputStream()`	Construye un objeto OutputStream
`close()`	Cierra el stream

Variables/Métodos	Descripción
`flush()`	Escribe de golpe cualquier dato que quede en el stream
`write()`	Escribe datos en el stream

Como podemos ver en la tabla 19.9, la definición de `Writer` es casi idéntica a la de `OutputStream`.

Tabla 19.9. Composición de la clase Write

Variables/Métodos	Descripción
`Writer()`	Construye un objeto Writer
`close()`	Cierra el stream
`flush()`	Escribe de golpe cualquier dato que quede en el stream
`write()`	Escribe datos en el stream

Existen cinco variedades del método `write()`:

```
public void write(int c) throws IOException
public void write(char c[]) throws IOException
public abstract void write(char c[], int off, int len) throws IOException
public void write(String s) throws IOException
public void write(String s, int off, int len) throws IOException
```

Estas dos clases son aún más sencillas que los streams de entrada y los lectores, ya que no soportan el mecanismo de marcación ni el de reestablecimiento, que no son necesarios para los datos de salida. Estas clases permiten escribir bytes o caracteres de uno en uno o en bloques.

El destino de estos datos puede ser cualquiera, en teoría, aunque en la práctica, se limita a archivos y conexiones network. (Estas últimas se discutirán en el capítulo 20.)

Nota

Aparte de la dirección del flujo de datos, hay poca diferencia entre la entrada y la salida de los mismos.

La figura 19.4 muestra las clases concretas que se derivan de `OutputStream`. La diferencia entre `writers` y los stream de salida es que los primeros gestionan caracteres, mientras que los segundos gestionan bytes. La figura 19.5 muestra las subclases de `Writer`.

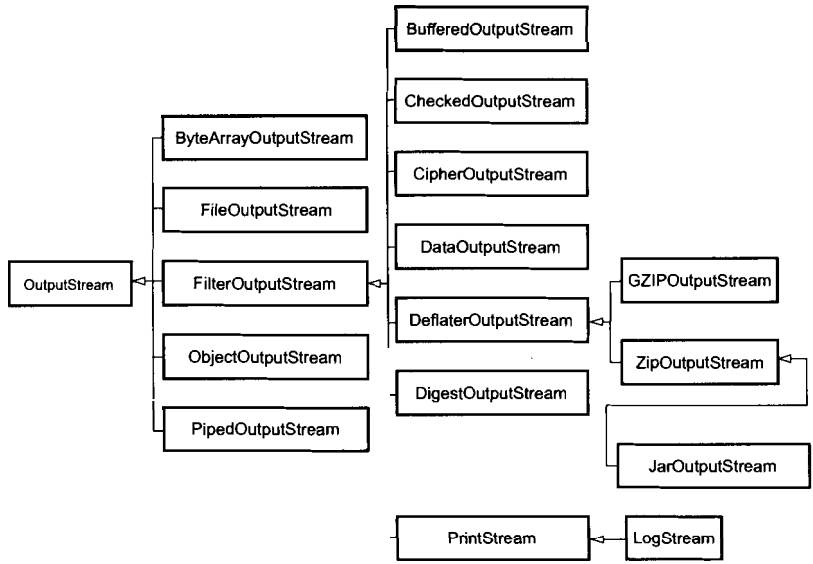


Figura 19.4. Jerarquia de la clase OutputStream

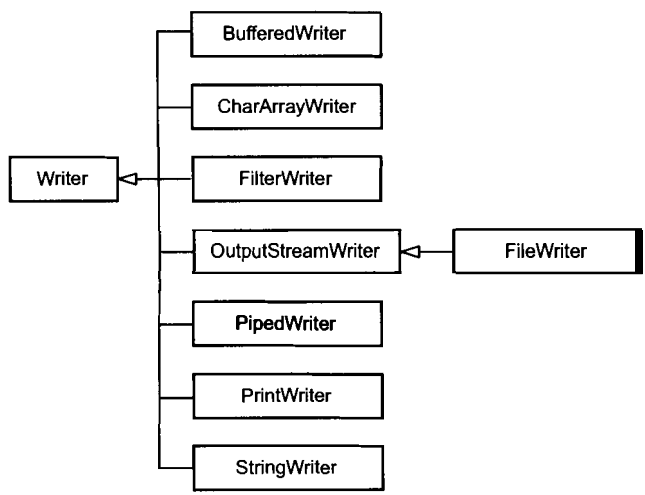


Figura 19.5. Jerarquia de la clase Writer

ByteArrayOutputStream y CharArrayWriter

Las clases `ByteArrayOutputStream` y `CharArrayWriter` **son las opuestas de** `ByteArrayInputStream` y `CharArrayReader`, **respectivamente. Proporcionan algunos métodos extra más allá de los limitados métodos** `write()` **definidos por** `OutputStream` **y** `Writer`. **La tabla 19.10 muestra la composición de** `ByteArrayOutputStream`.

Tabla **19.10.** Cornposicion de la clase ByteArrayOutputStream

Variables/Métodos	Descripción
buf	Contiene datos
count	Tamaño de los datos
ByteArrayOutputStream()	Construye un objeto ByteArrayOutputStrearn
close()	Cierra el stream
reset()	Reestablece el tamaño del bufer interno
size()	Devuelve el tamaño del bufer interno
toByteArray()	Devuelve el bufer interno como un array de bytes
toString()	Devuelve la representación en cadena del stream
write()	Escribe datos en el bufer interno
writeTo()	Escribe el bufer

La definición de **CharArrayWriter** (tabla 19.11) es casi idéntica a la de **ByteArrayOutputStream.**

Tabla **19.11.** Cornposicion de la clase CharArrayWriter

Variables/Métodos	Descripción
buf	Contiene datos
count	Tamaño de los datos
CharArrayWriter()	Construye un objeto CharArrayWriter
close()	Cierra el stream
flush()	Escribe de golpe cualquier dato que quede en el bufer interno
reset()	Reestablece el tamaño del bufer interno
size()	Devuelve el tamaño del bufer interno
toCharArray()	Devuelve el bufer interno como un array de caracteres
write()	Escribe datos en el bufer interno
writeTo()	Escribe el búfer

Una vez escritas estas clases, podemos recuperar los datos incluyendo **ByteArrayOutputStream** en toByteArray() o **CharArrayWriter** en CharArray().

Advertencia

`ByteArrayOutputStream` y `CharArrayWriter` introducen datos en la memoria, por lo que solo deberian usarse cuando la cantidad de datos puede guardarse en la memoria sin problema. Unos miles de bytes son una cantidad razonable, no asi unos megabytes.

FilterOutputStream y FilterWriter

Las clases `FilterOutputStream` y `FilterWriter` constituyen las superclases de los stream de salidad y de los writers. Están creadas para subclasificarse y no para ser iniciadas directamente.

BufferedOutputStream y BufferedWriter

El almacenamiento de los datos de salida puede mejorar el resultado de la tarea de escribir, de igual modo que el almacenamiento de los datos de entrada mejora los resultados de lectura. Las clases `BufferedOutputStream` y `BufferedWriter` son similares a `BufferedInputStream` y `BufferedReader`, excepto en que trabajan con datos de salida. El programa del listado 19.10 resalta las ventajas de utilizar un búfer de escritura. Para ello, este programa escribe un archivo con y sin bufer.

Listado 19.10. Comparación de resultados de la tarea de escribir con y sin bufer

```
import java.io.*;
import java.util.*;
public class BufferDiff {
  public static void main (String args[]) throws IOException {
    FileOutputStream      unbufStream;
    BufferedOutputStream bufStream;
    unbufStream = /* un stream de archivo */
      new FileOutputStream("test.one");
    bufStream   = new BufferedOutputStream(
      new FileOutputStream("test.two"));
    System.out.println ("Write file unbuffered: " +
      time (unbufStream) + "ms");
    System.out.println ("Write file   buffered: " +
      time (bufStream  ) + "ms");
  }
  static int time (OutputStream os) throws IOException {
    Date then = new Date();
    for (int i=0; i<500000; i++) {
      os.write(1);
    }
    os.close();
    return (int)((new Date()).getTime() - then.getTime());
  }
}
```

Al ejecutarse, el programa BufferDiff, se produce la siguiente estadistica:

```
Write file unbuffered: 1922ms
Write file   buffered: 46ms
```

Como podemos observar, al incluir un objeto BufferedOutputStream en el stream de destino final (es decir, en la última fase de la cadena de streams de salida) mejora substancialmente el resultado de la tarea de escritura.

DataOutputStream

La clase DataOutputStream implementa la interfaz DataOutput, descrita anteriormente en la seccidn referente a la clase RandomAccessFile. La interfaz DataOutput especifica métodos para guardar y escribir todos los tipos de Java primitivos, además de String. Aunque la representacidn exacta de los tipos transmitidos de este modo es irrelevante, lo cierto es que DataOutputStream genera a stream binario, es decir, un stream no legible para el usuario.

Cuando necesitemos mantener bases de datos externas en ambientes con problemas de capacidad, la representacidn binaria resulta lo mds eficaz. Digamos, por ejemplo, que estamos manipulando modelos 3D cuya definicidn consiste en grandes cantidades de trios coordinados (x, y, z), en los que cada tipo es doble y con datos extra de diferentes bases de datos. En este caso, serd recomendable guardar estos modelos en binario para ahorrar capacidad de almacenamiento.

Nota

No existe un tipo de writer equivalente a DataOutputStream, debido a que Writers trabajan con caracteres y no con representaciones binarias de otros tipos de datos.

Para entidades más pequefias, tales como archivos de configuración, las representaciones binarias no son la mejor opcibn. En su lugar, es preferible utilizar representaciones en formato ASCII, que resultan mucho mds atractivas. La clase PrintStream (quese describird mds adelante) puede utilizarse para crear un stream de datos legible. Sin embargo, esta clase no es tan útil como parece, debido a un fallo en la jerarquia de java.io, y es que no existe una clase de entrada para leer de nuevo estos datos, por lo que si necesitamos disefiar un formato para un archivo de configuracibn, lo mejor es utilizar una clase completamente diferente: Properties del paquete java.util.

Fin de los problemas de los formatos Endian

Siempre que los archivos binarios contengan numeros se desplacen de una a otra arquitectura, se nos plantea el problema de formatos "endian". Hoy en dia

los CPU organizan los bytes de un numero de acuerdo con el formato "little endian" (que coloca en la direccion mas baja el byte menos significativo) o con el formato "big endian" en el que el byte mas significativo obtiene la direccion mas baja.
Cuando se escribe un numero en un archivo mediante un procesador "little-endian", y se lee con un procesador "big-endian" (o viceversa), se produce un error (a no serque el numero este entre 0 o -1). Las clases `DataOutputStream` y `DataInputStream` nos evitan este problema ya que, aunque los archivos de datos Java se intercambien entre equipos fisicos muy distintos, este intercambio se realiza en realidad entre dos VM de Java identicas (que, además utilizan formatos "big-endian").

ObjectOutputStream

Como ya se explicó en la sección de `objectInputStream`, los streams de entrada y de salida leen y escriben objetos enteros. Más concretamente, escriben el estado de las variables de un objeto. Ésta es en realidad una operacidn mas complicada de lo que parece a primera vista, ya que algunas de estas variables son en si referencias a otros objetos. Y deben escribirse en un stream de salida o leerse en un stream de entrada.

El proceso de escribir el estado de un objeto recibe el nombre de "serialización", mientras que el de leer el estado de un objeto se llama "de-serialización". Estos procesos se describirán con más detalle en siguientes secciones.

PrintStream y PrintWriter

La clase `PrintStream` se asemeja a `DataOutputStream` puesto que los métodos que define son iguales que los métodos `write()` proporcionados por `DataOutputStream`. La diferencia es que se ofrecen en dos variedades: `print(..)` y `println(..)`. La tabla 19.12 contiene la composición de la clase `PrintStream`.

Tabla **19.12.** Composición de la clase PrintStream

Variables/Métodos	Descripción
`PrintStream()`	Construye un objeto PrintStream
`checkError()`	Devuelve el estado de error actual
`close()`	Cierra el stream
`flush()`	Escribe de golpe cualquier dato que quede en el stream
`print()`	Escribe datos en el stream
`println()`	Escribe datos en el stream, seguido de una nueva linea

Varkbles/Métodos	Descripción
`setError()`	Cambia el estado de error
`write()`	Escribe datos en el stream

La clase `PrintWriter` (tabla 19.13) es similar.

Tabla **19.13.** Composición de la clase PrintWriter

Variables/Métodos	Descripción
out	Strea mde caracteres
`PrintWriter()`	Construye un objeto PrintWriter
`checkError()`	Devuelve el estado de error actual
`close()`	Cierra el stream
`flush()`	Escribe de golpe cualquier dato que quede en el stream
`print()`	Escribe datos en el stream
`println()`	Escribe datos en el stream, seguido de una nueva línea
`setError()`	Cambia el estado de error
`write()`	Escribe datos en el stream

Existen tres constructores `PrintStream` disponibles. A menos que se especifique el codigo de los caracteres, todos los datos aparecen como caracteres de 8 bits, eliminando los bytes de orden superior:

```
public PrintStream(OutputStream out)
public PrintStream(OutputStream out, boolean autoFlush)
public PrintStream(OutputStream out, boolean autoFlush, String encoding)
```

Existen dos objetos integrados de `PrintStream`: `System.out` (salida estiindar) y `System.err` (error estiindar).

`PrintWriter` cuenta con cuatro constructores:

```
public PrintWriter(Writer out)
public PrintWritrc[Writer out, boolean autoFlush)
public PrintWriter(OutputStream out)
public PrintWriter(OutputStream out, boolean autoFlush)
```

Probablemente ya ha utilizado `PrintStream` desde sus primeros trabajos con Java:

```
System.out.println("Hello, World!");
```

`Object out` es una variable estática de `PrintStream` de la clase `System`. Varios ejemplos de los métodos `print()` y `println()` se han utilizado ya en muchos programas. La diferencia entre ambos es que `print()` no obliga a que se escriban los datos inmediatamente. Puede mantenerse almacenado en el stream hasta que se escriba el carácter de una nueva linea o hasta que se ejecute un `flush()` en el `PrintStream`. Las clases `PrintStream` y `PrintWriter` nos permiten transmitir cualquier objeto como un argumento a los métodos `print()` o `println()`. El mecanismo utilizado para cualquier objeto en una representación string consiste en invocar el método `toString()` del objeto. Si creamos una nueva clase que no anule el `toString()` del `Object`, obtendremos la implementación por defecto, que consiste en transmitir el nombre de la clase junto con el código hash del objeto, producido por el método `hashCode()` de `Object`.

El siguiente programa demuestra el funcionamiento de la clase `println()` en una clase recién creada:

```
public class Print {
  public static void main(String args[]) {
    Print anObject = new Print();
    System.out.println(anObject);
  }
}
```

La nueva clase `Print` no constituye un problema para el método `println (Object obj)`, ya que es capaz de invocar el método principal `Object toString()`, que en este caso produce el siguiente resultado para `anObject`:

```
Print@lec614
```

La ejecucion constante de los métodos `print()` y `println()` significa que podemos utilizar cualquier argumento. El resultado será el esperado: convertirán el argumento en una representacion string y la incluirán en el stream de salida.

DeflaterOutputStream y variables

El otro lado de `InflaterInputStream` y sus subclases lo forman `DeflaterOutputStream` y sus derivados: `GZIPOutputStream`, `ZipOutputStream`, y `JarOutputStream`. Como ya hemos mencionado anteriormente, los stream ZIP y JAR contienen entradas individuales comprimidas en un archivo mayor. Los streams GZIP, por el contrario, cornprimen todo un stream. Para comprimir, sólo tenemos que aiiadir un método `GZIPOutputStream` a las series de streams escritos. Para demostrar este proceso, el listado 19.11 incluye un programa que lee un archivo y comprime el contenido en un array de bytes en memoria. El archivo comprimido no se guarda, sino que se muestra el tamaiio reducido y el original.

Listado 19.11 Compresión de un archivo

```
import java.io.*;
import java.net.*;
```

```
import java.util.zip.*;
public class CompressIt {
  public static void main(String[] args) {
    String filename = args[0];
    try {
      File file = new File(filename);
      int length = (int)file.length();
      FileInputStream fis = new FileInputStream(file);
      BufferedInputStream bis =
        new BufferedInputStream(fis);
      ByteArrayOutputStream baos =
        new ByteArrayOutputStream(length);
      GZIPOutputStream gos =
        new GZIPOutputStream(baos);
      byte[] buffer = new byte[1024];
      int bytesRead;
      while ((bytesRead = bis.read(buffer)) != -1) {
        gos.write(buffer, 0, bytesRead);

      bis.close();
      gos.close();
      System.out.println("Input Length: " + length);
      System.out.println("Output Length: " +
        baos.size());
    } catch (FileNotFoundException e) {
      System.err.println("Invalid Filename");
    } catch (IOException e) {
      System.err.println("I/O Exception");

  }
}
```

Los grados de compresión varian dependiendo de lo que estemos comprimiendo. Por ejemplo, al comprimir un archivo fuente, lo reducimos de 1.045 bytes a 448, y al comprimir el archivo `test.one` de 500.000 bytes (creado en el ejemplo `BufferDiff`) se nos queda en 519 bytes.

Persistencia de objetos y Serialización

Por lo general, un objeto Java dura lo que el programa que lo creó. Un objeto puede dejar de existir durante el tiempo de ejecución si los destructores asi lo requieren. Aunque el objeto se libre de esta destrucción en este momento, desaparecerá cuando el usuario finalice de ejecutar el buscador (de un applet) o el tiempo de ejecución del objeto (para una aplicación).

En este contexto, la persistencia se refiere a la capacidad de un objeto para grabar su estado para que pueda reproducirse en el futuro. Por ejemplo, un objeto persistente puede almacenar su estado en un archivo. Este archivo puede entonces utilizarse para reestablecer el objeto en un tiempo de ejecución distinto No es el objeto en si lo que persiste sino la información necesaria para construir una replica

del objeto. Un objeto se graba asi mismo, escribiendo los valores que describen su estado. Este proceso se conoce como serialización, porque el objeto se representa mediante una serie ordenada de bytes. Java proporciona clases que escriben objetos en streams y reestablecen los objetos a partir de los streams. La tarea principal de la serializacion consiste en escribir los valores de las variables de un objeto. Si una variable es una referencia a otro objeto, éste también debe serializarse. Este proceso es recursivo ya que puede implicar toda una compleja estructura de árbol que representa todo el objeto original, los objetos del objeto, los objetos de los objetos del objeto, y asi sucesivamente. La jerarquia de un objeto se conoce gracias a su gráfico.

Criterio para la serialización

Sólo pueden serializarse con éxito aquellas clases que implementan las interfaces `Serializable` o `Externalizable` que se encuentran en el paquete `java.io`. Un objeto puede serializarse por un objeto externo que es en la prtictica un tipo de stream de salida. Un objeto puede escribir su propio estado sin necesitar otro objeto para hacerlo.

Podemos serializar cualquier clase que cumpla los siguientes requisitos:

- La clase o una de sus superclases debe implementar la interfaz `java.io.Serializable`.
- La clase debe participar en el método `writeObject()` para controlar datos que se estén guardando y aiiadir nuevos datos a los ya guardados.
- La clase debe participar en el método `readObject()` para leer los datos introducidos por el método correspondiente.

Si una clase tiene variables que no queremos serializar, debemos marcarlas con una clave para que el proceso de serialización las ignore.

Nota

La implementación de los metodos `writeObject()` y `readObject()` y `NotSerializableException` evitaran la serializacion de un objeto. `ObjectOutputStream` (u `ObjectInputStream`) detectara la excepción y abortara el proceso.

La interfaz Serializable

La interfaz `Serializable` no dispone de ningun método. Cuando una clase declara que implementa un objeto `Serializable`, lo hace mediante el protocolo de serializacion. Cuando un objeto puede serializarse y el estado del objeto está

escrito en un stream, este stream debe contener suficiente información para reestablecer el objeto y debe resultar `true`, incluso si la clase reestablecida se ha actualizado a una versión compatible más reciente.

Nota

Los miembros primitivos de datos, tal como `int` y `float` se serializan de manera automática por lo que no requieren marcarse de una manera especial para guardarse. Sin embargo, esto si es necesario si no queremos guardarlos.

La interfaz Externalizable

La interfaz `Externalizable` identifica objetos que pueden guardarse en un stream pero que son responsables de su propio estado. Cuando se escribe un objeto que pueda escribirse a sí mismo, éste es sólo responsable de guardar el nombre de clase del objeto y el objeto debe escribir su propio contenido. La interfaz `Externalizable` se define de la siguiente forma:

```
public interface Externalizable extends Serializable {
  public void writeExternal (ObjectOutput out)
    throws IOException;
  public void readExternal (ObjectInput in)
    throws IOException, ClassNotFoundException;
}
```

Una clase extraible debe adherirse a esta interfaz mediante un método `writeExternal()` que se encargará de guardar su estado durante el proceso de serialización y mediante un método `readExternal()` que reestablecerá su estado durante el mismo proceso.

Creación de streams de salida para serializar

Los objetos que pueden serializar otros objetos implementan la interfaz `ObjectOutput` del paquete `java.io`. Esta interfaz se implementa por las clases stream de salida. A continuación, se detalla la definición de la interfaz `ObjectOutput`:

```
public interface ObjectOutput extends DataOutput {
  public void writeObject(Object obj) throws IOException;
  public void write (int b) throws IOException;
  public void write(byte b[]) throws IOException;
  public void write(byte b[], int off, int len) throws IOException;
  public void flush() throws IOException;
  public void close() throws IOException;
}
```

El método fundamental de la interfaz `ObjectOutput` es `writeObject (Object obj)`, que escribe un `obj` en el stream. Los datos estdticos y temporales de `obj` se ignoran y se escriben todas las otras variables, incluyendo las privadas.

Las excepciones pueden darse cuando se estd accediendo al objeto o a sus campos, o cuando se estd intentando escribir el stream de almacenamiento. Cuando ocurren, el stream en el que se construye la interfaz se quedard en un estado desconocido e inutil y habrd errores en la representación externa del objeto.

La interfaz `ObjectOutput` extiende la interfaz `DataOutput`. Los métodos `DataOutput` soportan la escritura de tipos de datos primitivos. Por ejemplo, el método `writeDouble()` escribe datos dobles, mientras que `writeBoolean()` escribe datos booleanos. Estos primitivos métodos de escritura se emplean para escribir las variables primitivas de un objeto.

La clase principal que implementa la interfaz `ObjectOutput` es `ObjectOutputStream`. Esta clase es similar a otras clases stream de salida, descritas en secciones anteriores. Como podemos observar, los objetos se representan como streams y no como caracteres, por lo que utilizan `writers` de bytes.

Cuando va a serializarse un objeto en un archivo, el primer paso consiste en crear un stream de salida que se comunique con el archivo:

```
FileOutputStream fos = new FileOutputStream("obj.ser");
```

El siguiente paso consiste en crear una cadena de salida de objeto y vincularla a la cadena de salida de archivo:

```
ObjectOutputStream oo = new ObjectOutputStream(fos);
```

Truco

En la mayoría de los casos, querremos almacenar los datos de salida para lo cual debemos utilizar un `BufferedOutputStream` entre `FileOutputStream` y `ObjectOutputStream`.

El stream de salida escribe un encabezamiento que contiene un número y una versión. Estos datos se escriben de manera automdtica mediante el método `writeStreamHeader()` al crearse dicho objeto. Como se verá mds adelante, un stream de entrada lee este encabezamiento antes de ejecutar su estado.

Después de escribirlo, el stream de salida escribe una representación en bits de un objeto en el stream de salida mediante el método `writeObject()`. Para que sirva de ejemplo, el siguiente codigo construye una muestra de la clase `Point` y la serializa:

```
oo.writeObject(new Point(15, 20));
```

Esta linea muestra que este proceso no es muy diferente de escribir datos primitivos en un stream. La siguiente sección describe la manera de reestablecer objetos

serializados a partir de los stream de entrada. Más adelante, se incluye un ejemplo concreto de serialización que escribe objetos en un archivo. No debemos olvidar, sin embargo, que un stream de salida puede conectarse también de una manera muy sencilla a un stream de conexión Network.

Advertencia

El formato de salida de serializacion ha cambiado en la version 1.2, por lo que si necesita escribir streams en un ambiente de Java 1.2+ que van a leerse en sistemas de Java 1.1, debe invocar `objectOutputStream.useProtocolVersion (ObjectStreamConstants.PROTOCOL_VERSION 1)`. En la version I.2 y posteriores, la version por defecto es `PROTOCOL_VERSION_2`. La lectura automática detecta y soluciona problemas provocados por versiones diferentes.

Uso de streams de entrada para "deserialización"

La clase `ObjectInputStream` se encarga de mantener el estado del stream y de todos los objetos que se hayan serializado dentro del mismo. Mediante los métodos de esta clase, un programa puede reestablecer un objeto serializado a partir de un stream, además de toda la jerarquia de objetos que estén conectados con el objeto principal. Los tipos de datos primitivos pueden también leerse desde un stream de entrada.

La clase ObjectInputStream sólo dispone de un único constructor:

```
public ObjectInputStream(InputStream in)
  throws IOException, StreamCorruptedException
```

Este constructor invoca el método `readStreamHeader()` de la clase para verificar el encabezamiento y la versión escritos en el stream por el correspondiente stream de salida. Si se detecta algún problema se arroja un `StreamCorruptedException`.

El método fundamental de `ObjectInputStream` es `readObject()`, que reestablece un objeto desde el stream fuente que se devuelve como `Object`. El invocador es responsable de clasificarlo dentro del tipo adecuado.

Durante este proceso, el sistema mantiene una lista de los objetos que se han reestablecido que recibe el nombre de tabla de objetos conocidos.

Consideraciones de seguridad de los objetos serializados

La serializacion puede suponer el almacenamiento de datos de un objeto de un archivo de disco o la transmision de datos a traves de una red. En ambos casos,

existe un problema de seguridad potencial ya que los datos se encuentran fuera del ambiente de ejecucion de Java, y, por lo tanto, fuera del alcance de los mecanismos de seguridad de Java.

El método `writeExternal()` es público lo que permite que cualquier objeto puede hacer que un objeto se escriba a sí mismo en un stream. Debemos ser precavidos en nuestra decision de utilizar `writeExternal()` para serializar datos de un alto grado de privacidad. Al reestablecerse un objeto mediante `readExternal()`, sus valores se almacenan de nuevo como variables privadas, lo que no causa ningljn problema. Sin embargo, mientras los datos serializados se encuentren fuera del sistema cualquiera tiene acceso a ellos, decodificar su formato y obtener los valores. Del mismo modo, los valores de los datos podrian modificarse para por ejemplo cambiar una contraseña o incluso una cuenta bancaria. Sin llegar a estos extremos, se podrian simplemente inutilizar los datos serializados.

Cuando se serializa un objeto, todos los objetos de su grafico quedan expuestos. Un objeto serializado puede tener una referencia que a su vez tenga una referencia a una clase que contenga información de cache. Un pirata informatico podría acceder al sistema de archivos donde se originó el objeto serializado.

La mejor protección para un objeto que contiene campos que no deben almacenarse consiste en etiquetar dichos campos mediante una clave de transitoriedad. Los campos transitorios, al igual que los estaticos, no se serializan por lo que no quedan expuestos.

Si no es posible serializar una clase sin que implique un riesgo en el sistema que la contiene, debemos evitar que esa clase implemente la interfaz `Serializable`. Tampoco debe referirse a ella ninguna de las clases que van a serializarse.

Un objeto que pueda escribir sus propios datos utiliza a rnenudo la tecnica de incluir datos invariables entre sus variables sin una funcion concreta durante el funcionamiento normal de la clase. Estos datos se analizan tras la deserializacion y aparecera un valor inesperado en caso de que la representacion serializada externa contenga errores.

Si los datos conservados son de un tipo primitivo, se tratan como una secuencia de bytes y se reestablecen a partir del stream de entrada. Si los datos almacenados son un string, se leen mediante el código de formato UTF (Unicode Transfer *Format*) y el string se añadirá a la tabla de objetos conocidos. Si el objeto almacenado es un array, se determina su tipo y su longitud. Despuks, se localiza la memoria para el array, y se leen todos los elementos del array mediante el método correspondiente. Una vez reconstruido el array, se aiiade a la tabla de objetos conocidos. Si se trata de un array de objetos (a diferencia de los primitivos), se desearilizan todos los objetos y se aiiaden a la tabla. Cuando un objeto ordinario (es decir, que no sea un string ni un array) se reestablece, se añade también a la tabla. Despuks, los objetos a los que se refiere el objeto original se reestablecen y se aiiaden a la tabla.

Una vez que el objeto se ha recuperado del stream, debe validarse para que se convierta en un objeto completo y pueda ser utilizado por el programa. El método `validateObject()` se utiliza cuando se ha recuperado del stream un grafico completo de objetos. Si el objeto principal no puede validarse, el proceso se detiene y se arroja la excepción.

Serialización de excepciones

Durante la serialización o la deserialización de un objeto, pueden aparecer siete tipos distintos de excepciones, extensiones de `ObjectStreamException`, que a su vez es una extensión de `IOException`.

- `InvalidClassException`: Aparece por lo general cuando el stream de serialización no puede determinar el tipo de clase o cuando no es posible representar la clase devuelta en el sistema que recupera el objeto. Esta excepción aparece también si la clase deserializada no se declara pública, o si no dispone de un constructor público por defecto (sin argumento).
- `NotSerializableException`: Aparece con objetos capaces de escribirse a sí mismos y responsables de su propia serialización, durante la detección de un stream de entrada corrupto. Esto se indica mediante un valor invariable e inesperado.
- `StreamCorruptedException`: Aparece cuando el encabezamiento o los datos de control del objeto almacenado no son válidos.
- `NotActiveException`: Aparece si se invoca al método `registerValidation()` fuera del método `readObject()`.
- `InvalidObjectException`: Aparece cuando no puede validarse un objeto restaurado tras el proceso de serialización.
- `OptionalDataException`: Aparece cuando un stream debe contener un objeto pero en realidad contiene sólo datos primitivos.
- `WriteAbortedException`: Aparece durante el proceso de lectura de la serialización, cuando un stream de entrada detecta que sus datos son incompletos al finalizarse el proceso de escritura de un modo anormal.

Lectura y escritura de un stream de objeto

Escribir un objeto en un stream es un proceso sencillo, similar a escribir cualquier otro tipo de estructura de nivel superior. Para ello, debemos primero escribir un stream de salida de nivel inferior que proporcione acceso al medio externo (generalmente un archivo o una red). A continuación, se encadena un stream de nivel superior al stream de nivel inferior. Para el proceso de serialización, este stream de nivel superior es un stream de salida de objeto.

El siguiente código construye una muestra de `Point` y lo escribe en un archivo llamado `point.ser` en el sistema local de archivos.

```
Point p = new Point(13, 10);
FileOutputStream fos = new FileOutputStream("point.ser");
BufferedOutputStream bos = new BufferedOutputStream(fos);
ObjectOutputStream oos = new ObjectOutputStream(bos);
try {
  oos.writeObject (p);
  oos.flush ();
} catch (IOException e) {
}
```

Para reestablecer el objeto es necesario abrir un stream de entrada de archivo en el archivo, y encadenar un stream de entrada de objeto al stream de archivo. El objeto `Point` se lee mediante `readObject()`, invocado desde el stream de objeto. El valor devuelto es de tipo `Object` y debe clasificarse por el usuario. El siguiente código muestra cómo llevar a cabo esta operación.

```
Point p = null;
FileInputStream fis = new FileInputStream("point.ser");
BufferedInputStream bis = new BufferedInputStream(fis);
ObjectInputStream ois = new ObjectInputStream(bis);
try {
  p = (Point)ois.readObject ();
} catch (IOException e) {
```

Un ejemplo de serialización

En esta sección, vamos a desarrollar un programa sencillo que guarda y reestablece un objeto. Se trata de un simple programa de dibujo que puede almacenar su lista de muestras en un archivo. Una lista de muestras es una estructura de datos que contiene una descripción abstracta de lo que debe aparecer en pantalla. El programa nos permite dibujar rectángulos mediante el ratón. Al mantener pulsado el botón del ratón, definimos una esquina del rectángulo y al arrastrarlo y soltarlo, definimos la esquina opuesta. Esta lista de muestra se almacena en un `Vector` que contiene dos muestras de la clase `Point` para cada rectángulo. Uno de ellos representa la esquina inferior del rectángulo y el segundo la esquina superior.

La aplicación `PersisTest` es una subclase de `JFrame`. Un panel en la parte superior del marco contiene cuatro botones de control y otro en la parte inferior pone la pantalla en blanco y ejecuta el vector de la lista de muestras y dibuja un rectangulo en negro por cada par de puntos en el vector.

Los cuatro controles soportan las funciones de eliminar, guardar, reestablecer y salir. El controlador del botón **Guardar** utiliza la técnica de escritura descrita en la sección anterior para almacenar el Vector de la lista de muestras en un archivo. El nombre de archivo debe especificarse en el argumento de la linea de comandos. El controlador del botón **Restaurar** sustituye la antigua lista de muestras por el nuevo

`Vector`. Para probar la aplicacion, invóquela con un nombre de archivo como un argumento de linea de comandos:

```
java PersisTest filename
```

Utilice el ratón para dibujar algunos rectangulos y pulse el botón Guardar para escribir la lista de muestras en el archivo. Elimine los rectangulos de la pantalla o dibuje algunos mas, antes de pulsar el botón Restaurar. La pantalla mostrará el estado de los elementos en el momento de pulsar el boton Guardar. Pruebe a cerrar la aplicación y abrirla de nuevo y comprobará que se reestablece correctamente desde el archivo externo.

Este ejemplo lograria los mismos resultados si abriéramos un stream de salida de datos en lugar de un stream de salida de objetos, y escribiéramos cuatro valores `int` para cada rectángulo. La ventaja de la serialización recae en que el mantenimiento del programa se facilita enormemente. Si tuviéramos que almacenar y reestablecer la lista de muestras mediante streams de entrada y de salida de datos, cualquier cambio en el formato de esa lista nos obligaria a modificar el código. Con la serializacibn, sin embargo, el formato de la lista es irrelevante.

El listado 19.12 contiene el código del programa `PersisTest` y la figura 19.6 muestra el programa tras reestablecer un archivo guardado.

Listado 19.12. Aplicacion de la serialización

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.io.*;
import java.util.ArrayList;
public class PersisTest extends JFrame
    implements ActionListener {
  ArrayList displayList = new ArrayList();;
  String pathname;
  JButton clearBtn, saveBtn, restoreBtn, quitBtn;
  public static void main(String args[])    {
    if (args.length == 0) {
      System.err.println("Usage: java PersisTest filename");
      System.exit(-1);
    }
    new PersisTest(args[0]).show();
  }
  public PersisTest(String pathname) {
    super("Save Me");
    setDefaultCloseOperation(EXIT_ON_CLOSE);
    this.pathname = pathname;
    // Construye el GUI. Hace que este objeto sea el listener de todas las
    // acciones.
    JPanel pan = new JPanel();
    clearBtn = new JButton("Clear");
    clearBtn.addActionListener(this);
    pan.add(clearBtn);
    saveBtn = new JButton("Save");
```

```
    saveBtn.addActionListener(this);
    pan.add(saveBtn);
    restoreBtn = new JButton("Restore");
    restoreBtn.addActionListener(this);
    pan.add(restoreBtn);
    quitBtn = new JButton("Quit");
    quitBtn.addActionListener(this);
    pan.add(quitBtn);
    Container c = getContentPane();
    c.add(pan, BorderLayout.NORTH);
    c.add(new ClickPanel(), BorderLayout.CENTER);
    setSize(350, 200);

  public void actionPerformed(ActionEvent e) {
    if (e.getSource() == clearBtn) {
      // Dibuja con una lista de muestras vacía.
      displayList = new ArrayList();
      repaint();
    } else if (e.getSource() == saveBtn) {
      // Escribe los arrays de la lista de muestras en un stream de salida
      // de objetos.
      try {
        FileOutputStream fos = new FileOutputStream(pathname);
        ObjectOutputStream oos = new ObjectOutputStream(fos);
        oos.writeObject(displayList);
        oos.flush();
        oos.close();
        fos.close();
      } catch (IOException ex) {
        System.err.println("Trouble writing display list array list");
      }
    } else if (e.getSource() == restoreBtn) {
      // Lee nuevos arrays de la lista de rnuestras desde un stream de
      // entrada de objetos.
      try {
        FileInputStream fis = new FileInputStream(pathname);
        ObjectInputStream ois = new ObjectInputStream(fis);
        displayList = (ArrayList)(ois.readObject());
        ois.close();
        fis.close();
        repaint();
      } catch (ClassNotFoundException ex) {
        System.err.println("Trouble reading display list array list");
      } catch (IOException ex) {
        System.err.println("Trouble reading display list array list");
      }
    } else if (e.getSource() == quitBtn) {
      System.exit(0);


  class ClickPanel extends JPanel {
    ClickPanel() {
      MouseListener listener = new MouseAdapter() {
        public void mousePressed(MouseEvent e) {
          // Almacena x e y en los arrays de la lista de rnuestras
          Point p = new Point(e.getX(), e.getY());
```

```
            displayList.add(p);
          }
          public void mouseReleased(MouseEvent e) {
            // Almacena x e y en los arrays de la lista de muestras, y pide
            // que se dibuje do nuevo.
            Point p = new Point(e.getX(), e.getY());
            displayList.add(p);
            repaint();
          }
        };
        addMouseListener(listener);

      public void paintComponent(Graphics g) {
        // Elimina los elementos de la pantalla.
        g.setColor(Color.white);
        g.fillRect(0, 0, getSize().width, getSize().height.);
        // Ejecuta la lista de muestras, dibujando un rectángulo por cada dos
        // puntos de la lista de arrays.
        g.setColor(Color.black);
        int i = 0;
        while (i < displayList.size()) {
          Point p0 = (Point)(displayList.get(i++));
          Point p1 = (Point)(displayList.get(i++));
          int x = Math.min(p0.x, p1.x);
          int y = Math.min(p0.y, p1.y);
          int w = Math.abs(p0.x - p1.x);
          int h = Math.abs(p0.y - p1.y);
          g.drawRect(x, y, w, h);
        }
      }
    }
}
```

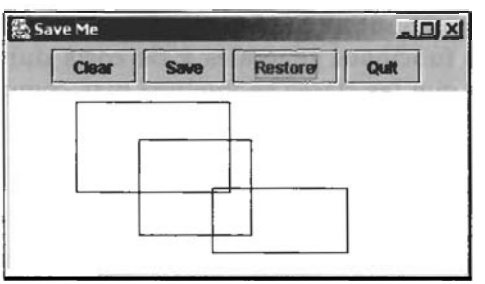


Figura 19.6. El programa Persistest en acción

transient y ObjectStreamField

En circunstancias normales, podemos marcar un campo como `transient` si no deseamos que se guarde durante la serialización. Este proceso es necesario para tipos de datos tales como `Thread` e `Image`, que no pueden serializarse. Existe un segundo mecanismo para especificar los tipos de datos que deseamos guardar. Si la

definición de la clase contiene una variable final estática y privada llamada `serialPersistentFields` de tipo `ObjectStreamField[]`, el array definirá lo que se almacena y cualquier marca **`transient`** se ignorard.

Por ejemplo, en la siguiente definición de clase se guardan **`Point`** e **`int`**, pero no así el **`String`**:

```
public class FieldSave implements Serializable {
  private Point point;
  private int area;
  private String passwd;
  public final static ObjectStreamField[]
    serialPersistentFields = {
      new ObjectStreamField("point", Point.class),
      new ObjectStreamField("area", int.class)
    };
}
```

Podemos elegir entre uno u otro mecanismo dependiendo de nuestras necesidades.

Truco

La ventaja de utilizar `ObjectStreamField`es que no guarda ni reestablece nada a no serque lo especifiquemos. Por motivos de seguridad, esto puede ser fundamental. Porotro lado, el inconveniente es que no debemos olvidar añadir los nuevos datos al array `serialPersistentFields`, o no se guardaran.

Código XML y decodificación

Las clases `ObjectInputStream` y `ObjectOutputStream` son de gran utilidad con clases con funciones generales y de corta duración, pero el proceso se complica a medida que las clases se vuelven más complejas. Si observamos, por ejemplo, el código de los componentes de **`swing`** en una versión de Java anterior a la **1.4,** encontraremos los siguientes comentarios:

```
 * <strong>Advertencia:</strong>
 * Objetos que pueden ser serializados de esta clase no son compatibles con
 * futuras versiones de Swing. La serialización actual es apropiada para
 * almacenamientos de corta duración o RMI entre aplicaciones que se
 * ejecuten en la misma
 * versión de Swing. Futuras versiones de Swing soportarán
   almacenamientos de más larga duración.
```

En la versión 1.4 el comentario sigue apareciendo, aunque con información añadida:

```
 * La versión 1.4, soporta el almacenamiento de larga duración
 * de JavaBeans<sup><font size="-2">TM</font></sup>
 * se ha añadido al paquete <code>java.beans</code>.
 * Por favor, consulte {@link java.beans.XMLEncoder}.
```

Con cierto trabajo, podiamos conseguir clases mds complejas y lograr asi que el proceso de serialización funcionase correctamente. Ahora, existe ya una manera mds sencilla para los componentes de JavaBean. Las clases `XMLEncoder` y `XMLDecoder`, del paquete `java.beans`, funcionan como las clases `ObjectOutputStream` y `ObjectInputStream`, con la diferencia de que en lugar de guardar el estado de la clase en un formato binario, lo guardan en un archivo XML. En el archivo se añade información sobre cómo decodificarlo. Para demostrar este proceso de codificación y decodificación, el listado 19.13 muestra una clase que contiene un array de `string`, un `object`, propiedades de datos de un `int` primitivo.

Listado 19.13 El componente SaveMe de JavaBean

```
import java.awt.*;
import java.io.*;
import java.beans.*;
public class SaveMe {
  private String names[];
  private Point p;
  private int length;
  public String[] getNames() {
    return names;
  }
  public Point getPoint() {
    return p;
  }
  public int getLength() {
    return length;
  }
  public void setNames(String newValue[]) {
    names = newValue;
  }
  public void setPoint(Point newValue) {
    p = newValue;
  }
  public void setLength(int newValue) {
    length = newValue;
  }
}
```

Los métodos `set` y `get` hacen de la clase un componente de JavaBean, que se detallard en el capitulo 21. Sin ellos, la codificación y decodificación de XML no funcionaria. Para demostrar la manera de guardar mediante codificación XML, sólo tenemos que crear una muestra y guardarla, como se hace a continuación.

```
public static void main(String args[]) throws IOException {
    // Crea
    SaveMe saveme = new SaveMe();
    saveme.setNames(new String[] ["one", "two", "three"));
    saveme.setPoint(new Point(15, 27));
    saveme.setLength(6);
    // Guarda
    XMLEncoder encoder = new XMLEncoder(
```

```
        new BufferedOutputStream(
          new FileOutputStream("saveme.xml")));
      encoder.writeObject(saveme);
      encoder.close();
    }
```

Del mismo modo que con `ObjectOutputStream`, podemos pasar al constructor `XMLEncoder` el nombre del stream que vamos a guardar. Después, escribimos el objeto mediante `writeObject()` y cerramos el stream con `close()`. Este es el archivo de salida de este caso concreto:

```
<?xml version="1.0" encoding="UTF-8"?>
<java version="1.4.0-rc" class="java.beans.XMLDecoder">
(object   class="SaveMe">
   <void property="length">
    <int>6</int>
   </void>
   <void property="names">
    <array class="java.lang.String" length="3">
     <void index="0">
      <string>one</string>
     </void>
     <void index="1">
      <string>two</string>
     </void>
     <void index="2">
      <string>three</string>
     </void>
    </array>
   </void>
   <void property="point">
    <object class="java.awt.Point">
     <int>15</int>
     <int>27</int>
    </object>
   </void>
 </object>
 </java>
```

Para leer un archivo XML, sólo nos resta utilizar un `XMLDecoder`:

```
XMLDecoder decoder = new XMLDecoder(
  new BufferedInputStream(
    new FileInputStream("saveme.xml")));
SaveMe saveMe2 = (SaveMe)decoder.readObject();
decoder.close();
```

El nuevo paquete E/S

Una de las funciones mas esperadas de Java 1.4 es el paquete "New I/O" ("Nuevo E/S"). Estas clases se encuentran en `java.nio` y en otros subpaquetes. En ellos,

se incluyen una gran variedad de nuevas funciones que complementan el paquete `java.io`, tales como búferes para almacenar tipos de datos primitivos. Encontraremos también codificadores y decodificadores entre grupos de caracteres, una biblioteca de expresiones usuales para asignar patrones y canales para abstraer las operaciones E/S.

Búferes NIO, canales y grupos de caracteres

Los búferes trabajan como un objeto especializado, combinación de `DataInputStream` y `DataOutputStream`, y sirven para leer y escribir un tipo primitivo especifico en la memoria. Existen siete clases principales de bufer, para los tipos de datos `byte`, `char`, `double`, `float`, `int`, `long` y `short`. Todas estas clases son subclases de `Buffer` cuya definición se muestra en la tabla 19.14.

Tabla 19.14. Composición de la clase Buffer

Variables/Métodos	Descripción
`capacity()`	Devuelve la capacidad
`clear()`	Elimina el contenido
`flip()`	Cambia el limite y la posicion actual
`hasRemaining()`	Indica los elementos entre el limite y la posicion actual
`isReadOnly()`	Indica si el bufer es solo de lectura
`limit()`	Devuelve/Cambia el limite del bufer
`mark()`	Marca la posicion dentro del búfer
`position()`	Devuelve la posicion actual
`remaining()`	Devuelve el nurnero de elernentos entre el lirnite y la posicion actual
`reset()`	Reestabelce la posicion actual al marcador
`rewind()`	Reestablece la posicion actual al principio y elimina los marcadores

Antes de examinar los diferentes tipos de búfer y cómo utilizarlos, es importante saber que la tarea de posicionar un búfer es similar a la de `RandomAccessFile`. Puesto que un búfer se utiliza para leer y escribir, no podemos escribir algo en el bufer y esperar que lo lea inmediatamente. Para ello, debemos cambiar la nueva posición al lugar en que se encontraba antes de leerse el elemento, mediante el método `flip()`.

En la tabla 19.14 no podemos ver la manera de obtener elementos del bufer y volverlos a incluir. Cada subclase cuenta con sus métodos correspondientes de

get() y put() para sus tipos de datos. Las subclases disponibles son ByteBuffer, CharBuffer, DoubleBuffer, FloatBuffer, IntBuffer, LongBuffer, y ShortBuffer, ninguna de las cuales dispone de un constructor. Para crear uno, debemos pasar los datos al método wrap() de cada subclase, o encontrar espacio para uno vacio, mediante allocate(). El siguiente código sirve para crear un CharBuffer a partir de un string:

```
String s = ...;
CharBuffer buff = CharBuffer.wrap(s);
```

Una vez hecho esto, podremos leer los caracteres del string a través del bufer mediante el siguiente bucle:

```
CharBuffer buff = CharBuffer.wrap(s);
for (int i=0, n=buff.length(); i<n; itt) {
  System.out.println(buff.get());
}
```

Hasta el momento, el resultado es el mismo que el de trabajar con streams normales de entrada y de salida, aparte de permitirnos leer y escribir el mismo bufer, como en el siguiente código:

```
buff.put('X');
buff.flip();
char c = buff.get(); // c == 'X'
```

Esto nos conduce al concepto de buferes directos o indirectos. Al crear un bufer, podemos hacer que sea un bufer directo. Aunque son más dificiles de crear, y sólo deberian usarse para un tipo de bufer de gran tamaño y de larga duración, permiten a JRE aplicar operaciones rápidas de E/S directamente al bufer, lo que, al menos en teoria, puede mejorar enormemente el resultado de los accesos a memoria.

Una de las formas mas utilizadas de bufer directo sirve para crear un mapa de todo el contenido de un archivo en la memoria. Se trata de una forma especializada de ByteBuffer llamada MappedByteBuffer.

Para crear el mapa de archivos en la memoria, debemos comprender el funcionamiento de los canales, que se encuentran en el paquete java.nio.channels. Los canales funcionan como los stream y las clases de lectura y escritura. Las diferentes conexiones se basan en canales tal como FileChannel, que conecta un FileInputStream o un FileOutputStream. Dependiendo del código del canal, éste puede leerse o escribirse.

El siguiente codigo abre un archivo, consigue su FileChannel, calcula su tamaño y crea un mapa del mismo en la memoria.

```
String filename = ...;
FileInputStream input = new FileInputStream(filename);
FileChannel channel = input.getChannel();
int fileLength = (int)channel.size();
MappedByteBuffer buffer = channel.map(
  FileChannel.MapMode.READ_ONLY, 0, fileLength);
```

Podemos leer el búfer mediante rápidas operaciones de E/S. Existen diferentes maneras de obtener un canal a partir de muchas de las clases stream que se conectan con objetos reales, tales como archivos, pipes y conexiones network. Los streams de filtrado, sin embargo, no disponen de canales.

Además de crear un mapa del archivo en la memoria, podemos utilizar `FileChannel` para bloquear los niveles de los archivos. La tabla 19.15 muestra las operaciones disponibles de `FileChannel`.

Tabla 19.15. Composición de la clase FileChannel

Variables/Métodos	Descripción
`FileChannel()`	Construye un objeto FileChannel
`force()`	Hace que se escriba el contenido
`lock()`	Bloquea un archivo o una parte del mismo
`map()`	Crea un mapa del archivo en el bufer
`position()`	Cambia el indicador de la posición actual
`read()`	Lee datos del bufer
`size()`	Devuelve el tamaño actual
`transferFrom()`	Transfiere bytes de otro canal
`transferTo()`	Transfiere bytes a otro canal
`tryLock()`	Intenta bloquear un archivo o parte del mismo
`write()`	Escribe datos del búfer

Como ocurria con `InputStream`, el `MappedByteBuffer` que obtenemos de `FileChannel` es un conjunto de bytes. Si deseamos tratar el contenido del archivo como caracteres, debemos convertir el `ByteBuffer` en un `CharBuffer` mediante la clase `Charset`. La clase `Charset` y otra relacionadas se encuentran en el paquete `java.nio.charset` y nos permiten pasar de bytes a caracteres Unicode. Para ello, sólo tenemos que encontrar el `Charset` adecuado, mediante `forName()`, así como su decodificador para traducir el búfer, de la siguiente manera:

```
Charset charset = Charset.forName("ISO-8859-1");
CharsetDecoder decoder - charset.newDecoder();
CharBuffer charBuffer = decoder.decode(buffer);
Once you have the buffer, you can read it, just like before:
for (int i=0, n= charBuffer.length(); i<n; itt) {
  System.out.println(charBuffer.get());
}
```

Si deseamos realizar la operación inversa, es decir, traducir caracteres a bytes, utilizaremos `CharsetEncoder`.

Nota

Podra encontrar un grupo de caracteres muy utiles en http:llwww.iana.orgl assignmentslcharacter-sets. Java solo necesita soportar: `US-ASCII`, `ISO-8859-1`, UTF-8, `UTF-16BE`, `UTF-16LE`, y UTF-16. Si necesitamos más, tendremos que definir nuestro propio `CharsetProvider`, aunque esto no suele ser necesario.

Antes de ver un ejemplo completo de como utilizar todos los elementos descritos, examinemos la nueva prestacidn de expresiones regulares. También veremos mapas de archivos y la manera de procesar el contenido de un archivo, modificando el ejemplo del listado 19.4, mediante buferes, canales, grupos de caracteres y expresiones normales.

Expresiones normales

El nuevo paquete `java.util.regex` contiene una biblioteca para la asignación compleja de patrones. A través de ella, podemos buscar secuencias concretas de caracteres así como procesarlos. La clase `Pattern` define la secuencia de caracteres a asignar y la clase `Matcher` realiza la asignacibn. Este paquete también incluye la excepción `PatternsyntaxException`.

Para asignar patrones debemos primero definir el patrbn a asignar, el cual puede ser una cadena de caracteres normales, o alguna combinación de caracteres normales y especiales. Los caracteres especiales ayudan a definir las clases de grupos de caracteres que se van a buscar o a organizar los caracteres.

En el caso mds sencillo, un patrón consiste de un unico cardcter tal como `\t` para tabulador o `\e` para escape. El patrón se escribe entre comillas y se envía al `Matcher`. Para buscar un tabulador uilizaremos `"//t"`.

Para especificar uno de los muchos caracteres debemos escribirlos dentro de paréntesis cuadrados. De este modo, emplearemos `[jaz]` para asignar los caracteres j, a, o z, y `[j-z]` para asignar desde la letra j a la z, ambas inclusive. También podemos negar un cardcter mediante el símbolo `^`. Por ejemplo, `[^jaz]` se refiere a cualquier cardcter excepto j, a, o z.

Esto nos conduce al siguiente grupo de símbolos encargados de asignar patrones especiales. Algunos patrones son tan comunes que vienen acompafiados de teclas de método abreviado.

\d	Cualquier digito
\D	Cualquier simbolo que no sea un digito
\s	Espacio en blanco
\S	Cualquier caracter que no sea un espacio en blanco
\w	Una palabra

Nota

Podra encontrar un grupo de caracteres muy útiles en http://www.iana.orgl assignments/character-sets. Java solo necesita soportar: `US-ASCII`, `ISO-8859-1`, `UTE-8`, `UTF-16BE`, `UTF-16LE`, y `UTF-16`. Si necesitamos mas, tendremos que definir nuestro propio `CharsetProvider`, aunque esto no suele ser necesario.

Antes de ver un ejemplo completo de cómo utilizar todos los elementos descritos, examinemos la nueva prestación de expresiones regulares. También veremos mapas de archivos y la manera de procesar el contenido de un archivo, modificando el ejemplo del listado 19.4, mediante búferes, canales, grupos de caracteres y expresiones normales.

Expresiones normales

El nuevo paquete `java.util.regex` contiene una biblioteca para la asignación compleja de patrones. A través de ella, podemos buscar secuencias concretas de caracteres asi como procesarlos. La clase `Pattern` define la secuencia de caracteres a asignar y la clase `Matcher` realiza la asignacibn. Este paquete también incluye la excepción `PatternSyntaxException`.

Para asignar patrones debemos primero definir el patrón a asignar, el cual puede ser una cadena de caracteres normales, o alguna combinación de caracteres normales y especiales. Los caracteres especiales ayudan a definir las clases de grupos de caracteres que se van a buscar o a organizar los caracteres.

En el caso más sencillo, un patrón consiste de un Dnico carácter tal como `\t` para tabulador o `\e` para escape. El patrón se escribe entre comillas y se envía al `Matcher`. Para buscar un tabulador uilizaremos `"//t"`.

Para especificar uno de los muchos caracteres debemos escribirlos dentro de paréntesis cuadrados. De este modo, emplearemos `[jaz]` para asignar los caracteres `j`, `a`, o `z`, y `[j-z]` para asignar desde la letra `j` a la `z`, ambas inclusive. También podemos negar un carácter mediante el simbolo `^`. Por ejemplo, `[^jaz]` se refiere a cualquier carácter excepto `j`, `a`, o `z`.

Esto nos conduce al siguiente grupo de simbolos encargados de asignar patrones especiales. Algunos patrones son tan comunes que vienen acompafiados de teclas de método abreviado.

\d	Cualquier digito
\D	Cualquier símbolo que no sea un digito
\s	Espacio en blanco
\S	Cualquier caracter que no sea un espacio en blanco
\w	Una palabra

\W	Cualquier elemento que no sea una palabra
\p{Lower}	Caracter en minúsculas
\p{Alnum}	Caracter alfanumerico

Nota

Para mas información sobre los patrones disponibles, consulte el javadoc.

Se puede trabajar con patrones de dos maneras distintas. La primera consiste en definir el patrón, pasarle los datos de entrada y que los divida. De este modo, obtenemos un array de elementos String (un elemento por palabra). El programa del listado 19.14 divide un argumento de línea de comandos basindose en el espacio en blanco.

Listado 19.14 Separación de un String en un array de elementos coincidentes

```
import java.util.regex.*;
public class RegOne {
  public static void main(String[] args) throws Exception {
    Pattern p = Pattern.compile("\\s");
    String split[] = p.split(args[0]);
    for (int i=0, n=split.length; i<n; i++) {
      System.out.println(split[i]);
    }
  }
}
```

Al ejecutar el programa RegOne con la siguiente línea de comandos, en la que se incluye un tabulador entre tres y cuatro:

```
java RegOne "One Two Three    Four"
```

se produce el siguiente resultado:

```
One
Two
Three
Four
```

La otra manera de trabajar con Pattern consiste en crear uno y pasarlo al Matcher. Después nos desplazamos por el resultado como en una Enumeration, pero mediante los métodos find() y group():

```
CharBuffer charBuffer    ...;
Pattern p    ...
Matcher m = p.matcher(charBuffer);
while (m.find()) {
  CharSequence line = m.group();
  // Procesa CharSequene
}
```

El listado 19.15 demuestra ambas maneras de trabajar `Pattern`. Como ya hemos dicho, este programa es una modificación del programa del listado 19.4.

Listado 19.15 Cálculo de frecuencias de palabras con expresiones regulares

```
import java.io.*;
import java.nio.*;
import java.nio.channels.*;
import java.nio.charset.*;
import java.util.*;
import java.util.regex.*;
public class WordCount {
  public static void main(String args[]) throws Exception {
    String filename = args[0];
    // Encuentra la ruta entre el nombre de archivo y el búfer de bytes
    FileInputStream input = new FileInputStream(filename);
    FileChannel channel = input.getChannel();
    int fileLength = (int)channel.size();
    MappedByteBuffer buffer = channel.map(
      FileChannel.MapMode.READ-ONLY, 0, fileLength);
    // Convierte a búfer de caracteres
    Charset charset = Charset.forName("ISO-8859-1");
    CharsetDecoder decoder = charset.newDecoder();
    CharBuffer charBuffer = decoder.decode(buffer);
    // Crea un patrón de linea
    Pattern linePattern = Pattern.compile(".*$", Pattern.MULTILINE);
    // Srea un patrón de palabra
    Pattern wordBreakPattern = Pattern.compile("[\\p{Punct}\\s]");
    // Asigna el patrón de linea al bhfer
    Matcher lineMatcher = linePattern.matcher(charBuffer);
    Map map = new TreeMap();
    Integer ONE = new Integer(1);
    // Para cada linea
    while (lineMatcher.find()) {
      // Obtiene linea
      CharSequence line = lineMatcher.group();
      // Obtiene el array de palabras de la linea
      String words[] = wordBreakPattern.split(line);
      // Para cada palabra
      for (int i=0, n=words.length; i<n; itt) {
        if (words[i].length() > 0) {
          Integer frequency = (Integer)map.get(words[i]);
          if (frequency == null) {
            frequency = ONE;
          } else {
            int value = frequency.intValue();
            frequency = new Integer(value t 1);
          }
          map.put(words[i], frequency);
        }

    }
    System.out.println(map);
  }
}
```

Registro

La última novedad de JDK 1.4 que vamos a discutir en este capitulo es el registro API. Esta nueva función nos permite crear en el código mensajes para subsanar errores, en forma de llamadas a `java.util.logging.Logger`. Los objetos registrados `Logger` procesan el mensaje basándose en filtrados de código, nivel, tipo u otras caracteristicas.

Un `Logger` mantiene un espacio para mostrar jerarquia, al igual que los paquetes, por lo que cuando obtenemos uno, pasamos el nombre del paquete:

```
Logger logger = Logger.getLogger("com.jzventures.sybex");
```

Esto permite registrar los controladores en los siguientes niveles:

- Root "".
- "com".
- "com.jzventures".
- "com.jzventures.sybes".

y que se notifiquen cuando los mensajes de registro se envian.

En el caso más sencillo, registramos un mensaje invocando el mensaje de `log()` y pasando el grado de severidad, un mensaje y una excepción opcional.

```
logger.log(Level.WARNING, "Oops", exception);
```

Como se observa en la tabla 19.16, hay otros métodos disponibles para registrar niveles y actividades concretas.

Tabla 19.16. Composición de la clase Logger

Variables/Métodos	Descripción
`global`	Referencia al logger global
`Logger`	Construye un objeto `Logger`
`addHandler()`	Aiiade un controlador para recibir el mensaje
`config()`	Registra el mensaje del nivel adecuado
`fine()`	
`finer()`	
`finest()`	
`info()`	
`severe()`	
`warning()`	

Variables/Métodos	Descripción
`entering()`	Registra mensaje para el método de entrada o el existente
`exiting()`	
`getAnonymousLogger()`	Encuentra o crea un `Logger`
`getLogger()`	
`getFilter()`	Devuelve el filtro
`getHandlers()`	Devuelve los objetos controladores registrados
`getLevel()`	Devuelve el nivel de registro actual
`getName()`	Devuelve el nombre
`getParent()`	Devuelve el padre
`getResourceBundle()`	Devuelve el *bundle* de codigo de localización
`getResourceBundleName()`	Devuelve el nombre del *bundle,* o envoltura
`getUseParentHandlers()`	Indica si se esta utilizando el controlador principal
`isLoggable()`	Indica el nivel de registro
`log()`	Registra el mensaje
`logp()`	Registra el mensaje "preciso"
`logrb()`	Registra el mensaje con el *bundle* del codigo
`removeHandler()`	Elimina el controlador
`setFilter()`	Cambia el filtro
`setLevel()`	Cambia el nivel de registro actual
`setParent()`	Cambia el Logger principal o padre
`setUseParentHandlers()`	Cambia la indicación de uso del controlador padre
`throwing()`	Registra la excepción que se arroja

Los niveles de severidad se representan mediante constantes de la clase `Level`. La severidad se incrementa en las siguientes fases: FINEST, FINER, FINE, CONFIG, INFO, WARNING, SEVERE. Los métodos `throwing()`, `entering()`, y `exiting()` se registran en el nivel FINER.

La interfaz `Filter` define un único método que es `boolean isLoggable(LogRecord)`, que nos permite acceder a un registro para determinar si estamos interesados en registrar el mensaje.

Para incluir nuestro propio controlador, utilizaremos la clase `Handler`. Existen subclases concretas dependiendo de cómo deseamos guardar los datos:

- `MemoryHandler` permite guardar en memoria.
- `ConsoleHandler` envía datos de salida a `System.err`.
- `FileHandler` envía datos XML a cualquier nombre de archivo designado.
- `SocketHandler` nos permite registrarnos a un host y un puerto.

Para enviar todos los mensajes de registro de todos los paquetes al archivo `file jaz.log`, crearemos un `FileHandler` para el nombre de archivo y añadiremos el `Handler` al `Logger`. Para añadir el `Logger`, debemos situarlo en el nivel jerárquico en el que queremos registrar los mensajes. Para registrarlo todo, disponemos del logger al nivel raíz, el que no tiene nombre, como se muestra a continuación:

```
Handler handler = new FileHandler("jaz.log");
Logger.getLogger("").addHandler(handler);
```

Una vez hecho esto, todos los mensajes registrados de cualquier paquete se enviarán a `jaz.log`.

Nota

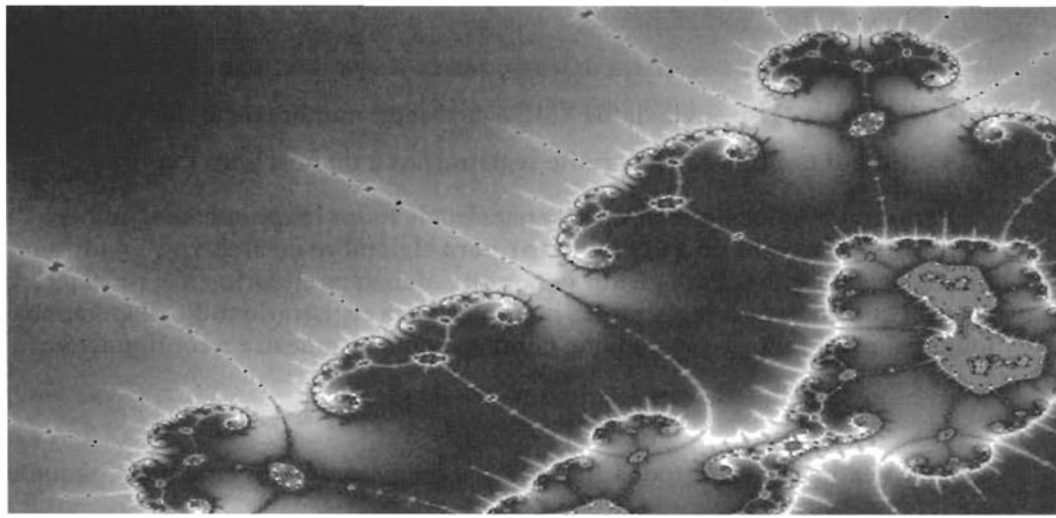
La funcion de registro API de Java 1.4 no funciona con lasversiones anteriores de Java. Si tenemos que trabajar en este ambiente, debemos emplear las bibliotecasde terceros. Una opción es el paquete `Log4j` de Apache, disponible en **http://jakarta.apache.org/log4j/doc/.**

Resumen

Las prestaciones E/S de Java 1.4 SDK son muy complejas y poderosas, y ademds, siguen desarrollándose en la actualidad.

De acuerdo con JSR 51 (http://www.jcp.org/jsr/detail/51.jsp) una de las funciones que faltan en la versión 1.4 es "una sencilla prestación para formatear estilos `printf`." Si necesita una prestación como ésta, pruebe con la sencilla utilidad de acme.com que encontrard gratuitamente en (http://www.acme.com/java/software/Acme.Fmt.html). La clase `Fmt` de Acme proporciona las bases para el formato `printf`. Es seguro que la versión oficial de Sun de Java 1.5, contendrá una biblioteca mucho más compleja, pero no es necesario esperar hasta que salga al mercado.

Para soporte adicional de las funciones E/S de Java visite la página http://www.jguru.com/faq/IO. Ademds de soluciones a las respuestas más frecuentes, encontrará foros donde podrd plantear sus dudas, y ayudar a otros usuarios a resolver las suyas.



20
Trabajos de impresión

El soporte de impresión de la plataforma de Java ha cambiado en cada nueva versión con el intento de mejorarlo. Parece que la ultima versión, por fin lo ha conseguido. Como todos los elementos de Java, el marco de impresión está disefiado para proporcionar soporte en todas las plataformas, ademds de en Microsoft Windows. Por impresión se entiende la conversión de datos visualizados a un formato de impresión ademds de la instalación de servicios de impresión personalizados. En este capitulo repasaremos la historia del soporte de impresión proporcionado por la plataforma de Java, y veremos cómo emplear las nuevas funciones. Revisaremos también las clases principales del marco antes de pasar a algunos ejemplos prácticos.

Historia del soporte de impresion

Java 1.4 es en realidad la quinta versión de la plataforma Java, pero la cuarta en incluir funciones de impresión. En Java 1.1, la clase `java.awt.PrintJob` proporcionaba el marco de impresión de componentes. El método posterior `print()`, que anula esta clase, ofrece lo que se va a imprimir (por defecto, este método sólo invoca a `paint()`). Con la versión 1.2. El soporte de impresión se incluyó en el paquete `java.awt.print`, con la clase `PrinterJob` como la responsable de hacer el trabajo. Podiamos imprimir un `Book` con diferentes formatos de página y painters. La versión 1.3 añadió nuevos atributos, incluyendo los de pdginas horizontales y de tareas como "diálogos de no impresión".

El listado 20.1 muestra el funcionamiento de la tarea de impresión en Java 1.1. El código todavia funciona en Java 1.4, pero existen otras maneras de hacer lo mismo, o de incluir funciones más útiles. El componente para imprimir utiliza el código de dibujo del ejemplo `Polys` del capitulo 10.

Listado 20.1. Tarea de impresión en Java 1.1

```
import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;
import javax.swing.*;
import java.util.*;
public class PrintingOneOne {
  static class MyComponent extends JPanel {
    Random random = new Random();
    public void paint(Graphics g) {
      Polygon convex, concave, selfintersecting;
      convex = new Polygon();
      convex.addPoint(20, 20);
      convex.addPoint(60, 24);
      convex.addPoint(50, 50);
      convex.addPoint(21, 75);
      convex.addPoint(10, 30);
      concave = new Polygon();
      concave.addPoint(100+20, 20);
      concave.addPoint(100+60, 24);
      concave.addPoint(100+25, 50);
      concave.addPoint(100+21, 75);
      concave.addPoint(100+10, 30);
      concave.addPoint(100+20, 20);
      selfintersecting = new Polygon();
      for(int i=0; i< 8; i++) {
        selfintersecting.addPoint(200 + random.nextInt(80),
                                  20  t random.nextInt(80));
      }
      g.fillPolygon(convex);
      g.drawPolygon(concave);
      g.fillPolygon(selfintersecting);
    }
  }
  public static void main(String args[]) {
    final JFrame frame = new ExitableJFrame("Printing 1.1");
    Container contentPane = frame.getContentPane();
    final Component printIt = new MyComponent();
    contentPane.add(printIt, BorderLayout.CENTER);
    JButton button = new JButton("Print");
    contentPane.add(button, BorderLayout.SOUTH);
    ActionListener listener = new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        Toolkit def = Toolkit.getDefaultToolkit();
        PrintJob job = def.getPrintJob(frame, "Print Job", null);
        if (job != null) {
          Graphics pg = job.getGraphics();
          printIt.print(pg);
```

```
            pg.dispose();
            job.end();
          }
        }
      };
      button.addActionListener(listener);
      frame.setSize(300, 200);
      frame.show();
    }
}
```

Este programa realiza esencialmente cuatro tareas. El primer paso consiste en encontrar la impresora que el usuario quiere utilizar.

```
Toolkit def = Toolkit.getDefaultToolkit();
PrintJob job = def.getPrintJob(frame, "Print Job", null);
```

Al invocar **get** `PrintJob()` de `Toolkit` aparece el cuadro de diálogo de selección de la impresora que aparece en la figura 20.1. El mecanismo de Java 1.1 no permitia imprimir sin que apareciera primero este cuadro de diálogo. El valor devuelto de la llamada a **get** `PrintJob()` es `PrintJob` de la impresora seleccionada, o `null` si el usuario pulsaba **Cancelar.**

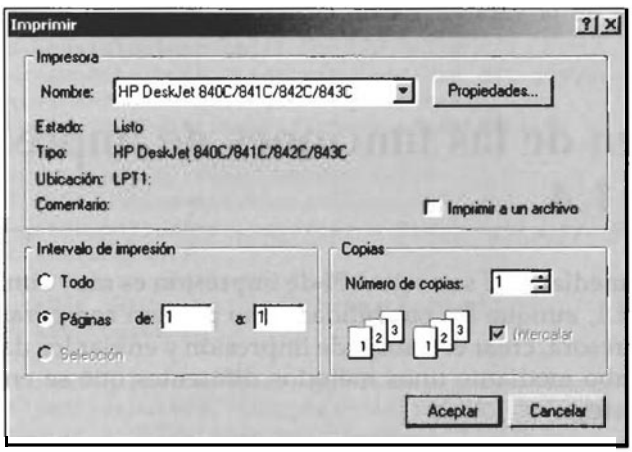


Figura 20.1. Cuadro de dialogo Print estandar

Asumiendo que el usuario no pulsó **Cancelar** se obtendrian los `Graphics` del trabajo.

```
Graphics pg = job.getGraphics();
```

Debido a que el programa fue diseiiado para ejecutarse en Java 1.1, no es posible convertir el objeto devuelto en un objeto `Graphics2D`, incluso si se ejecuta en 1.4.

Una vez obtenemos el objeto `Graphics`, informamos del componente cuyo contenido queremos que se imprima primero.

```
printIt.print(pg);
```

Como ya se ha mencionado anteriormente, la implementación por defecto de `print()` se limita a invocar `paint()`. Sin embargo, se emplean ambos métodos para imprimir de manera diferente de la que se pinta. Por ejemplo, si el contenido es un área de texto por la que nos podemos desplazar, `paint()` dibujaria únicamente lo que se puede ver en el área actual, mientras que `print()` imprimiria todas las páginas del contenido.

Al invocarse el método `dispose()` del objeto `Graphics` recuperado de `PrintJob` señalizaría el final de una página. Páginas múltiples implican múltiples llamadas a `getGraphics()`.

```
pg.dispose();
```

Por último, cerrariamos el trabajo de impresión mediante el método `end()` de `PrintJob`. No se imprimird nada hasta que no se concluya el trabajo.

```
job.end();
```

Esto es todo lo que se podia hacer con la función de impresión de Java 1.1. En la nueva versión, entra en juego el API de impresión unificado, también conocido como Servicio de impresidn de Java. Como se ha visto en el ejemplo anterior, las anteriores clases de apoyo de impresión están todavia disponibles aunque las nuevas funciones de la versidn 1.4 de la plataforma de Java 2 nos ofrecen mayores ventajas.

Resumen de las funciones de impresion de Java 1.4

Imprimir mediante el servicio API de impresión es muy similar a hacerlo con el API de Java 1.1, aunque las posibilidades no pueden compararse. Debemos seleccionar la impresora, crear el trabajo de impresidn y enviar los datos. Estas tareas las llevamos a cabo mediante unos métodos diferentes que se encuentran en cuatro nuevos paquetes.

- El paquete `javax.print` incluye las clases e interfaces principales.
- El paquete `javax.print.event` incluye las clases de gestión de eventos relacionados con el control de la impresión.
- El paquete `javax.print.attribute` contiene clases que describen los tipos de atributos relacionados con tareas de impresión.
- El paquete `javax.print.attribute.standard` incluye los ajustes para los atributos especificos de tareas de impresidn.

La primera tarea de impresión consiste en seleccionar la impresora, o más concretamente, el servicio de impresión, lo que se realiza preguntando a `PrintSer-`

viceLookup por el servicio registrado adecuado. Muchos de los mktodos de las clases no son invocados por el usuario ya que sólo las estáticas son importantes.

```
public static final PrintService lookupDefaultPrintService()
public static final PrintService[] lookupPrintServices(
  DocFlavor flavor, AttributeSet attributes)
public static final MultiDocPrintService[] lookupMultiDocPrintServices(
  DocFlavor[] flavors, AttributeSet attributes)
public static boolean registerServiceProvider(PrintServiceLookup sp)
public static boolean registerService(PrintService service)
```

Fijese en los tres mktodos `lookupXXX()`. En el caso más simple, se seleccionara el servicio de impresion determinado: `lookupDefaultPrintService()`. Es más probable, sin embargo, que tengamos que encontrar un servicio de impresion que soporte la impresión de tipos especificos de documentos: `lookupPrintServices()`. El tercer tipo, `lookupMultiDocPrintServices()`, imprime multiples documentos como un unico trabajo de impresión.

En los dos primeros casos, lo que obtenemos al buscar el servicio de impresión es una clase que implementa la interfaz `PrintService`:

```
public interface PrintService {
  public void addPrintServiceAttributeListener(
    PrintServiceAttributeListener 1);
  public DocPrintJob createPrintJob();
  public boolean equals(Object obj);
  public PrintServiceAttribute getAttribute(Class category);
  public PrintServiceAttributeSet getAttributes();
  public Object getDefaultAttributeValue(Class category);
  public String getName();
  public ServiceUIFactory getServiceUIFactory();
  public Class[] getSupportedAttributeCategories();
  public O ject getSupportedAttributeValues(Class category, DocFlavor favor,
    AttributeSet attributes);
  public DocFlavor[] getSupportedDocFlavors();
  public AttributeSet getUnsupportedAttributes(DocFlavor flavor,
    AttributeSet attributes);
  public int hashCode();
  public boolean isAttributeCategorySupported(Class category);
  public boolean isAttributeValueSupported(Attribute attribute,
    DocFlavor flavor, AttributeSet attributes);
  public boolean isDocFlavorSupported(DocFlavor flavor);
  public void removePrintServiceAttributeListener(
    PrintServiceAttributeListener l);
```

En el caso más sencillo, creamos una solicitud de impresión mediante `PrintService`. En el caso de API de Java 1.4, esta solicitud toma la forma de la interfaz `DocPrintJob`, que se asemeja al `PrintJob` anterior.

Para conseguirlo, sólo tenemos que invocar al mktodo `createPrintJob()` de `PrintService`:

```
DocPrintJob job = printService.createPrintJob();
```

El devuelto `DocPrintJob`, además de imprimir, proporciona los medios para obtener los atributos del trabajo de impresión:

```
public interface DocPrintJob {
  public void addPrintJobAttributeListener(PrintJobAttributeListener l,
    PrintJobAttributeSet attributes);
  public void addPrintJobListener(PrintJobListener l);
  public PrintJobAttributeSet getAttributes();
  public PrintService getPrintService();
  public void print(Doc doc, PrintRequestAttributeSet attributes)
    throws PrintException;
  public void removePrintJobAttributeListener(PrintJobAttributeListener l);
  public void removePrintJobListener(PrintJobListener l);
```

También es posible registrar un `PrintJobListener`, que se encuentra en el paquete `javax.print.event` para encontrar información sobre el trabajo de impresion. .Como muestra la siguiente definición de la interfaz, existen seis cosas que podemos descubrir sobre un trabajo de impresión:

```
public interface PrintJobListener {
  public void printDataTransferCompleted(PrintJobEvent e);
  public void printJobCompleted(PrintJobEvent e);
  public void printJobFailed(PrintJobEvent e);
  public void printJobCanceled(PrintJobEvent e);
  public void printJobNoMoreEvents(PrintJobEvent e);
  public void printJobRequiresAttention(PrintJobEvent e);
}
```

Nota

La clase `PrintJobAdapter` proporciona una implementación vacia de la interfaz `PrintJobListener`.

El trabajo concreto de impresión se realiza mediante el método `print()` de la interfaz `DocPrintJob`. Para ello, debemos enviar un `Doc` y especificar un `PrintRequestAttributeSet` para los atributos de la impresión.

`PrintRequestAttributeSet` es como un conjunto de ajustes de impresión: 3 copia, papel **A4,** en horizontal. Como se ve en la siguiente definicidn de la interfaz, es posible aiiadir ajustes de impresión de uno en uno, o en conjunto (`AttributeSet`):

```
public interface PrintRequestAttributeSet extends AttributeSet {
  public boolean add(Attribute attribute);
  public boolean addAll(AttributeSet attributes);
}
```

La clase que implementa la interfaz `PrintRequestAttributeSet` es `HashPrintRequestAttributeSet`. Mientras no establezcamos ningún atributo, todo lo que tenemos que hacer es crear una muestra de esta clase:

```
PrintRequestAttributeSet pras = new HashPrintRequestAttributeSet();
```

Por su parte, Doc también constituye una interfaz:

```
public interface Doc {
  public DocFlavor getDocFlavor();
  public DocAttributeSet getDocAttributeSet();
  public Object getPrintData();
  public Reader getReaderForText();
  public InputStream getStreamForBytes();
}
```

En este caso, la clase SimpleDoc (que ofrece un constructor) es la que implementa esta interfaz.

```
public SimpleDoc(Object printData, DocFlavor flavor,
  DocAttributeSet attributes)
```

El DocFlavor es el mismo que pasamos al método lookupPrintServices() de PrintServiceLookup si no estamos buscando el servicio preestablecido. Un DocFlavor funciona igual que los tipos de medio MIME (Extensiones multipropósito para Correo en Internet) Podemos tener, por ejemplo, un tipo MIME de un gif o imagen para un formato GIF de archivos gráficos. Para ello, no es necesario conocer si la imagen está destinada a formato GIF. Existen siete subclases de DocFlavor que nos ayudan a especificar los diferentes tipos mds utilizados.

Subclase	Categoria (Ver las tablas siguientes)
DocFlavor.BYTE_ARRAY	Orientado a bytes
DocFlavor.CHAR_ARRAY	Orientado a caracteres
DocFlavor.INPUT_STREAM	Orientado a bytes
DocFlavor.READER	Orientado a caracteres
DocFlavor.SERVICE_FORMATTED	Formato del servicio
DocFlavor.STRING	Orientado a caracteres
DocFlavor.URL	Orientado a bytes

Los *flavor* orientados a bytes (BYTE_ARRAY, INPUT_STREAM, y URL) disponen de los 19 *flavors* distintos que se muestran en la tabla 20.1. Los orientados a caracteres (CHAR_ARRAY, READER, y STRING) contienen los dos *flavors* de la tabla 20.2. Esto nos deja con SERVICE–FORMATTED para los tres *flavors* de la tabla **20.3.**

Tabla 20.1. Flavors orientados a bytes y sus tipos MIME

Constante	Tipo MIME
AUTOSENSE	"applicationloctet-stream"
GIF	"image/gif"

Constante	Tipo MIME
JPEG	"image/jpeg"
PCL	"application/vnd.hp-PCL"
PDF	"application/pdf"
PNG	"image/png"
POSTSCRIPT	"application/postscript"
TEXT_HTML_HOST	"text/html"
TEXT-HTML-US-ASCII	"text/html; charset=us-ascii"
TEXT_HTML_UTF_16	"text/html; charset=utf-16"
TEXT_HTML_UTF_16BE	"text/html; charset=utf-16be" (ordenarnientobig-endian)
TEXT_HTML_UTF_16LE	"text/html; charset=utf-16le" (ordenamientolittle-endian)
TEXT_HTML_UTF_8	"text/html; charset=utf-8"
TEXT-PLAIN-HOST	"text/plain"
TEXT_PLAIN-US-ASCII	"text/plain; charset=us-ascii"
TEXT-PLAIN-UTF-16	"text/plain; charset=utf-16"
TEXT_PLAIN_UTF_16BE	"text/plain; charset=utf-16be" (ordenamientobig-endian)
TEXT_PLAIN_UTF_16LE	"text/plain; charset=utf-16le" (ordenamientolittle-endian)
TEXT_PLAIN_UTF_8	"text/plain; charset=utf-8"

Tabla 20.2. Flavors orientados a caracteres y sus tipos MIME

Constante	Tipo MIME
TEXT_HTML	"text/html; charset=utf-16"
TEXT-PLAIN	"text/plain; charset=utf-16"

Tabla 20.3. Flavors formateados según el servicio de impresion y sus tipos MIME

Constante	Tipo MIME
PAGEABLE	"applicationlx-java-jvrn-local-objectref'
PRINTABLE	"applicationlx-java-jvrn-local-objectref'
RENDERABLE-IMAGE	"applicationlx-java-jvm-local-objectref'

Si los datos que queremos imprimir provienen de un archivo Postscript, necesitamos utilizar un DocFlavor de DocFlavor.INPUT_STREAM. POSTSCRIPT. Si estuviéramos generando los mismos datos de manera dinámica y los guardáramos en un array de bytes, emplearíamos, en su lugar, un DocFlavor.BYTE_ARRAY. POSTSCRIPT.

El tercer argumento del constructor SimpleDoc es DocAttributeSet, que funciona exactamente igual que un PrintRequestAttributeSet. En esta ocasión es HashDocAttributeSet quien implementa la interfaz.

Si ignoramos los atributos concretos de HashDocAttributeSet y PrintRequestAttributeSet, sólo nos queda el primer argumento del constructor de SimpleDoc, el cual representa los datos que se van a imprimir. Tomando como ejemplo un archivo PostScript, el único argumento del Object seria FileInputStream:

```
DocFlavor flavor = DocFlavor.INPUT-STREAM.POSTSCRIPT;
String filename = "myfile.ps";
FileInputStream fis = new FileInputSteam(filename);
DocAttributeSet das = new HashDocAttributeSet();
Doc doc = new SimpleDoc(fis, flavor, das);
```

Ahora que ya disponemos de un Doc, sólo nos queda imprimirlo:

```
job.print(doc, pras);
```

Después de imprimir, el programa no deberia detenerse inmediatamente. Mediante el PrintJobListener sabremos cuando la transferencia de datos se ha completado (printDataTransferCompleted()) y entonces será seguro cerrar el programa.

Ejemplo sencillo de impresion

Combinando todas las fases mencionadas, el listado 20.2 demuestra la manera de imprimir un archivo de imagen en la impresora predeterminada. (Si su impresora soporta Postscript, puede modificar el programa para probar a imprimir un archivo PostScript). Para ejecutar el programa sólo tenemos que pasar el archivo GIF como un argumento de linea de comandos, del siguiente modo:

```
java PrintingOneFour logo.gif
```

Listado 20.2. Tarea de impresion en Java 1.4

```
import javax.print.*;
import javax.print.attribute.*;
import javax.print.event.*;
import java.io.*;
public class PrintingOneFour {
  public static void main(String args[]) throws Exception {
    String filename = args[0];
    DocFlavor flavor = DocFlavor.INPUT_STREAM.GIF;
```

```
    PrintService printService =
      PrintServiceLookup.lookupDefaultPrintService();
    DocPrintJob job = printService.createPrintJob();
    PrintJobListener listener = new PrintJobAdapter() {
      public void printDataTransferCompleted(PrintJobEvent e) {
        System.out.println["Good-bye"];
        System.exit(0);

    };
    job.addPrintJobListener(listener);
    PrintRequestAttributeSet pras = new HashPrintRequestAttributeSet();
    FileInputStream fis = new FileInputStream(filename);
    DocAttributeSet das = new HashDocAttributeSet();
    Doc doc = new SimpleDoc(fis, flavor, das);
    job.print(doc, pras);
    Thread.sleep(10000);
  }
}
```

Cómo mostrar un cuadro de diálogo

Al ejecutar el programa del listado 20.2, comprobará que no aparece ningun cuadro de difilogo con opciones de impresora, sino que utiliza la preestablecida. Incluso si modificamos la linea de búsqueda de la impresora para que no utilice la predeterminada sigue sin aparecernos una ventana en la que podamos elegir la impresora que deseemos.

```
// Vieja linea
PrintService printService = PrintServiceLookup.lookupDefaultPrintService();
// Nueva linea
PrintService printService[] =
  PrintServiceLookup.lookupPrintServices(flavor, pras);
```

Para que el usuario pueda elegir una impresora diferente, debemos crear un cuadro de difilogo que muestre la lista de las impresoras disponibles. La clase `ServiceUI` hace esta operación por nosotros, a no ser que lo queramos crear manualmente. Esta clase incluye un método estfitico para mostrar la ventana emergente:

```
public static PrintService printDialog(GraphicsConfiguration gc,
  int x, int y, PrintService[] services, PrintService defaultService,
  DocFlavor flavor, PrintRequestAttributeSet attributes)
    throws HeadlessException
```

Sólo nos resta invocar este método estático para obtener el cuadro de difilogo de la figura 20.2.

Desde este cuadro, ademfis de seleccionar la impresora, podemos determinar la configuración de la página y otros atributos de apariencia, dependiendo de las opciones soportadas por la impresora. Todas estas opciones se muestran en las figuras 20.3 y 20.4.

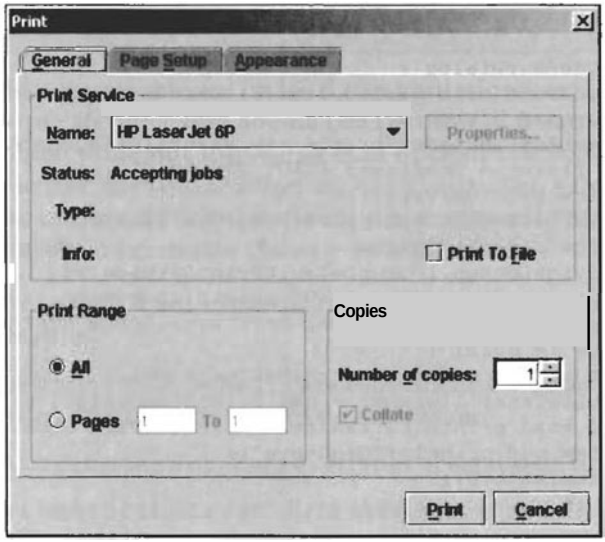


Figura 20.2. Ficha general del cuadro de dialogo de impresión ServiceUI

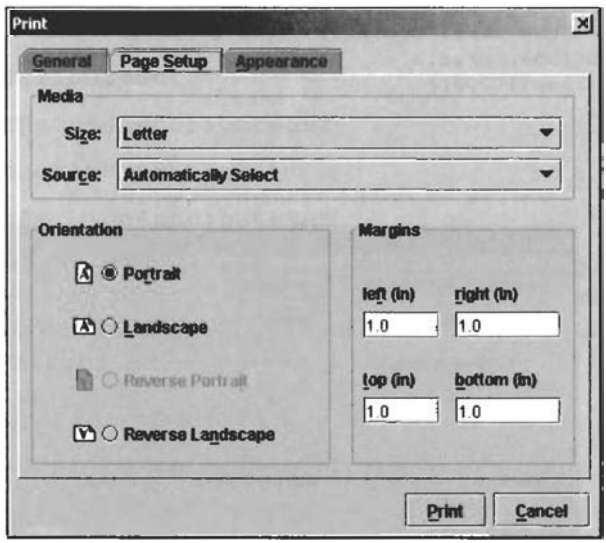


Figura 20.3. Ficha de configuración de pagina del cuadro de dialogo de impresion ServiceUI

El programa actualizado que muestra este cuadro de diálogo se representa un listado 20.3.

Listado 20.3. Prograrna con cuadro de dialogo para la irnpresora

```
import javax.print.*;
import javax.print.attribute.*;
```

```
import javax.print.event.*;
import java.io.*;
public class OneFourDialog {
  public static void main(String args[]) throws Exception {
    String filename = args[0];
    PrintRequestAttributeSet pras = new HashPrintRequestAttributeSet();
    DocFlavor flavor = DocFlavor.INPUT_STREAM.GIF;
    PrintService printService[] =
      PrintServiceLookup.lookupPrintServices(flavor, pras);
    PrintService defaultService =
      PrintServiceLookup.lookupDefaultPrintService();
    PrintService service = ServiceUI.printDialog(null, 200, 200,
      printService, defaultService, DocFlavor.INPUT_STREAM.GIF, pras);
    if (service != null) {
      DocPrintJob job = service.createPrintJob();
      PrintJobListener listener = new PrintJobAdapter() {
        public void printDataTransferCompleted(PrintJobEvent e) {
          System.out.println("Good-bye");
          System.exit(0);
        }
      };
      job.addPrintJobListener(listener);
      FileInputStream fis = new FileInputStream(filename);
      DocAttributeSet das = new HashDocAttributeSet();
      Doc doc = new SimpleDoc(fis, flavor, das);
      job.print(doc, pras);
      Thread.sleep(10000);
    }
  }
}
```

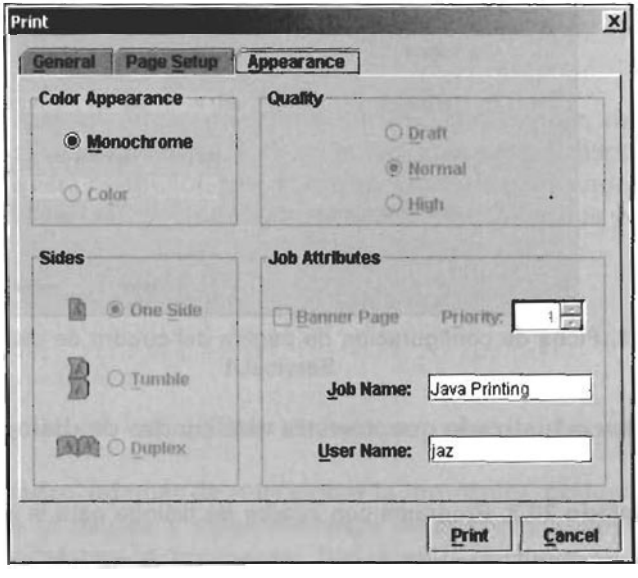


Figura 20.4. Ficha Apariencia del cuadro de diálogo de impresion ServiceUI

Utilizacion de atributos

Todos los cuadros representados en las figuras anteriores están escritos en puro código de Java, lo que significa que además de permitir al usuario elegir las opciones deseadas, también podemos introducir esas opciones directamente en el código. Esaesla tarea de las clases `PrintRequestAttributeSet` y `DocAttributeSet` del ejemplo anterior. Además de estas clases, disponemos de otros dos grupos. A continuacidn se muestran las cuatro clases y su utilización.

- `DocAttributeSet`: Para especificar caracteristicas de impresión para un unico documento.
- `PrintRequestAttributeSet`: Para especificar las caracteristicas de todos los documentos de un unico trabajo de impresión.
- `PrintJobAttributesSet`: Para averiguar el estado de un trabajo concreto.
- `PrintServiceAttributeSet`: Para averiguar el estado de un servicio de impresión concreto.

Estas cuatro interfaces se definen en el paquete `javax.print.attribute`. Junto a estas interfaces, se encuentran las cuatro clases que las implementan:

- `HashDocAttributeSet`.
- `HashPrintRequestAttributeSet`.
- `HashPrintJobAttributesSet`.
- `HashPrintServiceAttributeSet`.

Los atributos reales se definen en el paquete `javax.print.attribute.standard`. Existen al rededor de 70 clases para establecer los atributos de impresion. Estas clases se clasifican en ocho grupos, definidos como clases en el paquete `javax.print.attribute`:

- `DateTimeSyntax`: Para atributos de fecha y hora.
- `EnumSyntax`: Para atributos de enumeración "segura".
- `IntegerSyntax`: Para atributos de integrales.
- `Resolutionsyntax`: Para atributos de resolución de la impresora en puntos por centimetro (dpc)o puntos por pulgada (dpi).
- `SetOfIntegersyntax`: Para atributos de integrales positivas.
- `Size2DSyntax`: Para atributos que representen un tamafio bidimensional.
- `TextSyntax`: Para atributos de string.
- `URISyntax`: Para atributos con un valor URI (Identificador uniforme de recursos).

Al subclasificar una de estas clases sintacticas debemos conocer de manera implicita el tipo de valores que pueden guardarse en la clase de atributos.

Esto nos conduce al segundo grupo. Cada una de las clases de atributos implementa una interfaz adicional para indicar el grupo en el que se encuentran. Por ejemplo, la clase `Copies` implementa las interfaces `PrintRequestAttribute` y `PrintJobAttribute`. De este modo, podemos determinar el numero de copias para una solicitud y para un trabajo de impresión. La tabla 20.4 incluye todos los atributos disponibles.

Tabla 20.4. Atributos de impresion

Clase	Descripción
`Chromaticity`	Para especificar impresiones en color o monocromáticas
`ColorSupported`	Identifica si el dispositivo soporta color
`Compression`	Indica el formato de compresion de los datos
`Copies`	Especifica el numero de copias a imprimir
`CopiesSupported`	Indica los valores soportados para el nljmero de copias
`DateTimeAtCompleted`	Indica cuando un trabajo de impresion ha concluido
`DateTimeAtCreation`	Indica el tiempo de comienzo del trabajo de impresion
`DateTimeAtProcessing`	Indica el momento en se empezo a procesar el trabajo de impresion
`Destination`	Especifica un destino alternativo para el trabajo de impresion
`DocumentName`	Especifica el nombre del documento
`Fidelity`	Indica si el servicio de impresion debe rechazar la solicitud de impresion
`Finishings`	Indica si se necesita mas espacio para operaciones de acabado, tal como encuadernacion o grapado
`JobHoldUntil`	Indica la facha y la hora en que el trabajo de impresion estara disponible
`JobImpressions`	Indica el numero total de impresiones del documento
`JobImpressionsCompleted`	Indica el nljmero completo de impresiones
`JobImpressionsSupported`	Indica los valores que soportan las impresiones
`JobKOctets`	Indica el tamaño total en **(1.024)** Kilo octetos
`JobKOctetsProcessed`	Indica el tamaflo total procesado en Kilo octetos

Clase	Descripción
JobKOctetsSupported	Indica los tipos de valores soportados
JobMediaSheets	Indica el nurnero de hojas que se van a imprirnir
JobMediaSheetsCompleted	Indica el nurnero de hojas ya impresas
JobMediaSheetsSupported	Indica los valores soportados
JobMessageFromOperator	Envía un rnensaje al usuario desde el operador
JobName	Proporciona el nornbre del trabajo de irnpresion
JobOriginatingUserName	Indica el nornbre de usuario que inicia el trabajo
JobPriority	Indica la prioridad del trabajo de irnpresion
JobPrioritySupported	Indica los valores soportados
JobSheets	Indica si se deben irnprirnir la primera y la ultirna hoja
JobState	Indica el estado actual del trabajo de irnpresion
JobStateReason	Indica la razon del estado actual del trabajo de irnpresion
JobStateReasons	Indica las razones del estado actual del trabajo de irnpresion
Media	Indica los medios en los que se va a irnprirnir
MediaName	Nombres de los medios en los que se va a imprirnir
MediaPrintableArea	Especifica el area irnprirnible del rnedio
MediaSize	Especifica el tamaño de los rnedios. Clases internas de ayuda de **Engineering,** ISO, JIS, NA, entre otras
MediaSizeName	Indica el nornbre del rnedio
MediaTray	Indica la bandeja que va a utilizar el medio
MultipleDocumentHandling	Indica corno gestionar docurnentos multiples
NumberOfDocuments	Indica el nurnero de documentos en la lista de espera
NumberOfInterveningJobs	Indica el nurnero de documentos en la lista de espera por delante de un trabajo concreto
NumberUp	Indica el nurnero de páginas que enviar a un medio
NumberUpSupported	Indica los valores soportados
OrientationRequested	Especifica la orientación de la irnpresion
OutputDeviceAssigned	Nornbre del dispositivo de salida asignado al trabajo
PageRanges	Especifica las páginas a irnprirnir

Clase	Descripción
`PagesPerMinute`	Indica la velocidad de la irnpresora para copias en blanco y negro
`PagesPerMinuteColor`	Indica la velocidad de la irnpresora para copias en color
`PDLOverrideSupported`	Indica si la impresora debe ignorar instrucciones de procesarniento
`PresentationDirection`	Indica el diseño cuando se especifica un `NumberUp`
`PrinterInfo`	Incluye informacion sobre la irnpresora
`PrinterIsAcceptingJobs`	Indica si la irnpresora esta preparada para recibir trabajos de impresión
`PrinterLocation`	Indica la ubicacion de la irnpresora
`PrinterMakeAndModel`	Especifica la rnarca y el modelo de la impresora
`PrinterMessageFromOperator`	Muestra un rnensaje con el estado de la impresora desde el operador
`PrinterMoreInfo`	Proporciona un URI con inforrnacion adicional sobre la irnpresora
`PrinterMoreInfoManufacturer`	Proporciona un URI con información adicional sobre el fabricante
`PrinterName`	Especifica el nornbre de la irnpresora
`PrinterResolution`	Especifica la resolución de la irnpresora
`PrinterState`	Especifica el estado de la impresora
`PrinterStateReason`	Especifica la razón del estado actual de la irnpresora
`PrinterStateReasons`	Especifica las razones del estado actual de la irnpresora
`PrinterURI`	Proporciona un identificador unico del dispositivo
`PrintQuality`	Especifica un indicador de la calidad del dispositivo
`QueuedJobCount`	Indica el nurnero de trabajos en la lista de espera
`ReferenceUriSchemesSupported`	Indica de donde obtenerse los datos
`RequestingUserName`	Especifica el nornbre del usuario
`Severity`	Indica la severidad de `PrinterStateReason`

Clase	Descripción
`SheetCollate`	Indica cómo procesar copias multiples
`Sides`	Indica cómo realizar la impresión

Nota

Los atributos del servicio API de impresion se basan en RFC 2911, "El protocolo de Internet de impresion (IPP)/1.1: Modelo y Semántica". Para mas información sobre este protocolo, visite www.ietf.org/rfc/rfc2911.txt.

Para utilizar estos atributos sólo tenemos que añadirlos al `AttributeSet` correspondiente. En el siguiente código, se imprimirá un documento cinco veces en el archivo `/tmp/example`:

```
PrintRequestAttributeSet pras = new HashPrintRequestAttributeSet();
pras.add(new Copies(5));
pras.add(new Destination("file:/tmp/example"));
```

También podemos emplear los distintos atributos Supported:

```
CopiesSupported supported = (CopiesSupported)
  service.getSupportedAttributeValues(Copies.class, null, null);
if ((supported != null) && (copSupp.contains(5))) {
  pras.add(new Copies(5));
} else {
  ...
}
```

Utilización de StreamPrintService

Existe un segundo grupo de servicios de impresión, similares a cambiar el atributo `Destination` de un `PrintService`, que nos permiten convertir diferentes formatos de impresion, en los que el destino asignado automáticamente sea un `OutputStream`. La clase que se soporta es `StreamPrintService`, una subclase de `PrintService`; aunque el mecanismo de buscar el servicio de impresión es diferente.

Cuando trabajamos con un `StreamPrintService`, en lugar de buscar el servicio mediante un `PrintServiceLookup`, utilizamos un `StreamPrintServiceFactory`, de la siguiente manera:

```
public static StreamPrintServiceFactory[]
  lookupStreamPrintServiceFactories(DocFlavor flavor, String
outputMimeType)
```

Los tipos MIME pued.en obtenerse de la segunda columna de la tabla 20.1, aunque no es necesario. En su lugar, sólo tenemos que preguntar a la constante de la primera columna por el tipo MIME adecuado mediante `getMimeType()`, como en

```
DocFlavor.OUTPUT–STREAM. POSTSCRIPT.getMimeType()
```

Una vez tengamos el grupo de *factories* que convierten un flavor en tipo MIME, obtenemos el `StreamPrintService` adecuado y lo añadimos a un `Output-Stream`:

```
DocFlavor flavor = DocFlavor.INPUT_STREAM.GIF;
String mimeType  = DocFlavor.INPUT–STREAM. POSTSCRIPT.getMimeType();
StreamPrintServiceFactory[] factories =
  StreamPrintServiceFactory.lookupStreamPrintServiceFactories(
  flavor, mimeType);
String filename = "out.ps";
FileOutputStream fos = new FileOutputStream(filename);
StreamPrintService sps = factories[0].getPrintService(fos);
The conversion from GIF to PostScript would be done just like the normal
printing earlier:
FileInputStream fis = new FileInputStream("logo.gif");
DocPrintJob dpj = sps.createPrintJob();
PrintRequestAttributeSet pras = new HashPrintRequestAttributeSet();
Doc doc = now SimpleDoc(fis, flavor, null);
pj.print(doc, pras);
fos.close();
```

El programa del listado 20.4 demuestra este proceso.

Listado 20.4. Conversion de un GIF en un POSTSCRIPT

```
import javax.print.*;
import javax.print.attribute.*;
import javax.print.event.*;
import java.io.*;
public class StreamOneFour {
  public static void main(String args[]) throws Exception {
    String infile = args[0];
    DocFlavor flavor = DocFlavor.INPUT_STREAM.GIF;
    String mimeType  = DocFlavor.INPUT–STREAM. POSTSCRIPT.getMimeType();
    StreamPrintServiceFactory[] factories =
      StreamPrintServiceFactory.lookupStreamPrintServiceFactories(
        flavor, mimeType);
    String filename = "out.ps";
    FileOutputStream fos = new FileOutputStream(filename);
    StreamPrintService sps = factories[0].getPrintService(fos);
    FileInputStream fis = new FileInputStream(infile);
    DocPrintJob dpj = sps.createPrintJob();
    PrintRequestAttributeSet pras = new HashPrintRequestAttributeSet();
    Doc doc = new SimpleDoc(fis, flavor, null);
    dpj.print(doc, pras);
    fos.close();
  }
```

Nota

Debido al modo de funcionamiento de `StreamPrintService`, la impresion en el `OutputStream` se produce en el mismo thread, por lo que no se requiere ningún `PrintJobListener` ni ninguna llamada a `Thread.sleep()`.

Advertencia

Si queremos utilizar un `StreamPrintService` para `DocFlavor.INPUT_STREAM.PDF`, con la intención de crear un archivo PDF, el `lookupStreamPrintServiceFactories()` devolverá un arrayvacio. Esperemos que este tipo de conversor esté disponible pronto.

Impresión de pantallas

Aunque ya hemos visto la manera de imprimir archivos, eso no es todo lo que podemos hacer con las nuevas prestaciones de impresión de la versión 1.4. Anteriormente hemos visto la manera de imprimir pantallas mediante API 1.1. Para conseguir lo mismo con API 1.4, nos valemos en realidad de las funciones de impresión de API 1.2, junto con algunas pocas funciones nuevas.

Las funciones de Java 1.2 requieren que el objeto que vamos a imprimir implemente la interfaz `java.awt.print.Printable`, que contiene un unico método. Las dos constantes se utilizan para indicar si se debe imprimir otra página, y de ser así se invoca `print()` de nuevo, de manera automática:

```
public interface Printable {
  public static final int PAGE_EXISTS;
  public static final int NO_SUCH_PAGE;
  public int print(Graphics g, PageFormat pageFormat, int pageIndex)
    throws PrinterException;
}
```

A continuación, tenemos que pasar el objeto que implementa la interfaz `Printable` al constructor `SimpleDoc`, en el que el `DocFlavor` es `DocFlavor.SERVICE_FORMATTED.PRINTABLE`. El listado 20.5 demuestra esta forma combinada de impresión.

Listado 20.5. Impresion combinando los metodos de Java 1.2 y 1.4

```
import javax.print.*;
import javax.print.attribute.*;
import java.awt.print.*;
import java.awt.*;
```

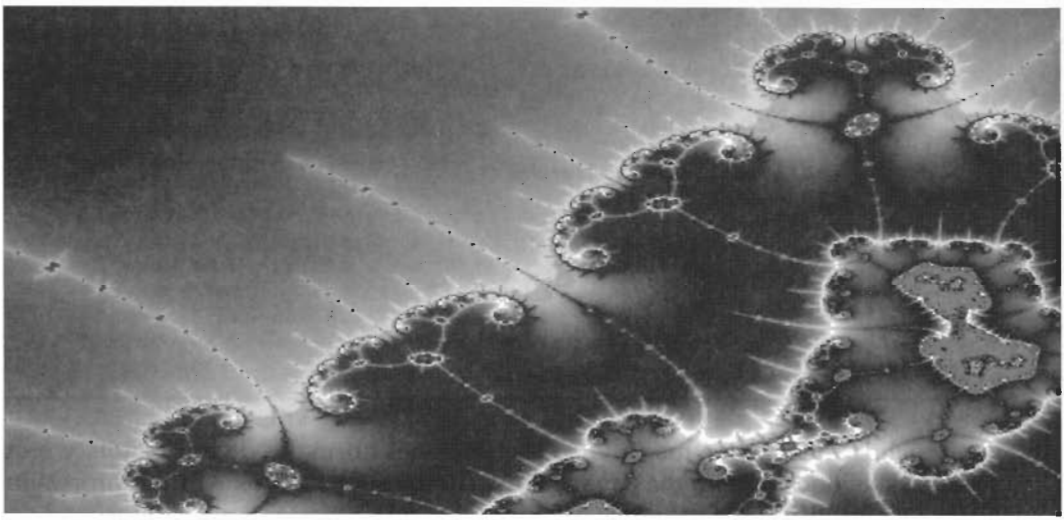
```
import java.awt.geom.*;
import java.awt.event.*;
import javax.swing.*;
import java.io.*;
public class PrintingOneTwoFour {
  static class MyComponent extends JPanel
      implements Printable {
    public void paint(Graphics g) {
      Graphics2D g2d = (Graphics2D)g;
      g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
        RenderingHints.VALUE_ANTIALIAS_ON);
      g2d.setPaint(Color.BLUE);
      g2d.setStroke(new BasicStroke(3));
      CubicCurve2D cubic = new CubicCurve2D.Float(10, 80, 60, 30, 110, 130,
        160, 80);
      g2d.draw(cubic);
      cubic = new CubicCurve2D.Float(60, 205, 150, 120, 20, 120, 110, 205);
      g2d.fill(cubic);
      cubic = new CubicCurve2D.Float(160, 130, 210, 30, 260, 30, 310, 130);
      g2d.fill(cubic);
      cubic = new CubicCurve2D.Float(160, 130, 260, 230, 210, 230, 310,
130);
      g2d.draw(cubic);
      // Puntos de la curva
      g.setColor(Color.BLACK);
      g.fillRect(58, 28, 5, 5);   g.fillRect(108, 128, 5, 5);
      g.fillRect(148, 118, 5, 5); g.fillRect(18, 118, 5, 5);
      g.fillRect(208, 28, 5, 5);  g.fillRect(258, 28, 5, 5);
      g.fillRect(258, 228, 5, 5); g.fillRect(208, 228, 5, 5);
    }
    public int print(Graphics g, PageFormat pageFormat, int pageIndex) {
      if (pageIndex == 0) {
        paint(g);
        return Printable.PAGE_EXISTS;
      } else {
        return Printable.NO_SUCH_PAGE;
      }
    }
  }
  public static void main(String args[]) throws Exception {
    final JFrame frame = new ExitableJFrame("Printing 1.2/4");
    Container contentPane = frame.getContentPane();
    final Component printIt = new MyComponent();
    contentPane.add(printIt, BorderLayout.CENTER);
    JButton button = new JButton("Print");
    contentPane.add(button, BorderLayout.SOUTH);
    ActionListener listener = new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        DocFlavor flavor = DocFlavor.SERVICE_FORMATTED.PRINTABLE;
        PrintService printService =
          PrintServiceLookup.lookupDefaultPrintService();
        DocPrintJob job = printService.createPrintJob();
        PrintRequestAttributeSet pras = new HashPrintRequestAttributeSet();
        DocAttributeSet das = new HashDocAttributeSet();
        Doc doc = new SimpleDoc(printIt, flavor, das);
        try {
```

```
            job.print(doc, pras);
          } catch (PrintException pe) {
            pe.printStackTrace();
          }
        }
      };
      button.addActionListener(listener);
      frame.setSize(350, 350);
      frame.show();
    }
}
```

Resumen

Muchos clientes van más allá de las funciones normales de impresión, y requieren prestaciones mucho m& avanzadas. Aunque los nuevos API de impresión de Java 1.4 simplifican en gran manera las tareas relacionadas con las interfaces de las impresoras, no existe ninguna función que permita pasar los datos de una `JTable` a una impresora, mediante páginas multiples. Para eso, debemos sacar partido de los paquetes de terceros. A continuación se incluye una lista de algunos de los paquetes disponibles:

- **Elixir Report:** Paquete que soporta XML y Java (http://www.elixirtech.com).
- **FOP para XSLT:** Un analizador de código abierto que soporta la generación PDF (http://xml.apache.org/fop).
- **i-net Crystal-Clear:** Paquete comercial (http://www.inetsoftware.de).
- **JasperReports:** Paquete para código abierto y orientado a pdginas (http://jasperreports.sourceforge.net).
- **JReport:** Paquete comercial (http://www.jinfonet.com).



21
JavaBeans

La tecnología de los Beans de Java vio la luz en la versión 1.1 de Java, y no ha cambiado mucho desde entonces, aunque su importancia aumenta con el tiempo. JavaBeans es en realidad un marco para crear componentes reutilizables, así como aplicaciones mediante sistemas de funcionamiento rápido.

Se pueden comprar o crear los complementos por separado, para después conectar los Beans y construir una aplicación mayor a partir de componentes más pequeños. Al crear estos elementos mediante JavaBean, es más sencillo conectarlos, incluso sin necesidad de crear código, ya que `JBuilder` y `Forte` se encargarán de ello.

Aunque por lo general se cree que los componentes de JavaBean sirven únicamente para diseñar ventanas gráficas en aplicaciones para clientes, lo cierto es que también se emplean para aplicaciones del servidor. Son una parte integral de las páginas *JavaServer* (JSP), un lenguaje de programación basado en Java y utilizado para crear páginas Web.

Los componentes de JavaBean no tienen nada que ver con *Enterprise* JavaBeans (EJB), a pesar de su parecido. Los componentes de JavaBean sirven para almacenar información de estado mientras que los componentes de EJB sirven para almacenar lógica empresarial.

Los JavaBeans pueden además tener una representación gráfica. Para más información sobre la diferencia entre ambos, consulte el capítulo 23.

Este capítulo explica cómo crear y reutilizar componentes de JavaBean, también conocidos como Beans. Repasaremos todas las ventajas que nos ofrecen y veremos por qué todos los métodos se llaman `getThis()` y `setThat()`.

Modelo de los componentes JavaBeans

Los JabaBeans se basan en un modelo de componentes de software para Java especialmente diseñado para permitir a otros fabricantes crear y vender componentes Java integrados en productos de software distintos. Un desarrollador puede comprar componentes, arrastrarlos y soltarlos en una herramienta de desarrollo, modificarlos y revisarlos, sin escribir ningún tipo de código. Dentro del modelo de Java, los componentes pueden modificarse o combinarse entre ellos para crear nuevos o completar aplicaciones. En el momento de ejecutar el programa, el usuario puede también modificar estos componentes a través de propiedades creadas por el desarrollador, tales como color o forma, u otras más sofisticadas que afectan totalmente el comportamiento del componente. El modelo descrito en las Especificaciones descritas en la página (http://java.sun.com/products/javabeans/docs/beans.101.pdf) define fundamentalmente cinco funciones:

- **Introspección:** Un mecanismo que permite a los componentes propagar las operaciones y propiedades que soportan, y que tiene además la capacidad de descubrir tales mecanismos en otros componentes.
- **Comunicación:** Un mecanismo de gestión de eventos para crear un evento que va a recibirse en forma de mensaje por otros componentes.
- **Persistencia:** Un medio para almacenar el estado de un componente.
- **Propiedades:** Un mecanismo para controlar el diseño de un componente, incluyendo el espacio físico que ocupa y la relación entre los componentes de un mismo contenedor.
- **Personalización:** Un mecanismo que permite al desarrollador controlar los cambios requeridos para cada componente. Los componentes deben proporcionar al constructor visibilidad para las propiedades y operaciones. El constructor debe a su vez proporcionar los medios para modificar los componentes para así construir la aplicación adecuada.

El modelo de componentes de JavaBeans permiten diseñar el software de modo que pueda modificarse. Cada aplicación de software contiene un grupo de propiedades, operaciones y controladores de eventos. Combinando varios componentes podemos obtener el comportamiento deseado. Los componentes se incluyen dentro de un contenedor o *kit* de herramientas, que proporciona el contexto para la aplicación.

Los Bean de Java

Un Bean de Java es un componente elemental reutilizable de software. Los Beans se manipulan en una herramienta de constructor (contenedor) que produce un

rendimiento operativo concreto. En definitiva, se trata de bloques de construcción que sirven para crear aplicaciones. Los Beans más comunes no son habitualmente más que pequeños o medianos programas de control, pero también se pueden generar Beans que abarquen una aplicación entera e integrarlos en un documento compuesto. Por ejemplo, la figura 21.1 ilustra el caso de un panel de contenedor que consta de tres componentes.

Figura 21.1. Grupo de componentes Bean en un contenedor

En general, los Beans se representan por medio de GUI sencillas. Pueden ser elementos de botones, barras deslizantes, listas de menús, etc. Estos componentes ilustran a la perfección las posibilidades que ofrecen los Beans. Sin embargo, éstos se pueden hacer invisibles para poder trabajar a escondidas. En cualquier caso, es más fácil entender su utilidad si los vemos como bloques de construcción de componentes destinados a percibir un evento y procesarlo para que se lleve a cabo una operación determinada.

Un Bean no es ni una biblioteca de clases ni un API como el marco de Colecciones. Este último sirve para almacenar objetos, mientras que un Bean conecta al usuario con el lugar de almacenamiento. Por ejemplo, se puede usar para crear un simple botón **Display** en la interfaz del usuario de una aplicación. Éste, a su vez, puede ser un Bean que enumere y muestre los elementos de la colección.

Arquitectura de los Beans

Los Beans se dividen en tres partes: propiedades, métodos y eventos. Estos elementos están representados en la figura 21.2.

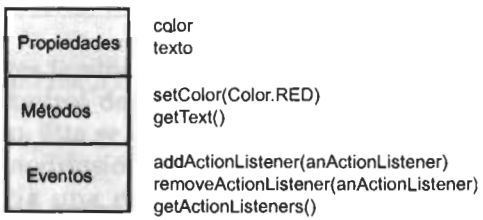


Figura 21.2. Arquitectura de los Beans

Las propiedades del Bean describen el estado de sus atributos, incluida su representación física. Son los principales mecanismos de transformación del Bean y se fijan o consultan a través de métodos, que también se emplean para desencadenar y recibir eventos.

Éste es el principal medio de comunicación de los Bean, que conectados por métodos de eventos, constituyen un sistema o aplicación completa, tal como hemos podido observar en la figura 21.1.

El modelo de eventos JavaBean

Como ya hemos dicho anteriormente, los eventos son el vehículo de información entre los Bean, así como entre éstos y los contenedores. Este mecanismo se utiliza para ordenar a otro Bean que realice una acción determinada, o para comunicarle los cambios de estado que se produzcan. La fuente registra o propaga los eventos (a través de una llamada a un método) entre uno o varios oyentes o listeners determinados.

Un evento de Bean pasa en forma de objeto, una muestra de una clase derivada de `java.util.EventObject`. El evento puede crearse directamente de una subclase, como en este ejemplo:

```
import java.util.*;
public class TheEvent extends EventObject {
  public TheEvent(Object source) {
    super(source);
  }
}
```

El modelo JavaBean reproduce con bastante exactitud el modelo de eventos Swing, lo que constituye una gran ventaja, porque muchos componentes Bean son elementos visuales GUI que forman parte de la jerarquía de los componentes de Swing.

Los Bean envían los eventos a los oyentes, que deben responder adecuadamente. No existe una convención concreta para la forma de envío, aunque, por lo general, se utiliza el método `fire` seguido del nombre del método para el oyente del evento, del siguiente modo:

```
protected void fireActionPerformed(ActionEvent e) {
  if (actionListeners != null) {
    Vector listeners = actionListeners;
    int count = listeners.size();
    for (int i = 0; i < count; i++) {
      ((ActionListener)listeners.elementAt(i)).actionPerformed(e);
    }
  }
}
```

En este caso, se envía un objeto de evento al grupo de oyentes, y el método que se dispara es `actionPerformed()`.

Fuentes y destinatarios de eventos

El modelo de eventos de los JavaBeans está constituido por las fuentes y los destinatarios de los mismos. Las primeras se encargan de iniciar el proceso al registrar uno o varios destinatarios de eventos. Ambos elementos establecen una serie de métodos, que la fuente usará para comunicar los sucesos a los oyentes.

El emisor trata de notificar un evento a un grupo arbitrario de destinatarios. Esta modalidad funciona por omisión y se llama multidifusión. La fuente lleva un registro de los oyentes de cada tipo de evento para comunicárselo al que corresponda en cada caso.

El sistema de multidifusión permite al emisor difundir los eventos entre varios destinatarios a la vez, lo cual se consigue conociendo en todo momento el estado de los oyentes registrados:

```
private transient Vector actionListeners;
...
public synchronized void removeActionListener(ActionListener l) {
  if ((actionListeners != null) && (actionListeners.contains(l))) {
    Vector v = (Vector)actionListeners.clone();
    v.removeElement(l);
    actionListeners = v;
  }
}
public synchronized void addActionListener(ActionListener l) {
  Vector v = ((actionListeners == null) ?
    new Vector(2) : (Vector)actionListeners.clone());
  if (!v.contains(l)) {
    v.addElement(l);
    actionListeners = v;
  }
}
```

Nota

El dilema `add/remove ActionListener` se suele sincronizar para evitar las "carreras" de multitareas. (Las carreras se producen cuando muchos procesos intentan realizar una acción con datos compartidos y el resultado de la misma depende del orden de operaciones.) No obstante, las clases `AWTEvent-Multicaster` y `EventListener` efectúan todas las labores de sincronización.

Las fuentes de eventos también pueden ser de monodifusión, cuando solamente tienen que llevar un control de un oyente por cada uno de los tipos de suceso difundidos. Por lo tanto, ésta se comunica únicamente al destinatario en cuestión.

Los eventos de monodifusión sólo permiten registrar un oyente; pues, de lo contrario, se produciría una excepción llamada `java.util.TooManyListenersException`. El receptor del evento de monodifusión únicamente almacena una muestra de oyente:

```
private transient ActionListener actionListeners = null;
public synchronized void addActionListener(ActionListener l)
    throws java.util.TooManyListenersException {
  if (l == null) {
    actionListeners = l;
  } else {
    throw new java.util.TooManyListenersException();
  }
}
public synchronized void removeActionListener(ActionListener l) {
  if (actionListeners == l) {
    actionListeners = null;
  }
}
```

El destinatario del evento es una muestra de una clase que implementa algunas (o todas) las interfaces `EventListener`; concretamente, los métodos que interesan a dicha clase. Cada tipo de evento está asociado a un método específico, y éstos se agrupan por aplicaciones.

```
import java.awt.event.*;
public class MyActionListener implements ActionListener {
   public void actionPerformed(ActionEvent e) {
      String command = e.getActionCommand();
      if (command.equals ("add")) {
         ...
      }
   }
}
```

Algunos programadores prefieren que cada oyente sólo escuche los eventos de una fuente. En ese caso, no hace falta el bloque `if`, puesto que ya se sabe a qué fuente corresponde el suceso. Así es como las herramientas IDE suelen generar los códigos.

El programa del listado 21.1, que combina los fragmentos de código mostrados hasta ahora, demuestra cómo un oyente de eventos se registra con una fuente y cómo ésta envía un evento a cada uno de los oyentes registrados.

Listado 21.1. Mantenimiento y notificación de una lista de oyentes de eventos

```
import java.awt.event.*;
import java.util.*;
public class MyBean {
  private transient Vector actionListeners;
  public synchronized void removeActionListener(ActionListener l) {
    if (actionListeners != null && actionListeners.contains(l)) {
      Vector v = (Vector) actionListeners.clone();
      v.removeElement(l);
      actionListeners = v;
    }
  }
  public synchronized void addActionListener(ActionListener l) {
    Vector v = (actionListeners == null) ?
```

```
        new Vector(2) : (Vector) actionListeners.clone();
      if (!v.contains(l)) {
        v.addElement(l);
        actionListeners = v;
      }
    }
    protected void fireActionPerformed(ActionEvent e) {
      if (actionListeners != null) {
        Vector listeners = actionListeners;
        int count = listeners.size();
        for (int i = 0; i < count; i++) {
          ((ActionListener) listeners.elementAt(i)).actionPerformed(e);
        }
      }
    }
}
```

Propiedades de los Beans

Las propiedades de los Beans son los atributos asociados a éstos, como el color, el tamaño o el texto, que se debe utilizar como etiqueta. Hay varias formas de utilizarlas, según el entorno en el que accedamos al Bean. Las propiedades se pueden cambiar durante el tiempo de ejecución, mediante sus métodos `get/set`, en un entorno de *scripting*, tal como JSP, o a través de la hoja de propiedades incluida en la herramienta constructor/personalización de los Beans.

Se definen cuatro tipos de propiedades: simples, indexadas, dependientes y restrictivas.

Propiedades simples

El usuario final (otro programador) puede modificar las propiedades simples mediante el par de métodos `get/set` correspondiente a las mismas. Por ejemplo, supongamos que un Bean tiene un determinado color y el usuario puede cambiarlo a través de un cuadro de diálogo de propiedades incluido en el producto. El Bean le ofrecerá dos métodos para hacerlo:

```
private Color background;
public Color getBackgroundColor() {...}
public void setBackgroundColor(Color c) {...}
```

Propiedades indexadas

Las propiedades se pueden indexar para que admitan una serie de valores, de forma que los índices vengan dados por valores `int`. Una vez indicadas, hay cuatro métodos para acceder a ellas, a través de un único elemento o de un conjunto entero de valores:

```
void setLabel(int index, String label)
String getLabel(int index)
void setLabel(String labels[])
String[] getLabel()
```

Los métodos indexados deben comprobar los límites del array y activar una `java.lang.ArrayIndexOutOfBoundsException` si el índice los rebasa. Estas propiedades, sin embargo, no se utilizan demasiado.

Propiedades dependientes

Una propiedad dependiente manda un mensaje de alteración de valor a otros Beans, cuando se altera la propiedad, y, por tanto, se difunde un evento.

Este proceso de comunicación se realiza vinculando el tipo de propiedad a un oyente de eventos `PropertyChangeListener`. Cualquier Bean que quiera comunicarse consigo mismo, con otro o con la herramienta del contenedor de Beans deberá llevar incluidos dos métodos de registro de oyentes de eventos de multidifusión, y un tercer método opcional para obtener el grupo de oyentes actual:

```
public void addPropertyChangeListener (PropertyChangeListener l)
public void removePropertyChangeListener (PropertyChangeListener l)
public PropertyChangeListener[] getPropertyChangeListeners()
```

El paquete `java.beans` contiene una clase compatible con ambas propiedades que sirve para gestionar la lista de oyentes. Por ejemplo:

```
private Color color;
private transient PropertyChangeSupport propertyChangeListeners =
  new PropertyChangeSupport(this);
...
public void setColor(Color color) {
  Color  oldColor = color;
  this.color = color;
  propertyChangeListeners.firePropertyChange("color", oldColor, newColor);
}
```

El método `firePropertyChange()` llamará a `propertyChange()` a través del objeto que implementa la interfaz `PropertyChangeListener`. La escucha de los eventos de cambio de propiedades permite que muchos Bean compartan un mismo valor, como por ejemplo su color de fondo. Cuando uno de ellos cambia, los demás lo hacen simultáneamente, siempre que todos estén registrados como `PropertyChangeListener`.

Propiedades restrictivas

Las propiedades restrictivas autorizan o rechazan los cambios efectuados. Al usuario o programador se le comunica ese extremo mediante una excepción. Por ejemplo:

```
public void setSize(Dimension size) throws PropertyVetoException {
...
}
```

Las propiedades restrictivas utilizan la interfaz `VetoableChangeListener` para autorizar los cambios. El Bean las implementa incluyendo un par de métodos `add/remove`, además de `get/set` para la propiedad:

```
public void addVetoableChangeListener (VetoableChangeListener v)
public void removeVetoableChangeListener (VetoableChangeListener v)
VetoableChangeListener[] getVetoableChangeListeners()
```

Por supuesto, el método de propiedades de Bean deberá difundir un evento antes de que se cambie la propiedad:

```
private Dimension size;
private transient VetoableChangeSupport vetoableChangeListeners =
  new VetoableChangeSupport(this);
private transient PropertyChangeSupport propertyChangeListeners =
  new PropertyChangeSupport(this);
...
public void setSize(Dimension size) throws PropertyVetoException {
  Dimension  oldSize = getSize();
  vetoableChangeListeners.fireVetoableChange("size", oldSize, newSize);
  // si se rechaza el cambio, no se llega hasta aquí
  this.size = size;
  propertyChangeListeners.firePropertyChange("size", oldSize, newSize);
}
```

Nota

El método `setSize` muestra el tamaño en forma de propiedad dependiente y restrictiva. Normalmente, éstas siempre son dependientes. Si no lo fueran, todo aquel que hubiera estado escuchando a través de un `VetoableChangeListener` ignoraría si otro oyente ha vetado el cambio. Al escuchar por ambos, puede desautorizarlo si el nuevo valor es inadecuado o retocarlo si nadie lo ha vetado.

Los métodos Bean

Los métodos Bean son las operaciones que reciben llamadas

- Desde otros componentes (que tengan un ejemplo del Bean).
- Desde el contenedor.
- Desde un entorno de *scripting*.

Los métodos Bean pueden exportarse mediante su publicación; de esta forma, se permite ver los métodos con una herramienta de constructor, mediante el sistema de exploración de Java, o JSP.

Estos procedimientos sirven para fijar y obtener propiedades, difundir y captar eventos. Los métodos Bean no siempre son públicos. Aquellos que son privados, amigables o están protegidos pueden ser invisibles a las herramientas de constructor.

Grabación de Beans

Los Beans se graban en un formato JAR (Archivo de Java). Básicamente, los ficheros JAR son documentos formateados en Zip con un componente optativo llamado fichero manifiesto, que puede llevar información adicional sobre su contenido.

Las especificaciones de JavaBeans no obligan a las aplicaciones que utilizan Beans a emplear ficheros JAR, ni siquiera a grabar éstas como tales, pero sí sugiere que los Beans se envíen en un principio como ficheros JAR.

Revisión y personalización de Beans

Los JavaBeans se adquieren a los vendedores como partes de software o son directamente programados por las compañías, probablemente como componente genérico; es decir, apto para su personalización en el momento de diseñar la aplicación.

Esto se logra mediante dos sistemas de Java que están madurando a medida que se desarrolla el API de JavaBean:

- El Java Reflection API, una serie de clases usada para examinar los ficheros de clase y descubrir las propiedades (variables) de ésta y sus métodos.
- El Java Serialization API, que sirve para almacenar permanentemente una clase, incluido su estado actual.

Estas dos técnicas se utilizan para permitir el estudio de los Beans y su descubrimiento por parte de una herramienta de constructor, para su posterior modificación y grabación dentro de una aplicación.

El proceso de exploración de Beans

El procedimiento de exploración JavaBean permite el análisis de las propiedades, métodos y eventos de un Bean. Este proceso se ha vuelto bastante rutinario. Se considera que las clases de Beans tienen propiedades si hay algún método que establezca o consiga un tipo de ellas:

```
public PropertyType getPropertyName()
public void setPropertyName(PropertyType p)
```

Si solamente se descubre uno de los métodos `get/set`, entonces se considera que `PropertyName` es sólo de lectura o de escritura.

Las propiedades booleanas (es decir, las que dan un tipo `boolean`) también pueden llevar un método `boolean`:

```
public boolean isPropertyName()
```

Las propiedades indexadas también se descubren cuando entre las signaturas del método se incluyen:

```
public PropertyType getPropertyName(int index)
public PropertyType[] getPropertyName()
public void setPropertyName(int index, PropertyType value)
public void setPropertyName(PropertyType value[])
```

Los eventos se hallan a través de dos métodos `add/remove`, que se supone empiezan por `add` y `remove` y toman un argumento `<EventListenerType>` que extiende la interfaz `java.util.EventListener` cuando el nombre del tipo acaba por `Listener`. Por ejemplo:

```
public void addListenerType(ListenerType l)
public void removeListenerType(ListenerType l)
```

Existe otro método opcional para obtener oyentes de eventos:

```
public ListenerType[] getListenerTypes()
```

Los métodos descubren si el acceso a los mismos es público, incluidos los de propiedades y eventos.

La interfaz BeanInfo

El JavaBean API también ofrece una interfaz explícita, que permite a los programadores de Beans exponer las propiedades, eventos, métodos y todo tipo de datos correspondientes a las mismas. Los vendedores de éstas incluyen la interfaz `BeanInfo` mediante una clase que la extiende y a continuación añade al nombre:

```
public class MyBeanBeanInfo implements java.util.BeanInfo {...
```

La interfaz `BeanInfo` proporciona una serie de métodos de acceso a los datos del Bean, pero el programador de éste puede incluir, por su parte, algunos ficheros privados de descripción que le sirvan a la clase `BeanInfo` para completar la información. Por omisión, se crea un objeto `BeanInfo` cada vez que se ejecuta la exploración en el Bean.

En resumen, el proceso de análisis correspondiente a un Bean consta de las siguientes etapas:

- Recorrido de la cadena de clases y superclases de cada una de las clases analizadas.
- Búsqueda de un nombre de clase `BeanInfo` (un nombre de clase con la coletilla `BeanInfo`).
- Uso de la reflexión a bajo nivel para estudiar la clase y crear un objeto `BeanInfo` con los resultados.

Por regla general, la interfaz `BeanInfo` no se implementa directamente, sino que se emplea la subclase `SimpleBeanInfo` y se personaliza.

Persistencia del Bean

Los Beans de Java son unos componentes que dependen mucho del estado en que se encuentren. Cuando éste cambia, el programador puede desear grabar, o hacer que persista dicho cambio. Estas modificaciones pueden producirse como consecuencia de alguna acción, ya sea durante el tiempo de ejecución o a lo largo de la programación. Los Beans se memorizan de tres maneras distintas:

- Automáticamente, mediante el mecanismo de procesamiento en serie de Java.
- A través de un dispositivo personalizado de exteriorización de corriente, que proporciona al objeto Beans un control absoluto sobre la escritura de su estado, incluida la posibilidad de imitar formatos arbitrarios existentes de datos.
- Mediante un mecanismo de serialización XML, distinto del mecanismo estándar que es generalmente para almacenamientos temporales durante la ejecución de una aplicación.

Normalmente, los Beans memorizan las partes de su estado interno que se van a usar para definirlos al recrearlos. Eso es muy típico en ellos. Un Bean que dé referencias de otros puede querer conservarlas, pero esta operación es intrínsecamente peligrosa porque da por sentado que los Beans referidos están grabados. En lugar de ello, hay que reconstruir las referencias de otros Beans durante el proceso de recreación. De esta forma, los Beans podrán marcar las variables como `transient` para dar a entender que la referencia es volátil y se reconstruirá si hace falta.

El taller de programación de Beans (BDK)

El BDK es una herramienta proporcionada gratuitamente por Sun para probar los componentes de JavaBean, que, sin embargo, no se utiliza demasiado al contar con otras herramientas más avanzadas, tal como JBuilder y Forte. De todos modos, BDK nos ofrece la posibilidad de examinar los componentes reutilizables que siguen el modelo de eventos de Bean.

Sólo depende de la versión 1.2 del JDK y está disponible tanto para Solaris Unix como para Windows 95 y NT. El BDK no es un producto de carácter comercial, sino que sirve para iniciarse en el conocimiento de los Beans: cómo se construyen y cómo se aplican a través de una sencilla y manejable aplicación de constructor.

Nota

La nueva aplicación BeanBuilder reemplaza en gran medida a BDK, aunque de momento la primera se emplea más para demostrar algunas de las nuevas

funciones de Java 1.4 tal como la de guardar el estado de los Bean con XML y con el gestor `SpringLayout`. Para descargar esta aplicación, visite la página http://java.sun.com/products/javabeans/beanbuilder/.

Instalación del BDK

El BDK es un sistema completo que contiene código fuente para todas las aplicaciones, ejemplos y documentación. También incluye un constructor de Bean de muestra y una aplicación personalizada que se denomina BeanBox.

- Las versiones de BDK vienen determinadas por el mes y año de su lanzamiento. Para Solaris Unix, el BDK se desempaqueta ejecutando el archivo contenedor % sh bdk1_1-solsparc.bin.

  Para Windows 95, 98, Me, 2000 y NT, se emplea un Win32 EXE (bdk1_1-win.exe). Para Windows, el BDK se instala copiando el fichero bdk1_1-win.exe en una carpeta y haciendo doble clic sobre él.

- Para otros sistemas, el procedimiento se realiza a través de un archivo ZIP (bdk1_1.zip). En todo caso, le aconsejamos que siga las instrucciones que van apareciendo al instalar el BDK.

Advertencia

No instale el BDK en un directorio que incluya un espacio en su nombre de archivo, lo que quiere decir que no debe instalarse en el directorio que aparece por defecto.

En todos estos procesos se le pide al usuario que acepte las condiciones del acuerdo de licencia de distribución y, a continuación cree un directorio y el lugar de almacenamiento desde donde se va a ejecutar Java 1.2+. Se crearán los siguientes subdirectorios:

- `beanbox`: Contiene BeanBox, la herramienta del constructor de Beans de muestra, y el código fuente.
- `demo`: Contiene los Bean de muestra del BeanBox.
- `doc`: Contiene ficheros HTML que describen el Bean API, el BeanBox y los ejemplos.
- `jars`: Lleva incluidos los ficheros JAR de los Bean de muestra que la aplicación debe leer y cargar.
- `lib`: Contiene la biblioteca utilizada por uno de los Bean de muestra.
- `UninstallerData`: Proporciona información para desinstalar el BDK.

Para que funcione el BDK, debemos ejecutar `run.bat` o `run.sh` correspondientes al directorio `beanbox`.

Uso del BDK BeanBox

La aplicación BDK BeanBox (situada en el directorio beanbox, bajo el de instalación de Bean) es un sencillo contenedor de pruebas que le permitirá realizar las siguientes funciones:

- Descargar Beans en una paleta.
- Seleccionar Beans desde la paleta.
- Desplazar los Bean a una ventana de composición.
- Mover y cambiar el tamaño de los Bean.
- Modificar los Bean mediante hojas de propiedades.
- Personalizar los Bean con la aplicación correspondiente.
- Conectar los Bean entre sí.
- Grabar los Beans durante el procesamiento en serie.
- Programar *Applet* a partir de las conexiones existentes.
- Restaurar los Bean.

La figura 21.3 muestra la ventana de composición de BeanBox, (centro), la paleta (izquierda), la ventana **Properties** (arriba a la derecha), y la ventana **Method Tracer** (abajo a la derecha). El menú **ToolBox** muestra los Bean que están listos para ser depositados en la ventana de composición. Cuando se selecciona uno de ellos, la ventana **Properties** nos presenta las propiedades del Bean que está en condiciones de ser editado.

Para situar un Bean en la ventana de composición, seleccione el Bean mediante el icono correspondiente o el nombre del mismo, que aparece en la paleta **ToolBox**, y luego pinche con el ratón en el lugar donde quiera colocar el centro del Bean. Éste se arrastrará hasta la ventana de composición dentro de un cuadro discontinuo en blanco y negro, como se muestra en la figura 21.4.

El menú **File** de la ventana de composición de BeanBox sirve para guardar, borrar o salir de la composición en curso. El submenú **Load** se utiliza para restablecer el estado de una composición grabada. Las opciones del submenú **LoadJar** le permiten ampliar la paleta del componente. La opción **MakeApplet** sirve para convertir los contenidos actuales del BeanBox en un applet.

El menú **Edit** se emplea para cambiar el Bean seleccionado en ese momento (o la ventana de composición en caso de que no haya ninguno). En él, se incluyen las opciones **Cut**, **Copy** y **Paste** para aplicarlas al Bean seleccionado. Según las características de éste, el menú **Edit** puede también ofrecer la opción **Events**, que le permitirá conectar los Bean entre sí.

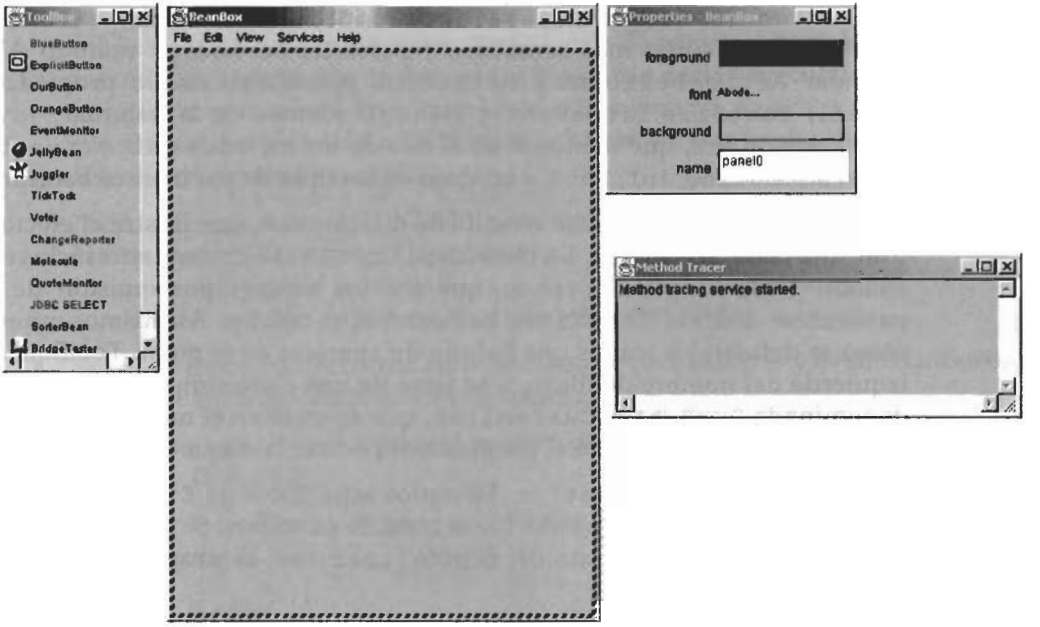


Figura 21.3. La aplicación BeanBox

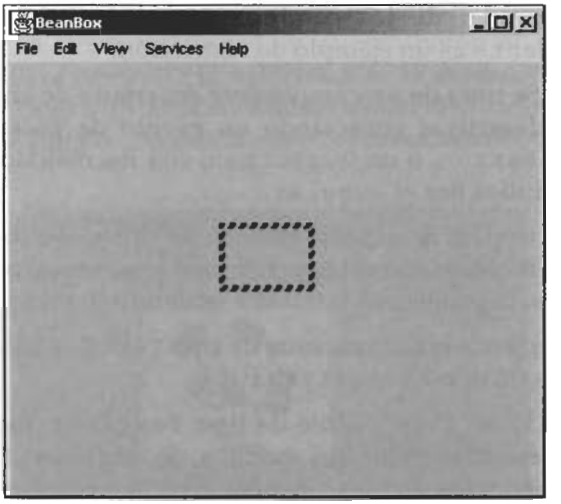


Figura 21.4. Un JellyBean en la ventana de BeanBox

Los Beans de muestra

El BeanBox contiene un total de 16 Bean de muestra. Cada uno de ellos tiene una función diferente y ofrece un ejemplo de distintos aspectos del JavaBean API

- `OurButton`: Subclase de un `java.awt.Component`. Este Bean es el componente GUI de botón más sencillo. Al pulsarlo, enviará un método AWT estándar `ActionPerformed`. `OurButton` ejemplifica cuatro propiedades (`label`, `fontSize`,`largeFont` y `debug`), además de la habitual `java.awt.Component`, que demuestran el uso de los métodos `getter/setter` de la clase `String`, `int`, `Font`, y también de los tipos de parámetros booleanos.
- `ExplicitButton`: Subclase sencilla de `OurButton`, que ilustra el efecto de usar una clase `BeanInfo`. La clase `ExplicitButtonBeanInfo` incluye un método `PropertyDescriptor` que fija los valores por omisión de los parámetros `label`, `fontSize`, `largeFont` y `debug`. Asimismo, muestra cómo se definen los iconos que habrán de aparecer en el menú **ToolBox**, a la izquierda del nombre del Bean, y se sirve de una clase simple `customizer`, denominada `OurButtonCustomizer`, que aparece en el menú **Edit** al seleccionar este Bean, y permite al programador editar la etiqueta del botón.
- `OrangeButton` y `BlueButton`: Ejemplos específicos de `ExplicitButton` con diferentes colores de fondo. No se trata de subclases. Se graban (procesan en serie) y restauran desde un fichero `.ser` con la propiedad de fondo establecida.
- `JellyBean`: Un sencillo componente visual que dibuja un pequeño gráfico ovalado de color. Este Bean ilustra el uso de las propiedades dependientes y restrictivas. La propiedad de `color` es un componente vinculado e informa al `JellyBean` de los cambios que experimenta. La propiedad de `priceInCents` es un ejemplo de restricción.
- `Juggler`: Se trata de un componente ensartado de animación, que se puede activar y desactivar conectando un evento de pulsación de botón de un `ExplicitButton`, o un `OurButton`, con los métodos de eventos `start` y `stop` admitidos por el `Juggler`.
- `Voter`: Sirve para manipular eventos de tipo `vetoableChange`. Por defecto, el Bean rehusará todas las peticiones de cambios, pero estas solicitudes se aceptarán si la propiedad `vetoAll` se determina como `false`.
- `ChangeReporter`: Componente de tipo `TextField` que sirve para mostrar los eventos de tipo `PropertyChange`.
- `EventMonitor`: Componente de tipo `TextArea` que se puede programar para que escuche todos los eventos de un Bean determinado. Controla dinámicamente los sucesos y genera el código correspondiente.
- `SorterBean`: Applet en serie que acepta "`BubbleSort`", "`QSort`" o "`BidirBubbleSort`" como posibles algoritmos de clasificación.
- `Molecule`: Basado en un concepto similar al de `Juggler`. El Bean representa una molécula en 3 dimensiones y acepta el uso del ratón para hacerla girar. Esto mismo también se puede hacer asociando botones a los métodos `rotateX()` y `rotateY()`.

- `QuoteMonitor`: Utiliza la Invocación de Métodos Remotos (RMI) para comunicarse con un servidor remoto (o local) de presupuestos. Éste se activa pasando los repertorios de ficheros al directorio `demo` y ejecutando `gnumake -f quote,gmk run &` en Solaris Unix, o bien `start nmake -f quote,mk run` desde una llamada a DOS en Win32. Se da por hecho que el sistema dispone de `gnumake` o `nmake`.
- `JDBC Select`: Usa el JDBC API para conectarse a una base de datos de un determinado URL y genera una sentencia de selección. Es bastante complejo y requiere su propio `customizer` para configurar la cadena JDBC URL, el nombre del usuario de la base de datos y la contraseña.
- `BridgeTester`: Ofrece una serie de tipos de propiedades y eventos idóneos para probar otros componentes del Bean.
- `TransitionalBean`: Aplica el modelo de eventos JDK 1.0.2 y funciona con JDK 1.1 y JDK 1.0.2.
- `TickTock`: Un Bean no visual que manda `PropertyChangeEvents` a intervalos regulares.

Conexión de Beans

Los Beans se pueden conectar visualmente en BeanBox. La figura 21.5 muestra el Bean `Juggler` (el malabarista de golosinas) con otros dos Beans de tipo `ExplicitButton`. El siguiente paso consiste en conectar uno de los botones al método de eventos `startJuggling()`, y el otro botón al método de eventos `stopJuggling()`. Para controlar la velocidad de la animación, conectaremos la propiedad `priceInCents` (precio en céntimos) del `JellyBean` a la propiedad `animationRate` del Juggler. Al modificar el primero, aumentaremos o disminuiremos la velocidad.

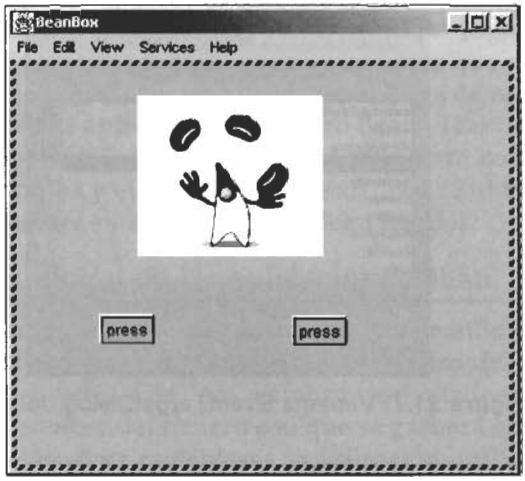


Figura 21.5. La aplicación Juggler

En este caso concreto, debemos seleccionar uno de los `ExplicitButton`, seleccionar Events del menú Edit y los submenús button push y actionPerformed, como se indica en la figura 21.6.

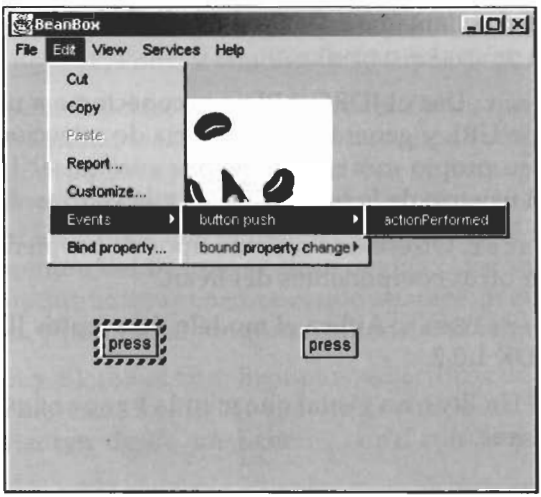


Figura 21.6. Selección del evento actionPerformed

Tras seleccionar el evento actionPerformed, la aplicación traza una línea desde el Bean de tipo `ExplicitButton`. Esta raya muestra la fuente del evento. El destinatario del mismo (oyente) se selecciona pinchando en el Bean al que deberá dirigirse el emisor. En este caso concreto, el Bean Juggler va a recibir un evento de tipo `actionPerformed`. Este elemento consta de dos métodos para captarlo, `startJuggling()` y `stopJuggling()`, que aparecen situados en la ventana EventTargetDialog (figura 21.7) junto con otros que no aceptan ningún parámetro.

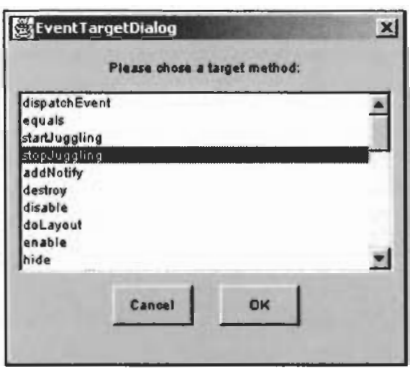


Figura 21.7. Ventana EventTargetDialog del Juggler

Cuando se selecciona el método `stopJuggling()`, se conectará el evento `actionPerformed` de uno de los botones con el método `stopJuggling()` del

Bean Juggler. Ahora ya están ambos comunicados. Si se pincha el botón con el ratón, el evento se transmite al Bean Juggler, y éste parará la animación (que se inició automáticamente).

Para conectar el otro botón con el método `startJuggling()`, se repite el mismo proceso, seleccionándose este último. Para conectar el precio del `JellyBean` (la golosina) con la velocidad de la animación, seleccione el componente del `JellyBean`, y elija la opción **Bind Property** del menú **Edit** para que aparezca el cuadro de diálogo **PropertyNameDialog** de la figura 21.8. Para conectar el precio del `JellyBean`, seleccione la opción **priceInCents** y pulse **OK**. Aparecerá otro cuadro de líneas discontinuas alrededor del botón. Pulse sobre el malabarista y se abrirá otro cuadro de diálogo **PropertyNameDialog**. Esta vez, en lugar de mostrar una lista con las propiedades del Juggler, contiene las propiedades con los mismos tipos de datos que la propiedad fuente. Puesto que `priceInCents` es un `int`, sólo podemos conectar la propiedad `animationRate` del mismo tipo. Pulse **OK**.

Figura 21.8. Ventana PropertyNameDialog del JellyBean

Tras realizar este proceso, la velocidad de la animación habrá aumentado. El precio por defecto de la golosina es "2", por lo que el Juggler tiene esta misma velocidad de animación. Para modificarla, sólo tenemos que cambiar el precio desde la ventana **Properties** del `JellyBean`. Al disponer ya de un oyente conectado, el cambio de la propiedad se aplica también al otro Bean. Tras unas cuantas pulsaciones de ratón, la aplicación funciona perfectamente. Para completar la operación, abra la ventana **Properties** y etiquete los dos `ExplicitButtons` **Start** y **Stop**. De esta forma, se determinará su efecto sobre el Bean Juggler.

Grabación y restauración de las aplicaciones Bean

Se pueden guardar las aplicaciones creadas con el BeanBox en su estado presente (tanto si están activas como detenidas, como en el caso del Juggler) seleccionando la opción **Save** del menú **File**. Así, se abrirá un cuadro de diálogo donde se puede determinar el nombre del fichero con que se grabará el grupo de Beans de la ventana de composición. Para restablecer la aplicación, utilice la opción **Load** del menú **File**.

Nota

El BeanBox no pretende ser un herramienta de programación de Bean de tipo comercial. Simplemente, ofrece un entorno de experimentación a los programadores de Java. Si busca un producto más profesional, tiene a su disposición el JBuilder de Borland, VisualAge para Java de IBM y VisualCafé de Symantec.

Advertencia

Las opciones Clear y Load del menú File eliminarán el grupo de Beans de la ventana de composición sin previo aviso, aunque no se hayan grabado.

Advertencia

La opción Save del menú File no le preguntará si quiere sobrescribir las aplicaciones Bean existentes.

Cómo crear un nuevo Bean

El BDK también proporciona un método para añadir nuevos Beans al BeanBox y probarlos. En el ejemplo siguiente, el nuevo Bean va a funcionar como si fuera un semáforo de tráfico. Por tanto, se representa en forma de rectángulo con tres discos de colores: rojo, ámbar y verde (figura 21.9). Estas señales simbolizan las mismas órdenes que en un semáforo real: parada obligatoria, precaución y franquía de paso. El simulador reaccionará a las señales del botón **Walk** para oscilar entre los dos estados opuestos (rojo o verde, ya que el ámbar indica una transición entre ellos), de forma que los peatones puedan cruzar la calle.

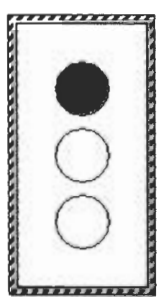


Figura 21.9. El Bean TrafficLight

Éste es un componente visual, por lo que se debe crear como una subclase de un componente Swing. Además de responder a las indicaciones del botón **Walk**, ten-

drá que comunicarse con un segundo semáforo auxiliar que aparecerá en un cruce de calles.

> **Nota**
>
> JBuilder podría generar gran parte del código necesario para el Bean TrafficLight de manera automática. Sólo nos restaría añadir el código de dibujo así como cualquier otra propiedad lógica distinta de los métodos `getter` y `setter`, y los eventos de registro y notificación.

El Bean TrafficLight

Para crear este Bean debemos crear un directorio para el componente que llamaremos `jlight` y que situaremos debajo del de instalación del BDK.

El Bean TrafficLight dispone de tres subclases de `javax.swing.JComponent`:

- `defaultLightState`, que es un `String` que representa el color inicial del disco en su estado de descanso.
- `currentLightState`, que representa el color actual del disco.
- `boolean debug`, que se puede encender y apagar a través del editor de propiedades para comprobar el resultado de los eventos (mientras se reciben y transmiten desde el TrafficLight)

El listado 21.2 incluye el código del componente TrafficLight.

Listado 21.2. El componente TrafficLight

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
import java.util.*;
public class TrafficLight extends JComponent {
  public static String STATE_RED    = "RED";
  public static String STATE_YELLOW = "YELLOW";
  public static String STATE_GREEN  = "GREEN";
  private static int DELAY = 3000;
  private String defaultLightState;
  private String currentLightState;
  private boolean debug;
  private Thread runner;
  private transient ActionListener listener;
  public TrafficLight() {
    defaultLightState = currentLightState = STATE_RED;
    setPreferredSize(new Dimension(100, 200));
  }
  public void paintComponent(Graphics g) {
    super.paintComponent(g);
```

```
    // Crea un borde negro alrededor
    g.setColor(Color.black);
    g.drawRect(1, 1, 98, 198);
    // Crea el borde del semáforo
    g.setColor(Color.white);
    g.fillRect(2, 2, 96, 196);
    // Crea el borde de las luces
    g.setColor(Color.black);
    g.drawOval(30, 30, 40, 40);
    g.drawOval(30, 80, 40, 40);
    g.drawOval(30, 130, 40, 40);
    // Limpia el código
    if (debug) {
      System.out.println("Current light state is: " + currentLightState);
    }
    // ¿Qué luz está emcendida?
    if (currentLightState.equals(STATE_RED)) {
      g.setColor(Color.red);
      g.fillOval(30, 30, 40, 40);
    } else if (currentLightState.equals(STATE_YELLOW)) {
      g.setColor(Color.yellow);
      g.fillOval(30, 80, 40, 40);
    } else if (currentLightState.equals(STATE_GREEN)) {
      g.setColor(Color.green);
      g.fillOval(30, 130, 40, 40);
    }
  }
  public String getCurrentLightState() {
    return currentLightState;
  }
  public void setCurrentLightState(String currentLightState) {
    this.currentLightState = currentLightState;
    if (currentLightState != STATE_YELLOW) {
      if (debug) {
        System.out.println("Firing action event");
      }
      ActionEvent ae = new ActionEvent(
        this, ActionEvent.ACTION_PERFORMED, currentLightState);
      fireActionPerformed(ae);
    }
    repaint();
  }
  public void setDefaultLightState(String defaultLightState) {
    this.defaultLightState = defaultLightState;
    currentLightState = defaultLightState;
    repaint();
  }
  public String getDefaultLightState() {
    return defaultLightState;
  }
  public void setDebug(boolean debug) {
    this.debug = debug;
  }
  public boolean isDebug() {
    return debug;
  }
```

```
// Envía un mensaje a la luz para comenzar el ciclo
public void initiate() {
  // Debug
  if (debug) {
    System.out.println("Initiate traffic light cycle!");
  }
  startCycle();
}
// Comienza el ciclo de luces
private void startCycle() {
  if (runner == null) {
    Runnable runnable = new Runnable() {
      public void run() {
        if (debug) {
          System.out.println("Started cycle");
        }
        while (runner != null) {
          try {
            Thread.sleep(DELAY);
          } catch (InterruptedException e) {
          }
          // Obtiene el estado actual de la luz
          if (currentLightState.equals(STATE_RED)) {
            // Pasa a verde
            setCurrentLightState(STATE_GREEN);
          } else if (currentLightState.equals(STATE_GREEN)) {
            // Pasa de ambar a rojo
            setCurrentLightState(STATE_YELLOW);
          } else {
            // De lo contrario, está en ambar,
            // Pasa a rojo
            setCurrentLightState(STATE_RED);
          }
          // Cambia el ciclo en el estado por defecto
          if (currentLightState.equals(defaultLightState)) {
            runner = null;
          }
        }
      }
    };
    runner = new Thread(runnable);
    runner.start();
  }
}
// Obtiene un evento que indica un cambio de color
// En cuyo caso, se actúa como semáforo secundario
public void lightChange(ActionEvent x) {
  // Comprueba el estado de este evento
  String command = x.getActionCommand();
  // Limpia el código
  if (debug) {
    System.out.println("Received event from traffic light: "
      + defaultLightState + " command: go to " + command);
  }
  // Si el otro semáforo está en rojo, comprueba el valor por defecto
  if (command.equals(STATE_RED)) {
```

```
      if (!currentLightState.equals(STATE_GREEN)) {
        // Ciclo
        currentLightState = STATE_GREEN;
        repaint();
      }
    } else if (command.equals(STATE_GREEN)) {
      if (!currentLightState.equals(STATE_RED)) {
        // Ciclo
        currentLightState = STATE_YELLOW;
        repaint();
        try {
          Thread.sleep(DELAY);
        } catch (InterruptedException e) {
        }
        currentLightState = STATE_RED;
        repaint();
      }
    }
  }
  public synchronized void removeActionListener(ActionListener l) {
    // Limpia el código
    if (debug) {
      System.out.println("Deregistering listener");
    }
    if (listener == l) {
      listener = null;
    }
  }
  public synchronized void addActionListener(ActionListener l)
      throws TooManyListenersException {
    // Limpia el código
    if (debug) {
      System.out.println("Registering listener");
    }
    if (listener == null) {
      listener = l;
    } else {
      throw new TooManyListenersException();
    }
  }
  protected void fireActionPerformed(ActionEvent e) {
    // Limpia el código
    if (debug) {
      System.out.println("Firing action event");
    }
    if (listener != null) {
      listener.actionPerformed(e);
    }
  }
}
```

El componente TrafficLight usa un par de métodos `set/get` para mostrar las propiedades `debugging`, `defaultLightState` y `currentLightState`. El método `set` de la propiedad `currentLightState` consiste en algo más que en notificar y registrar la acción del listener cuando sea adecuado. El `paintComponent()`,

pinta un rectángulo blanco y tres círculos vacíos para representar los discos rojo, ámbar y verde del Bean TrafficLight.

Los métodos públicos sin argumentos o que toman un argumento de tipo `ActionEvent` pueden conectarse a un evento de tipo `actionPeformed` (como en el ejemplo del Bean Juggler que ya vimos en el apartado anterior). El Bean TrafficLight aplica dos de ellos: uno de tipo `initiate()` para llamar a un método privado de tipo `startCycle()`, que a su vez crea un thread e inicia el ciclo de conmutaciones del TrafficLight, desde su estado actual hasta el opuesto, y así una y otra vez. Este método está especialmente ideado para su combinación con el botón **Walk**.

El método `lightChange()` sirve para añadir un semáforo auxiliar conectando dos Bean de tipo TrafficLight. Si un disco está en estado cíclico, al otro se le comunica con un evento que la luz debe cambiar. Por ejemplo, si el primer disco pasa del verde al rojo, el segundo (en el otro cruce) lo hará a la inversa.

El método `fireActionPerformed()` se usa para notificar a otro Bean de tipo TrafficLight que ha sido registrado como `ActionListener`. Este es un buen ejemplo de fuente de eventos de tipo monodifusión y por tanto, no admite más de un cruce de calles.

El BeanInfo de TrafficLight

A modo de experimento instructivo, vamos a configurar un editor especial de propiedades para el color del disco. Como sólo puede tener tres valores (las constantes `STATE_RED`, `STATE_YELLOW` y `STATE_GREEN`), nos basta sólo con incluirlos en una matriz y devolverla desde el método `getTags()` correspondiente a una subclase de `PropertyEditorSupport`. De esa forma, generamos lo que se llama un editor de propiedades, puesto que nos permite especializar la edición de las propiedades del Bean.

```
import java.beans.*;
public class LightColorEditor extends PropertyEditorSupport {
  public String[] getTags() {
    String values[] = {
      TrafficLight.STATE_RED,
      TrafficLight.STATE_YELLOW,
      TrafficLight.STATE_GREEN};
    return values;
  }
}
```

Para usar el editor de propiedades, tenemos que dotar al Bean TrafficLight de una clase `BeanInfo`. Eso significa que hay que configurar el descriptor de propiedades para la propiedad `defaultLightState` del Bean.

```
import java.beans.*;
public class TrafficLightBeanInfo extends SimpleBeanInfo {
  public PropertyDescriptor[] getPropertyDescriptors() {
    try {
      PropertyDescriptor pd1 =
        new PropertyDescriptor("debug", TrafficLight.class);
```

```
            PropertyDescriptor pd2 =
              new PropertyDescriptor("defaultLightState", TrafficLight.class);
            pd2.setPropertyEditorClass(LightColorEditor.class);
            PropertyDescriptor result[] = { pd1, pd2 };
            return result;
        } catch (IntrospectionException e) {
            System.err.println("Unexpected exception: " + e);
            return null;
        }
    }
}
```

Como podemos observar, ni el editor de propiedades ni el BeanInfo alteran el Bean TrafficLight original. Estas clases son imprescindibles a la hora de programar cuando se diseña el Bean utilizando una herramienta de constructor. Una vez que ha sido creada la aplicación final, no hace falta ya ponerlas a disposición del usuario.

Compilación del Bean TrafficLight y el BeanInfo

El BDK contiene un grupo de ficheros de creación que pueden modificarse para crear un archivo JAR e incluir así el semáforo del ejemplo dentro del BeanBox. Para compilarlos, tenemos que ejecutar el compilador del siguiente modo:

```
javac TrafficLightBeanInfo.java LightColorEditor.java TrafficLight.java
```

Nota

El compilador no es capaz de compilar sólo `TrafficLight.java` y obtener la compilación de los otros dos archivos, ya que no depende de ninguno de ellos.

Para que el BeanBox (o cualquier otro constructor de Beans) reconozca el específico del archivo JAR, debemos identificarlo. Para ello, debemos crear un archivo manifiesto y añadírselo al archivo JAR.

Como ya se ha visto, un archivo manifiesto es un archivo de texto que describe las clases. Para que reconocer alguna de esas clases como Bean, debemos añadir la siguiente línea:

```
Java-Bean: True
```

Puesto que sólo hay una clase Bean en TrafficLight, el archivo (manifest.tmp) será del siguiente modo:

```
Manifest-Version: 1.0
Name: TrafficLight.class
Java-Bean: True
```

Asegúrese de incluir un retorno de carro al final de la última línea o se ignorará.

Una vez tenemos el manifiesto, ya podemos crear el archivo JAR:

```
jar cfm Traffic.jar manifest.tmp *.class
```

e incluirlo en el BeanBox.

Cómo incluir el nuevo Bean en el BeanBox

De regreso en el BeanBox, elimine el Juggler mediante la opción **Clear** del menú **File**. Para abrir el TrafficLight en la ventana, seleccione **LoadJar**. Busque el directorio tlight, seleccione TRaffic.jar y haga clic en **Open**. La caja de herramientas o **Toolbox**, debería incluir ahora la opción **TrafficLight**, como se muestra en la figura 21.10.

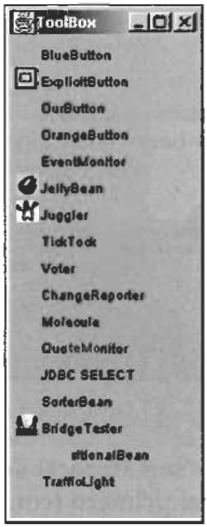


Figura 21.10. El menú Toolbox con el Bean TrafficLight

Prueba del nuevo Bean

Para probar TrafficLight, se colocan dos muestras del nuevo Bean y uno de un `ExplicitButton` (para la señal de *Walk*) en la ventana de composición, como se ve en la figura 21.11.

Mediante la opción **Event** del menú **Edit**, el evento `actionPerformed` del `ExplicitButton` se conecta al método de eventos `initiate()` correspondiente al primer semáforo. Cuando el usuario pulse el botón del ratón, el semáforo pasará periódicamente del verde al ámbar y de éste al rojo (con un intervalo de tres segundos entre dichos cambios).

Los dos Beans TrafficLight están en estado `RED` por defecto, por lo que el `defaultLightState` del segundo Bean pasa al `GREEN` si seleccionamos el valor correspondiente de la ventana **Properties** (figura 21.12).

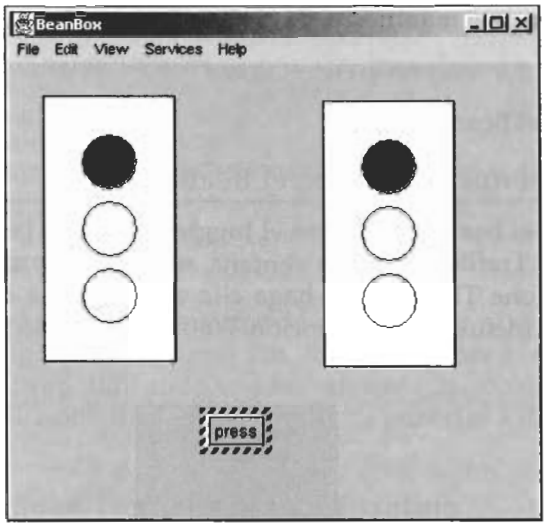


Figura 21.11. Dos muestras del bean TrafficLight y un ExplicitButton en la ventana de composición

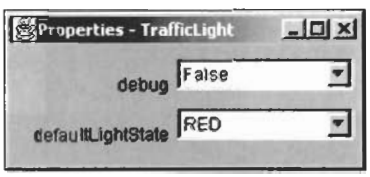


Figura 21.12. El cuadro de diálogo Properties

Luego se conecta el primer semáforo al segundo, a través de la conexión del método `actionPerformed` del primero (con el método `lightChange()` de este último). Esto se hace repitiendo el mismo proceso que para el `ExplicitButton`. Por último, la etiqueta del `ExplicitButton` se sustituye por *Walk* (figura 21.13)

Al pulsar el botón **Walk** con el ratón, el primer semáforo envía una señal al segundo para que cambie, y así se inicia su propio ciclo. El segundo pasa del verde al ámbar y luego al rojo, mientras el primero lo hace del rojo al verde, después al ámbar, y vuelta a empezar.

Resumen

Aunque BDK es de gran utilidad para realizar algunas operaciones que no son aún posibles con las herramientas de desarrollo en el mercado, lo cierto es que probablemente dejará de utilizarse por completo en breve, ya que cada herramienta conecta Beans de manera diferente.

A continuación encontrará algunas de las herramientas más importantes del mercado:

- JBuilder, de Borland (http://www.borland.com/jbuilder).
- VisualAge para Java de IBM (http://www.software.ibm.com/ad/vajava).
- Code Warrior para Java de Metrowerks (http://www.metrowerks.com/desktop/java).
- Forte para Java/NetBeans de Sun Microsystems (http://www.sun.com/forte/ffj).
- VisualCafé de WebGain (http://www.webgain.com/products/visual_cafe).

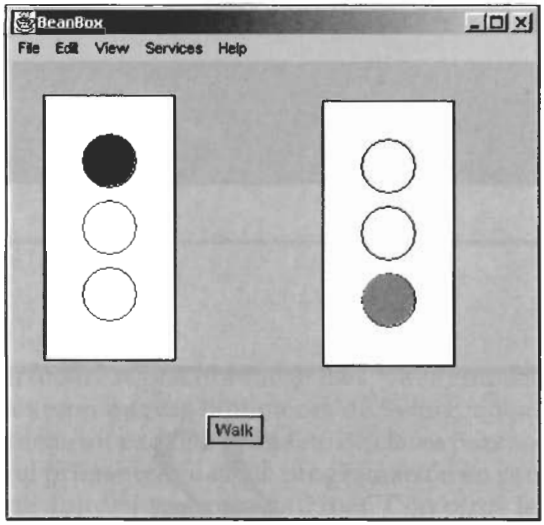
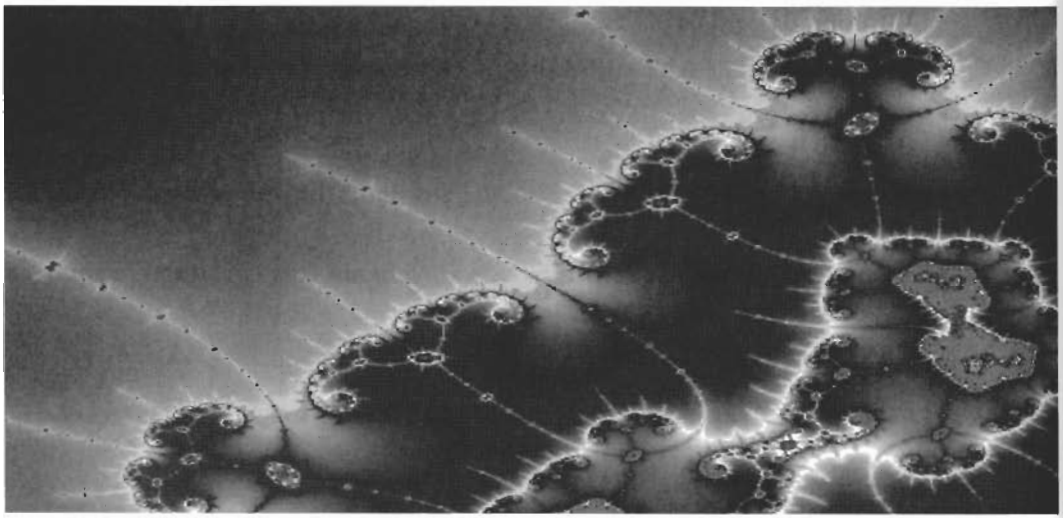


Figura 21.13. La aplicación TrafficLight completa

Tanto Oracle9i JDeveloper (http://technet.oracle.com/products/jdev/) como IntelliJ IDEA (http://www.intellij.com/idea/) son cada vez más importantes.

La elección depende de la forma de trabajo que emplee y el ambiente en el que desee integrarlo. VisualAge, por ejemplo, funciona bien con WebSphere y otras herramientas de IBM.

Muchos programadores utilizan únicamente un editor (emacs) y un compilador de líneas de comando, que es suficiente para código sencillo, pero si deseamos crear ventanas, es indispensable utilizar alguna de las herramientas mencionadas. Además, muchas de ellas tienen una versión de prueba gratuita, por lo que es recomendable que las utilice todas para encontrar la que más se adecua a su perfil. Recuerde que el CD que acompaña al libro incluye una versión gratuita de JBuilder.



22
Programación en Red

Antes de que se desarrollara la plataforma de Java, ya se empleaban las interfaces GUI, por lo que se incluyeron nuevas bibliotecas de Swing, además de AWT. Esta plataforma también contiene un paquete completo de clases para soporte de Internet.

Java fue en realidad el primer lenguaje de programación en proporcionar soporte para programación de aplicaciones para Internet. Con otros lenguajes, la única manera de crearlas era profundizando en los tecnicismos de los sistemas operativos y en las dependencias de API. Con Java, sin embargo, esta operación es muy sencilla y no implica investigar los volúmenes de referencias del soporte de red del sistema.

Este capítulo describe los protocolos más importantes y las clases incluidas en el paquete `java.net`. Los ejemplos que figuran en el mismo le enseñarán a conectarse a servidores remotos y buscar recursos, así como a elaborar un sistema simple de servidor. También aprenderá las nuevas prestaciones de la biblioteca E/S, las normas de diseño de `Factory` y las interfaces correspondientes, incorporadas en el paquete `java.net`.

Protocolos de Java: TCP/IP

TCP/IP (*Transmission Control Protocol/Internet Protocol* o Protocolo de control de transmisiones y Protocolo de Internet), los dos protocolos de comunicación de datos en los que se basa Internet. SMPT (*Simple Mail Transfer Protocol* o Protocolo de

transferencia simple de correo), y NNTP (*Network News Transfer Protocol* o Protocolo de transferencias de noticias en Internet) son dos de los antiguos protocolos que aún se emplean. El nuevo protocolo HTTP (*Hypertext Transfer Protocol* o Protocolo de transferencia de hipertexto), que define la World Wide Web, se ha convertido en parte fundamental de Internet y, por lo tanto de TCP/IP, hasta tal punto que mucha gente confunde la Red con Internet.

IP: El corazón de las comunicaciones de datos en Internet

No importa qué protocolo se emplea para implementar un servicio de Internet, IP es la base fundamental. IP es un protocolo que no garantiza la recepción de paquetes de información enviados. Los paquetes IP no forman parte de un flujo de paquetes relacionados, sino que se transmiten individualmente. La figura 22.1 muestra la estructura de un paquete IP.

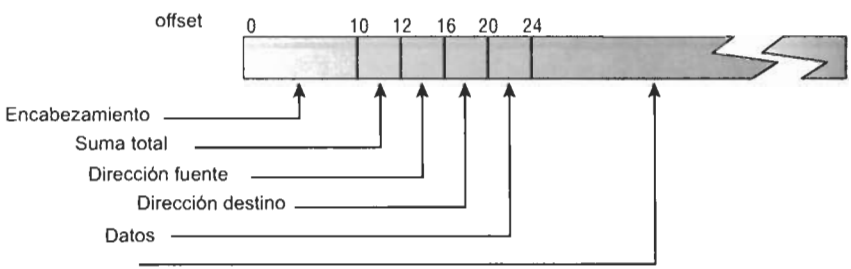


Figura 22.1. Formato de un paquete IP

Los distintos campos al principio de un paquete (llamado también marco) se conocen como Encabezamiento del marco y determinan la funcionalidad del protocolo IP además de sus limitaciones, incluyendo las estructuras utilizadas para crear el código (remitente) y el receptor. En la versión 4 del Protocolo de Internet (IPv4), cada uno de estos campos contiene 23 bits, lo que significa que el total permitido de equipos conectados a la red global es de 2 elevado a 32 (4.294.967.296).

Nota

La versión 6 (IPv6) proporciona 128 bits para cada dirección. Aunque la cantidad anterior parece más que suficiente, lo cierto es que casi se ha llegado al límite permitido. Java 1.4 proporciona soporte para ambas versiones de este protocolo. Para más información sobre la arquitectura de IPv6, visite la página www.ietf.org/rfc/rfc2373.txt.

En lugar de escribirse con cadenas de bits de 32 dígitos, tal como 110011101 10000110001011111010000, las direcciones de Internet casi siempre se expresan en

forma de texto legible (por ejemplo, www.sybex.com). En las contadas ocasiones en las que es necesario expresar la dirección de manera numérica, se hace mediante cuatro expresiones decimales (por ejemplo, 192.31.32.255). El resto del encabezamiento incluye una colección de campos con la longitud del paquete completo en bytes. Para este campo se reservan 16 bits, por lo que un paquete IP no puede exceder los 64KB de longitud.

TCP: Para garantizar la recepción

Puesto que la recepción de los paquetes IP no está asegurada, TCP proporciona un servicio básico que sí la garantiza., y que consiste en utilizar IP como un bloque de construcción. La estructura de un paquete se representa en la figura 22.2.

TCP presenta una cadena de datos conectados. Antes de enviar datos a través de TCP, el sistema necesita conectarse con el sistema receptor. Otra diferencia es que el protocolo TCP envía y recibe cantidades arbitrarias de datos como una gran cadena de datos de bytes. IP, por el contrario, tiene un límite de 65.536 bytes, que resultan insuficientes para la mayoría de los archivos, o páginas con gráficos. TCP soluciona este problema separando la cadena de datos en paquetes IP que numera, para después agruparlos de nuevo en el sistema receptor. Esa es la función de los campos Número de secuencia y Reconocimiento del número de secuencia.

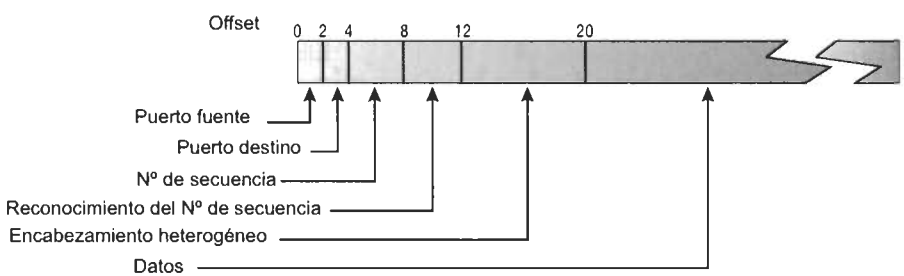


Figura 22.2. Formato de un paquete TCP

Los campos más importantes, desde el punto de vista del usuario, son los correspondientes a los puertos remitentes y receptores. IP permite enviar un paquete IP a un equipo conectado a la red, mientras que TCP no lo permite sin antes haber incluido la dirección del puerto receptor. Los equipos que soportan los protocolos TCP/IP tienen 65.536 puertos TCP a través de los que comunicarse, como se aprecia en la figura 22.3. Se ha definido una gran número de puertos estándar. La tabla 22.1 muestra algunas direcciones de puertos para los servicios de Internet más comunes. Por ejemplo, el puerto 21 se utiliza para transmitir archivos mediante el Protocolo de transferencia de archivos (FTP), mientras que el puerto número 80 se emplea para todo tipo de comunicaciones con los servidores HTML de WWW. En ejemplos posteriores, veremos cómo comunicarnos con el puerto 25 para enviar un correo electrónico, y con el puerto 80, para recibir una página Web.

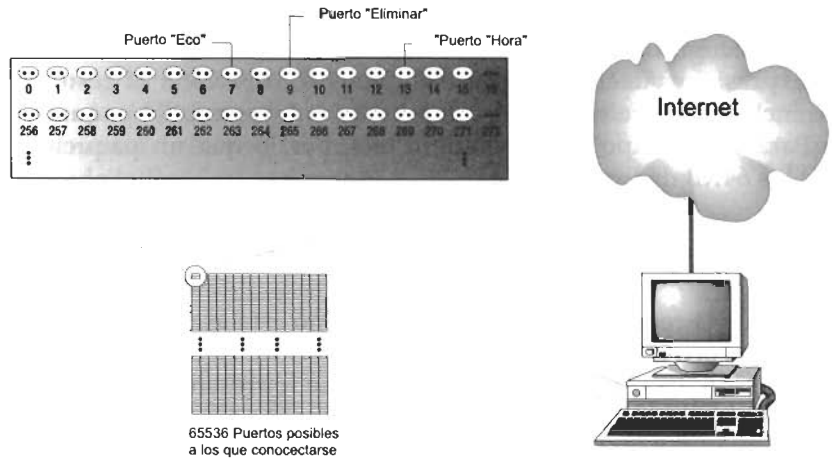


Figura 22.3. Puertos TCP y números de puertos conocidos

Tabla 22.1. Números de puertos TCP estándares

Nombre del puerto	Número del puerto	Descripción
`echo`	7	Repite lo que reciba
`discard`	9	Elimina lo que reciba
`daytime`	13	Produce la hora local del equipo receptor
`qotd`	17	Produce el día en el equipo
`charge`	19	Produce una cadena de caracteres
`ftp-data`	20	Puerto de datos FTP
`ftp`	21	Puerto de control de FTP
`telnet`	23	Puerto del protocolo Telnet
`smtp`	25	Puerto SMTP
`finger`	79	Indica el puerto de protocolo
`http`	80	Puerto del servidor Web
`pop3`	110	Puerto de la versión 3 de POP
`nntp`	119	Puerto NNTP
`https`	443	Puerto del servidor de Web seguro

La mayoría de los protocolos TCP/IP orientado a aplicaciones, tal como el protocolo SMTP de transferencia de correo electrónico, se basan en TCP y no en IP para

su funcionamiento, ya que necesitan garantizar la transmisión sin errores de los datos.

Truco

En el archivo SERVICES de configuración de TCP/IP de su equipo, encontrará más información sobre los números estándar de puertos. En Windows 95, 98 y Me, el archivo se encuentra en C:\WINDOWS\SERVICES, y en NT 4 y 2000, está en el directorio C:\WINNT\system32\drivers\etc. En los sistemas Unix, el archivo está en el directorio /etc. Si accede a este archivo, recuerdo no modificarlo ya que corre el riesgo de deshabilitar la conexión a Internet.

El protocolo UDP (*User Datagram Protocol*) es un protocolo de datagramas con el mismo límite que IP (64KB), aunque permite especificar la dirección del puerto. Cada equipo tiene en realidad dos grupos de 65.536 puertos a través de los que comunicarse: un grupo para TCP y otro para UDP. El protocolo TFTP (*Trivial File Transfer Protocol*), o protocolo de transferencia de archivos triviales, desempeña la misma función que el protocolo FTTP, pero para comunicaciones UDP. TFTP se encarga de solucionar problemas relacionados con pérdidas de paquetes de datos.

La siguiente sección describe las clases del paquete `java.net`.

Gestión de direcciones de Internet

Una de las clases de `java.net` sirve para gestionar cualquier dirección IP de 32 bits (es decir, una dirección anfitrión de Internet) a un nivel más alto que si aplicamos un simple número entero de 32 bits. Principalmente, la clase `InetAddress` le permite convertir una dirección textual de Internet del tipo `anfitrión.subdominio.dominio` en un objeto que la represente.

La tabla 22.2 incluye la definición de esta clase.

Tabla 22.2. Composición de la clase InetAddress

Variables/Métodos	Descripción
`getAllByName()`	Devuelve un `InetAddress` o un grupo de direcciones
`getByAddress()`	
`getByName()`	
`getLocalHost()`	
`equals()`	Busca dos objetos iguales

su funcionamiento, ya que necesitan garantizar la transmisión sin errores de los datos.

Truco

En el archivo SERVICES de configuración de TCP/IP de su equipo, encontrará más información sobre los números estándar de puertos. En Windows 95, 98 y Me, el archivo se encuentra en C:\WINDOWS\SERVICES, y en NT 4 y 2000, está en el directorio C:\WINNT\system32\drivers\etc. En los sistemas Unix, el archivo está en el directorio /etc. Si accede a este archivo, recuerdo no modificarlo ya que corre el riesgo de deshabilitar la conexión a Internet.

El protocolo UDP (*User Datagram Protocol*) es un protocolo de datagramas con el mismo límite que IP (64KB), aunque permite especificar la dirección del puerto. Cada equipo tiene en realidad dos grupos de 65.536 puertos a través de los que comunicarse: un grupo para TCP y otro para UDP. El protocolo TFTP (*Trivial File Transfer Protocol*), o protocolo de transferencia de archivos triviales, desempeña la misma función que el protocolo FTTP, pero para comunicaciones UDP. TFTP se encarga de solucionar problemas relacionados con pérdidas de paquetes de datos.

La siguiente sección describe las clases del paquete `java.net`.

Gestión de direcciones de Internet

Una de las clases de `java.net` sirve para gestionar cualquier dirección IP de 32 bits (es decir, una dirección anfitrión de Internet) a un nivel más alto que si aplicamos un simple número entero de 32 bits. Principalmente, la clase `InetAddress` le permite convertir una dirección textual de Internet del tipo `anfitrión.subdominio.dominio` en un objeto que la represente.

La tabla 22.2 incluye la definición de esta clase.

Tabla 22.2. Composición de la clase InetAddress

Variables/Métodos	Descripción
`getAllByName()`	Devuelve un `InetAddress` o un grupo de direcciones
`getByAddress()`	
`getByName()`	
`getLocalHost()`	
`equals()`	Busca dos objetos iguales

Variables/Métodos	Descripción
`getAddress()`	Devuelve el protocolo IP de la dirección (`byte[ ] {127, 0, 0, 1}`)
`getCanonicalHostName()`	Devuelve el nombre de dominio de la dirección ("www.example.com")
`getHostAddress()`	Devuelve la dirección IP ("127.0.0.1")
`getHostName()`	Devuelve el nombre del anfitrión ("www.example.com")
`hashCode()`	Devuelve el código hash computado
`isAnyLocalAddress()`	Indica el tipo de dirección
`isLinkLocalAddress()`	
`isLoopbackAddress()`	
`isMCGlobal()`	
`isMCLinkLocal()`	
`isMCNodeLocal()`	
`isMCOrgLocal()`	
`isMCSiteLocal()`	
`isMulticastAddress()`	
`isSiteLocalAddress()`	
`toString()`	Convierte la dirección en una cadena

La clase se diferencia de la norma orientada a objetos en que no proporciona ningún constructor y en que, en su lugar, emplea métodos de clase estática, `getByName()`, `getByAddress()` y `getAllByName()`, para crear ejemplos de `InetAddress`. Estos métodos toman como argumento la dirección textual de cualquier anfitrión de Internet, en forma de un `String`. También se puede convertir la dirección de Internet del propio ordenador (`localhost`) en una `InetAddress` introduciendo la secuencia `InetAddress.getLocalHost()`.

Para probar los métodos `InetAddress` y la mayoría de los demás métodos y clases de `java.net`, hay que tener el ordenador conectado a Internet, o lo que es lo mismo, a una red operativa TCP/IP. Ello se debe, entre otras razones, a que las clases de `java.net` tienen que estar en condiciones de realizar búsquedas completas (de nombres de dominio) a través del Sistema de Nombres de Dominio (DNS). Como quizá ya sabe, las direcciones de Internet están codificadas como números enteros de 32 bits dentro de los paquetes IP intercambiados en Internet. Las direcciones mnemónicas del estilo de www.miningco.com o web.mit.edu se deben convertir en direcciones de 32 bits para que se establezca una comunicación real por Internet a través de las mismas.

El Sistema de Nombres de Dominio

¿Cómo se pasa del formato textual al formato numérico? ¿Contiene el ordenador un enorme archivo con todas las máquinas de Internet del mundo y de sus direcciones numéricas? La respuesta a esta segunda pregunta es que ya no es así en la actualidad, aunque eso era exactamente lo que ocurría en los primeros años de vida de Internet, cuando sólo existían un par de cientos de ordenadores en todo el mundo. Por el contrario, ahora, y debido al crecimiento exponencial de huéspedes conectados a Internet, esto sería imposible. En la actualidad, cada máquina configurada para comunicarse con TCP/IP necesita conocer al menos la dirección numérica del servidor DNS, con el que puede comunicarse directamente sin necesidad de convertir el formato textual en una dirección de 32 dígitos. Este servidor es responsable de convertir la dirección textual en el formato numérico, mucho menos comprensible, pero más eficaz.

El sistema DNS es como una guía telefónica global de todas las máquinas huéspedes de Internet. Al introducir una dirección textual, el sistema DNS devuelve la dirección IP del huésped, que sería como su número de teléfono.

Búsqueda de direcciones textuales

El método `getByName()` de la clase `InetAddress` constituye la interfaz transparente del servicio DNS. Cuando se invoca, el ordenador se pone directamente en contacto con el servidor DNS (mediante su dirección numérica) y solicita una búsqueda y devolución de la dirección numérica correspondiente a la textual previamente transmitida a `getByName()`. Si el equipo no se encuentra conectado, este dispositivo de búsqueda fallará, y entonces se lanzará una `UnknownHostException` (es por eso por lo que hay que estar conectado para trabajar con las clases de `java.net`), a no ser que se busque la propia dirección IP, introduciendo el nombre de su ordenador como cadena.

El programa del listado 22.1 demuestra las posibilidades de búsqueda de direcciones de la clase `InetAddress`.

Listado 22.1. Búsqueda de direcciones de un anfitrión

```
import java.net.*;
public class DNSLookup {
  public static void main (String args[]) {
    try {
      InetAddress host;
      if (args.length == 0) {
        host = InetAddress.getLocalHost();
      } else {
        host = InetAddress.getByName(args[0]);
      }
```

```
            System.out.println("Host "" + host.getHostName() +
              "" has address: " + host.getHostAddress());
            byte bytes[] = host.getAddress();
            int fourBytes[] = new int[bytes.length];
            for (int i=0, n=bytes.length; i<n; i++) {
              fourBytes[i] = bytes[i] & 255;
            }
            System.out.println("\t" + fourBytes[0] + "." + fourBytes[1] +
              "." + fourBytes[2] + "." + fourBytes[3]);
          } catch (UnknownHostException e) {
            e.printStackTrace();
          }
        }
      }
```

El programa toma la cadena del nombre del anfitrión como argumento de la línea de mandatos. Si no se especifica el nombre del anfitrión, la búsqueda se ejecutará a través de la dirección de su propio ordenador. He aquí el resultado que se obtiene al introducir el servidor de Sybex como el argumento de la línea de comandos:

```
> java DNSLookup www.sybex.com
Host "www.sybex.com" has address: 63.99.198.12
        63.99.198.12
```

Nota

Al buscar la clase `InetAddress` se devuelve en realidad una subclase, que dependerá de la versión IP de la dirección que se ha transmitido. Se obtendrá una instancia de `Inet4Address` o de `Inet6Address`.

Obtención de direcciones numéricas

Para conseguir una dirección numérica, utilice `getHostAddress()` que a cambio le proporcionará un array de cuatro bytes. Estos bytes van de 0 a 225, mientras que la clase de bytes de Java van de -128 a 127, por lo que es necesario convertir los primeros en datos con signo y almacenarlos en un tipo de datos mayor. Entonces habrá que copiar dichos bytes en una matriz de números enteros mientras se efectúan combinaciones de 255 caracteres para anular la extensión de signo. La dirección numérica se muestra dos veces para indicar qué hacer con los resultados de `getAddress()`.

Observe que la clase `InetAddress` cancela la función `toString()`, con lo que aquí, se genera un `String` con, básicamente, la misma información elaborada manualmente. Al añadir el código `System.out.println(host)` al programa anterior, obtenemos el siguiente resultado:

```
www.sybex.com/63.99.198.12
```

El programa también permite obtener el nombre mediante la dirección numérica:

```
> java DNSLookup 63.99.198.12
Host "www.sybex.com" has address: 63.99.198.12
        63.99.198.12
```

Comunicación con sistemas remotos

Ahora que ya hemos visto cómo se especifica un destino en Internet mediante ejemplos de la clase `InetAddress`, vamos a examinar a continuación la forma de comunicarse con un sistema remoto. El paquete `java.net` nos ofrece varias opciones para conseguirlo.

Comunicación a bajo nivel mediante UDP

El procedimiento más elemental para comunicarse con un sistema remoto es por medio de datagramas UDP. Éstos se hallan incorporados a un ejemplo de la clase `DatagramPacket`, cuya definición se incluye en la tabla 22.3.

Tabla 22.3. Composición de la clase DatagramPacket

Variables/Métodos	Descripción
`DatagramPacket()`	Construye un objeto DatagramPacket
`getAddress()`	Devuelve información sobre el canal de comunicación
`getPort()`	
`getSocketAddress()`	
`getData()`	Devuelve el búfer de datos
`getLength()`	Devuelve la longitud del búfer de datos
`getOffset()`	Devuelve el desajuste del búfer de datos
`setAddress()`	Cambia la dirección de destino
`setSocketAddress()`	
`setData()`	Cambia los datos a transmitir
`setLength()`	Cambia la longitud de los datos a transmitir

La clase `DatagramPacket` proporciona cuatro constructores: dos de recepción de datagramas y otros dos de transmisión de los mismos.

```
public DatagramPacket (byte buf[], int length)
public DatagramPacket (byte buf[], int offset, int length)
public DatagramPacket (byte buf[], int length, InetAddress addr, int port)
public DatagramPacket (byte buf[], int offset, int length, InetAddress addr,
  int port)
public DatagramPacket (byte buf[], int length, SocketAddress addr)
public DatagramPacket (byte buf[], int offset, int length, SocketAddress addr)
```

En ambos casos hay que determinar el búfer de bytes y su longitud (la matriz `buf` puede ser mayor que `length`, pero nunca más pequeña). Además, los constructores de transmisión requieren la máquina de destino y el número de puerto del datagrama (en todo caso, el búfer de bytes contiene el mensaje correspondiente).Un `SocketAddres` no es más que una manera especial de guardar en el mismo paquete el equipo y el puerto.

Como podemos observar, la clase no ofrece los medios para enviar o recibir datagramas. De ello se encarga una clase complementaria, denominada `DatagramSocket`, cuya definición se muestra en la tabla 22.4.

Tabla 22.4. Composición de la clase DatagramSocket

Variables/Métodos	Descripción
`DatagramSocket()`	Construye un objeto `DatagramSocket`
`bind()`	Enlaza la conexión con el destino
`close()`	Cierra la conexión
`connect()`	Realiza la conexión con el destino
`disconnect()`	Desconecta
`getBroadcast()`	Indica si la opción de publicación está activada
`getChannel()`	Devuelve el canal para la conexión
`getInetAddress()`	Devuelve la dirección de la conexión
`getLocalAddress()`	Devuelve la dirección local
`getLocalPort()`	Devuelve el puerto local utilizado para la conexión
`getLocalSocketAddress()`	Devuelve información sobre el puerto local
`getPort()`	Devuelve el puerto remoto para la conexión
`getReceiveBufferSize()`	Devuelve el tamaño del búfer correspondiente
`getSendBufferSize()`	
`getRemoteSocketAddress()`	Devuelve información sobre el puerto remoto
`getReuseAddress()`	Indica si la opción de reutilizar la dirección está disponible

Variables/Métodos	Descripción
`getSoTimeout()`	Indica si la opción de compás de espera del puerto está disponible
`getTrafficClass()`	Devuelve el tipo de servicio incluido en el encabezamiento del paquete
`isBound()`	Comprueba si el puerto está vinculado
`isClosed()`	Comprueba si el puerto está cerrado
`isConnected()`	Comprueba si el puerto está conectado
`receive()`	Recibe el paquete
`send()`	Envía el paquete
`setBroadcast()`	Cambia el estado de publicación del puerto
`setDatagramSocketImplFactory()`	Cambia la implementación del `DatagramSocket` empleado
`setReceiveBufferSize()`	Cambia el tamaño del búfer
`setSendBufferSize()`	
`setReuseAddress()`	Cambia el indicador de reutilizar direcciones
`setSoTimeout()`	Cambia el indicador de la opción de compás de espera del puerto
`setTrafficClass()`	Cambia el tipo de datos del encabezamiento

Los métodos que nos interesan en este caso son los denominados `send()` y `receive()`. El primero de ellos se limita a tomar un ejemplo de `DatagramPacket` y enviar los datos del datagrama al anfitrión y la dirección del puerto predeterminada. El método `receive()` también usa un ejemplo de `DatagramPacket`, pero en esta ocasión para utilizarlo como receptor de los datagramas que se le envíen. A través del método `getData()` de la clase `DatagramPacket`, se pueden extraer los datos contenidos en los mismos. Para valorar las posibilidades de transmisión y recepción de las dos clases, el listado 22.2 contiene un ejemplo breve en el que se direcciona un servidor Web del MIT y se le pregunta la hora local (en el MIT; es decir, la Hora Estándar en la Costa Este).

Listado 22.2. Utilización de datagramas para obtener la fecha

```
import java.net.*;
public class GetDate {
  final static int PORT_DAYTIME = 13;
  public static void main(String args[]) throws Exception {
    byte msg[] = new byte[256];
    DatagramSocket dgSocket = new DatagramSocket();
    InetAddress destination = InetAddress.getByName("web.mit.edu");
```

```
        DatagramPacket datagram =
          new DatagramPacket(msg, msg.length, destination, PORT_DAYTIME);
        dgSocket.send(datagram);
        datagram = new DatagramPacket(msg, msg.length);
        dgSocket.receive(datagram);
        String received = new String(datagram.getData());
        System.out.println("The time in Cambridge is now: " + received);
        dgSocket.close();
    }
}
```

En primer lugar, el programa crea un terminal de datagramas que le servirá tanto para transmitirlos como para recibirlos. Después, produce un ejemplo de paquete de transmisión de datagramas para enviarlo al servicio de horas de puertos del servidor del MIT. La respuesta automática de éste siempre consiste en remitir un datagrama con la hora marcada (en ASCII), que se debe recoger en un ejemplo nuevo de `DatagramPacket`. El programa no vuelve a utilizar el paquete de transmisión para recibir envíos; para lo único que se empleará de nuevo es para procesar datagramas de nombre variable. Una vez recibido el datagrama de respuesta, se saca la hora del mismo en forma de matriz de bytes en ASCII y se convierte en un `String` que se imprime en la consola.

Sin embargo, la sencillez de este ejemplo enmascara un grave problema. Nada nos garantiza que los datagramas enviados o recibidos vayan a llegar a su destino. Ello puede suponer que el servidor nunca reciba el primer datagrama o que, aunque sí llegue a obtenerlo, su respuesta se pierda en el camino. La mayor utilidad de UDP es para difundir información de escaso valor o cuando hay que intercambiar mensajes a menudo, de forma que una interrupción ocasional de la comunicación no afecte demasiado al proceso. No obstante, en la mayoría de los casos hace falta una transmisión fiable de datos, y es ahí precisamente donde entran en juego todas las virtudes de TCP.

Conexión a servidores mediante TCP

El método de programación para las comunicaciones a través de TCP es similar al de UDP, con la diferencia de que no se basa en el uso de clases para encapsular sus paquetes. El protocolo TCP permite al usuario enviar datos en cantidades ilimitadas. Su clase principal, cuya definición se muestra en la tabla 22.5, se llama simplemente `Socket`:

Tabla 22.5. Composición de la clase Socket

Variables/Métodos	Descripción
`Socket()`	Construye un objeto Socket
`bind()`	Enlaza la conexión con el destino

Variables/Métodos	Descripción
`close()`	Cierra la conexión
`connect()`	Realiza la conexión con el destino
`getChannel()`	Devuelve el canal para la conexión
`getInetAddress()`	Devuelve la dirección de la conexión
`getInputStream()`	Devuelve el flujo de entrada para al conexión
`getKeepAlive()`	Indica si la opción de mantener los datos disponibles está activada
`getLocalAddress()`	Devuelve la dirección local
`getLocalPort()`	Devuelve el puerto local utilizado para la conexión
`getLocalSocketAddress()`	Devuelve información sobre el puerto local
`getOOBInLine()`	Indica si la opción fuera de rango está disponible
`getOutputStream()`	Devuelve el flujo de salida para la conexión
`getPort()`	Devuelve el puerto remoto para la conexión
`getReceiveBufferSize()`	Devuelve el tamaño del búfer correspondiente
`getSendBufferSize()`	
`getRemoteSocketAddress()`	Devuelve información sobre el puerto remoto
`getReuseAddress()`	Indica si la opción de reutilizar la dirección está disponible
`getSoLinger()`	Indica si la opción de finalizar la disponibilidad de los datos está activada
`getSoTimeout()`	Indica si la opción de compás de espera del puerto está disponible
`getTcpNoDelay()`	Indica si la opción de "no espera" del TCP está activada(algoritmo de Nagle)
`getTrafficClass()`	Devuelve el tipo de servicio incluido en el encabezamiento del paquete
`isBound()`	Comprueba si el puerto está vinculado
`isClosed()`	Comprueba si el puerto está cerrado
`isConnected()`	Comprueba si el puerto está conectado
`isInputShutdown()`	Comprueba si se ha cerrado el flujo de entrada
`isOutputShutdown()`	Comprueba si se ha cerrado el flujo de salida
`sendUrgentData()`	Envía bytes de datos urgentes
`setKeepAlive()`	Cambia el indicador de mantener datos disponibles

Variables/Métodos	Descripción
`setOOBInLine()`	Cambia el indicador de fuera de rango
`setReceiveBufferSize()`	Cambia el tamaño del búfer
`setSendBufferSize()`	
`setReuseAddress()`	Cambia el indicador de reutilizar direcciones
`setSocketImpleFactory()`	Cambia la implementación del `DatagramSocket` empleado
`setSoLinger()`	Cambia el indicador de finalizar la disponibilidad de los datos
`setSoTimeout()`	Cambia el indicador de la opción de compás de espera del puerto
`setTcpNoDelay()`	Cambia el indicador de no espera de TCP
`setTrafficClass()`	Cambia el tipo de datos del encabezamiento
`shutdownInput()`	Cierra el flujo de entrada
`shutdownOutput()`	Cierra el flujo de salida
`toString()`	Convierte los datos del estado del puerto en una cadena

Para utilizar la clase `Socket`, debemos conocer los cinco constructores de `Socket` y los dos métodos de acceso a los flujos de datos.

```
public Socket()
public Socket(InetAddress address, int port) throws IOException
public Socket (InetAddress address, int port, InetAddress localAddr,
  int localPort) throws IOException
public Socket (String host, int port) throws UnknownHostException,
IOException
public Socket (String host, int port, InetAddress localAddr, int localPort)
  throws IOException
public InputStream getInputStream() throws IOException
public OutputStream getOutputStream() throws IOException
```

Los constructores deben llevar un ejemplo de objeto `InetAddress` o un `String` para especificar el ordenador de destino al que uno se quiere conectar. Si éste es de tipo basado en IP, no hace falta nada más. Sin embargo, si es de la clase TCP, los constructores han de incluir asimismo una dirección de puerto del ordenador remoto. Al usar esta modalidad asumirá el papel de cliente (según el modelo de aplicación cliente/servidor), por lo que no podrá asignarle ningún número antiguo a esa dirección de puerto. Tendrá que utilizar alguno de los más conocidos, como los de la tabla 22.1.

Todas las conexiones TCP implican el uso de dos puertos, uno en el ordenador remoto y otro en el local, a través del cual se comunica el cliente. En este caso, no

hay que especificar el número de puerto local, ya que el propio software TCP/IP los va asignando sobre la marcha. A esta clase de puertos se les llama efímeros porque, a diferencia de los del servidor, sólo existen durante una operación concreta de tipo cliente/servidor.

Por el contrario, los puertos del servidor permanecen activos mientras el software del mismo (también llamado *daemon)* siga funcionando.

Nota

En los sistemas Unix, los programas del servidor que controlan los puertos se implementan como tareas *daemon*. Esta es la razón por la que todos los nombres terminan en "d" (smtpd para el servicio SMTP, ftpd, para el servicio FTP, y telnetd, para el servicio Telnet).

Veamos ahora algunos ejemplos prácticos que se basan en la clase `Socket` para comunicarse con Internet.

Cómo conseguir las especificaciones de los protocolos de Internet

Las especificaciones completas de todos los protocolos de Internet (IP, UDP, TCP, SMTP, FTP, HTTP, etc.) se pueden obtener a través de los propios documentos estándar, que se denominan *Request for Comments* (RFC). Por ejemplo, los RFC 821 y 822 contienen toda la información sobre SMTP. Estos RFC se extraen mediante un FTP anónimo de diversos sitios de Internet, como `mit.edu`. Otra forma aún más eficaz de conseguirlos consiste en enviar un mensaje por correo electrónico a rfc-info@isi.edu, con un texto de dos líneas, como por ejemplo:

```
Retrieve: RFC
Doc-ID: RFC0821
```

El servidor enviará el documento por correo electrónico (en ASCII, por supuesto) en un plazo de 24 horas.
Naturalmente, también se puede acudir a http://info.internet.isi.edu/1/in-notes/rfc y extraer directamente el documento.

Conexión con un servidor de correo SMTP

Si quiere enviar correo electrónico a un ordenador, tiene que llamar al puerto número 25 del mismo. Después, deberá comunicarse con la entidad que se halla en ese puerto a través de un rigurosísimo, a la par que sencillo, protocolo de comunicaciones llamado SMTP. He aquí un resumen de su modo de funcionamiento:

1. El servidor SMTP envía una cadena inicial de identificación.
2. Le contestamos diciéndole desde qué ordenador está trabajando.
3. Si todo está en orden, el servidor nos proporciona un acuse de recibo de su mensaje.
4. Le respondemos comunicándole la dirección de correo electrónico "From:" (la dirección del remitente).
5. Si todo está conforme, el servidor nos vuelve a dar acuse de recibo.
6. A continuación, le facilitaremos la dirección "To:" (del destinatario).
7. Si todo va bien, el servidor da un nuevo acuse de recibo.
8. Enviamos el mensaje completo, línea a línea, y lo concluimos con un renglón simple acabado en un punto final (".").
9. El servidor vuelve a dar acuse de recibo.
10. Nos desconectamos mediante la opción "QUIT".

Todo este intercambio de mensajes se produce en lenguaje ASCII legible (en este protocolo no existen señalizadores binarios ni tampoco campos que sean crípticos). Los protocolos de comunicación de datos fuera de la esfera de Unix se basan por lo general en estructuras complejas de paquetes codificados en binario. Pocos de los protocolos para aplicaciones de Internet emplean una representación binaria, que aunque más compleja es también más eficaz, para las unidades de datos de protocolo (PDU), que son los tipos de paquetes utilizados para gestionar un protocolo de comunicación entre dos equipos.

Nota

Sun proporciona una biblioteca de clases llamada JavaMail API para trabajar de manera más sencilla con servicios SMTP, POP, IMAP, y NNTP.

Un ejemplo de cómo enviar un correo electrónico

El programa del listado 22.3 demuestra lo fácil que es comunicarse con un TCP a través de cualquier puerto de un equipo remoto. Este programa envía un mensaje de correo electrónico desde la línea de comandos, al especificar el nombre de archivo del mensaje que se va a enviar, las direcciones Desde y Para, y la dirección de un anfitrión de correo que aceptará y/o reenviará el mensaje.

Listado 22.3. Envío de un correo mediante la clase socket

```
import java.io.*;
import java.net.*;
```

```
public class SMTPDemo {
  public static void main(String args[]) throws
      IOException, UnknownHostException {
    String msgFile;
    String from, to, mailHost;
    if (args.length != 4) {
      System.err.println("Usage: java SMTPDemo msgFile from to mailHost");
      System.exit(-1);
    }
    msgFile   = args[0];
    from      = args[1];
    to        = args[2];
    mailHost  = args[3];
    checkEmailAddress(from);
    checkEmailAddress(to);
    SMTP mail = new SMTP(mailHost);
    if (mail != null) {
      if (mail.send(new FileReader(msgFile), from, to) ) {
        System.out.println("Mail sent.");
      } else {
        System.out.println("Connect to SMTP server failed!");
      }
    }
    System.out.println("Done.");
  }
  static void checkEmailAddress(String address) {
    if (address.indexOf("@") == -1) {
      System.err.println("Invalid e-mail address "" + address + """);
      System.exit(-2);
    }
  }
  static class SMTP {
    private final static int SMTP_PORT = 25;
    InetAddress mailHost;
    InetAddress localhost;
    BufferedReader in;
    PrintWriter out;
    public SMTP(String host) throws UnknownHostException {
      mailHost = InetAddress.getByName(host);
      localhost= InetAddress.getLocalHost();
      System.out.println("mailhost = " + mailHost);
      System.out.println("localhost= " + localhost);
      System.out.println("SMTP constructor done\n");
    }
    public boolean send(FileReader msgg, String from, String to)
        throws IOException {
      Socket smtpPipe;
      InputStream inn;
      OutputStream outt;
      BufferedReader msg;
      msg = new BufferedReader(msgg);
      smtpPipe = new Socket(mailHost, SMTP_PORT);
      if (smtpPipe == null) {
        return false;
      }
      // Obtiene el flujo de datos
```

```
        inn  = smtpPipe.getInputStream();
        outt = smtpPipe.getOutputStream();
        // Los convierte en datos utilizables
        in   = new BufferedReader(new InputStreamReader(inn));
        out = new PrintWriter(new OutputStreamWriter(outt), true);
        if (inn==null || outt==null) {
          System.out.println("Failed to open streams to socket.");
          return false;
        }
        String initialID = in.readLine();
        System.out.println(initialID);
        System.out.println("HELO " + localhost.getHostName());
               out.println("HELO " + localhost.getHostName());
        String welcome = in.readLine();
        System.out.println(welcome);
        System.out.println("MAIL From:<" + from + ">");
               out.println("MAIL From:<" + from + ">");
        String senderOK = in.readLine();
        System.out.println(senderOK);
        System.out.println("RCPT TO:<" + to + ">");
               out.println("RCPT TO:<" + to + ">");
        String recipientOK = in.readLine();
        System.out.println(recipientOK);
        System.out.println("DATA");
               out.println("DATA");
        String line;
        while ((line = msg.readLine()) != null) {
          out.println(line);
        }
        System.out.println(".");
               out.println(".");
        String acceptedOK = in.readLine();
        System.out.println(acceptedOK);
        System.out.println("QUIT");
               out.println("QUIT");
        return true;
      }
    }
```

El programa de demostración de SMPT define una nueva clase llamada `SMPT`, que a su vez proporciona un único método `send()`. Este método permite a los clientes enviar un mensaje de correo electrónico a un receptor. Esta implementación de `send()` absorbe la respuesta del servidor sin comprobar los errores. El programa principal, por el contrario, realiza una comprobación sencilla de las direcciones para ver si al menos contienen el carácter @.

El código del programa se basa en métodos de entrada y salida de flujos de datos, donde se oculta la lógica del protocolo. La clase `Socket` permite a los clientes comunicarse a través de flujos de entrada y de salida comunes que se conectan al equipo. Dichos flujos se obtienen mediante los métodos `getInputStream()` y `getOutputStream()`. Es posible, por lo tanto, actualizar los flujos `InputStream` y `OutputStream`, incluyéndolos en flujos de nivel superior, tal como `BufferReader` para leer y `PrintWriter`, para escribir. Mediante estas clases, podemos tratar los

datos de entrada y de salida como líneas de texto ASCII, lo que resulta más apropiado en el caso del protocolo SMPT.

Conexión a un servidor HTTP de Web

Otro de los protocolos que funciona en este formato transparente lineal de ASCII es HTTP, donde se apoya la Web. Este protocolo está basado en un intercambio de encabezamientos multilineales de consultas y respuestas. Como siempre ocurre en los sistemas cliente/servidor, es aquél (es decir, el buscador) el que inicia la comunicación enviando al servidor un encabezamiento de petición. A continuación este último, contesta a dicha solicitud mandándole un encabezamiento de respuesta, que habitualmente contiene todos los recursos pedidos (página Web, fichero de imágenes, fragmento sonoro, etc.) en forma de datos anexos.

El tipo más común de formato de encabezamiento de petición se llama petición GET. Su formato es el siguiente:

```
GET <URL> HTTP/1.0
```

Este encabezamiento de petición de cliente tiene tres elementos:

- El método de petición (GET).
- El recurso URL.
- La versión del protocolo HTTP utilizada en el intercambio (HTTP/1.0).

Si el servidor puede satisfacer esa petición, replica con un encabezamiento de respuesta. He aquí un ejemplo de este tipo de cabecera:

```
HTTP/1.0 200 OK
Server:Apache/1.0.2
Content-type: text/html
Last-Modified: Fri, 04 Sep 1998 12:34:56 PST
<html><head>
...
```

Los rasgos más notables de los encabezamientos de respuesta de HTTP son los siguientes:

- Comienzan con la respuesta sobre el estado (los 200 OK).
- Contienen el tipo de recurso de retorno (en el campo `Content-type`).
- Llevan una línea en blanco que separa el encabezamiento de los datos propiamente dichos.

En el susodicho ejemplo de encabezamiento de respuesta, podemos ver el principio de una corriente solicitada de ficheros HTML (justo a continuación de la línea en blanco). Análogamente, cualquier otro recurso que no fuera de ASCII se desarrollaría a partir de dicha línea en blanco. Así, en el caso de un clip de audio, esos datos irán codificados en un sistema binario ilegible.

Un ejemplo: Descarga de una página Web

El programa del listado 22.4 nos permite bajar de Internet cualquier página, desde la línea de comandos.

Listado 22.4. Lectura de páginas Web con Streams

```
import java.io.*;
import java.net.*;
public class GetWebPage {
  public static void main(String args[])
      throws IOException, UnknownHostException {
    String resource, host, file;
    int slashPos;
    if (args.length != 1) {
      System.err.println("Usage: java GetWebPage <URL>");
      System.exit(-1);
    }
    if (!args[0].startsWith("http://")) {
      System.err.println("Please specify a legal http URL.");
      System.exit(-2);
    }
    resource = args[0].substring(7);    // ignora HTTP://
    slashPos = resource.indexOf("/");   // encuentra el separador anfitrión/
                                        // archivo
    if (slashPos < 0) {
      resource = resource + "/";
      slashPos = resource.indexOf("/");
    }
    file = resource.substring(slashPos); // separa las partes del anfitrión
                                         // y el archivo
    host = resource.substring(0,slashPos);
    System.out.println("Host to contact: "" + host +"""");
    System.out.println("File to fetch  : "" + file +"""");
    HTTP webConnection = new HTTP(host);
    if (webConnection != null) {
      BufferedReader in = webConnection.get(file);
      String line;
      while ((line = in.readLine()) != null) {  // lee hasta EOF
        System.out.println( line );
      }
    }
    System.out.println("\nDone.");
  }
  static class HTTP {
    public final static int HTTP_PORT = 80;
    InetAddress wwwHost;
    DataInputStream in;
    PrintStream out;
    public HTTP(String host) throws UnknownHostException {
      wwwHost = InetAddress.getByName(host);
      System.out.println("WWW host = " + wwwHost);
    }
    public BufferedReader get(String file) throws IOException {
```

```
      Socket httpPipe;
      InputStream inn;
      OutputStream outt;
      BufferedReader ir;
      PrintWriter out;
      httpPipe = new Socket(wwwHost, HTTP_PORT);
      if (httpPipe == null) {
        return null;
      }
      // Obtiene flujos simples
      inn  = httpPipe.getInputStream();
      outt = httpPipe.getOutputStream();
      // Los convierte en datos utilizables
      ir   = new BufferedReader(new InputStreamReader(inn));
      out  = new PrintWriter(new OutputStreamWriter(outt), true);
      if (inn==null || outt==null || ir==null || out==null) {
        System.out.println("Failed to open streams to socket.");
        return null;
      }
      // Envía la petición GET
      System.out.println("GET " + file + " HTTP/1.0\n");
             out.println("GET " + file + " HTTP/1.0\n");
      // Leę la respuesta hasta la línea en blanco del separador
      String response;
      while ((response = ir.readLine()).length() > 0) {
        System.out.println(response);
      }
      return ir;  // devuelve BufferedReader para que el cliente lea la fuente
    }
  }
}
```

Como en el programa del listado 22.3, la clase `HTTP` se crea para contener los detalles del protocolo. En lugar de un método `send()` el programa implementa el método `get()`. Este método crea un nuevo `Socket` del mismo modo que hizo la clase `SMPT`. A continuación, obtiene los flujos de entrada y salida asociados con el `socket`, para así, enviar y recibir datos mediante el vínculo HTTP. La parte fundamental del método `get()` consiste en enviar el encabezamiento `GET HTTP` seguido del análisis del encabezamiento de la respuesta del servidor Web.

La implementación mínima que aquí se produce se limita a leer y repetir el encabezamiento de respuesta hasta encontrar la línea del separador. A partir de entonces, todos los datos restantes forman parte de los datos solicitados por el programa principal, de modo que el flujo de entrada se devuelve al invocador, que puede entonces leer el flujo fuente (sin tener en cuenta que estaban precedidos por un encabezamiento ASCII).

El siguiente código forma parte de una sesión del programa `GetWebPage` del listado 22.4:

```
> java GetWebPage http://www.jguru.com
Host to contact: "www.jguru.com"
File to fetch  : "/"
WWW host = www.jguru.com/64.49.216.133
```

```
GET / HTTP/1.0

HTTP/1.0 200 OK
Server: Resin/2.0.1
Cache-Control: no-cache
Pragma: no-cache
Set-Cookie: JSESSIONID=aaaf4rkQQmJjpr; Path=/
Content-Type: text/html
Content-Length: 1789
Date: Mon, 28 Jan 2002 10:16:10 GMT

<html>
<head>
<title>
---------------- Bulk of HTML file cut ------------
</html>
Done.
```

Si especificamos un archivo no existente en un servidor, obtenemos el mensaje "404: Archivo no encontrado" El contenido exacto depende del servidor empleado.

Cómo obtener otros recursos de la Web

Los campos `content-type` devueltos por todos los servidores de Web utilizan un formato estándar del tipo denominado MIME (Extensiones Multipropósito para correo en Internet). La tabla 22.6 muestra algunos de los tipos MIME más comunes y su significado.

Tabla 22.6. Tipos MIME estándar

Tipo MIME	Origen
application/octet-stream	Corriente binaria genérica de bytes procedente de una aplicación indeterminada
application/pdf	Fichero de Adobe Acrobat
application/postscript	Fichero en lenguaje PostScript
application/rtf	Fichero del procesador de textos Rich Text Format
application/x-java-jnlp-file	Fichero del protocolo JNLP
audio/basic	Fichero de clip de audio .snd o .au
audio/x-aiff	Fichero de audio IFF
audio/x-wav	Fichero .wav
image/gif	Fichero de imagen .gif
image/jpeg	Fichero de imagen .jpg
image/tiff	Fichero de imagen .tif

Tipo MIME	Origen
image/x-xbitmap	Fichero de imagen .xbm
text/html	Fichero .html o .htm
text/plain	Ficheros .txt, .c, .cpp, .pl,
video/mpeg	Fichero .mpg
video/quicktime	Fichero .mov o .qt de Apple QuickTime

Para obtener otros recursos de la Web, como imágenes o ficheros de sonido, se puede utilizar el mismo procedimiento que en el programa `GetWebPage`, con la única diferencia de que se necesitan distintos métodos o clases para tratar los diversos tipos de contenido emitidos por el servidor.

Antes de dedicarle un gran esfuerzo a la identificación de cada uno de esos tipos de contenido, debemos tener en cuenta que Java nos ofrece una solución general, que se demuestra en el listado 22.5.

Listado 22.5. Lectura de páginas Web con URL

```
import ja
va.net.*;
import java.io.*;
public class GetContent {
  public static void main(String args[])
      throws MalformedURLException, IOException {
    Object obj;
    obj = (new URL("http://www.jguru.com/images/logo.gif"
      )).getContent();
    System.out.println(obj.getClass().getName());
  }
}
```

Con este sencillo programa conseguimos prácticamente lo mismo que con el programa más complejo denominado `GetWebPage`. La única diferencia es que `GetContent` extrae un fichero de imagen de un servidor Web en vez de un archivo HTML.

Sin *sockets*, sin flujos de entrada o salida, sin especificaciones de protocolo, entonces, ¿cómo funciona? ¿Y cómo es que sirve para todos los tipos MIME enumerados en la tabla 22.6? La respuesta a estas preguntas se encuentra en una nueva clase incorporada a este programa: la clase `URL`, cuyo funcionamiento veremos a continuación.

Operaciones en URL

La clase `URL` define el URL (Buscador de Recursos Uniformes) de la Web, así como otras operaciones que se pueden realizar en los URL. Existe además un URI

(Identificador de Recursos Uniformes) que empleamos cuando no necesitamos las operaciones. En cuanto a sus funciones más elementales, la clase URL se asemeja a InetAddress en que sólo nos permite crear un objeto que direcciona o apunta hacia algo.

En esta última clase, sus muestras apuntan hacia anfitriones de Internet; en cambio, las muestras de la clase URL apuntan a los recursos de la Web (páginas Web, ficheros de texto, ficheros de imágenes, clips de audio, etc.). La tabla 22.7 contiene la definición de la clase URL.

Tabla 22.7. Composición de la clase URL

Variables/Métodos	Descripción
URL()	Construye un objeto URL
equals()	Busca dos objetos iguales
getAuthority()	Devuelve la parte del componente
getFile()	
getHost()	
getPath()	
getPort()	
getProtocol()	
getQuery()	
getRef()	
getUserInfo()	
getContent()	Devuelve el objeto al que se refiere el URL
getDefaultPort()	Devuelve el puerto por defecto para el protocolo URL
hashCode()	Devuelve el código hash computado
openConnection()	Devuelve la conexión del tipo URL
openStream()	Devuelve la conexión como un InputStream
sameFile()	Busca semejanzas entre los URL
set()	Configura los campos internos
setURLStreamHandlerFactory()	Cambia la implementación de la conexión URL utilizada
toExternalForm()	Convierte la URL en una cadena
toString()	

Existen seis constructores para esta clase:

```
public URL(String spec) throws MalformedURLException
public URL(String protocol, String host, int port, String file)
  throws MalformedURLException
public URL(String protocol, String host, int port, String file,
  URLStreamHandler handler) throws MalformedURLException
public URL(String protocol, String host, String file)
  throws MalformedURLException
public URL(URL context, String spec) throws MalformedURLException
public URL(URL context, String spec, URLStreamHandler handler)
  throws MalformedURLException
```

Los argumentos del segundo constructor reflejan la estructura básica de todas las direcciones URL bien elaboradas:

```
<protocol> <host address> [<:port number>] <resourse>
```

Los campos contienen la siguiente información:

- El campo `protocol` puede ser `http:`, `ftp:`, `gopher:`, `news:`, `telnet:`, `mailto:` o cualquier otro que sea compatible.
- El `host address` puede ser cualquier dirección de anfitrión que tenga carácter legal, como por ejemplo www.apple.com.
- El campo `port number` es opcional e indica el número de puerto al que nos queremos conectar si el puerto de este protocolo por defecto se desactiva.
- El campo `resource` equivale habitualmente a la ruta completa de un fichero del sistema del ordenador remoto, aunque puede ser el que solicite en un momento dado el protocolo; por ejemplo, el nombre de un grupo de noticias en el protocolo `news:`

Los siguientes ejemplos corresponden a varias URL legales:

```
http://www.example.org:8080/index.htm
http://java.sun.com/jdc/earlyAccess/index.html#jdkdocs
http://user:password@www.example.com/protected/
ftp://ftp.uni-paderborn.de/pub/Aminet/README
news:comp.lang.java.programmer
gopher:gopher.ucdavis.edu
mailto:president@whitehouse.gov
jdbc:odbc:mydatabase
```

El primer URL, que anula el puerto del servidor estándar para Web (el 80) y lo reemplaza por un puerto menos común (el 8080). El segundo URL, especifica una referencia, es decir, un fragmento, marcador ancla, o situación, dentro del documento index.html, al añadir el símbolo de almohadilla "#" seguido del nombre de la referencia en el documento. El tercer URL indica cómo especificar una autoridad, a través del texto que precede el símbolo @.

El segundo constructor de objetos de URL reproduce los elementos de estas direcciones en su propia lista de argumentos. El tercer constructor funciona de

manera similar, con la salvedad de que no obliga al usuario a especificar claramente el número de puerto; en su lugar, utiliza el número (por defecto) del protocolo en cuestión (el 21 para FTP, el 80 para HTTP, etc.). Por su parte, el programa `GetContent` utiliza el primer constructor (el más compacto) para crear un ejemplo de clase `URL`. Simplemente, toma una cadena URL, tal como un usuario la transcribiría en cualquier cuadro de texto del URL del navegador de la Web.

Control de vínculos HTTP

Una vez construido el ejemplo de la clase `URL`, se puede extraer cualquiera de los campos de los elementos de la misma mediante los métodos `getPort()`, `getProtocol()`, `getHost()`, `getFile()` y `getRef()`. El más importante de todos ellos es `getContent()`, que se utiliza en el programa de demostración. En él no hace falta especificar claramente el tipo de recurso que se busca, el programa lo localiza y nos lo ofrece de la forma más adecuada. Por ejemplo, un objeto de `Image` nos lo devolvería en forma de recurso gráfico GIF o JPEG. El secreto de la clase `URL` estriba en que depende en gran medida de una clase estrechamente relacionada con ella, la clase `URLConnection`, que la ayuda a realizar todas esas laboriosas tareas. Esta última clase se describe en la tabla 22.8.

Tabla 22.8. Composición de la clase URLConnection

Variables/Métodos	Descripción
allowUserInteraction	Utilizado para mantener el estado interno de la conexión
connected	
doInput	
doOutput	
ifModifiedSince	
url	
useCaches	
URLConnection()	Construye un objeto `URLConnection`
`addRequestProperty()`	Añade pares de claves y valores para determinar las propiedades de solicitud
`connect()`	Abre el vínculo de comunicación
`getAllowUserInteraction()`	Indica la conexión interactiva
`getContent()`	Devuelve el objeto al que se hace referencia
`getContentEncoding()`	Devuelve el código del objeto

Variables/Métodos	Descripción
`getContentLength()`	Devuelve la longitud del objeto
`getContentType()`	Devuelve el tipo del objeto
`getDate()`	Devuelve la fecha asociada con el objeto
`getDefaultAllowUserInteraction()`	Devuelve la indicación por defecto de la conexión interactiva
`getDefaultUseCaches()`	Devuelve la indicación por defecto del protocolo que está utilizando el caché
`getDoInput()`	Devuelve la indicación de la interacción con flujos de entrada permitidos
`getDoOutput()`	Devuelve la indicación de la interacción con flujos de salida permitidos
`getExpiration()`	Devuelve la indisponibilidad del objeto
`getFileMapName()`	Devuelve el mapa actual de tipo MIME
`getHeaderField()`	Devuelve el valor del campo del encabezamiento específico
`getHeaderFieldDate()`	
`getHeaderFieldInt()`	
`getHeaderFieldKey()`	
`getHeaderFields()`	Devuelve el valor de todos los campos del encabezamiento
`getIfModifiedSince()`	Devuelve el campo de modificación del objeto
`getInputStream()`	Devuelve el flujo de entrada asociado con la conexión
`getLastModified()`	Devuelve el último elemento modificado
`getOutputStream()`	Devuelve el flujo de salida asociado con la conexión
`getPermission()`	Devuelve el permiso para realizar la conexión
`getRequestProperties()`	Devuelve el mapa sólo de lectura de todas las propiedades solicitadas
`getRequestProperty()`	Devuelve el valor de la propiedad de la solicitud especificada
`getURL()`	Devuelve el URL empleado para realizar la conexión

Variables/Métodos	Descripción
`getUseCaches()`	Devuelve la indicación de la conexión que está empleando el caché
`guessContentTypeFromName()`	Supone el tipo MIME a partir del código
`guessContentTypeFromStream()`	
`setAllowUserInteraction()`	Cambia el indicador de conexión interactiva
`setContentHandlerFactory()`	Cambia la implementación del `ContentHandler` empleado
`setDefaultAllowUserInteraction()`	Cambia el indicador por defecto de las conexiones interactivas
`setDefaultUseCaches()`	Cambia el indicador por defecto de uso del caché
`setDoInput()`	Cambia el indicador del uso del flujo de entrada
`setDoOutput()`	Cambia el indicador del uso del flujo de salida
`setFileNameMap()`	Cambia el mapa de tipo MIME
`setIfModifiedSince()`	Cambia el valor de la modificación del objeto
`setRequestProperty()`	Cambia el valor de la propiedad de la solicitud (o lo añade si no existe)
`setUseCaches()`	Cambia el indicador del uso del caché
`toString()`	Convierte la dirección en una cadena

Como se deduce del tamaño de la clase `URLConnection`, podemos controlar mucho mejor el vínculo HTTP que se crea al activar (abrir) una conexión URL. Los siguientes métodos nos permiten solicitar los valores de los campos del encabezamiento de la respuesta HTTP devuelta por el servidor:

```
String getContentType()
int getContentLength()
String getContentEncoding()
long getDate()
long getExpiration()
long getLastModified()
```

Sólo el campo de tipo de contenido es obligatorio, para que así todos los demás métodos puedan devolver el valor `Null` para los tipos `String`, o el valor `0`, para los tipos numéricos, en caso de que el servidor no proporcione la información necesaria.

Nota

Los contenidos `type` y `encoding` son diferentes, ya que el segundo nos informa sobre el tipo de código del Recurso. Los tipos más comunes son los binarios de 8 bits, los de código UU, y los que se basan en 64 bits. Los dos últimos se emplean cuando la respuesta de HTTP tiene que darse de manera que el bit más importante de cada byte sea cero.

Hay dos métodos `URLConnection` con un funcionamiento similar al de los métodos de `Socket` vistos anteriormente: `getInputStream()` y `getOutputStream`. La clase `URL` también dispone de un procedimiento `openStream()` (aunque carece del correspondiente método de flujo de salida), que se pone en marcha llamando al `getInputStream` implícito en la clase `URLConnection`.

En ésta hay varios métodos para tratar los asuntos relacionados con el almacenamiento temporal de los recursos. Habitualmente, los navegadores de la Web almacenan temporalmente las imágenes en línea, y a veces incluso los códigos completos de algunas páginas, para su uso posterior, con lo que se agiliza la carga y presentación de las mismas. El inconveniente de todo esto es que las páginas o ilustraciones originales de la Web pueden sufrir cambios importantes, que pasarían inadvertidos si la memoria de almacenamiento temporal del navegador guardará indefinidamente todos los recursos memorizados de esta manera.

Para resolver este problema, se utilizan los cuatro métodos siguientes:

```
void setDefaultUseCaches(boolean defaultusecaches)
boolean getDefaultUseCaches()
void setUseCaches(boolean usecaches)
boolean getUseCaches()
```

Por último, se han incluido dos utilidades para averiguar los contenidos:

```
public static String guessContentTypeFromName(String fname)
public static String guessContentTypeFromStream(InputStream is)
```

Nota

Además de `URLConnection`, las dos subclases que contiene, denominadas `HttpURLConnection` y `JarURLConnection`, ofrecen información especializada sobre las conexiones URL.

Uso de la identificación por contraseñas

Aunque la clase `URL` nos permite incluir la información sobre la identificación dentro de una cadena URL, por lo general será necesario que el usuario incluya esta

información en lugar de tener que codificar todos los datos. Si almacenamos la contraseña en un `String` no tendremos ningún tipo de control cuando el liberador de espacio libere los datos y queden visibles.

En lugar de proporcionar el URL con la información de la identificación, podemos instalar previamente un `Authen-ticator` para cuando dicha identificación sea necesaria. De esta forma, nos dirá cuándo hace falta introducir un nombre de usuario y contraseña y poder actuar en consecuencia para obtener la información.

Nota

El `Authenticator` y los datos de identificación con una cadena URL sólo sirven para identificaciones en HTTP.

El `Authenticator` es una clase abstracta, por lo que hay que crear una subclase para poder utilizarlo. Aunque desde un punto de vista técnico no hay ningún método abstracto, para que el sistema de identificación funcione de verdad hay que desactivar el método `getPasswordAuthentication` y sustituirlo por un `PasswordAuthentication`, con el nombre deseado de usuario y la clave. Las tablas 22.9 y 22.10 contienen la definición de las clases `Authenticator` y `PasswordAuthentication`.

Tabla 22.9. Composición de la clase Authenticator

Variables/Métodos	Descripción
`Authenticator()`	Construye un objeto Authenticator
`getPasswordAuthentication()`	Devuelve el objeto PasswordAuthentication apropiado cuando se necesita
`getRequestingHost()`	Devuelve información sobre la conexión solicitada
`getRequestingPort()`	
`getRequestingPrompt()`	
`getRequestingProtocol()`	
`getRequestingScheme()`	
`getRequestingSite()`	
`requestPasswordAuthentication()`	Pide la contraseña al identificador solicitado
`setDefault()`	Instala el identificador por defecto

Tabla 22.10. Composición de la clase PasswordAuthentication

Variables/Métodos	Descripción
`PasswordAuthentication()`	Construye un objeto PasswordAuthentication
`getPassword()`	Devuelve la contraseña del usuario
`getUsername()`	Devuelve el nombre de usuario

Como podemos ver, todo en el `Authenticator`, a excepción del constructor, es `static` o `protected`. El único método `protected`, aparte de `final`, `getPasswordAuthentication()`, nos da la clave del método que hay que desactivar por si se nos olvida. Si hay que pedir el nombre de usuario y la contraseña, se puede mostrar el texto de `getRequestedPrompt()` y solicitar la entrada correspondiente. Si no, también existe la posibilidad de pasar la información requerida al constructor de subclases, tal como se verá en el próximo ejemplo. Una vez que se ha obtenido la información, se crea un ejemplo de `PasswordAuthentication` y se devuelve. El programa del listado 22.6 ofrece una demostración de este procedimiento, extrayendo en este caso el URL, nombre de usuario y contraseña de la línea de comandos. La clase `MyAuthenticator` realiza la identificación y se instala en la línea `Authenticator.setDefault()`. Deberemos encontrar por nuestros propios medios un URL cuya contraseña y nombre de usuario conozca.

Listado 22.6. Lectura de una URL protegida con identificación HTTP

```
import java.io.*;
import java.net.*;
public class AuthDemo {
  public static void main(String args[])
      throws MalformedURLException, IOException {
    String urlString, username, password;
    if (args.length != 3) {
      System.err.println("Usage: java AuthDemo URL username password");
      System.exit(-1);
    }
    urlString = args[0];
    username  = args[1];
    password  = args[2];
    Authenticator.setDefault(new MyAuthenticator(username, password));
    URL url = new URL(urlString);
    InputStream content = (InputStream)url.getContent();
    BufferedReader in   = new BufferedReader(
      new InputStreamReader(content));
    String line;
    while ((line = in.readLine()) != null) {
      System.out.println(line);
    }
    System.out.println("Done.");
  }
```

```
  static class MyAuthenticator extends Authenticator {
    private String username, password;
    public MyAuthenticator(String user, String pass) {
      username = user;
      password = pass;
    }
    protected PasswordAuthentication getPasswordAuthentication() {
      System.out.println("Requesting Host     : " + getRequestingHost());
      System.out.println("Requesting Port     : " + getRequestingPort());
      System.out.println("Requesting Prompt   : " + getRequestingPrompt());
      System.out.println("Requesting Protocol: " + getRequestingProtocol());
      System.out.println("Requesting Scheme   : " + getRequestingScheme());
      System.out.println("Requesting Site     : " + getRequestingSite());
      return new PasswordAuthentication(username, password.toCharArray());
    }
  }
}
```

Nota

Si se conecta a un URL sin instalar un `Authenticator` y le pide que se identifique, no podrá soslayar este requisito. Probablemente, el servidor Web le mostrará un mensaje de denegación de acceso, que el programa leerá a continuación. A modo de prueba, elimine el comentario de la línea `Authenticator.setDefault()` o introduzca una combinación nula de nombre de usuario y contraseña.

Para eliminar un `Authenticator`, llame a `setDefault()` utilizando para ello el parámetro `null`.

Lectura de canales de socket

Ya hemos visto en otros capítulos cómo leer canales de archivos. La comunicación entre puertos puede realizarse mediante las técnicas ya comentadas (flujos y `URLConnection`) o mediante canales de *socket*. En este caso concreto un `SocketChannel` es simplemente otra implementación de la interfaz `Channel`. Se construye un `SocketChannel` para comunicarse con el *socket*, mientras que el `FileChannel` se construye para comunicarse con archivos.

La conexión a un `FileChannel` se realiza de la siguiente manera:

```
String filename = ...;
FileInputStream input = new FileInputStream(filename);
FileChannel channel = input.getChannel();
```

Conectarse a través de un *socket* requiere que especifiquemos el anfitrión y el puerto. Para trabajar con canales, lo hacemos mediante el método `InetSocketAddress`, que cuenta con tres constructores:

```
public InetSocketAddres(InetAddress addr, int port)
public InetSocketAddres(int port)
public InetSocketAddres(String host, int port)
```

De este modo, creamos el `InetSocketAddress`, abrimos un `SocketChannel`, y lo conectamos a la dirección. El proceso es un poco diferente al de conectar un archivo, aunque el resultado es el mismo: Contamos con un canal con el que trabajar.

```
InetSocketAddress socketAddress = new InetSocketAddress(host, 80);
SocketChannel channel = SocketChannel.open();
channel.connect(socketAddress);
```

Para demostrar cómo utilizar los canales de *socket*, el programa del listado 22.7 actualiza el ejemplo `GetWebPage` anterior, para emplear canales.

Listado 22.7. Lectura de páginas Web con canales Socket

```
import java.io.*;
import java.net.*;
import java.nio.*;
import java.nio.channels.*;
import java.nio.charset.*;
public class GetWebPage2 {
  public static void main(String args[]) throws Exception {
    String resource, host, file;
    int slashPos;
    if (args.length != 1) {
      System.err.println("Usage: java GetWebPage2 <URL>");
      System.exit(-1);
    }
    if (!args[0].startsWith("http://")) {
      System.err.println("Please specify a legal http URL.");
      System.exit(-2);
    }
    resource = args[0].substring(7);   // ignora HTTP://
    slashPos = resource.indexOf("/");  // encuentra el separador
                                       // anfitrión/archivo
    if (slashPos < 0) {
      resource = resource + "/";
      slashPos = resource.indexOf("/");
    }
    file = resource.substring(slashPos); // aisla las partes del anfitrión
                                         // y del archivo
    host = resource.substring(0,slashPos);
    System.out.println("Host to contact: "" + host +"""");
    System.out.println("File to fetch  : "" + file +"""");
    SocketChannel channel = null;
    try {
      Charset charset = Charset.forName("ISO-8859-1");
      CharsetDecoder decoder = charset.newDecoder();
      CharsetEncoder encoder = charset.newEncoder();
      // Encuentra el búfer
      ByteBuffer buffer = ByteBuffer.allocateDirect(1024);
```

```
        CharBuffer charBuffer = CharBuffer.allocate(1024);
        // Conecta
        InetSocketAddress socketAddress = new InetSocketAddress(host, 80);
        channel = SocketChannel.open();
        channel.connect(socketAddress);
        // Envía la solicitud
        String request = "GET " + file + " \r\n\r\n";
        channel.write(encoder.encode(CharBuffer.wrap(request)));
        // Lee la respuesta
        while ((channel.read(buffer)) != -1) {
          buffer.flip();
          // Decodifica el búfer
          decoder.decode(buffer, charBuffer, false);
          // Lo muestra
          charBuffer.flip();
          System.out.println(charBuffer);
          buffer.clear();
          charBuffer.clear();
        }
      } catch (UnknownHostException e) {
        System.err.println(e);
      } catch (IOException e) {
        System.err.println(e);
      } finally {
        if (channel != null) {
          try {
            channel.close();
          } catch (IOException ignored) {
          }
        }
      }
      System.out.println("\nDone.");
    }
}
```

Nota

Las clases `channel` y las nuevas bibliotecas E/S I/O se encuentran en el paquete `java.nio`. Al comunicarnos directamente con HTTP mediante conexiones *socket* debemos prestar atención a los cambios de direcciones, mientras que el mecanismo de `URLConnection` realiza esta tarea de manera automática.

Lecturas Nonblocking

Los canales `Socket` no se limitan a leer y escribir operaciones, sino que tienen la habilidad de configurar un canal para que no esté bloqueado, función que heredan directamente de la clase `AbstractSelectableChannel`. Esto significa que en

lugar de tener que esperar que se reciban datos para ser leídos, cuando empleamos una conexión de *socket*, podemos realizar la conexión y seguir realizando otras tareas. Cuando se recibe algún dato que requiere lectura, otro thread distinto recibe dicha notificación. (Por supuesto, sería posible crear un segundo thread que esperara a recibir algún dato de lectura, pero este thread bloquearía la operación, dificultando la operación de finalizar la conexión.)

Para configurar un `channel` con una arquitectura distinta de *nonblocking* sólo tenemos que emplear una llamada más a la configuración básica de la conexión.

```
channel.configureBlocking(false);
```

Aquí es donde concluyen las similitudes. Con operaciones *nonblocking*, es necesario registrar el canal mediante un `Selector`, especificando las operaciones que nos interesen. El primer paso consiste en abrir el `Selector` mediante `open()`, para después registrar el canal mediante `register()`:

```
Selector selector = Selector.open();
channel.register(selector,
  SelectionKey.OP_CONNECT | SelectionKey.OP_READ);
```

Estas operaciones del `Selector` se describen en la tabla 22.11.

Tabla 22.11. Composición de la clase Selector

Variables/Métodos	Descripción
`Selector()`	Construye un objeto Selector
`close()`	Cierra el selector
`isOpen()`	Indica si el selector está abierto
`keys()`	Devuelve el grupo de claves registradas
`open()`	Abre el selector
`provider()`	Devuelve el suministrador que creó el selector
`select()`	Selecciona una clave disponible
`selectedKeys()`	Devuelve el grupo de claves seleccionadas
`selectNow()`	Comprueba si existe alguna clave disponible y elimina el efecto del método *wake-up*
`wakeup()`	Hace que la primera operación de selección se devuelva inmediatamente

La clase `SelectionKey` define las diversas operaciones compatibles: `OP_CONNECT`, `OP_READ`, `OP_ACCEPT`, y `OP_WRITE`. La línea de `register()` muestra nuestro interés cuando exista algo que leer, o a lo que conectarnos. La tabla 22.12

muestra lo que podemos hacer con un `SelectionKey`; que no se limita, por cierto, a tareas con constantes.

Tabla 22.12. Composición de la clase SelectionKey

Variables/Métodos	Descripción
`OP_CONNECT`	Indicador de la operación
`OP_READ`	
`OP_ACCEPT`	
`OP_WRITE`	
`SelectionKey()`	Construye un objeto SelectionKey
`attach()`	Asocia un objeto a una clave
`attachment()`	Recupera el objeto asociado
`cancel()`	Cancela el registro del selector
`channel()`	Devuelve el canal asociado
`interestOps()`	Devuelve y cambia el grupo de las operaciones más importantes
`isAcceptable()`	Indica la clave lista para una operación de especificación
`isConnectable()`	
`isReadable()`	
`isWritable()`	
`isValid()`	Indica la validez de la clave
`readyOps()`	Devuelve el grupo de operaciones disponibles
`selector()`	Devuelve el selector asociado

Para leer un canal a través de un selector, debemos esperar a recibir algo con lo que trabajar.

Es aquí donde el método `select()` del `Selector` nos es de utilidad, ya que nos permite especificar los milisegundos que deseamos esperar, y no devolverá nada hasta que no haya recibido algo con lo que trabajar, o hasta que el tiempo especificado pase, o el selector comience a funcionar. Este método devuelve el número de objetos listos.

```
while (selector.select(500) > 0) {
   ...
}
```

Una vez que sabemos que hay algún elemento disponible, ya podemos obtener el conjunto de claves mediante el método `selectedKeys()`:

```
Set readyKeys = selector.selectedKeys();
```

Podemos inspeccionar este conjunto y eliminar la que vamos a procesar:

```
Iterator readyItor = readyKeys.iterator();
while (readyItor.hasNext()) {
  SelectionKey key = (SelectionKey)readyItor.next();
  readyItor.remove();
  ...
}
```

El procesamiento implica descubrir la razón por la que la clave se encuentra en la lista (si es `connectable`, cuando puede conectarse con otro elemento, o `readable`, si puede leerse), y trabajar con ella de la manera adecuada.

```
SocketChannel keyChannel = (SocketChannel)key.channel();
if (key.isConnectable()) {
  ...
} else if (key.isReadable()) {
  ...
}
```

Esta tarea es bastante tediosa para leer un único recurso de un URL, por lo que para leer un gran número de ellos, es casi insoportable. Con el mecanismo `Channel/Selector`, sin embargo, podemos emplear un único thread para leer todos los URL que necesitemos, que además no tendría que esperar la llegada de más datos para una única conexión. Todas las conexiones se gestionarían simultáneamente.

El listado 22.8 constituye una tercera versión del programa para leer un solo URL.

Listado 22.8. Lectura de páginas Web con canales de arquitectura *nonblocking*

```
import java.io.*;
import java.net.*;
import java.nio.*;
import java.nio.channels.*;
import java.nio.charset.*;
import java.util.*;
public class GetWebPage3 {
  static Selector selector;
  public static void main(String args[]) throws Exception {
    String resource, host, file;
    int slashPos;
    if (args.length != 1) {
      System.err.println("Usage: java GetWebPage3 <URL>");
      System.exit(-1);
    }
    if (!args[0].startsWith("http://")) {
      System.err.println("Please specify a legal http URL.");
      System.exit(-2);
```

```
}
resource = args[0].substring(7);    // ignora HTTP://
slashPos = resource.indexOf("/");   // encuentra el separador anfitrión/
                                    // archivo
if (slashPos < 0) {
  resource = resource + "/";
  slashPos = resource.indexOf("/");
}
file = resource.substring(slashPos); // aísla las partes del anfitrión
                                     // y del archivo
host = resource.substring(0,slashPos);
System.out.println("Host to contact: "" + host +""");
System.out.println("File to fetch  : "" + file +""");
SocketChannel channel = null;
try {
  // Configuración
  Charset charset = Charset.forName("ISO-8859-1");
  CharsetDecoder decoder = charset.newDecoder();
  CharsetEncoder encoder = charset.newEncoder();
  // Encuentra los búferes
  ByteBuffer buffer = ByteBuffer.allocateDirect(1024);
  CharBuffer charBuffer = CharBuffer.allocate(1024);
  // Conexión
  InetSocketAddress socketAddress = new InetSocketAddress(host, 80);
  channel = SocketChannel.open();
  channel.configureBlocking(false);
  channel.connect(socketAddress);
  // Abre el selector
  selector = Selector.open();
  channel.register(selector,
    SelectionKey.OP_CONNECT | SelectionKey.OP_READ);
  // Espera a que ocurra algo interesante
  while (selector.select(500) > 0) {
    // Obtiene le grupo de objetos disponibles
    Set readyKeys = selector.selectedKeys();
    try {
      Iterator readyItor = readyKeys.iterator();
      // Explora el grupo
      while (readyItor.hasNext()) {
        // Obtiene la clave del grupo, elimina la entrada actual y
        // averigua el canal
        SelectionKey key = (SelectionKey)readyItor.next();
        readyItor.remove();
        SocketChannel keyChannel = (SocketChannel)key.channel();
        if (key.isConnectable()) {
          // Termina la conexión
          if (keyChannel.isConnectionPending()) {
            keyChannel.finishConnect();
          }
          // Envía la solicitud
          String request = "GET " + file + " \r\n\r\n";
          keyChannel.write(encoder.encode(CharBuffer.wrap(request)));
        } else if (key.isReadable()) {
          // Lee lo que está preparado
          keyChannel.read(buffer);
          buffer.flip();
```

```
                // Decodifica el búfer y el display
                decoder.decode(buffer, charBuffer, false);
                charBuffer.flip();
                System.out.print(charBuffer);
                // Elimina todo para pasar al siguiente paso
                buffer.clear();
                charBuffer.clear();
              } else {
                System.err.println("Unknown key");
              }
            }
          } catch (ConcurrentModificationException e) {
            // Ignora, o reinicia con el grupo siguiente
          }
        }
      } catch (UnknownHostException e) {
        System.err.println(e);
      } catch (IOException e) {
        System.err.println(e);
      } finally {
        if (channel != null) {
          try {
            channel.close();
          } catch (IOException ignored) {
          }
        }
      }
      System.out.println("\nDone.");
    }
}
```

Programación de sistemas de servidor

Hasta ahora, este capítulo se ha centrado en la faceta del cliente de arquitectura cliente/servidor, puesto que la mayoría de los programadores en Java contemplan su actividad desde ese punto de vista. No obstante, si forma parte de esa minoría que programa en el ámbito del servidor, no del cliente, esta sección le interesará. El paquete `java.net` contiene todo lo que necesita para programar cualquier sistema de servidor, mediante la clase `ServerSocket`, como se muestra en la tabla 22.13.

Tabla 22.13. Composición de la clase ServerSocket

Variables/Métodos	Descripción
`ServerSocket()`	Construye un objeto ServerSocket
`accept()`	Detecta la conexión del socket
`bind()`	Conecta la conexión con el destino

Variables/Métodos	Descripción
`close()`	Cierra la conexión
`getChannel()`	Devuelve el canal para la conexión
`getInetAddress()`	Devuelve la dirección de la conexión
`getLocalPort()`	Devuelve el puerto local utilizado para la conexión
`getLocalSocketAddress()`	Devuelve información sobre el socket local
`getReceiveBufferSize()`	Devuelve el tamaño del búfer adecuado
`getReuseAddress()`	Indica si la opción de reutilizar la dirección está disponible
`getSoTimeout()`	Indica si la opción de compás de espera del socket está disponible
`implAccept()`	Asegura que se emplee el tipo adecuado de socket conectado a la subclase
`isBound()`	Comprueba si el socket está conectado
`isClosed()`	Comprueba si el socket está cerrado
`setReceiveBufferSize()`	Cambia el tamaño del búfer
`setReuseAddress()`	Cambia el indicador de la opción de reutilizar la dirección
`setSocketFactory()`	Cambia la implementación del socket utilizado
`setSoTimeout()`	Cambia el indicador de la opción de compás de espera del socket
`toString()`	Convierte el socket del servidor en una cadena

La clase `ServerSocket` proporciona cuatro constructores, que especifican el puerto local a utilizar. El primer constructor es independiente.

```
public ServerSocket() throws IOException
public ServerSocket(int port) throws IOException
public ServerSocket(int port, int backlog) throws IOException
public ServerSocket(int port, int backlog,
  InetAddress bindAddr) throws IOException
```

Lo único que le hace falta para comenzar a trabajar es el segundo constructor, que crea un nuevo receptor de escucha en su ordenador capaz de admitir conexiones de cualquier cliente de la red.

El argumento del número de puerto determina la entrada por la que cualquier usuario puede acceder a su servidor. Para programar un servidor estándar, como un SMTP o un FTP, necesitamos las direcciones de puerto de la tabla 22.1. Por otra parte, si lo que queremos es crear un nuevo servicio de Internet, utilizaremos un

número de puerto exclusivo. Como la gama de números de puerto va del 0 al 65.535 hay una gran cantidad de opciones, siempre que evitemos los siguientes intervalos:

- El rango 0-1.023 está reservado para protocolos "estándar" de Internet. Estos números están controlados por la IANA (Dirección de Asignación de Números de Internet).
- El intervalo que va del 1024 en adelante se utiliza para puertos efímeros de cliente.
- Algunos sistemas utilizan también para puertos efímeros el rango que va desde 32.768 en adelante, motivo por el que es conveniente evitar estos números.

En los ensayos se utiliza a menudo el puerto 8001, aunque todo lo que se sitúe en los intervalos de 8 a 16K o de 48 a 64K es perfectamente aceptable.

Una vez que acabemos de crear el nuevo objeto de `ServerSocket`, aún no estará listo para escuchar las consultas de los hipotéticos clientes, sólo se consigue cuando se llama al método `accept()` en el objeto de `ServerSocket`. Dicho método no retornará hasta que se conecte un cliente al servidor. Otra opción es llamar al método `setSoTimeout()` para asignar un compás de espera al receptor del servidor (esto hay que hacerlo antes de la llamada a `accept()`). Si no se conecta ningún cliente al servidor antes de que ese plazo expire, se lanzará una `java.io.Interrupted-IOException`.

Como podemos fácilmente deducir de la lista de métodos que proporciona `ServerSocket`, no existe ningún procedimiento de lectura o escritura, ni tampoco el receptor le permite dirigir los flujos de entrada o salida hacia él mismo. Ello se debe a que el `ServerSocket` no se utiliza en la comunicación real, sino que produce un nuevo ejemplo de `Socket` para que el software del servidor dialogue con el cliente conectado. Este ejemplo se crea (y se devuelve) cuando la conexión se acepta mediante el método `accept()`. Significa que el sistema de programación del servidor es casi idéntico al del cliente. La única diferencia es que hay que utilizar las corrientes de entrada y salida conectadas a un receptor para implementar el protocolo adecuado.

Un programa simple de servidor

El ejemplo siguiente presenta un sencillo servidor que, sin embargo, se comporta como un servidor real. Cuando un cliente se conecta a su puerto, el servidor envía una cadena de identificación de bienvenida y espera la orden del cliente. El protocolo del servidor se basa en SMTP.

Los únicos comandos que el programa implementa son `HELP` Y `QUIT`, que, aunque no son muy funcionales, aparecen en todos los protocolos basados en líneas de comandos, y además son soportados por SMTP, NNTP y FTP. El listado 22.9 contiene el código de este servidor.

Listado 22.9. Definición del servidor

```
import java.util.*;
import java.io.*;
import java.net.*;
public class ServerTest {
  final static int SERVER_PORT = 8001;  // puerto del servidor
  public static void main(String args[]) {
    Server server;
    String clientRequest;
    BufferedReader reader;
    PrintWriter writer;
    server = new Server(SERVER_PORT);
    reader = new BufferedReader(new InputStreamReader(server.in));
    writer = new PrintWriter(new OutputStreamWriter(server.out), true);
    // Envía la cadena inicial al cliente.
    writer.println("Java Test server v1.04, " + new Date());
    while (true) {
      try {
        // ¿Qué ordena el cliente?
        clientRequest = reader.readLine();
        System.out.println("Client says: " + clientRequest);
        if (clientRequest.startsWith("HELP")) {
          writer.println("Vocabulary: HELP QUIT");
        } else {
          if (clientRequest.startsWith("QUIT")) {
            System.exit(0);
          } else {
            writer.println("ERR: Command "" + clientRequest +
              "" not understood.");
          }
        }
      } catch (IOException e) {
        System.err.println("IOEx in server " + e);
      }
    }
  }
  static class Server {
    private ServerSocket server;
    private Socket socket;
    public InputStream in;
    public OutputStream  out;
    public Server(int port) {
      try {
        server = new ServerSocket(port);
        System.out.println("ServerSocket before accept: " + server);
        System.out.println("Java Test server v1.04, on-line!");
        // Espera un cliente que se conecte al puerto
        socket = server.accept();
        System.out.println("ServerSocket after accept: " + server);
        in  = socket.getInputStream();
        out = socket.getOutputStream();
      } catch (IOException e) {
        System.err.println("Server constructor IOEx: " + e);
      }
```

```
    }
  }
}
```

Al ejecutar este programa, la consola mostrará lo siguiente:

```
> java ServerTest
ServerSocket before accept:
  ServerSocket[addr=0.0.0.0/0.0.0.0,port=0,localport=8001]
Java Test server v1.04, on-line!
```

El cliente para el servidor de muestra

El servidor de muestra parece funcionar correctamente y, sin embargo, no tenemos ningún cliente que conozca el protocolo que acabamos de inventar, cuando lo que espera un servidor es que los clientes dialoguen solamente con dicho protocolo. Este problema se puede solucionar de inmediato escribiendo un programa cliente especialmente elaborado para hablar con su nuevo servidor. Tanto éste como el cliente se someterán a prueba en el mismo equipo, pero no importa, ya que el programa cliente (listado 22.10) se conectará simplemente al ordenador "anfitrión local" a través del puerto 8001 (el puerto de este servidor).

Listado 22.10. Definición del cliente

```
import java.io.*;
import java.net.*;
public class ClientTest {
  public static void main(String args[]) {
    String welcome, response;
    Client client;
    BufferedReader reader;
    PrintWriter writer;
    client = new Client("localhost", 8001);
    try {
      reader = new BufferedReader(new InputStreamReader(client.in));
      writer = new PrintWriter(new OutputStreamWriter(client.out), true);
      welcome = reader.readLine();
      System.out.println("Server says: ""+ welcome +""");
      System.out.println("HELLO");
      writer.println("HELLO");
      response =  reader.readLine();
      System.out.println("Server responds: ""+ response +""");
      System.out.println("HELP");
      writer.println("HELP");
      response = reader.readLine();
      System.out.println("Server responds: ""+ response +""");
      System.out.println("QUIT");
      writer.println("QUIT");
    } catch (IOException e) {
      System.err.println("IOException in client.in.readln()");
      System.err.println(e);
    }
  }
```

```
  static class Client {
    // Pone a disposición de las clases del usuario los flujos de entrada y
    // salida
    public InputStream in;
    public OutputStream out;
    private Socket client;
    public Client(String host, int port) {
      try {
        client = new Socket(host, port);
        System.out.println("Client socket: " + client);
        in = client.getInputStream();
        out= client.getOutputStream();
      } catch (IOException e) {
        System.err.println("IOExc : " + e);
      }
    }
  }
}
```

Si ejecuta este programa cliente en una nueva ventana de la consola mientras el servidor sigue comunicado y a la espera de conexiones de clientes, debería ver pasar la siguiente prueba:

```
> java ClientTest
Client socket: Socket[addr=localhost/127.0.0.1,port=8001,localport=1976]
Server says: "Java Test server v1.04, Mon Jan 28 13:42:08 EST 2002"
HELLO
Server responds: "ERR: Command "HELLO" not understood."
HELP
Server responds: "Vocabulary: HELP QUIT"
QUIT
```

Mientras el cliente imprimía esas líneas, su servidor debió escribir las siguientes, que reflejan su punto de vista sobre los intercambios:

```
ServerSocket after accept:
  ServerSocket[addr=0.0.0.0/0.0.0.0,port=0,localport=8001]
Client says: HELLO
Client says: HELP
Client says: QUIT
```

Como podemos observar, las dos partes se comunican sin problemas. Por supuesto, esto se debe a que el protocolo utilizado en este caso es bastante simple (para empezar, no presenta ningún estado en particular, lo cual siempre facilita las cosas), y a que en realidad no ha entrado en juego ninguna red. Aunque se han creado verdaderos paquetes TCP, no han viajado muy lejos. Se quedaron en forma de bucle dentro de la pila TCP/IP. Los protocolos reales suelen depender en gran medida de la variedad de estados en que se puedan encontrar. Ello obliga al usuario a implementar máquinas de estado para gestionar el protocolo. Los aparatos de este tipo que disponen de numerosos estados y aceptan una gran variedad de posibles incidencias se hacen muy complicados de manejar, puesto que exigen el uso de métodos matemáticos formales para demostrar su corrección.

Por desgracia, las máquinas de estado de protocolo no suelen ser demasiado sencillas, ya que las redes reales pueden ocasionar incidencias y situaciones muy variopintas. Los paquetes se pueden corromper a causa de eventuales ruidos en la línea, e incluso no llegar juntos si el equipo de la red falla de repente. Asimismo se pueden retrasar tanto, que su destinatario crea que se han perdido y súbitamente, el primer paquete aparece como por arte de magia, provocando un asincronismo entre el emisor y el receptor. Hay que tener en cuenta todas estas posibilidades al elaborar un protocolo, a menos que éste se cree a partir de otro que ya prevea tales contingencias y ésa es justamente la función de TCP, en el que se basa la mayoría, aunque no todos, de los protocolos de Internet.

Truco

Programar y depurar protocolos cliente/servidor es mucho más fácil si se usa un protocolo lineal de ASCII y se emplea Telnet para probar el servidor. Se puede incluso revisar el programa ServerTest conectándose a él a través de Telnet, en vez de usar ClientTest. Pruebe a hacerlo con `telnet localhost 8001` después de volver a poner en marcha el servidor. También descubrirá que programar su propia utilidad Telnet (en Java) es sumamente fácil.

Servidores Nonblocking

Los servidores Nonblocking son más importante que los clientes del mismo tipo ya que un único thread es capaz de gestionar múltiples solicitudes de un cliente en lugar de tener que emplear un thread para cada solicitud. Hay pocas diferencias entre este programa y el listado 22.8.

El primero es que la configuración del servidor requiere un `ServerSocketChannel` en lugar de un `SocketChannel`, y un `InetSocketAddress` para especificar el puerto desde el que realizar las conexiones:

```
ServerSocketChannel channel = ServerSocketChannel.open();
channel.configureBlocking(false);
InetSocketAddress isa = new InetSocketAddress(port);
channel.socket().bind(isa);
```

Cuando el servidor espera al método `accept()` para conectarse a un socket, `ServerSocketChannel` requiere que nos registremos con el `Selector` para un evento `OP_ACCEPT`:

```
channel.register(selector, SelectionKey.OP_ACCEPT);
```

o `OP_ACCEPT` y `OP_READ` si además de conectarnos, deseamos leer los datos:

```
channel.register(selector, SelectionKey.OP_ACCEPT | SelectionKey.OP_READ);
```

Para devolver un mensaje no es necesario leer nada; únicamente esperar un OP_ACCEPT. Para leer los comandos del cliente necesitamos ambos tipos de selección del Selector.

El ejemplo del listado 22.11 demuestra la manera de trabajar sólo con un OP_ACCEPT.

Aunque podemos utilizar el programa ClientTest para conectarnos, el programa ChannelServer se limita a devolver un mensaje.

Listado 22.11. Definición del servidor de arquitectura non-blocking

```
import java.io.*;
import java.net.*;
import java.nio.*;
import java.nio.channels.*;
import java.util.*;
public class ChannelServer {
  final static int SERVER_PORT = 8001;  // puerto del servidor
  public static void main(String args[]) throws Exception {
    Selector selector = Selector.open();
    ServerSocketChannel channel = ServerSocketChannel.open();
    channel.configureBlocking(false);
    InetSocketAddress isa = new InetSocketAddress(SERVER_PORT);
    channel.socket().bind(isa);
    // Registra la acción al conectarse
    channel.register(selector, SelectionKey.OP_ACCEPT);
    // Espera alguna acción
    while (selector.select() > 0) {
      // Obtiene el conjunto de objetos disponibles
      Set readyKeys = selector.selectedKeys();
      Iterator readyItor = readyKeys.iterator();
      // Explora el conjunto
      while (readyItor.hasNext()) {
        // Obtiene la clave del conjunto y elimina la entrada actual
        SelectionKey key = (SelectionKey)readyItor.next();
        readyItor.remove();
        if (key.isAcceptable()) {
          // Obtiene el canal y el socket y acepta la solicitud
          ServerSocketChannel keyChannel = (ServerSocketChannel)key.channel();
          ServerSocket serverSocket = keyChannel.socket();
          Socket socket = serverSocket.accept();
          // Envía la respuesta
          PrintWriter writer = new PrintWriter(socket.getOutputStream(), true);
          writer.println("Welcome");
          writer.close();
        } else {
          System.err.println("Ooops");
        }
      }
    }
  }
}
```

Truco

En la página de Java World (www.javaworld.com/javaworld/jw-09-2001/jw-0907-merlin.html) encontrará un ejemplo más complejo de servidores multitarea que emplean las nuevas clases E/S.

Las factory

Al echar una ojeada a las clases de `java.net`, habrá comprobado que la palabra *factory* aparece una y otra vez. Dicho paquete contiene tres interfaces que la llevan en sus nombres. Para entender cómo funcionan éstos, hay que saber lo que son las clases *factory* en el contexto de la programación orientada a objetos.

Como probablemente sabrá, la palabra inglesa *factory* significa fábrica, que es una entidad productora de bienes diversos. En la programación orientada a objetos, esta metáfora se aplica a las clases capaces de construir objetos de características dispares, aunque relacionadas, sin necesidad de invocar a los constructores en cada clase de objeto concreta. El modelo de diseño de fábricas se puede usar siempre que una clase necesite ejemplificar objetos de otras que aún no conoce.

La mayoría de los recursos Web se transmiten a través de buscadores mediante el protocolo HTTP. Sin embargo, hay ocasiones en que la dirección de la página comienza por ftp:// en lugar del típico "http://", que indica que el buscador recibió la orden de buscar un recurso de un protocolo distinto a HTTP. Si la URL de un recurso especifica un protocolo distinto de los que se soportan (HTTP, FTP, NNTP, etc.), el buscador puede tener problemas, ya que no lo reconocerá.

Un obstáculo análogo ocurre en las mismas páginas Web. La mayoría de los buscadores no tienen problemas para mostrar imágenes GIF o JPEG, pero sí con otros formatos diferentes que puedan llegar a ser bastante populares.

El buscador HotJava de Sun, escrito en código Java, es capaz de descargar clases que soportan nuevos protocolos y nuevos formatos de imágenes y archivos, basándose para ello en el diseño `Factory`. Las clases `factory` son capaces de construir nuevos protocolos y controladores que permiten al buscador soportar nuevos estándares sin modificar la estructura del mismo.

Evidentemente, las clases `factory` son incapaces de fabricar objetos nuevos sin más para afrontar retos inéditos (para que ello fuera posible, deberíamos pedir un esfuerzo mucho mayor a los especialistas en inteligencia artificial). El modo de funcionamiento de las clases `factory` consiste en cargar nuevas clases de protocolo o manipuladores de formato fuera de Internet. Esta operación se efectúa la primera vez que se encuentra el nuevo protocolo o formato; a partir de entonces, simplemente cargan las clases fuera del disco local del cliente, al igual que se hace con todas las demás. En resumen, el modelo de diseño de `factory` consiste en desacoplar los sistemas entre sí, aumentando de esta forma su grado de flexibilidad.

La figura 22.4 muestra el diseño de un `Factory` y cómo la clase `URL` se basa en ella para descubrir el contenido del recurso, cuyo proceso se explica a continuación con más detalle:

1. Una clase `client` quiere obtener el contenido de un recurso definido por una URL.
2. La clase `URL` abre una conexión HTTP con el servidor Web y crea un objeto `URLConnection`.
3. El objeto `URL` llama a `getContent()` a través del objeto `URLConnection`.
4. El `URLConnection` determina el tipo de recurso de que se trata desde el encabezamiento de respuesta HTTP (el campo `Content-type`).
5. El `URLConnection` llama a su `ContentHandlerFactory` para dotarle de una subclase `ContentHandler` adecuada.
6. Durante la ejemplificación que se realiza de este manipulador de contenidos, el `URLConnection` lo llama para que se encargue de este recurso específico.

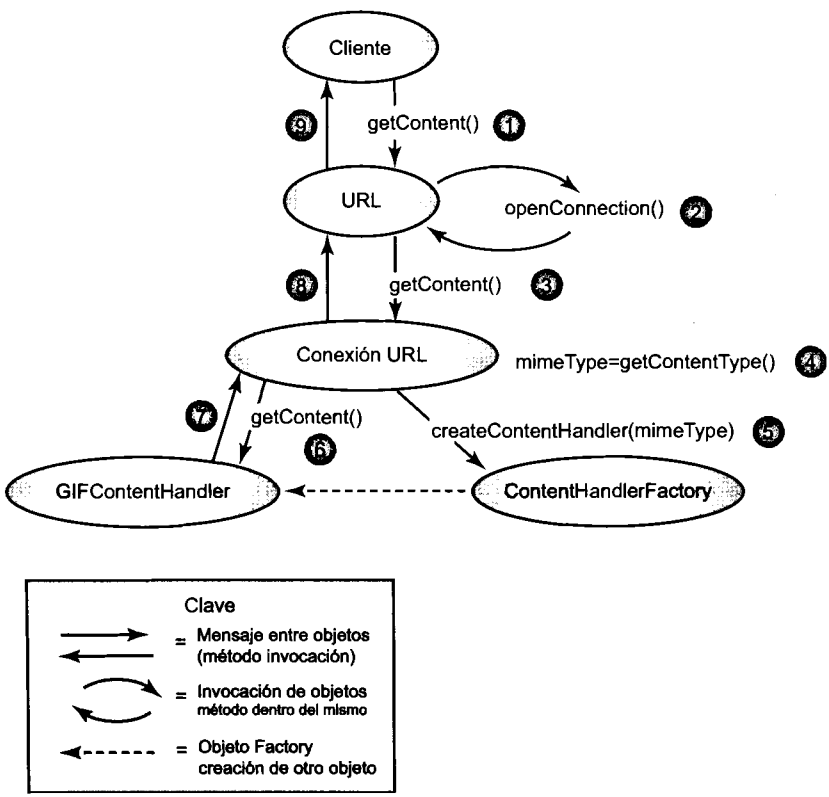
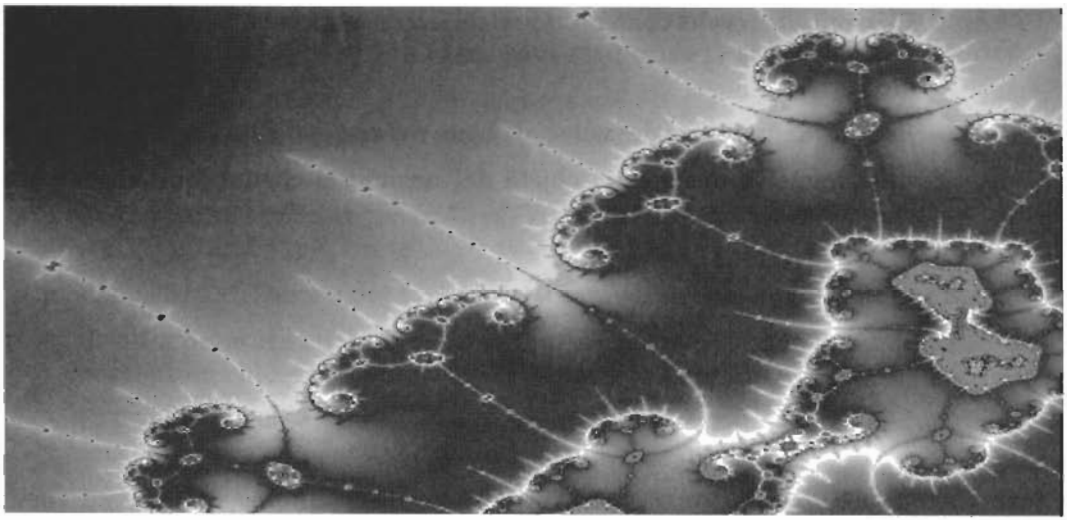


Figura 22.4. Cómo la clase URL utiliza una clase factory

7. El manipulador del contenido prefabricado devuelve el recurso bajo la forma más adecuada (`Image` para imágenes GIF, `String` para ficheros de texto, etc.).
8. El `URLConnection` devuelve el objeto del recurso a `URL`.
9. `URL` devuelve después el objeto del recurso al cliente (que ahora ya está contento).



23
Bibliotecas de extensiones de Java

Hace algún tiempo, toda la plataforma de Java podía explicarse en un único manual de mil páginas. Afortunada o desafortunadamente, en la actualidad, únicamente los cambios realizados en nuevas versiones ocupan más que lo que ocupaba la descripción de la primera versión de Java.

Para entender mejor los elementos disponibles en la edición estándar (J2SE) y la edición Enterprise (J2EE), exploraremos en este capítulo las numerosas novedades de los paquetes estándares y de las extensiones, para hacernos al menos una idea aproximada de las posibilidades que nos ofrecen las bibliotecas Java.

La edición estándar de Java (J2SE)

La edición estándar de Java es la que obtenemos al descargar el Kit de desarrollo de Software de Java (SDK) de http://java.sun.com/j2se/. Esta versión contiene un conjunto de las bibliotecas disponibles en las diferentes versiones de la plataforma. La plataforma 1.4, por ejemplo, contiene la biblioteca XML (de lenguaje extensible de marcado) mientras que en las versiones anteriores, era necesario instalarla manualmente.

Si exploramos la información disponible en http://java.sun.com/j2se/1.4/docs/ descubriremos que casi todos los temas correspondientes a las funciones generales se tratan en este libro. En este capítulo describiremos algunos temas que aún no se han comentado, incluidas algunas de las funciones de la versión Enterprise.

XML

El lenguaje XML es similar a HTML, aunque no dispone de un conjunto de etiquetas predefinidas como es el caso del segundo. Podemos utilizar XML para crear un formato común para compartir información en Internet. A diferencia de HTML, que emplea etiquetas que describen el formato de una página Web, las etiquetas de XML identifican los datos que contiene la página. El siguiente documento es un ejemplo que describe una agenda telefónica:

Listado 23.1. Phonebook.xml

```
<PHONEBOOK>
<PERSON>
  <NAME>George Bush</NAME>
  <EMAIL>president@whitehouse.gov</EMAIL>
  <TELEPHONE>202-456-1414</TELEPHONE>
  <WEB>www.whitehouse.gov</WEB>
</PERSON>
<PERSON>
  <NAME>Karol Josef Wojtyla</NAME>
  <EMAIL>pope@gabriel.vatican.va</EMAIL>
  <TELEPHONE>39 (6) 69.89.15.02</TELEPHONE>
  <WEB>www.vatican.va</WEB>
</PERSON>
<PERSON>
  <NAME>Newt Gingrich</NAME>
  <EMAIL>ngingrich@gingrichgroup.com</EMAIL>
  <TELEPHONE>202-414-1330</TELEPHONE>
  <WEB>www.newt.org</WEB>
</PERSON>
<PERSON>
  <NAME>Miss Cleo</NAME>
  <EMAIL>support@mindandspirit.com</EMAIL>
  <TELEPHONE>800-940-6194</TELEPHONE>
  <WEB>www.mindandspirit.com</WEB>
</PERSON>
</PHONEBOOK>
```

La etiqueta `PHONEBOOK` indica que se trata de una agenda telefónica, dentro de la cual existe un conjunto de elementos `PERSON`. Para cada persona, se asigna un nombre, una dirección de correo electrónico, un número de teléfono y una dirección de Internet.

Las bibliotecas de J2SE nos permiten transformar documentos XML mediante XSLT, y analizarlos mediante SAX o DOM.

Transformación de documentos XML mediante XSLT

El lenguaje de estilo extensible para transformaciones XSLT crea un segundo documento que describe las transformaciones que deseamos realizar y nos permite obtener el contenido del primer documento para crear un tercero. Java 1.4 propor-

ciona un procesador XSLT que realiza gran cantidad de transformaciones distintas, para lo cual sólo es necesario cambiar el archivo XSL que describe estas transformaciones.

Nota

Para aprender más sobre la sintaxis XSLT, visite la página http://www.zvon.org.

Si quisiéramos crear una tabla HTML a partir del listado 23.1, las transformaciones que se muestran en el listado 23.2 extraerían el nombre, la dirección de correo electrónico y la dirección de Internet del archivo XML, organizaría los resultados y los enviaría a la consola. El programa que realiza estas transformaciones está representado en el listado 23.3.

Listado 23.2. directory.xsl

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">
<xsl:template match="/">
<html>
<head>
<title>Directory</title>
</head>
<body>
<table border="1">
<tr>
  <th>Name</th>
  <th>Telephone</th>
  <th>Email</th>
</tr>
<xsl:for-each select="PHONEBOOK/PERSON">
  <xsl:sort/>
  <tr>
    <td><xsl:value-of select="NAME"/></td>
    <td><xsl:value-of select="TELEPHONE"/></td>
    <td><xsl:value-of select="EMAIL"/></td>
  </tr>
</xsl:for-each>
</table>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

Listado 23.3. Transform.java

```
import java.io.*;
import javax.xml.transform.*;
import javax.xml.transform.stream.*;
```

```
public class Transform {
  public static void main(String args[]) {
    if (args.length != 2) {
      System.err.println("Usage: java Transform xmlfile.xml
stylesheet.xsl");
      System.exit(-1);
    }
    try {
      StreamSource source = new StreamSource(args[0]);
      StreamSource stylesource = new StreamSource(args[1]);
      TransformerFactory factory = TransformerFactory.newInstance();
      Transformer transformer = factory.newTransformer(stylesource);
      StreamResult result = new StreamResult(System.out);
      transformer.transform(source, result);
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

Si ejecutáramos el programa `Transform` incluyendo la línea de comandos que se incluye a continuación, y mostráramos el archivo resultante, obtendríamos lo que se muestra en la figura 23.1.

```
java Transform phonebook.xml directory.xsl > phone.html
```

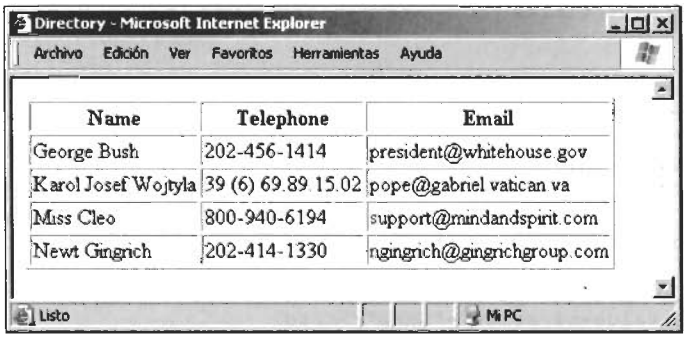


Figura 23.1. Ejemplo de búsqueda en la agenda

Además de estas transformaciones XSLT, otro aspecto importante son la asignación de formato y las instrucciones de procesamiento.

Análisis mediante SAX y DOM

SAX, o un API simple para XML, nos permite examinar el documento y reaccionar a eventos tales como el inicio o el final de una etiqueta. DOM, o Modelo de objetos del documento, guarda todo el documento XML en la memoria en forma de modelo de estructura de árbol.

El listado 23.4 demuestra el análisis realizado por SAX. El analizador utiliza un mecanismo de rellamada que invoca métodos de la clase `DefaultHandler` al

principio y al final de una etiqueta. Por ejemplo, cada vez que en este ejemplo aparece una etiqueta NAME se imprime su valor.

Listado 23.4. Análisis de archivos XML mediante SAX

```
import java.io.*;
import javax.xml.parsers.*;
import org.xml.sax.*;
import org.xml.sax.helpers.*;
public class NameLister {
  public static void main(String args[]) {
    if (args.length != 1) {
      System.err.println("Usage: java NameLister xmlfile.xml");
      System.exit(-1);
    }
    try {
      SAXParserFactory factory = SAXParserFactory.newInstance();
      SAXParser saxParser = factory.newSAXParser();
      DefaultHandler handler = new DefaultHandler() {
        boolean name = false;
        public void startElement(String uri, String localName,
                                 String qName, Attributes attributes)
            throws SAXException {
          if (qName.equalsIgnoreCase("NAME")) {
            name = true;
          }
        }
        public void characters(char ch[], int start, int length)
            throws SAXException {
          if (name) {
            System.out.println("Name: " + new String(ch, start, length));
            name = false;
          }
        }
      };
      saxParser.parse(args[0], handler);
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

Si ejecutamos el programa NameLister con la siguiente línea de comandos:

```
java NameLister phonebook.xml
```

obtendremos los siguientes resultados:

```
Name: George Bush
Name: Karol Josef Wojtyla
Name: Newt Gingrich
Name: Miss Cleo
```

Para inquirir sobre la base de datos, en lugar de simplemente mostrar el nombre, emplearíamos SAX.

La estructura básica de DOM es igual que la de SAX, aunque para trabajar con DOM, obtendríamos un `DocumentBuilder` del `DocumentBuilderFactory`, en vez de un `SAXParserFactory` de `javax.xml.parsers`. Una vez analizado el archivo de entrada, obtenemos una estructura de árbol que podemos utilizar para añadir nuevos nódulos, reorganizar los ya existentes, o guardar los resultados en un nuevo archivo XML.

Seguridad de Java

Como con XML, existe un número casi ilimitado de libros sobre la seguridad de Java. La versión 1.4 de la plataforma Java 2 incluye aún más funciones estándar relacionadas con seguridad, entre las que se encuentran:

- Extensión Criptográfica de Java (JCE).
- Extensión de Seguridad de Socket (JSSE).
- Servicio de Acceso y Autentificación (JAAS).
- GSS-API (Servicio de Seguridad Genérica) para soporte de la versión 5 de Kerberos.
- Ruta de Certificado de Java.

Ya hace tiempo que los tres primeros API están disponibles, aunque anteriormente no eran parte estándar de J2SE. Las dos últimas bibliotecas, por el contrario, son nuevas.

Las bibliotecas JCE proporcionan apoyo para las tareas de encriptación y descriptación de datos formados por flujos de bytes. Podemos también escribir directamente dentro del flujo ya encriptado, mediante `CipherOutputStream` o tratarlo como un filtro para añadirle otro flujo de datos y así disponer de otros elementos más complejos que simples bytes o flujos. En el listado 23.5, se añade un `ObjectOutputStream` al `CipherOutputStream` para poder así encriptar los elementos. Antes de encriptar la información es necesario generar primero la clave. La clase `DESKeySpec` utilizada en el listado 23.5 aplica una encriptación DES basada en generar una única clave para las tareas de encriptación y descriptación. Java también dispone de soporte para los algoritmosTriple-DES, Diffie-Hellman, PBE, RC2, y RC5.

Una vez generada la clave, el siguiente paso consiste en generar un `Cipher`, que es el que se encarga en realidad de realizar la encriptación. La plataforma Java 2 se construye basándose en una estructura suministrador por razones de seguridad. Al crear un objeto suministrador de seguridad, nunca se llama a un constructor, sino al método `getInstance()` del objeto correspondiente, transmitiendo una cadena que describe un algoritmo. La arquitectura Java transmite la cadena a todos los suministradores registrados para que el primero que la entienda cree el suministrador necesario. Gracias a esta técnica, es muy sencillo añadir suministradores adicionales en caso de que los que tengamos no sean suficientes.

Una vez tenemos el objeto `Cipher`, es necesario inicializarlo antes de poder utilizarlo. Como vemos en el listado 23.5, el método `init()` requiere un modo. Para la tarea de encriptación, transmitimos el modo `Cipher.ENCRYPT_MODE`, y para la de descriptación, el modo `Cipher.DECRYPT_MODE`. Si deseamos realizar estas tareas en otro programa distinto, debemos proporcionar una manera para que la clave `cipher` pase de un programa a otro

Listado 23.5. Tareas de encriptación y descriptación mediante JCE

```
import java.io.*;
import javax.crypto.*;
import javax.crypto.spec.*;
import java.awt.*;
import java.util.*;
public class TestCipher {
  // La contraseña debe contener al menos 8 caracteres
  private static final String password = "zukowski";
  public static void main(String args[]) throws Exception {
    Set set = new HashSet();
    Random random = new Random();
    for (int i=0; i<10; i++) {
      Point point = new Point(random.nextInt(1000), random.nextInt(2000));
      set.add(point);
    }
    int last = random.nextInt(5000);
    // Crea la clave
    byte key[] = password.getBytes();
    DESKeySpec desKeySpec = new DESKeySpec(key);
    SecretKeyFactory keyFactory = SecretKeyFactory.getInstance("DES");
    SecretKey secretKey = keyFactory.generateSecret(desKeySpec);
    // Crea el Cipher
    Cipher desCipher = Cipher.getInstance("DES/ECB/PKCS5Padding");
    desCipher.init(Cipher.ENCRYPT_MODE, secretKey);
    // Crea el flujo
    FileOutputStream fos = new FileOutputStream("out.des");
    BufferedOutputStream bos = new BufferedOutputStream(fos);
    CipherOutputStream cos = new CipherOutputStream(bos, desCipher);
    ObjectOutputStream oos = new ObjectOutputStream(cos);
    // Escribe objetos
    oos.writeObject(set);
    oos.writeInt(last);
    oos.flush();
    oos.close();
    // Cambia el modo del cipher
    desCipher.init(Cipher.DECRYPT_MODE, secretKey);
    // Crea flujo
    FileInputStream fis = new FileInputStream("out.des");
    BufferedInputStream bis = new BufferedInputStream(fis);
    CipherInputStream cis = new CipherInputStream(bis, desCipher);
    ObjectInputStream ois = new ObjectInputStream(cis);
    // Lee los objetos
    Set set2 = (Set)ois.readObject();
    int last2 = ois.readInt();
    ois.close();
```

```
    // Compara el original con la lectura anterior
    int count = 0;
    if (set.equals(set2)) {
      System.out.println("Set1: " + set);
      System.out.println("Set2: " + set2);
      System.out.println("Sets are okay.");
      count++;
    }
    if (last == last2) {
      System.out.println("int1: " + last);
      System.out.println("int2: " + last2);
      System.out.println("ints are okay.");
      count++;
    }
    if (count != 2) {
      System.out.println("Problem during encryption/decryption");
    }
  }
}
```

Al ejecutar el programa `TestCipher` obtenemos el resultado que se muestra a continuación, o algo muy similar, ya que los datos se incluyen arbitrariamente.

```
>java TestCipher
Set1: [java.awt.Point[x=684,y=1269], java.awt.Point[x=846,y=130],
  java.awt.Point[x=205,y=56], java.awt.Point[x=602,y=819],
  java.awt.Point[x=20,y=1727], java.awt.Point[x=173,y=280],
  java.awt.Point[x=750,y=1121], java.awt.Point[x=829,y=1174],
  java.awt.Point[x=71,y=210], java.awt.Point[x=188,y=515]]
Set2: [java.awt.Point[x=188,y=515], java.awt.Point[x=71,y=210],
  java.awt.Point[x=829,y=1174], java.awt.Point[x=750,y=1121],
  java.awt.Point[x=173,y=280], java.awt.Point[x=20,y=1727],
  java.awt.Point[x=602,y=819], java.awt.Point[x=205,y=56],
  java.awt.Point[x=846,y=130], java.awt.Point[x=684,y=1269]]
Sets are okay.
int1: 4530
int2: 4530
ints are okay.
```

Nota

Una lectura rápida del contenido de `Set1` y `Set2` del resultado de `TestCipher` muestra que los elementos son diferentes. Estos conjuntos no necesitan mantener un orden específico, sólo que se incluyan dentro de la misma colección.

Las bibliotecas de encriptación, parte estándar de la versión 1.4, no proporcionan en realidad un potencial ilimitado; Para obtener este potencial, visite la página http://java.sun.com/j2se/1.4/index.html#jcepolicyfiles desde donde descargar el programa adecuado. La nueva biblioteca de extensión Socket de seguridad de Java (JSSE) ofrece una nueva función, antes sólo disponible para los applets y que permi-

te leer solicitudes de HTTP de manera segura, conocido también como HTTPS. Los applets se ejecutan en buscadores que, a su vez, soportan HTTPS, por lo que los applets pueden servirse de ello. Las aplicaciones de J2SE 1.4 pueden también beneficiarse de estas funciones, sin necesidad de instalar la implementación de referencia u otras bibliotecas de terceros.

Conectividad de las bases de datos de Java (JDBC)

El API (JDBC) permite a los desarrolladores crear bases de datos de comunicaciones sin tener que rescribir el código continuamente. A pesar de los estándares establecidos por el comité ANSI, cada distribuidor de sistemas de bases de datos tiene una manera propia de conectarse, utilizando incluso su sistema particular.

Esta necesidad de crear aplicaciones independientes y aplicaciones para la Web ha llevado a considerar la utilización de las soluciones de conectividad de Java. Los desarrolladores de software de terceros han intentado solucionar este problema de conectividades mediante soluciones de propiedad, a través de métodos nativos para integrar las bibliotecas de parte del cliente o para crear un tercer nivel y un nuevo protocolo.

La División de Software de Java ha trabajado en conjunción con los distribuidores de herramientas de bases de datos para crear un mecanismo independiente de DBMS que permita escribir aplicaciones de cliente sin importar el tipo de bases de datos utilizado. El resultado es el API JDBC.

JDBC proporciona un único API uniforme e independiente que proporciona el estándar de escritura, considerando todas las diferentes aplicaciones en el mercado. El secreto reside en un conjunto de interfaces de Java implementadas por un driver que convierte las llamadas JDBC en las llamadas específicas requeridas por la base de datos soportada. En la figura 23.2, la aplicación se escribe sólo una vez y después se transmite a los diferentes drivers. Los driver pueden utilizarse para crear el nivel medio de una base de datos de múltiples niveles, conocida como middleware, y que se representa en la imagen 23.2.

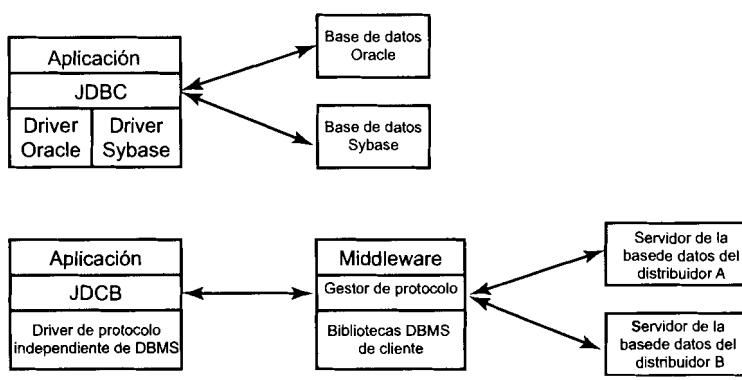


Figura 23.2. Diseño de la base de datos JDBC

Además de esta estructura uniforme e independiente, JDBC permite mantener la funcionalidad específica ofrecida por los distintos distribuidores. Los drivers compatibles con JDBC deben soportar el nivel de entrada estándar de ANSI SQL-2, pero además JDBC permite transmitir cadenas de solicitud directamente al driver conectado. Estas cadenas no tienen por qué ser ANSI SQL, ni siquiera SQL, ya que el driver decide la manera de emplearlas. Esta función limita sin embargo la libertad del desarrollador para cambiar los modos de comunicación de las bases de datos.

Existen dos niveles distintos de interfaces en el API de JDBC: la de la aplicación en sí, que es desde la que el desarrollador utiliza el API para llamar a la base de datos mediante SQL y obtener los resultados, y el correspondiente al driver que gestiona todas las comunicaciones con la implementación del driver concreto.

Cada aplicación JDBC (o applet) debe tener al menos un driver JDBC específico según el tipo de DBMS utilizado, que no tiene que estar obligatoriamente conectado con una base de datos específica.

Los componentes API

De estos dos niveles mencionados, el más importante de entender es el correspondiente al driver que se va a implementar por los diferentes distribuidores, y sobre todo el hecho de que el driver crea algunos de los objetos que se van a utilizar en el nivel de la aplicación. La figura 23.3 muestra la conexión entre ambos niveles.

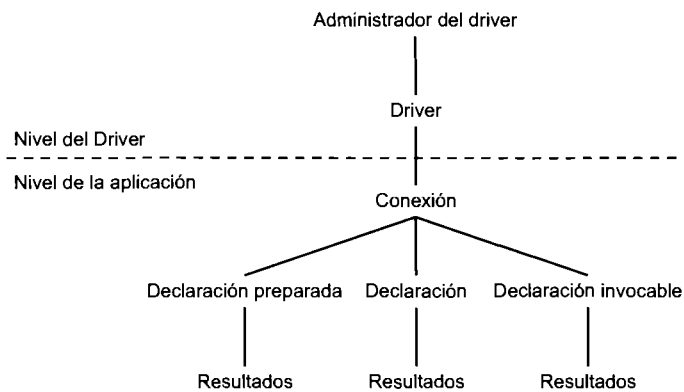


Figura 23.3. Componente API de JDCB

Afortunadamente, los desarrolladores de aplicaciones sólo tienen que utilizar las interfaces estándares de API para garantizar que sea compatible con JDBC. El desarrollador del driver es responsable de crear el código que se interrelacione con la base de datos y soporte las llamadas realizadas en el nivel de la aplicación.

Cada driver debe implementar cuatro interfaces distintas, y una clase que comunique ambos niveles. Las cuatro interfaces son `Driver`, `Connection`, `Statement`, y `ResultSet`. La implementación de la primera interfaz es donde se produce la conexión con la base de datos. En la mayoría de las aplicaciones, se accede a esta

interfaz mediante la clase `DriverManager`, que proporciona un nivel más de abstracción.

EL distribuidor del driver implementa las interfaces `Connection`, `Statement`, y `ResultSet`, permitiéndolas que representen los métodos que el desarrollador de la aplicación va a tratar como clases de objetos reales. También permite al desarrollador crear nuevas declaraciones y obtener resultados. La distinción que aquí estamos haciendo entre los dos niveles es artificial, pero permite al desarrollador crear aplicaciones de bases de datos sin preocuparse por el origen de los objetos, ni por el driver que va a utilizar dicha aplicación.

Drivers JDBC

Uno de los verdaderos atractivos del API de JDBC reside en que nos permite crear aplicaciones con el conocimiento de que los distribuidores de las bases de datos más importantes disponen de drivers adecuados. Estos drivers están disponibles a través del distribuidor de la base de datos o por desarrolladores de terceros. Es una buena idea comparar las funciones que ofrecen y los precios antes de decidirnos por uno en concreto.

Sun define cuatro categorías de drivers de bases de datos:

- **Tipo 1:** Un driver que hace de puente entre JDBC y ODBC implementado con un código binario ODBC, y que se divide en tres partes: Un grupo de bibliotecas C que conectan el programa JDBC al administrador de ODBC; este administrador y el driver ODBC.

Truco

ODBC es un API muy común de acceso a la base de datos introducido por Microsoft. Hay drivers ODBC disponibles para todos los sistemas de bases de datos de PC, que además permiten migrar aplicaciones más antiguas a JDBC sin tener que encontrar el driver específico para JDBC.

- **Tipo 2:** Una implementación de la biblioteca de Java. Este driver utiliza las llamadas de la biblioteca nativa C para transmitir las llamadas JDBC a la biblioteca del cliente. Las bibliotecas de lenguaje C proporcionan una funcionalidad específica del distribuidor que las comunica con el JDBC.
- **Tipo 3:** Un driver de protocolo de red independiente de DBMS que convierte las llamadas de JDBC en un protocolo independiente de DBMS y envía un servidor de medio nivel al socket, cuyo código puede contactar con una gran variedad de bases de datos, en nombre del cliente. Este tipo de driver es el más popular y flexible, y está además relacionado con seguridad en la red, incluyendo la transmisión de datos a través de *firewalls*.
- **Tipo 4:** Un driver de protocolo nativo que permite convertir las llamadas de JDBC directamente en el tipo de protocolo que utiliza el servidor DBMS. El

distribuidor de la base de datos soporta un socket de red, y el driver JDBC conecta directamente el socket y el servidor a la base de datos. El código del cliente puede escribirse en Java, lo cual cuenta con la ventaja de ser una de las maneras más sencillas y prácticas de realizar implementaciones para Intranet. Aunque el protocolo de red viene definido por el distribuidor, el driver proviene del distribuidor de la base de datos.

Nivel de la aplicación

En el nivel de la aplicación encontramos tres interfaces que se implementan al nivel del driver, pero que utiliza el desarrollador de la aplicación. En Java, una interfaz admite un nombre general para indicar un objeto concreto, que además define los métodos que las clases de ese objeto deben implementar. Para el desarrollador de la aplicación, esto significa que la implementación de la clase `Driver` es irrelevante, ya que sólo será necesario utilizar el código estándar de los API de JDBC.

No debemos olvidar, que para ello, la base de datos debe soportar al menos un nivel de entrada ANSI SQL-2.

Las tres interfaces que se utilizan en este nivel son `Connection`, `Statement`, y `ResultSet`. Un objeto `Connection` se obtiene a partir de la implementación del driver, mediante la llamada del método `DriverManager.getConnection()`. Una vez dispone de este objeto, el desarrollador de la aplicación puede crear un objeto `Statement` que tiene como resultado un objeto `ResultSet`.

La interfaz `Connection` representa una sesión con la conexión a la base de datos realizada por el `Driver`. Las conexiones más frecuentes tienen la habilidad de controlar los cambios producidos en los datos almacenados mediante transiciones. Una transición es un conjunto de operaciones completas.

La primera función de la interfaz `Connection` consiste en crear una declaración:

```
Connection msqlConn;
Statement stmt;
msqlConn = DriverManager.getConnection(url);
stmt = msqlConn.createStatement();
```

Esta declaración puede emplearse para enviar líneas de código SQL que devuelven un único resultado en una referencia al objeto `ResultSet`, o el número de elementos afectados por dicho código. Las proposiciones que deben llamarse varias veces con pequeñas variaciones pueden ejecutarse de una manera más eficaz mediante `PreparedStatement`. La interfaz `Connection` se utiliza también para crear un `CallableStatement` cuya función principal es ejecutar procedimientos almacenados.

Las declaraciones pueden devolver o no un objeto `ResultSet`, dependiendo del método `Statement` utilizado. El método `executeUpdate()`, por ejemplo, se utiliza para ejecutar declaraciones SQL que no esperan un resultado, excepto el estado de número de filas.

```
int rowCount;
rowCount = stmt.executeUpdate(
  "DELETE FROM customer WHERE CustomerID = "Example"");
```

Las declaraciones SQL que no devuelven un único conjunto de resultados pueden emplear el método `executeQuery()`, que devuelve un objeto `ResultSet`, que representa la información sobre la fila:

```
ResultSet results;
results = stmt.executeQuery ("SELECT * FROM stock");
```

Las declaraciones SQL que ejecutan procedimientos almacenados (o producen uno) pueden devolver más de un conjunto de resultados. El método `execute()` es un método general que puede devolver un único conjunto de resultados, el número, o ambos. Este método devuelve un marcador booleano que sirve para determinar si existen o no más resultados. Puesto que un conjunto de resultados puede componerse de datos o de número de filas, se deben utilizar los métodos `getResultSet()`, `getMoreResults()`, y `getUpdateCount()`:

```
// Si SQLString devuelve varios resultados
// execute() devuelve el valor true si se devuelve ResultSet
boolean result = stmt.execute(SQLString);
int count = stmt.getUpdateCount();
// Se desplaza por el código hasta que no hay más resultados ni nuevas
// actualizaciones del número de filas
while (result || (count != -1)) {
  // ¿Es ResultSet el resultado?
  if (result) {
    results = stmt.getResultSet();
    // Procesa el resultado
  } else if (count != -1) {
    // Procesa el número de filas
  }
  result = stmt.getMoreResults();
  count = stmt.getUpdateCount();
}
```

La interfaz `ResultSet` define métodos para acceder a tablas de datos generadas como resultado de ejecutar un `Statement`. A los valores de columnas de `ResultSet` se puede acceder en cualquier orden, a partir de un nombre o del número de la columna (de 1 a n). `ResultSet` mantiene la posición de la fila actual, comenzando por la primera fila de los datos devueltos. El método `next()` desplaza la posición hasta la siguiente fila de datos:

```
// Transmite una solicitud al objeto statement
ResultSet rs = stmt.executeQuery("SELECT * FROM stock WHERE quantity = 0");
// Obtiene los resultados en forma de sus tipos de Java
// Observe que las columnas están indexadas por una integral que comienza
// por 1, o a partir del nombre de la columna, como en "ItemID"
System.out.println("Stock replenishment list");
while (rs.next()) {
```

```
    System.out.println("Item ID: " + rs.getString("ItemID"));
    System.out.println("Next ship date: " + rs.getDate(2));
    System.out.println();
}
```

Para más información sobre los API de JDBC consulte la página http://java.sun.com/docs/books/tutorial/jdbc/index.html.

Invocación de método remoto (RMI), CORBA, y Java IDL

La invocación RMI de Java permite que un programa en un equipo virtual de Java (JVM) realice llamadas de métodos a un objeto que se encuentra en un servidor remoto. Esta invocación se utilizaba en principio para comunicarse sólo con otro equipo JVM, pero las versiones más recientes de RMI permiten comunicarse con servicios CORBA.

CORBA (Arquitectura gestora de solicitud de objetos) es un marco diseñado para soportar arquitecturas heterogéneas. La RMI de Java proporcionaba soporte sólo para arquitecturas homogéneas de Java que estuviesen conectadas con un cable. CORBA (así como la última versión de RMI), por el contrario, permite que dos sistemas diferentes se conecten. Estos sistemas pueden tener la CPU y la memoria distintas, pero también su sistema operativo y el lenguaje de programación que emplean.

El lenguaje IDL (Lenguaje descriptivo de la interfaz) proporciona una implementación de la especificación CORBA 2.0. Tanto la nueva versión de RMI como CORBA se comunican mediante un protocolo conocido como protocolo de red específico de OMG (IIOP).

RMI

RMI permite a los programadores de Java distribuir códigos a través de la red. El diseño orientado a objetos requiere que cada tarea se ejecute por el objeto más apropiado para la misma. RMI comparte este concepto y lo lleva un paso más allá, requiriendo además que esta tarea se ejecute en el equipo más adecuado para ella.

RMI define una interfaz `Remote` para crear objetos remotos. Un cliente puede invocar los métodos de un objeto remoto con la misma sintaxis empleada para invocar métodos de un objeto local. El API de RMI proporciona clases y métodos para cubrir todos los aspectos de la comunicación, así como los requisitos de referencia a parámetros para acceder a métodos remotos. RMI también se encarga de la serialización de objetos que se pasan en forma de argumentos de los objetos remotos. Los paquetes `java.rmi` y `java.rmi.server` contienen las interfaces y las clases que definen los bloques de construcción para crear los objetos del servidor y las stub, o redes aisladas, de los objetos del cliente. Un stub es una representación local de un objeto remoto. El cliente realiza llamadas al stub, que a su vez se comunica con el servidor, de manera automática.

RMI consiste en tres niveles: el nivel de Stubs/Skeleton el de referencia remota, y el de transporte. La relación entre estos niveles se representa en la figura 23.4. Cuando un cliente invoca un método remoto, esta solicitud se realiza en el nivel superior, con el stub del cliente. El cliente se refiere al stub como el objeto proxy del equipo remoto. Toda la funcionalidad representada en la figura 23.4 es en realidad invisible para el cliente. El código stub se genera con un compilador `rmic` y utiliza el nivel de referencia remota (RRL) para transmitir las solicitudes de invocaciones a métodos a los objetos del servidor.

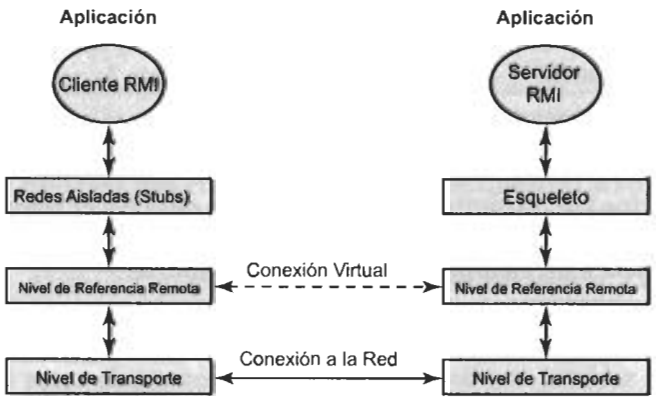


Figura 23.4. Arquitectura de RMI

Nivel Stubs/Skeleton

El stub es el proxy del cliente que representa al objeto remoto. Los stub definen todas las interfaces que la implementación del objeto remoto soporta y se hace referencia a ellos del mismo modo que si se tratara de un objeto local a través de un programa que se ejecuta en el equipo del cliente. El stub es un objeto local del cliente que además mantiene una conexión con el objeto del servidor. El nivel de referencia remota (RRL) del cliente devuelve un flujo de datos recodificados (marshal) al stub que utiliza el RRL para comunicarse con el RRL del servidor. El stub serializa los datos de parámetros, transmitiéndolos al flujo de estos datos.

Una vez ejecutado el método remoto, el RRL devuelve todos los valores serializados al stub, que se encarga de la tarea de deserialización.

La parte servidora o *skeleton* recibe las solicitudes de invocación a los métodos desde el RRL del cliente. El RRL del servidor debe deshacer la remodificación de cualquier argumento enviado al método remoto. También se encarga de llamar a la implementación real del objeto del servidor y de recibir cualquier valor devuelto por el objeto remoto y de recodificar de nuevo en el flujo (marshal).

Nivel de referencia remota (RRL)

El RRL se encarga de mantener un protocolo de referencia independiente que no sea específico de ningún modelo de stub o de *skeleton*. Esta flexibilidad nos permite

cambiar el RRL sin que los otros dos niveles se vean afectados. Este nivel funciona con la interfaz de transporte de nivel más bajo y proporciona un flujo de datos a los niveles de las partes del cliente (stub) y de las del servidor (skeleton).

EL RRL utiliza un componente de cliente y de servidor para comunicarse a través del nivel de transporte. El componente del cliente contiene información específica del servidor remoto que se transmite al componente del servidor, que, por lo tanto, es dependiente sólo del RRL del servidor. Éste es responsable de las referencias y de gestionarlas antes de transmitir la llamada al método remoto. El nivel de transporte se encarga de la comunicación entre los componentes del servidor y del cliente.

El nivel de transporte

Este nivel se encarga de crear y mantener conexiones entre el cliente y el servidor y consiste en cuatro abstracciones:

- Un `endpoint`, o punto final, que se utiliza para hacer referencia al espacio de la dirección que contiene un JVM. En otras palabras, un `endpoint` es una referencia a un ejemplo específico de transporte.
- Un `channel`, o canal, es la ruta entre dos espacios de direcciones que gestiona la comunicación entre el cliente y el servidor, y viceversa.
- Una `connection` es una abstracción para transferir datos (tanto argumentos como valores) entre el cliente y el servidor.
- La abstracción `transport` se encarga de determinar un canal entre un espacio de dirección local y un `endpoint` remoto y de aceptar conexiones al espacio de dirección que contiene la abstracción.

El nivel de transporte establece conexiones, gestiona las ya existentes, así como los objetos remotos de su espacio de direcciones. Cuando recibe una solicitud del RRL del cliente, este nivel establece una conexión de socket con el servidor, la transmite al RRL del cliente y añade una referencia al objeto remoto en la tabla interna. En este punto, el cliente se conecta al servidor.

El nivel de transporte controla la conexión, de modo que si pasa algún tiempo sin que exista ningún tipo de actividad, se encarga de cerrar dicha conexión. El tiempo de espera por defecto es de 10 minutos.

Un ejemplo de RMI

Esta sección describe todos los pasos a seguir para crear, compilar y ejecutar una aplicación RMI que sea accesible para clientes remotos:

1. Definir las interfaces para las clases remotas.
2. Crear y compilar las clases de implementación para las clases remotas.
3. Crear clases `stub` y `skeleton` mediante el comando `rmic`.

4. Crear y compilar una aplicación de servidor.
5. Comenzar el registro RMI y la aplicación del servidor.
6. Crear y compilar un programa de cliente para acceder a los objetos remotos.
7. Comprobar el cliente.

Siguiendo estos pasos, podemos desarrollar un ejemplo detallado de una aplicación RMI que representa un sistema muy sencillo de tarjeta de crédito. El servidor soportará la creación de una nueva cuenta, además de la realización de transacciones. Ya que el objetivo de este ejemplo consiste en mostrar el funcionamiento de un RMI, no se incluye una interfaz de usuario del cliente, sino que el cliente realizará algunas invocaciones al código.

Paso 1: Definir las interfaces para las clases remotas

El programa utilizará dos clases remotas: la clase `CreditCardImpl`, que mantiene el nombre de usuario, el balance, el crédito disponible, y el número de identificación para una tarjeta de crédito, y la clase `CreditManager`, que mantiene una lista de objetos `Account` y crea nuevos cuando es necesario. La aplicación del servidor construye una sola muestra de de `CreditManagerImpl` y la hace disponible para clientes remotos.

Cada uno de estos elementos debe describirse por una interfaz (`CreditCard` y `CreditManager`). Los stubs del cliente implementan estas interfaces. Estas clases se crean en un paso posterior, mediante la utilidad `rmic` que debe cumplir los siguientes requisitos:

- Las interfaces deben ser públicas y ser extensiones de la interfaz `Remote`.
- Cada método debe arrojar un `RemoteException`.
- El stub y el código de implementación deben estar en el paquete.

La definición de la interfaz `CreditCard` se muestra a continuación.

```
package credit;
import java.rmi.*;
public interface CreditCard extends Remote {
  /** Este método devuelve una línea de crédito para la tarjeta */
  public float getCreditLine() throws RemoteException;
  /** Este método permite al propietario de la tarjeta pagar todo o parte
      del balance. Arroja un InvalidMoneyException si el parámetro money no
      es válido. */
  public void payTowardsBalance(float money) throws
    RemoteException, InvalidMoneyException;
  /** Este método permite al propietario realizar compras con la línea de
      crédito. Arroja una CreditLineExceededException
      Si la compra supera el límite aceptado. */
  public void makePurchase(float amount, int signature) throws
    RemoteException, InvalidSignatureException,
    CreditLineExceededException;
```

```
  /** Este método establece el número identificador. */
  public void setSignature(int pin) throws RemoteException;
}
```

La estructura de `CreditManager` es similar, aunque sólo define dos métodos:

```
package credit;
import java.rmi.*;
public interface CreditManager extends Remote {
  /** Este método encuentra una tarjeta ya existente asignada al nombre de
      un cliente. Si el cliente no dispone de cuenta, se emite una con un
      número personal preestablecido y una línea de crédito de $5000*/
  public CreditCard findCreditAccount(String Customer) throws
    DuplicateAccountException, RemoteException;
  /** Este método crea una nueva cuenta de crédito con un número personal
      preestablecido y una línea inicial de crédito de $5000*/
  public CreditCard newCreditAccount(String newCustomer) throws
    RemoteException;
}
```

En este caso existen cuatro clases de excepción: `CreditLineExceededException`, `DuplicateAccountException`, `InvalidMoneyException`, e `InvalidSignatureException`. Estas clases son extensiones de `Exception` sin añadir nuevos datos ni métodos.

```
package credit;
public class CreditLineExceededException extends Exception {
}
package credit;
public class DuplicateAccountException extends Exception {
}
package credit;
public class InvalidMoneyException extends Exception {
}
package credit;
public class InvalidSignatureException extends Exception {
}
```

Paso 2: Crear y compilar las clases de implementación

Las clases de implementación son clases del servidor que implementan las interfaces ya mencionadas. La clase `CreditCardImpl` implementa la interfaz `CreditCard` definida en el paso primero. Esta clase debe implementar todos los métodos de la interfaz `CreditCard` y complementar un `UnicastRemoteObject`. El listado 23.6 contiene el código de la clase `CreditCardImpl`.

> **Nota**
>
> Cuando se trabaja con RMI, los nombres de las clases de implementación suelen terminar en `Impl`. En el lado del cliente, sin embargo, se hace referencia a la interfaz y no a los nombres de las clases.

Listado 23.6. CreditCardImpl.java

```
package credit;
import java.rmi.*;
import java.rmi.server.*;
import java.io.Serializable;
/** Esta clase es el objeto remoto al que el nivel skeleton va a hacer
    referencia en el lado del servidor y al que el stub hará referencia en
    el lado del cliente */
public class CreditCardImpl extends UnicastRemoteObject
    implements CreditCard, Serializable {
  private float currentBalance = 0;
  private float creditLine = 5000f;
  private int signature = 0;          // Como un número pin
  private String accountName;         // Nombre del propietario
  /** El constructor de la clase genera un pin inicial.*/
  public CreditCardImpl (String customer) throws
      RemoteException, DuplicateAccountException {
    accountName = customer;
    signature = (int)(Math.random() * 10000);
  }
  /** Devuelve la línea de crédito. */
  public float getCreditLine() throws RemoteException {
    return creditLine;
  }
  /** Paga algunas deudas. */
  public void payTowardsBalance(float money) throws
      RemoteException, InvalidMoneyException {
    if (money <= 0) {
      throw new InvalidMoneyException ();
    } else {
      currentBalance -= money;
    }
  }
  /** Cambia la identificación. */
  public void setSignature(int pin) throws RemoteException {
    signature = pin;
  }
  /** Realiza una compra asegurando que existe crédito suficiente, y
      aumenta o disminuye el crédito disponible. */
  public void makePurchase (float amount, int signature) throws
      RemoteException, InvalidSignatureException,
      CreditLineExceededException {
    if (signature != this.signature) {
      throw new InvalidSignatureException();
    }
    if (currentBalance+amount > creditLine) {
      throw new CreditLineExceededException();
    } else {
      currentBalance += amount;
      creditLine -= amount;
    }
  }
}
```

La clase `CreditManagerImpl` representada en el listado 23.7 se encarga de crear y almacenar nuevas cuentas como objetos `CreditImpl`. Esta clase almacena los objetos de la cuenta dentro de un mapa `HashMap`, accesible mediante el nombre del cliente.

Listado 23.7. CreditManagerImpl.java

```
package credit;
import java.rmi.*;
import java.rmi.server.*;
import java.util.HashMap;
public class CreditManagerImpl extends UnicastRemoteObject
    implements CreditManager {
  private static transient HashMap accounts = new HashMap();
  /** Este es el constructor de la clase por defecto que se limita a llamar
      a super(). */
  public CreditManagerImpl() throws RemoteException {
  }
  /** Crea una nueva cuenta. Incluye el nombre del cliente y la tarjeta en
      la hashtable. */
  public CreditCard newCreditAccount(String customerName)
      throws RemoteException {
    CreditCardImpl newCard = null;
    try {
      newCard = new CreditCardImpl(customerName);
    } catch (DuplicateAccountException e) {
      return null;
    }
    accounts.put(customerName, newCard);
    return newCard;
  }
  /** Busca una cuenta ya existente en la hashtable. Si no existe se crea
      una nueva y se añade a la tabla. Devuelve la cuenta. */
  public CreditCard findCreditAccount(String customer)
      throws DuplicateAccountException, RemoteException {
    CreditCardImpl account = (CreditCardImpl)accounts.get(customer);
    if (account != null) {
      return account;
    }
    // Crea una nueva cuenta. Añade la tarjeta a la hashtable.
    account = new CreditCardImpl(customer);
    accounts.put(customer, account);
    return account;
  }
}
```

Todas las interfaces, clases y excepciones declaran que pertenecen al paquete de crédito. Cada uno de los archivos fuente debe compilarse con la opción `-d directoryname` para especificar un directorio destino para los archivos `.class` resultantes.

Dentro del directorio destino, el compilador creará de manera automática un subdirectorio llamado credit (si aún no existe uno); los archivos de las clases se

crean en el subdirectorio de credit. El directorio destino proporcionado a la opción `-d` debe estar en CLASSPATH.

Un modo sencillo de compilar las interfaces y las clases consiste en especificar el directorio de trabajo que se esté utilizando como el directorio destino, de la siguiente manera:

```
javac -d . CreditLineExceededException.java
javac -d . DuplicateAccountException.java
javac -d . InvalidMoneyException.java
javac -d . InvalidSignatureException.java
javac -d . CreditCard.java
javac -d . CreditCardImpl.java
javac -d . CreditManager.java
javac -d . CreditManagerImpl.java
```

Paso 3: Crear clases stub y skeleton

Una vez compiladas las clases de implementación, el siguiente paso consiste en crear los archivos para las clases `stub` y `skeleton` que sirven para acceder a las primeras. El código del cliente utiliza las clases `stub` para comunicarse con el código de las clases `skeleton` del servidor.

El comando `rmic` crea automáticamente el código de `stub` y `skeleton` a partir de las definiciones de las clases, para la interfaz y la implementación. La sintaxis de `rmic` es la siguiente:

```
rmic [options] package.interfaceImpl ...
```

El siguiente código crea los `stub` y `skeleton` para las clases remotas `CreditCard` y `CreditManager`:

```
rmic -d . credit.CreditCardImpl credit.CreditManagerImpl
```

Como podemos observar, el comando requiere la especificación del paquete en que se encuentra la clase. Esta es la razón por la que todos los módulos de código del paso 2 declaraban que pertenecían al paquete de crédito.

El comando `rmic` crea cuatro archivos de clase en el directorio del paquete `credit`:

```
CreditCardImpl_Skel.class
CreditCardImpl_Stub.class
CreditManagerImpl_Skel.class
CreditManagerImpl_Stub.class
```

Una vez creadas las clases `skeleton` y `stub`, el siguiente paso es crear una aplicación de servidor que ponga estas clases a disposición de la invocación remota.

Paso 4: Crear y compilar la aplicación del servidor

Todo está listo para crear la aplicación del servidor. Habrá una clase de aplicación llamada `CardBank`, cuya definición se muestra en el listado 23.8, que se encar-

ga de construir un ejemplo de `CreditManager`. Exceptuando la línea que llama al constructor `CreditManager`, todo el código de `CardBank` se encarga de hacer disponible el objeto del administrador para clientes remotos.

Listado 23.8. CardBank.java

```
package credit;
import java.rmi.*;
import java.rmi.registry.LocateRegistry;
public class CardBank {
  public static void main(String args[]) {
    if (args.length != 2) {
      System.err.println("Usage:");
      System.err.println("java credit.CardBank <server> <port>");
      System.exit(-1);
    }
    String server = args[0];
    int port = Integer.parseInt(args[1]);
    // Crea e instala un administrador de seguridad.
    System.setSecurityManager(new RMISecurityManager());
    try {
      // Especifica la ubicación del registro
      LocateRegistry.createRegistry(port);
      System.out.println("Card Bank: defined registry port");
      // Crea un ejemplo del manager de crédito.
      System.out.println("CreditManagerImpl: create a CreditManager");
      CreditManagerImpl cmi = new CreditManagerImpl();
      // Conecta el objeto con el registro remoto. Utiliza el método
      // estático rebind() para evitar conflictos.
      System.out.println("CreditManagerImpl: bind it to a name");
      String urlString = "//" + server + ":" + port + "/" + "cardManager";
      Naming.rebind(urlString, cmi);
      System.out.println("CreditManager is now ready");
    } catch (Exception e) {
      System.out.println("An error occured");
      e.printStackTrace();
      System.err.println(e.getMessage());
    }
  }
}
```

Las aplicaciones se ejecutan sin administradores de seguridad. En el método principal mostrado anterior a la llamada, `System.setSecurityManager()` instala un administrador de seguridad RMI.

Código local seguro mediante el Administrador de seguridad

Al instalar el administrador de seguridad, de manera automática se pone en duda todo el código que no se haya descargado de la ruta de propiedades de sistema `sun.boot.class.path`. Para que se considere seguro el ejemplo de RMI para que así funcione, es necesario añadir el directorio adecuado al

archivo java.policy, como se muestra a continuación, para asegurarse de que se encuentra dentro de la ruta de la clase de arranque. Si está trabajando en un directorio distinto, la modificación será diferente. Puede además restringir aún más el acceso.

```
grant codeBase "file:/MasteringJava/ch23/" {
  permission java.security.AllPermission;
};
```

Encontrará el archivo java.policy en el subdirectorio de seguridad jre/lib/, dentro del directorio en el que se instaló 1.4 SDK.

El Registro RMI busca un puerto concreto en el que registrar información y solicita su servicio. Este puerto se especifica con una simple llamada a `LocateRegistry.createRegistry(port)`.

El servidor publica una instancia del objeto al incluir un nombre específico a dicha instancia y registrando el nombre mediante el Registro RMI. Existen dos métodos que permiten registrar y asignar un nombre a la instancia:

```
public static void bind(String name, Remote obj) throws
AlreadyBoundException,
   MalformedUrlException, UnknownHostException, RemoteException
public static void rebind(String name, Remote obj) throws
MalformedUrlException,
   UnknownHostException, RemoteException
```

Como puede observar, ambos métodos son estáticos y solicitan el nombre para hacer referencia al objeto y a la instancia del objeto remoto que se le ha asignado al nombre. En este caso, el nombre del objeto es `cardManager`. Cualquier equipo conectado a la Red puede hacer referencia a este objeto al especificar la máquina huésped del Registro, el puerto, y el nombre del objeto.

El argumento requerido para `bind()` y `rebind()` es una cadena URL, que puede tener el siguiente formato:

```
protocol://host:port/bindingName
```

donde el protocolo es rmi, el *host* es el nombre del servidor RMI, el *port* corresponde al número del puerto que el servidor va a utilizar para recibir solicitudes, y *bindingName* es el nombre exacto que debe utilizar el cliente al solicitar acceso al objeto. Si la cadena sólo incluye un nombre, se utilizan los siguientes valores por defecto: rmi para el protocolo, localhost para el nombre del servidor, y 1099 para el número de puerto. Tanto `bind()` como `rebind()` asocian un nombre a un objeto; sin embargo se comportan de manera distinta cuando el nombre asignado ya ha sido utilizado para asignarse a un objeto. En ese caso, `bind()` arroja la excepción `AlreadyBound-Exception`,mientras que `rebind()` ignora la asignación anterior y utiliza la nueva. En este punto, se compila el banco:

```
javac -d . CardBank.java
```

Paso 5: Comenzar el registro RMI y la aplicación del servidor

El Registry RMI es una aplicación que proporciona un servicio de búsqueda de nombres. Cuando `CardBank` llama a `rebind()`, es este registro el que mantiene la vinculación. Se trata de un programa independiente que debe ejecutarse antes de invocar a la aplicación del servidor y que se encuentra en el directorio java\bin. La manera de invocarlo consiste en incluir `rmiregistry` en la línea de comandos.

Las siguientes dos líneas de comandos invocan el Registry e inicializa el servidor `CardBank`. Si hay varios equipos conectados a la red, es posible ejecutar `CardBank` en un equipo distinto y emplear el nombre de ese equipo en lugar del anfitrión local.

```
start rmiregistry
java credit.CardBank localhost 12345
```

La aplicación `CardBank` imprime varias líneas de estado al comenzar el servicio. Si no se produce ningún error, aparecerá el siguiente resultado:

```
CardBank: puerto de registro definido
CreditManagerImpl: crea un CreditManager
CreditManagerImpl: lo vincula a un nombre
CreditManager está listo
```

Una vez transmitido un objeto al Registry, un cliente puede pedir al RMI que proporcione una referencia para el objeto remoto. La siguiente sección muestra como se realiza esta operación.

Paso 6: Crear y compilar el programa cliente

La aplicación del comprador necesita encontrar un objeto `manager` de crédito en el servidor remoto. El programa asume que el nombre del servidor se ha introducido en el primer argumento de la línea de comandos y que el segundo puerto se define en el segundo argumento. Se utiliza este nombre de servidor para crear una cadena del tipo URL y con el formato `rmi://hostname:port/cardManager`. Esta cadena se transmite al método estático `lookup()` de la clase `Naming`, y se comunica con el servidor, devolviendo un controlador del objeto remoto que se construyó y registró en el paso 5. Más concretamente, lo que en realidad se devuelve es un controlador para un `stub` que se comunica con el objeto remoto.

El tipo devuelto por el método `lookup()` es `Remote`, que es el elemento padre de todas las interfaces `stub`. Cuando el valor devuelto es de tipo `CreditManager`, sus métodos pueden invocarse en él. El siguiente código, correspondiente a la clase `Shopper`, muestra la manera en que se produce esta operación.

El cliente espera tres argumentos de líneas de comandos; el primero especifica el servidor; y utiliza el mismo nombre que se detalló al comenzar `CardBank`. (Para comprobarlo en un solo equipo, debemos especificar el `locahost` para el nombre del servidor.) El segundo argumento especifica un puerto y emplea el número de puerto que se ha detallado al comenzar el `CardBank`. El tercer argumento es una cadena que proporciona un nombre de cuenta.

El programa cliente pide al objeto `manager` de crédito del programa cliente un administrador para el objeto de la tarjeta de crédito que represente esta cuenta del cliente. El número PIN inicial se cambia a algo que el usuario pueda recordar fácilmente. Después, el programa cliente realiza varias compras y pagos, indicando el crédito disponible después de cada transacción. El listado 23.9 corresponde al código del cliente.

Después sólo resta compilar el comprador:

```
javac -d . Shopper.java
```

Listado 23.9. Shopper.java

```
package credit;
import java.rmi.*;
public class Shopper {
  public static void main(String args[]) {
    CreditManager cm = null;
    CreditCard account = null;
    // Comprueba la línea de comandos.
    if (args.length != 3) {
      System.err.println("Usage:");
      System.err.println(
        "java credit.Shopper <server> <port> <account name>");
      System.exit(-1);
    }
    String server = args[0];
    int port = Integer.parseInt(args[1]);
    String name = args[2];
    // Crea e instala un manager de seguridad.
    System.setSecurityManager(new RMISecurityManager());
    // Obtiene la referencia al manager de la tarjeta.
    try {
      String url = "//" + server + ":" + port + "/cardManager";
      System.out.println("Shopper: lookup cardManager, url = "
        + url);
      cm = (CreditManager)Naming.lookup(url);
    } catch (Exception e) {
      System.err.println("Error in getting card manager" + e);
    }
    // Obtiene la cuenta del usuario.
    try {
      account = cm.findCreditAccount(name);
      System.out.println("Found account for " + name);
    } catch (Exception e) {
      System.err.println("Error in getting account for " + name);
    }
    // Realiza algunas transacciones.
    try {
      System.out.println("Available credit is: "
        + account.getCreditLine());
      System.out.println("Changing pin number for account");
      account.setSignature(1234);
      System.out.println("Buying a new watch for $100");
```

```
            account.makePurchase(100.00f, 1234);
            System.out.println("Available credit is now: " +
              account.getCreditLine());
            System.out.println("Buying a new pair of shoes for $160");
            account.makePurchase(160.00f, 1234);
            System.out.println("CardHolder: Paying off $136 of balance");
            account.payTowardsBalance(136.00f);
            System.out.println("Available credit is now: " +
              account.getCreditLine());
        } catch (Exception e) {
            System.err.println("Transaction error for " + name);
        }
    }
}
```

Paso 7: Comprobar el cliente

El último paso consiste en ejecutar el código del cliente, operación que puede realizarse desde cualquier equipo que tenga acceso al servidor y a las clases que soporta. El siguiente ejemplo representa el resultado de una sesión en un sistema Unix, con el servicio remoto ejecutándose en un anfitrión llamado `snoopy`. La primera línea es la invocación y el resto, el resultado:

```
% java credit.Shopper snoopy 12345 linus
Shopper: lookup cardManager, url = //snoopy:12345/cardManager
Found account for linus
Available credit is: 5000.0
Changing pin number for account
Buying a new watch for $100
Available credit is now: 4900.0
Buying a new pair of shoes for $160
CardHolder: Paying off $136 of balance
Available credit is now: 4740.0
```

Una vez el programa cliente ha terminado de ejecutarse, los objetos remotos siguen vigentes. El ejemplo anterior creaba una nueva cuenta para el cliente. Una segunda llamada al cliente se realizará en esa cuenta. Los números de créditos disponibles en la siguiente lista reflejan el estado actual de la cuenta:

```
% java credit.Shopper snoopy 12345 linus
Shopper: lookup cardManager, url = //snoopy:12345/cardManager
Found account for linus
Available credit is: 4740.0
Changing pin number for account
Buying a new watch for $100
Available credit is now: 4640.0
Buying a new pair of shoes for $160
CardHolder: Paying off $136 of balance
Available credit is now: 4480.0
```

Las posibilidades de RMI que se describen en este capítulo son sólo una muestra de su funcionalidad. Para utilizar RMI con IIOP, será necesario compilar las clases `stub` y `skeleton` con la opción `-iiop` de `rmic`.

La interfaz nativa de Java (JNI)

La interfaz JNI es un mecanismo que permite a los programas conectarse a bibliotecas de terceros escritas en lenguajes de programación distintos de Java. Debido a que las bibliotecas estándar y Enterprise de Java han crecido considerablemente, apenas existe ya la necesidad de emplear código propio para crear aplicaciones. El único caso en que tengamos que crear nuestro propio código será para conectarnos a hardware que no disponga de las interfaces de Java necesarias.

Para más información sobre este tema, visite la página I http://java.sun.com/tutorial/native1.1/. Además de SDK, necesitaremos un compilador C o C++.

Edición Enterprise de Java 2

La edición Enterprise de Java 2 (J2EE) contiene una serie de extensiones API que complementan la edición estándar (J2SE). Mientras la edición estándar es básicamente para necesidades del cliente, la Enterprise está dirigida a las necesidades del servidor. En esta sección se describen los componentes fundamentales de J2EE, entre los cuales se incluyen las bibliotecas para servlets, las páginas de JavaServer (JSP), JavaMail, la interfaz de directorios e identificación de Java (JNDI), el servicio de mensajes de Java (JMS), y los JavaBeans Enterprise (EJB).

Algunos componentes, tales como XML y partes de las bibliotecas JDBC, ya aparecían en J2EE y se incorporan de nuevo en la edición estándar. El resto de las interfaces API de J2EE están disponibles para descargar de manera individual o agrupada en el Kit de desarrollo de software de Java 2, edición Enterprise, desde la página http://java.sun.com/j2ee/download.html.

Las siguientes secciones cubren la versión 1.3 de J2EE, que incluye las tecnologías JDBC y XML mencionadas. Una vez descargado el SDK (de 10 a 15 MB), necesitaremos añadir el archivo j2ee.jar que se encuentra en el subdirectorio de CLASSPATH para comprobar algunos de los ejemplos.

Servlets

Los Servlets son la manera que tiene Java de complementar el servidor Web. Del mismo modo que un applet puede complementar las capacidades de un buscador, el servlet realiza una función semejante en el servidor. La biblioteca de servlets se encuentra dentro de las clases J2EE, aunque la implementación de referencia del API es parte de algo llamado Tomcat que pertenece al código abierto del grupo Apache, y que podemos obtener en http://jakarta.apache.org/tomcat/.

Desde la creación de Java, los applets han sido un mecanismo de las páginas Web para presentar información interactiva y dinámica. Como ya hemos visto, con un applet, el proceso se realiza en la propia máquina del buscador. Por lo general, esta configuración constituye una ventaja ya que mejora el tiempo de respuesta y

reduce los requisitos de banda ancha de la Red, aunque puede presentar problemas si el programa tiene alguno de los siguientes requisitos:

- **Si se requiere acceso restringido a las facilidades del servidor**. Un applet por lo general, no dispone de ningún acceso especial a los servicios e información del servidor. Ni siquiera el servidor del applet es capaz de distinguir entre una solicitud segura de un applet y una solicitud normal, en caso de que no exista una firma digital. Por lo tanto, un applet no puede garantizar la lectura de una base de datos en el servidor a menos que se conceda acceso total a la solicitud HTTP.
- **Si se requiere protección de algoritmos de propiedad**. El código de bytes de Java es más sencillo de modificar que muchos otros lenguajes. Esto se debe a la dificultad que implica generar código deficiente o confuso ya que el verificador lo rechazará. Por esta razón, no debería encomendarse al applet ningún algoritmo de propiedad de valor significante.

Nota

Uno de los motivos por los que se utilizan los servlet tiene que ver con la incompatibilidad entre buscadores. Si desconocemos la versión del buscador utilizado por el usuario final, puede que no podamos emplear algunas de las nuevas ventajas de Java. La solución reside a veces en pasar la ejecución de parte del programa al servidor Web.

Si alguna de estas consideraciones es aplicable a nuestro caso, la solución sería utilizar un servlet, gracias al cual el servidor concedería acceso total a las facilidades locales y confiar en que el servlet se encargará de controlar el acceso al usuario externo. De este modo, podemos limitar el rango de solicitudes y controlar y verificar su origen.

Si construimos un algoritmo de propiedad dentro de un servlet, el código nunca va más allá del servidor; sólo los resultados; si el código no llega hasta el cliente, éste no podrá guardarlo ni decodificarlo.

Pueden utilizarse en combinación un servlet y un applet para conseguir la optimización del flujo de datos, mediante el uso adecuado de los métodos de las clases `GZIPInputStream` y `GZIPOutputStream` del paquete `java.util`, o mediante la serialización de objetos, como se describe en el capítulo 19. La encriptación se controla mediante clases como `SSLSocket`, que implementan un nivel de sockets de seguridad para utilizar en programas de Java, y que se encuentran en las bibliotecas de JSSE.

Para utilizar esta API se requiere una configuración adicional, más allá de descargar y desempaquetar Tomcat. En el directorio lib en el que instalamos Tomcat, se encuentra el archivo servlet.jar; Para compilar servlets, debemos primero situar este archivo en el CLASSPATH (o en el directorio jre\lib\ext).

La interfaz API de Servlet

En el nivel más superficial, un servlet se asemeja mucho a un applet. Por ejemplo, no se ejecuta en la aplicación ni comienza con un método estático `main()`; sino que se carga y se crea una muestra. El siguiente paso consiste en proporcionar el ambiente de trabajo sobre el que determinará los detalles, tales como los parámetros con los que se ha invocado. El comportamiento de una clase `Applet` se determina a través de unos pocos métodos invocados por el buscador: `init()`, `start()`, `stop()`, `destroy()`, y `paint()`. En una clase que implementa la interfaz `Servlet` se emplean los mismos conceptos, aunque con unos métodos ligeramente distintos.

Cuando se crea una clase que implementa la intefaz `Servlet`, se invoca a su método `init(ServletConfig)`, lo que permite al servlet inicializarse por sí mismo. El parámetro `ServletConfig` proporciona acceso a los parámetros de inicialización del servlet y al `ServletContext`, como se demuestra en la siguiente definición:

```
public interface ServletConfig {
  public String getInitParameter(String name);
  public Enumeration getInitParameterNames();
  public ServletContext getServletContext();
}
```

Advertencia

Si implementa la interfaz `Servlet` no olvide incluir el parámetro `ServletConfig` al método `init()`, o de lo contrario el método nunca será invocado. Al subclasificar uno de los implementadores de la interfaz `Servlet`, soportarán la ausencia de argumento de `init()`, que se llamará después de `init(ServletConfig)`.

Un servlet no necesita estar activo o inactivo como es el caso de un applet al entrar y salir de la página. Además, un servlet no dispone de su propio GUI. La interfaz `Servlet`, por lo tanto, no define los métodos `start()`, `stop()`, ni `paint()`. Una nueva conexión al servidor resulta en una llamada al método `service()` del servlet. Este método tiene dos parámetros, que se definen a través de las interfaces `ServletRequest` y `ServletResponse` que además proporcionan métodos de acceso que permiten al servlet examinar su ambiente, determinar cómo se ha ejecutado, y decidir entonces la solicitud que ha recibido y la manera de provocar una respuesta adecuada. Los flujos de entrada y salida conectados al programa invocador o buscador son accesibles a través de sus parámetros.

El implementador básico de la interfaz `Servlet`, `GenericServlet`, no es específico de un único protocolo, aunque debido a que la gran mayoría de los servlets utilizan HTTP como base de comunicación, se proporciona una subclase de la clase `GenericServlet` (`HttpServlet`) como soporte adicional. El siguiente ejemplo emplea `HttpServlet` así como dos interfaces adicionales, `HttpServletRequest`

y `HttpServletResponse`, que complementan las interfaces `ServletRequest` y `ServletResponse`, proporcionando definiciones adicionales de métodos de acceso para gestionar los aspectos HTTP del servlet, tales como el encabezamiento, parte inicial de una transacción HTTP.

La información que se puede leer en un `HttpServletRequest` incluye los detalles de la solicitud, tal como la ruta y la cadena, además del tipo de autorización y los campos del encabezamiento. La clase `HttpServletResponse` soporta el envío de errores específicos de HTTP así como la redirección.

Ejemplo de un Servlet sencillo

El programa del listado 23.10 muestra el ambiente de trabajo de un objeto `HttpServlet`, devolviendo los valores de parámetros específicos según son proporcionados por la solicitud recibida.

Listado 23.10. Primer Servlet

```
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class HelloServlet extends HttpServlet {
  public void init() {
    System.out.println("HelloServlet got an init()!");
    File f = new File(System.getProperty("user.dir"));
    f = new File(f.getAbsolutePath());
    System.out.println("HelloServlet current directory is " + f);
  }
  public void service(HttpServletRequest request,
                      HttpServletResponse response) {
    try {
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
      response.setStatus(HttpServletResponse.SC_OK);
      out.println("<html>");
      out.println("<head><title>Hello World</title></head>");
      out.println("<body>");
      out.println("<H1>Hello World!</H1>");
      out.println("<H2>This servlet was invoked with:</H2>");
      out.println("<dl>");
      Enumeration e = request.getHeaderNames();
      while (e.hasMoreElements()) {
        String name = (String)e.nextElement();
        out.println("<dt>" + name + "</dt><dd>" +
          request.getHeader(name) + "</dd>");
      }
      out.println("<dt>Request method:</dt><dd>" +
        request.getMethod() + "</dd>");
      out.println("<dt>Translated Request path info:</dt><dd>" +
        request.getPathTranslated() + "</dd>");
      out.println("<dt>Request query string:</dt><dd>" +
        request.getQueryString() + "</dd>");
```

```
            out.println("</dl>");
            out.println("</body></head></html>");
        } catch (IOException e) {
            // Ignorelo.
        }
    }
}
```

Ejecución del Servlet

Para ejecutar servlets, necesitamos un servidor Web que los soporte. La mayoría de ellos lo hacen, pero, a diferencia de Tomcat, requieren conocer el lugar donde copiar el archivo . class del servlet. Una vez compilada la clase `servlet`, es necesario que este archivo esté disponible para el servidor de modo que pueda encontrarlo cuando un buscador lo solicite. En el caso de Tomcat, el directorio raíz de los archivos web es el subdirectorio webapps. Podemos situar este archivo en ROOT\WEB-INF\classes. Aunque dependerá de la versión de Tomcat que hayamos descargado, la ruta de directorio suele ser la siguiente:

```
jakarta-tomcat-3.3\webapps\ROOT\WEB-INF\classes
```

Una vez hayamos copiado el archivo al subdirectorio webapps, ya podemos ejecutarlo. La operación consiste en solicitar el servlet desde el buscador. Con el servidor iniciado y el servlet instalado, el siguiente paso consiste en dirigir el buscador al URL:

```
http://localhost:8080/servlet/HelloServlet?Hello+World
```

Este ejemplo da por supuesto que hemos configurado nuestro servidor para que utilice el puerto 8080, que es el que se utiliza por defecto en Tomcat y que, además, permite al servidor funcionar en un sistema Unix.

Tras abrir el buscador, la página debería mostrar un encabezamiento y varias entradas, como muestra la figura 23.5.

Modificación de la cadena de solicitud

Podemos experimentar realizando modificaciones en el URL. La primera parte que describe el protocolo, el anfitrión, y la ruta, no puede cambiarse, pero sí podemos cambiar las partes que aparecen después del signo de interrogación (?), aunque con ciertas restricciones. No se deben incluir espacios ni caracteres que no sean alfanuméricos, exceptuando el símbolo (+), que significa un espacio. Para representar otros caracteres, utilizaremos el código ASCII correspondiente, en hexadecimal, y precedido del símbolo del tanto por ciento, que se representa como %25.

Al cambiar la cadena de solicitud, observaremos que todo el resultado, exceptuando la última línea, permanece sin cambios. Si omitimos toda la cadena y utilizamos la siguiente dirección URL

```
http://localhost:8080/servlet/HelloServlet
```

el resultado indica que la cadena es `null`, lo cual ejemplifica el intento de convertir una referencia nula en un servlet.

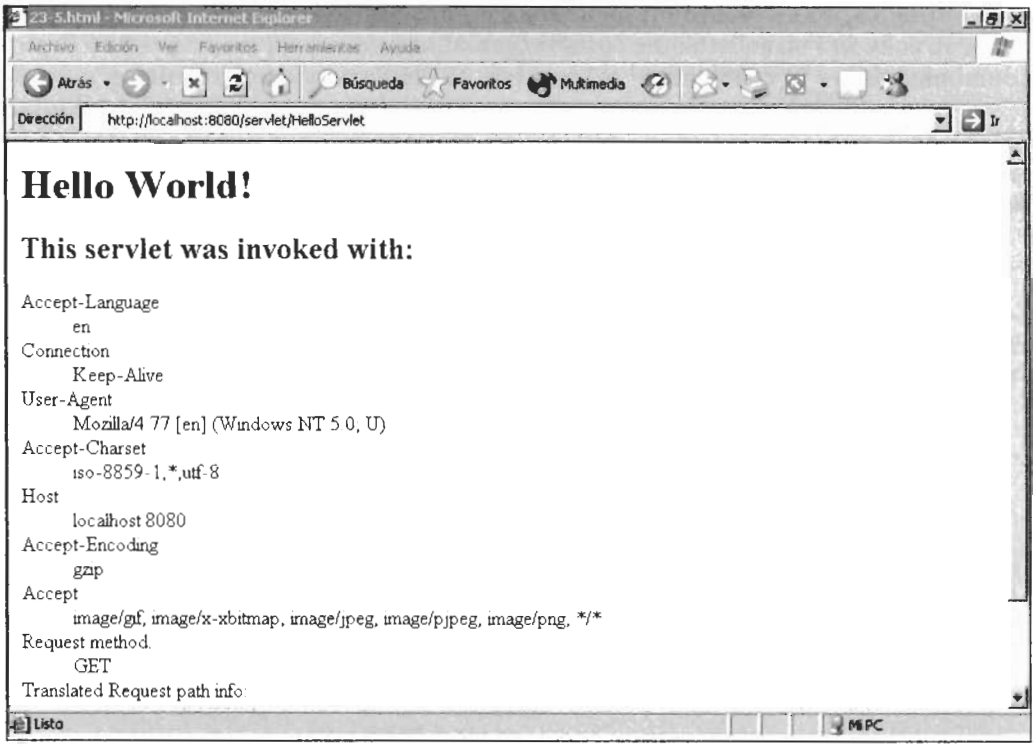


Figura 23.5. Resultado de HelloServlet

Clases y métodos de HelloServlet

El ejemplo de `HelloServlet` extrae partes disponibles de la información sobre el ambiente de trabajo, proporcionadas por los objetos al invocador mediante el flujo de salida proporcionado por el servidor.

La clase `HelloServlet` es una subclase de `HttpServlet` y no de `GenericServlet` debido a que varios métodos de utilidad y definiciones de utilidad provienen de la clase `HttpServlet`, y la mayoría de los servlets tendrán HTTP como su protocolo. Para escribir un servlet que emplee otros protocolos, será necesario extender la clase `GenericServlet`.

La clase `GenericServlet` (más concretamente, la interfaz `Servlet`) define un método llamado `service()`, que tiene dos argumentos en forma de objetos, y que implementan las interfaces `ServletRequest` y `ServletResponse`. La clase `HttpServlet` define otro método `service()` que tiene diferentes tipos de argumentos. Estos dos argumentos son objetos que implementan las interfaces `HttpServletRequest` y `HttpServletResponse` respectivamente. El nuevo método `service()` sobrecarga el nombre del método, en lugar de anularlo. Debido a que el servidor web considera el servlet como un objeto que implementa la interfaz `Servlet`, y no sabe nada sobre la clase `HttpServlet`, llamará sólo al

primer método `service()`. En la clase `HttpServlet`, este método `service()` del servlet sirve para transmitir la llamada al nuevo método `service()`.

Dentro de un servlet, hay disponibles dos flujos para la comunicación con el cliente remoto. El ejemplo del listado 23.10 produce sólo una página HTML; y no lee datos de entrada del cliente, y, por lo tanto, requiere sólo un flujo de salida. En el caso de los objetos `ServletResponse` y `HttpServletResponse`, es posible obtener el flujo de salida mediante los métodos `getOutputStream()` o `getWriter()`. Si estamos implementando comunicación bidireccional, como requiere el protocolo POST, debemos usar además los métodos `getInputStream()` o `getReader()` en el objeto `ServletRequest`.

Generando los datos de salida

Los datos de salida del servlet se componen de dos partes: la del estado y la del contenido. En la parte del estado, el ejemplo `HelloServlet` especifica que el tipo de contenido de su respuesta es un texto en html y que el estado es OK. Ambas operaciones utilizan métodos del objeto `HttpServletResponse`. El primero, `setContentType()`, utiliza un argumento de la clase `String`, que sirve para especificar la cadena de tipo MIME de la respuesta a enviar. Debemos establecer el tipo de contenido mediante `setContentType()` antes de obtener el escritor (`getWriter()`); de lo contrario, se produce un error de ejecución. El estado se especifica mediante el método `setStatus()`, aunque si la respuesta es OK, es en realidad redundante. Existen varias constantes `SC_` dentro de la clase `HttpServletResponse` que proporcionan el código de estado apropiado.

Nota

El método `setContentType()` forma parte de la clase `ServletResponse`, por lo que se puede emplear en servlets para protocolos distintos de HTTP. El método `setStatus()`, sin embargo, se define en la clase `HttpServletResponse` y no en la suya superior.

Después de establecer las partes de estado de la respuesta, `HelloServlet` genera la parte del contenido del resultado como un texto HTML. Este contenido normalmente se presenta al usuario en la ventana del buscador del cliente. El flujo de salida obtenida del objeto `HttpServletResponse` es una muestra de la clase `PrintWriter`. Si empleamos `getOutputStream()`, devuelve una muestra de `javax.servlet.ServletOutputStream`, que define métodos de `println()` con una gran variedad de versiones que soportan diferentes tipos de argumentos.

El método `getWriter()` arroja una excepción `IOException` si se produce algún problema con la conexión. Debido a ello, el cuerpo principal del método `service()` se incluye dentro de un bloque de prueba. Si ocurre una excepción en `HelloServlet`, simplemente se ignora. En un servlet real, el error probablemente se registra.

Una vez obtenemos el escritor del flujo de salida, el servlet debe construir la página para el buscador del cliente. Por lo general, esto implica algún tipo de respuesta condicionada o calculada o, de lo contrario, un archivo de texto HTML es el encargado de definir la página. El objeto `HelloServlet` devuelve una página HTML, lo cual constituye una acción típica, especialmente para servlets basados en la clase `HttpServlet`. La página HTML se genera mediante una serie de declaraciones `println`.

A continuación se crea la lista de definiciones, que en HTML se compone de una serie de entradas, pares de encabezamientos y texto. Las etiquetas HTML que definen estas entradas son <DT> y <DD>. El resultado tiene la forma de la parte final de los datos de salida de un servlet, como se vio en el listado 23.10.

Para construir listas no se necesitan argumentos ya que la definición de HTML no tiene un título propio, sólo elementos individuales. Una vez construida, líneas como la que se muestra a continuación añaden estos elementos:

```
out.println ("<dt>Request method:</dt><dd>" +
   req.getMethod() + "</dd>");
```

Como podemos observar, en este caso se proporcionan dos argumentos. El primero es el título del elemento y el segundo, el texto. La información dada en esta definición se obtiene a partir de la información solicitada y transmitida por el primer argumento del método `service()`. Este argumento es una muestra de la clase `HttpServletRequest`. En el ejemplo de `HelloServlet`, se extrae bastante información que se incluye en la lista, comenzando con los valores del encabezamiento.

Una solicitud HTTP tiene un número de encabezamientos que definen aspectos generales de la conexión. El grupo exacto de encabezamientos dependerá del buscador que se utilice. Como se veía en el resultado de `HelloServlet`, uno de los campos del encabezamiento estaba formado por la información que describe el buscador actual. Para obtener los encabezamientos, un servlet usa uno de los métodos de `getHeader()`. En este caso, como sólo se van a mostrar los encabezamientos, el servlet emplea una variación que obtiene la enumeración de los nombres de los encabezamientos y ejecuta un bucle hasta que no se encuentra ninguno más.

Por lo general, se requiere un encabezamiento concreto, tal como la lista `Accept`, que describe los tipos de respuesta preferidos, lo que se consigue mediante el método `getHeader()`. Este método acepta un objeto `String` como un argumento e intenta encontrar el encabezamiento especificado con un nombre en ese argumento. Si ninguno de estos métodos `getHeader()` es capaz de encontrar el encabezamiento solicitado, se devuelve el resultado de nulo.

En el servlet existen disponibles otros tipos de datos. En el ejemplo de `HelloServlet` podemos observar tres de estos tipos, que se obtienen mediante los métodos `getMethod()`, `getPathTranslated()`, y `getQueryString()`.

El método `getMethod()` devuelve el tipo de solicitud, que es casi siempre `GET`, `POST`, o `HEAD`, aunque la definición de HTTP permite otras cadenas distintas. El método `GET` se emplea para solicitudes generales desde un buscador. Este tipo de solicitud transmite toda la información desde el buscador hasta el servidor como

parte de una cadena URL. Las solicitudes desde formas HTML utilizan generalmente el método `POST`, que envía el cuerpo de los datos al servidor como un flujo en lugar de como parte del URL. En un servlet, este flujo de datos corresponde al flujo de entrada obtenido del objeto `ServletRequest` mediante el método `getInputStream()`. En el controlador de protocolo para HTTP de SDK, al abrir el flujo de salida desde el applet del cliente, convierte la solicitud en un tipo `POST` de manera automática. El método `HEAD` solicita sólo información del encabezamiento, y se utiliza para determinar si una entrada de caché puede reutilizarse, o si debe reemplazarse por información nueva.

Análisis de parámetros de solicitud

Si la cadena de solicitud tiene la forma estándar según la cual los argumentos son nombres y valores separados por signos de igualdad (aKey=aValue& anotherKey=another+Value), podemos emplear el método `parseQueryString()` de `HttpUtils` para decodificar la cadena URL. Este método introduce los parámetros en una `Hashtable`, de la que es posible obtener los valores. El siguiente código demuestra la manera de examinar dichos parámetros. Como podemos observar, cada clave devuelve valores múltiples que deben controlarse de la manera adecuada.

```
Hashtable hash = HttpUtils.parseQueryString(req.getQueryString());
Enumeration enum = hash.keys();
while (enum.hasMoreElements()) {
  String name = (String)enum.nextElement();
  out.println("<dt>" + name + "</dt><dd>");
  String values[] = (String[])hash.get(name);
  for (int i=0,n=values.length; i<n; i++) {
    out.print(values[i]);
    if (i!=n-1) out.print("/");
  }
  out.println ("</dd>");
}
```

Para solicitudes `POST` de HTTP, utlizaremos el método `parsePostData()` de `HttpUtils`.

La información de la ruta, obtenida mediante el método `getPathTranslated()`, describe la ruta en el equipo del servidor desde el que se abrió el servlet. Como se ve en la figura 23.5, algunos servidores pueden devolver una ruta nula, por lo que no debemos confiar demasiado en esta función. En muchas de las solicitudes convencionales a servicios, el URL transmite información adicional sobre los argumentos. El resultado mostrado anteriormente provenía del URL:

```
http://localhost:8080/servlet/HelloServlet?Hello+World
```

y la cadena de solicitud era "Hello+World".

El método `getQueryString()` devuelve el texto completo de la cadena de solicitud del URL invocado. El valor devuelto está aún en formato URL y puede que tenga que convertirse antes de utilizarse.

Por último, después de construir la lista de definiciones, se añaden las declaraciones finales de HTML.

Páginas de JavaServer (JSP)

En el ejemplo de `HelloServlet`, gran parte del resultado consistía en etiquetas de HTML. Si esto ocurre, es posible que el servlet no constituya la mejor solución. Si tenemos código HTML dentro de los servlets implica que estos tienen que recompilarse si queremos modificar el código. Las páginas de JavaServer (JSP), al igual que los servlets, requieren una extensión al servidor Web para abrirse, y las implementaciones de referencia se llevan a cabo mediante Tomcat.

JSP, sin embargo, funciona colocando etiquetas del tipo HTML dentro de un archivo de tipo HTML. Dentro de esas etiquetas está el código de Java. Tanto este código como el resto de los archivos de tipo HTML se compilan dentro de un servlet cuando se abre la página por primera vez. Sin necesidad de realizar esta compilación, hacemos accesible el archivo .jsp que contiene estas etiquetas al servidor Web como una página normal. Cuando el usuario solicita la página, el servidor comprueba si el archivo .jsp es más reciente que el archivo .class a él asociado. Si el archivo .jsp es más reciente (o si el archivo .class no existe), el servidor lo compila antes de ejecutarlo. El servidor, por lo tanto, no necesita recompilar la clase hasta que el archivo .jsp cambia.

El listado 23.11 proporciona una página JSP que realiza las mismas funciones que el programa `HelloServlet`, por lo que el resultado debe ser el mismo. Nómbrelo HelloJSP.jsp y guárdelo en el directorio webapps\ROOT.

Listado 23.11. Servlet JSP

```
<%@ page import="java.io.*,java.util.*" %>
<%!
  public void init() {
    System.out.println("HelloServlet got an init()!");
    File f = new File(System.getProperty("user.dir"));
    f = new File(f.getAbsolutePath());
    System.out.println("HelloServlet current directory is " + f);
  }
%>
<%
    response.setContentType("text/html");
    response.setStatus(HttpServletResponse.SC_OK);
%>
<html>
<head><title>Hello World</title></head>
<body>
<H1>Hello World!</H1>
<H2>This servlet was invoked with:</H2>
```

```
<dl>
<%
  Enumeration e = request.getHeaderNames();
  while (e.hasMoreElements()) {
    String name = (String)e.nextElement();
%>
    <dt> <%= name %> </dt><dd>
      <%= request.getHeader(name)  %> </dd>
<%
  }
%>
  <dt>Request method:</dt><dd>
    <%= request.getMethod() %> </dd>
  <dt>Translated Request path info:</dt><dd>
    <%= request.getPathTranslated() %> </dd>
  <dt>Request query string:</dt><dd>
    <%= request.getQueryString() %></dd>
</dl>
</body></head></html>
```

Todo el código que se encuentra entre los símbolos de tanto por ciento corresponde al código JSP. La tabla 23.1 describe los elementos y su sintaxis.

Tabla 23.1. Elementos de JSP

Nombre	Sintaxis
`Directive`	<%@ directive %>
`Declaration`	<%! declaration %>
`Expression`	<%= expression %>
`Scriptlet`	<% code fragment %>
`Comment`	<%-- comment --%>

Los directivos no se utilizan para generar flujos de salida, sino que se encargan de informar al equipo JSP (en el que se ha abierto el archivo JSP) de la manera de compilar el código. La línea:

```
<%@ page import="java.io.*,java.util.*" %>
```

indica que el equipo debe importar los paquetes `java.io` y `java.util`.

Las declaraciones se utilizan para variables de nivel de página y métodos de nivel de clase.

Las expresiones muestran el valor de la variable o de la expresión transmitida. Es importante recordar que al final de una expresión no debemos escribir un punto y coma, como es el caso de las declaraciones de variables.

Los Script no son más que series de código. En las páginas JSP, este código está escrito en Java.

Los componentes de JSP son comentarios que deben permanecer dentro de la página JSP. Pueden utilizarse comentarios de Java que se generan en el archivo utilizado para compilar la página JSP. Los comentarios de HTML o de JavaScript están a disposición del usuario a través de la opción de **ver código fuente**.

En el listado 23.11 no fue necesario obtener el flujo de salida de JSP la solicitud, ni los objetos resultantes. Sólo utilizamos las variables declaradas en el servlet. Las páginas de JSP tienen acceso a una gran cantidad de objetos denominados implícitos, que se describen en la tabla 23.2.

Tabla 23.2. Objetos implícitos de JSP

Nombre	Descripción
`request`	Solicitud del cliente
`response`	Respuesta de la página
`pageContext`	Aquí se gestionan los atributos de la página
`session`	Sesión asociada a la solicitud
`application`	Contexto en que se ejecuta la página
`out`	Flujo de `Writer` para la respuesta
`config`	Configuración del servlet
`page`	La página JSP
`exception`	Para páginas de error

Gracias a estos objetos implícitos, podemos obtener una línea como la siguiente:

```
<%= request.getQueryString() %>
```

sin necesidad de inicializar el objeto solicitado. No podemos, sin embargo, utilizar un objeto implícito `out` en la declaración de `init()`. Los objetos implícitos son sólo válidos dentro de las etiquetas `scriptlet`.

JavaMail

La interfaz API de `JavaMail` soporta el envío y la recepción de mensajes electrónicos. Las clases pueden descargarse desde la implementación J2EE o desde la página http://java.sun.com/products/javamail/. También se requiere el marco de activación de JavaBeans disponible en http://java.sun.com/beans/glasgow/jaf.html. Por último, es necesario copiar el archivo mail.jar de JavaMail, y el activation.jar del marco de activación.

Revisemos antes algunos de los acrónimos relacionados con los correos electrónicos:

- El protocolo SMPT de transferencia de correo electrónico sencillo.
- El protocolo POP de Post Office y el protocolo IMAP de acceso de mensajes a través de Internet, que definen la manera de recibir mensajes. La utilización de uno u otro, dependerá del suministrador de servicio de Internet (ISP). La mayoría de los usuarios emplean POP, o más concretamente POP3, donde el número 3 se refiere a la versión tercera. En el mundo empresarial, se utilizan ambos.
- MIME o extensiones multipropósito de correo de Internet describen el formato interno de los mensajes y no el método de transferencia.

El envío de un correo electrónico se basa en cuatro pasos: Primero, debemos conectarnos al servidor de correo, mediante el objeto `javax.mail.Session`. A partir de la sesión, se crea el mensaje a enviar, más concretamente un `MimeMessage`, que es en realidad una subclase. El siguiente paso consiste en asignar una dirección. La clase `Address` se sirve de su subclase `InternetAddress` para realizar esta tarea.

Por último, se envía el mensaje mediante la clase `Transport`. Todo este proceso se demuestra en el listado 23.12.

Listado 23.12. Envío de un correo electrónico

```
import java.util.Properties;
import javax.mail.*;
import javax.mail.internet.*;
public class MailExample {
  public static void main (String args[]) throws Exception {
    if (args.length != 3) {
      System.err.println("Usage: java MailExample host from to");
      System.exit(-1);
    }
    String host = args[0];
    String from = args[1];
    String to = args[2];
    // Get system properties
    Properties props = System.getProperties();
    // Determina el servidor de correo
    props.put("mail.smtp.host", host);
    // Obtiene la sesión
    Session session = Session.getDefaultInstance(props, null);
    // Define el mensaje
    MimeMessage message = new MimeMessage(session);
    message.setFrom(new InternetAddress(from));
    message.addRecipient(Message.RecipientType.TO,
      new InternetAddress(to));
    message.setSubject("The Subject");
    message.setText("The Message");
    // Envía el mensaje
    Transport.send(message);
  }
}
```

Para ejecutar el programa `MailExample` tenemos que transmitir el nombre del servidor SMTP que estemos empleando, la dirección del remitente, y la del destinatario, como se muestra a continuación:

```
java MailExample smtp.example.com from@example.net to@example.org
```

Para recibir un correo, el proceso es similar en cuanto al establecimiento de la sesión. Después debemos conectarnos a un `Store`, basado en el protocolo de correo que estemos empleando.

El siguiente paso consiste en obtener el `Folder` y leer los objetos `Message` que en él se encuentren, como se ve en el listado 23.13.

Listado 23.13. Lectura de un correo electrónico

```
import java.io.*;
import java.util.Properties;
import javax.mail.*;
import javax.mail.internet.*;
public class GetMessageExample {
  public static void main (String args[]) throws Exception {
    if (args.length != 3) {
      System.err.println("Usage: java MailExample host username password");
      System.exit(-1);
    }
    String host = args[0];
    String username = args[1];
    String password = args[2];
    // Create empty properties
    Properties props = new Properties();
    // Obtiene la sesión
    Session session = Session.getDefaultInstance(props, null);
    // Obtiene el lugar de almacenamiento
    Store store = session.getStore("pop3");
    store.connect(host, username, password);
    // Obtiene la carpeta
    Folder folder = store.getFolder("INBOX");
    folder.open(Folder.READ_ONLY);
    BufferedReader reader = new BufferedReader (
      new InputStreamReader(System.in));
    // Obtiene el directorio
    Message message[] = folder.getMessages();
    for (int i=0, n=message.length; i<n; i++) {
       System.out.println(i + ": " + message[i].getFrom()[0]
         + "\t" + message[i].getSubject());
      System.out.println("Read message? [YES to read/QUIT to end]");
      String line = reader.readLine();
      if ("YES".equalsIgnoreCase(line)) {
        System.out.println(message[i].getContent());
      } else if ("QUIT".equalsIgnoreCase(line)) {
        break;
      }
    }
    // Cierra la conexión
```

```
        folder.close(false);
        store.close();
    }
}
```

Para ejecutar este programa, debemos proporcionar un nombre de usuario y una contraseña en la línea de comandos:

```
java GetMessageExample pop.example.com username password
```

Truco

Una mejor manera de identificación es mediante el método `Authenticator`. Puede crear uno que pedirá al usuario la contraseña siempre que sea necesario.

Nota

En la página http://java.sun.com/jdc/onlineTraining/JavaMail/ encontrará un tutorial completo sobre JavaMail API.

La interfaz de directorios e identificación de Java (JNDI)

La interfaz de directorios e identificación de Java (JNDI) proporciona los medios para buscar servicios dentro de directorios. Si por ejemplo queremos visitar una página Web, antes de poder acceder al buscador, debe encontrar su dirección, buscando en el mapa de sistema de nombres de dominio (DNS), y obteniendo una dirección numérica del tipo 12.13.14.15. La interfaz JNDI se emplea cuando necesitamos buscar un recurso para una de las otras interfaces de Java, tal como el driver de JDBC o una conexión JMS, como se mostrará más adelante.

Nota

Para más información sobre JNDI, visite http://java.sun.com/products/jndi/.

Servicio de mensajes de Java (JMS)

La interfaz API de JMS constituye el marco de mensajes de la versión Enterprise de Java. JMS no es un sistema de mensajes, sino una substracción del subsistema.

Un subsistema de mensajes sería algo como las MQSeries de IBM (http://www-4.ibm.com/software/ts/mqseries/api/mqjava.html) o el SonicMQ (http://www.sonicsoftware.com/). Como una abstracción, JMS nos permite cambiar de suministradores sin modificar el código.

Este API soporta dos modelos de mensajes: de suscripción pública y de punto a punto también conocidos como pub/sub y p2p respectivamente. Estos dos modelos se diferencian en el modo en que el suministrador de mensajes se comunica con el consumidor. En el primer modelo, el publicador anuncia la disponibilidad del tema. Por su parte, los subscriptores expresan interés en dicho tema. Cualquier número de subscriptores puede recibir el mismo mensaje según este modelo, como se representa en la figura 23.6.

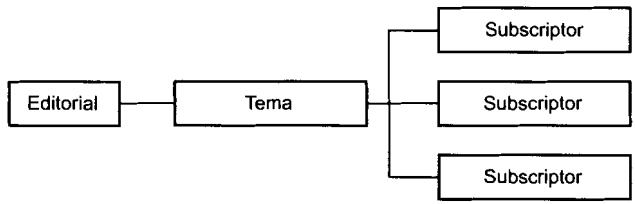


Figura 23.6. Modelo de mensajes des suscripción pública (pub/sus)

En el modelo p2p, los remitentes crean canales en los que se pueden registrar los receptores potenciales para solicitar un mensaje determinado. Los receptores no reciben el mensaje todos a la vez, sino que el remitente lo envía de uno en uno. Es como una carrera de relevos, en las que sin recibir el testigo no se puede empezar a correr. Mientras no se reciba el mensaje, hay que esperar. La figura 23.7 muestra el diseño básico de este modelo.

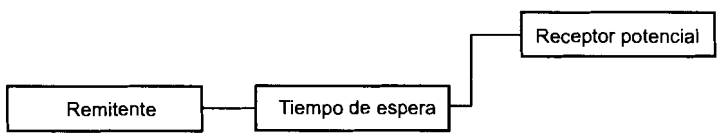


Figura 23.7. Modelo de mensajes des punto a punto (p2p)

JMS no define el orden en que el remitente envía los mensajes, sino el mecanismo básico para enviarlos. La implementación del distribuidor puede distribuir los mensajes de cualquier manera.

El listado 23.14 muestra el diseño básico de un JMS para pub/sub.

Listado 23.14. Uso básico de Pub/Sub de JMS

```
import javax.jms.*;
import javax.naming.*;
...
// Obtiene el contexto JNDI
Properties env = new Properties();
// Establece las propiedades
```

```
InitialContext jndi = new InitialContext(env);
// Busca la conexión Factory
TopicConnectionFactory conFactory = (TopicConnectionFactory)jndi.lookup(
  "TopicConnectionFactory");
// Crea la conexión
TopicConnection connection = conFactory.createTopicConnection(
  username, password);
// Crea los objetos de la sesión
TopicSession pubSession = connection.createTopicSession(false,
  Session.AUTO_ACKNOWLEDGE);
TopicSession subSession = connection.createTopicSession(false,
  Session.AUTO_ACKNOWLEDGE);
// Busca el tema
Topic topic = (Topic)jndi.lookup(name);
// Crea el publicador
TopicPublisher publisher = pubSession.createPublisher(topic);
// Crea el subscriptor
TopicSubscriber subscriber = subSession.createSubscriber(topic);
// Establece el MessageListener para notificar la recepción de mensajes
MessageListener listener = ...
subscriber.setMessageListener(listener);
// Comienza la conexión
connection.start
```

En este punto, el subscriptor necesita proporcionar un método `onMessage()` para responder cuando el publicador llame a `publish()` con un `Message`. Este método proviene de la interfaz `MessageListener`.

La especificación JMS define varios tipos de mensajes:

- `BytesMessage`: Utilizado para ocultar datos a la implementación JMS.
- `MapMessage`: Utilizado para enviar pares de valores y claves, igual que la interfaz `Map` aunque los valores pueden ser bases de datos primitivas.
- `ObjectMessage`: Para enviar objetos serializables, de manera similar que `ObjectInput` y `ObjectOutput`.
- `StreamMessage`: Similar a leer y escribir flujos de datos con `DataInput` y `DataOutput`.
- `TextMessage`: Para enviar un `String` de texto.

Para más información sobre JMS, visite http://java.sun.com/products/jms/.

JavaBeans Enterprise (EJB)

La tecnología de JavaBeans Enterprise (EJB) es la base de Sun para desarrollar aplicaciones de transición con una arquitectura distribuida. EJB se encuentra dentro de los contenedores del mismo nombre de los servidores de la aplicación, tales como WebLogic de BEA, y sus objetos provienen de servicios remotos para clientes de la Red. Es muy similar a RMI y CORBA; de hecho, RMI es el protocolo de comunicación básico empleado.

Los EJB no tienen ninguna vinculación con los JavaBean normales, aunque comparten el mismo nombre básico y ambos se escriben en lenguaje Java. La manera más sencilla de entender las diferencias es examinando sus funciones respectivas. Los componentes de JavaBean sirven para comunicarse con otros componentes de JavaBean, o con otros objetos, dentro de una única aplicación. Por el otro lado, los componentes EJB se comunican a través de espacios de direcciones múltiples.

La función de los componentes EJB consiste en ocultar temas relacionados con un sistema de desarrollo de transición complejo. Entre dichos temas se incluyen la administración de la seguridad, la escalabilidad, y la persistencia. Cuando se crean sistemas a larga escala, los EJB permiten encapsular la lógica dentro de componentes reutilizables.

Trabajar con los componentes EJB es bastante complejo, por lo que las bases fundamentales que aquí se dan sirven sólo para introducir la terminología de EJB. Existen tres tipos fundamentales de tipos de bean:

- Los beans de entidad son como objetos de bases de datos almacenados en la memoria. Todos los clientes pueden acceder a ellos de manera simultánea, y el contenedor EJB proporcionará el soporte para administrar la transacción. Si el programador controla la persistencia del bean de entidad, se conoce como persistencia de bean controlado. Si, por el contrario, es el contenedor de EJB quien se encarga de la gestión, la persistencia se conoce como persistencia controlada por el contenedor. Aunque el programador puede, por lo general, realizar las tareas de manera más rápida, no merece la pena perder tiempo en escribir el código cuando la misma tarea la puede realizar mediante la persistencia controlada por el contenedor de manera automática.
- Los beans de sesión pueden ser de estado o no. Los beans de sesión de estado pueden almacenar información entre llamadas de métodos, de forma similar a almacenar información relacionada con la sesión con un servlet o un JSP. Los beans de sesión que no son de estado no pueden hacer esto ya que no disponen de memoria. Esencialmente, los beans de sesión sólo almacenan los datos que el programa necesita cuando se está ejecutando.
- Los beans de mensajes son una novedad de las especificaciones de la versión 2.0 de EJB. Como su nombre indica, estos beans pueden participar en comunicaciones de JMS. Los mensajes pueden ser del tipo pub/sub o p2p y llegan fuera de sincronización a través de la lista de espera o de la suscripción.

Para más información sobre los componentes de JavaBeans Enterprise consulte la página http://www.titan-books.net/download.html.

Comunicaciones de Java (COMM)

Las interfaces API de comunicaciones de Java no forman parte de las especificaciones de J2EE, aunque son de gran interés para los desarrolladores ya que propor-

cionan soporte para acceder directamente a la serialización del equipo o a los puertos paralelos. Se trata de una biblioteca extraña en el sentido que no es puramente de Java. Es necesario descargar una versión del paquete específico según las diferentes plataformas. Sun proporciona versiones de la biblioteca COMM para Solaris y Windows en la página http://java.sun.com/products/javacomm/. La versión Linux está disponible en http://www.interstice.com/kevinh/linuxcomm.html, aunque no se ha actualizado desde 1998.

Su instalación es un poco complicada ya que tras descargarla de la página de Sun, es necesario copiar el archivo JAR para las clases en el CLASSPATH, y crear un archivo de bibliotecas compartido en código nativo, así como un archivo de propiedades disponible para el ambiente de ejecución. Desafortunadamente, y como ocurre con la versión Linux de COMM, el archivo Readme.html que acompaña las descargas para Windows y Solaris no se actualiza desde la versión 1.1.6 de Java.

Con Java 1.4, debemos realizar las siguientes copias para Windows:

```
copiar comm.jar \jdk1.4\jre\lib\ext
mkdir \jdk1.4\jre\lib\ext\x86
copiar win32com.dll \jdk1.4\jre\lib\ext\x86
copiar javax.comm.properties \jdk1.4\jre\lib
```

Para usuarios de Solaris en lugar de crear un directorio x86 y copiar un archivo .dll, se crea un directorio sparc en el que se copia un archivo .so.

Una vez tenemos los archivos en el directorio adecuado, estamos listos para utilizar el paquete `javax.comm`. Para descubrir los puertos disponibles, debemos preguntar a `CommPortIdentifier` mediante `getPortIdentifiers()`, lo que devuelve una enumeración en la que realizar la búsqueda. El listado 23.15 demuestra esta función.

Listado 23.15. Búsqueda de puertos disponibles

```
import javax.comm.*;
import java.util.Enumeration;
public class ListPorts {
  public static void main(String args[]) {
    Enumeration ports =
      CommPortIdentifier.getPortIdentifiers();
    while (ports.hasMoreElements()) {
      CommPortIdentifier port =
        (CommPortIdentifier)ports.nextElement();
      String type;
      switch (port.getPortType()) {
        case CommPortIdentifier.PORT_PARALLEL:
          type = "Parallel";
          break;
        case CommPortIdentifier.PORT_SERIAL:
          type = "Serial";
          break;
        default:  /// no debería ocurrir
          type = "Unknown";
          break;
```

```
        }
        System.out.println(port.getName() + ": " + type);
      }
    }
}
```

También es posible abrir un puerto concreto y obtener su `OutputStream`:

```
CommPort port = portId.open(name, timeout);
OutputStream os = port.getOutputStream();
```

Mediante este `OutputStream`, podemos imprimir en la impresora. Para demostrarlo, el listado 23.16 toma cualquier archivo de entrada y lo envía a la impresora. Este archivo puede ser de texto, aunque si deseamos imprimir gráficos, debemos primero grabar los gráficos en el archivo en un formato que la impresora comprenda y después enviar ese archivo al programa `PrintFile`. Los archivos PostScript pueden enviarse directamente, siempre y cuando la impresora los entienda.

Listado 23.16. Impresión de un archivo

```
import java.io.*;
import javax.comm.*;
public class PrintFile {
  public static void main(String args[]) throws Exception {
    if (args.length != 2) {
      System.err.println("usage : java PrintFile port file");
      System.err.println("sample: java PrintFile LPT1 sample.prn");
      System.exit(-1);
    }
    String portname = args[0];
    String filename = args[1];
    // Obtiene el puerto
    CommPortIdentifier portId =
      CommPortIdentifier.getPortIdentifier(portname);
    // Abre el puerto
    // Requiere el nombre del propietario y el tiempo de espera
    CommPort port = portId.open("Java Printing", 30000);
    // Configura la lectura del archivo
    FileInputStream fis = new FileInputStream(filename);
    BufferedInputStream bis = new BufferedInputStream(fis);
    // Configura la salida
    OutputStream os = port.getOutputStream();
    BufferedOutputStream bos = new BufferedOutputStream(os);
    int c;
    while ((c = bis.read()) != -1) {
      bos.write(c);
    }
    // Se cierra
    bos.close();
    bis.close();
    port.close();
  }
}
```

Truco

Para imprimir en un archivo en las plataformas de Windows, debemos ir a la opción de Impresoras del Panel de control y hacer clic con el botón derecho del ratón sobre la impresora que vamos a utilizar. En la ficha Detalles, del cuadro de diálogo Propiedades debemos seleccionar la opción FILE (Crear un archivo en el disco) de la lista desplegable Imprimir en el siguiente puerto. Tras hacer clic varias veces, la próxima vez que vayamos a imprimir, se nos pedirá un nombre de archivo. Una vez guardado, podremos utilizar dicho archivo como parámetro de entrada del programa `PrintFile`.

La API de COMM no proporciona acceso a los puertos USB. Para más detalles, visite la página http://www.jcp.org/jsr/detail/80.jsp.

Resumen

Aunque la versión 1.4 de Java es muy reciente, ya se está trabajando en una versión futura que soporte tipos de datos genéricos, así como plantillas y tipos distintos de parámetros.

Esencialmente, los datos genéricos ofrecen la posibilidad de trabajar con colecciones compatibles con distintos tipos.

Al trabajar con el marco de colecciones de Java, podemos añadir elementos nuevos de cualquier tipo a la colección, aunque es nuestra tarea determinar el tipo de datos adecuado, como en el siguiente ejemplo:

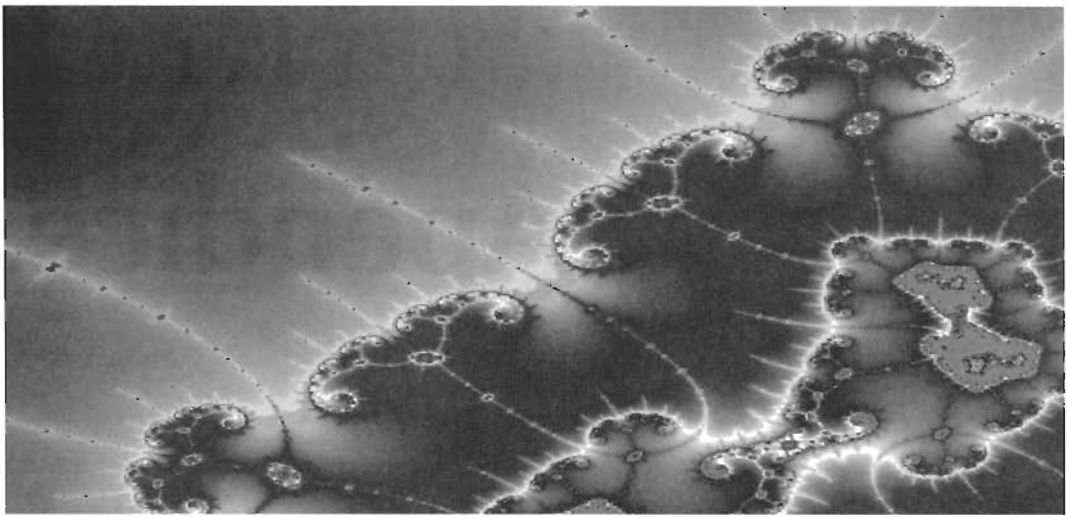
```
String head = (String)list.get(0);
```

Los elementos genéricos nos permiten añadir únicamente un tipo determinado de elementos, que por lo tanto no tienen que ser clasificados, lo que proporciona una mayor seguridad en el tiempo de la compilación, ya que los errores se detectan en esta fase y no durante la comprobación.

Para demostrarlo, la siguiente línea crea una `List`, que sólo acepta elementos de tipo `String`:

```
List<String> list = new ArrayList<String>();
```

Éste es el marco básico definido por una implementación de acceso anterior, que no es necesariamente cómo se implementará el resultado final, por lo que es fundamental que nos mantengamos informados sobre las nuevas funciones de Java 1.5. De todos modos, no existen garantías de que los elementos genéricos sean una de las nuevas capacidades de la versión 1.5. Para más información, consulte periódicamente la página JSR 14 en http://jcp.org/jsr/detail/014.jsp.



Apéndice A. Sintaxis del lenguaje Java

Tabla A.1. Lista de palabras reservadas

Keyword (Clave)		
abstract	assert	boolean
break	byte	case
catch	char	class
const*	continue	default
do	double	else
extends	false**	final
finally	float	for
goto*	if	implements
import	instanceof	int
interface	long	native
new	null**	package
private	protected	Keyword
public	return	short
static	strictfp	super

switch	synchronized	this
throw	throws	transient
true**	try	void
volatile	while	

* Reservada pero sin utilizar.
** Técnicamente, true, false, y null son términos literales y no claves.

Tabla A.2. Tipos numéricos

Tipo	Tamaño	Por defecto	Mínimo	Máximo
byte	80	0	-128	127
short	16	0	-32768	32767
int	32	0	-2147483648	2147483647
long	64	0	-9223372036854775808	9223372036854775807
float	32	0.0f	1.4012984632481707e-45f*	3.40282346638528860e+38f
double	64	0.0	4.94065645841246544e-324*	1.79769313486231570e+308

* Valor positivo más pequeño.

Tabla A.3. Constantes más utilizadas

Nombre	Descripción
Math.E	Base de logaritmos naturales
Math.PI	Ratio de la circunferencia y el diámetro
Float/Double.NEGATIVE_INFINITY	Valor infinito negativo
Float/Double.POSITIVE_INFINITY	Valor infinito positivo
Float/Double.NaN	Distinto de un número
Byte/Short/Integer/Long/Float/ Double.MAXIMUM_VALUE	Máximo valor del tipo de datos
Byte/Short/Integer/Long/Float/ Double.MINIMUM_VALUE	Valor mínimo del tipo de datos*
System.in Standard input	Normalmente el teclado
System.out Standard output	Normalmente la consola
System.err Standard error	Normalmente la consola

* = Para Float y Double, el valor positivo más pequeño.

Tabla A.4. Notaciones literales

Tipo de datos	Notación
`int`	Decimal: dígitos 0-9 (1234)
	Octal: cero (0) seguido de los dígitos 0-7 (01234)
	Hexadecimal: 0x o 0X seguido de los dígitos 0-9, A-F (0x1234)
`long`	Añade l o L al final del término literal `int`
`float`	Añade f o F al final del término literal `double`, y sustituye a d/D en caso de existir
`double`	Notación científica como en 1.345e+10, donde el exponente puede expresarse con `e`
	`EOptional d o D` al final del elemento literal `double`.
`char`	Dentro de indicaciones simples; utilizar \ para evitar caracteres especiales (ver tabla A.5)
`String`	Dentro de indicaciones dobles; los caracteres " y \ deben evitarse con \
`boolean`	verdadero, falso
`Null`	nulo

Tabla A.5. Secuencias de caracteres de escape

Secuencia de escape	Descripción
\b	Tecla de retroceso
\t	Tabulador
\n	Línea de alimentación (LF)
\f	Forma de alimentación (FF)
\r	Retorno de carro (CR)
\"	Anotación doble
\'	Anotación simple
\\	Línea oblicua
Secuencia Unicode	\u Seguida de cuatro dígitos hexadecimales
Secuencia Octal	Representa \u0000 a través de \u00FF\ seguido de dígitos octal de 1-3, si hay tres dígitos, el primero debe ser 0-3 Por ejemplo, "A" equivale a \101 (65)

Tabla A.6. Lista de precedencia de operadores

Tipo	Operadores	Orden
Postfix	[ ], ., (*expression*), var++, var--	Izquierda a derecha
Prefijo/Unario	++var, --var, +var, -var, ~, !	Derecha a izquierda
Creación/Clasificación	new, (*type*)*expression*	Derecha a izquierda
Multiplicación	*, /, %	Izquierda a derecha
Adición	+, -	Izquierda a derecha
Modificación	<<, >>, >>>	Izquierda a derecha
Comparación	<, >, <=, >=,	Izquierda a derecha
Igualdad	==, !=	Izquierda a derecha
Operador AND	&	Izquierda a derecha
Operador XOR	^	Izquierda a derecha
Operador OR	|	Izquierda a derecha
Lógica AND	&&	Izquierda a derecha
Lógica OR	||	Izquierda a derecha
Condicional	?:	Derecha a izquierda
Tarea	=, +=, -=, *=, /=, %=, &=, ^=, |=, <<=, >>=, >>>=	Derecha a izquierda

Tabla A.7. Modificadores

Modificador	Descripción
`abstract`	Válido para métodos y clases. La definición debe proporcionarse en algunas subclases no abstractas. No puede crear muestras de clases abstractas.
`final`	Válido para variables, campos, métodos y clases. Su valor no se puede cambiar en variables y campos. Para los métodos, no pueden anularse. Para las clases, no pueden subclasificarse.
`native`	Válido para métodos. Definición del método descrita fuera de Java.
`none`	Válido para campos métodos, clases e inicializadores. Los elementos marcados con este modificador son visibles en el paquete en el que son declarados.
`private`	Válido para campos, métodos y clases internas. Los elementos marcados con este modificador son visibles en todas las muestras de la clase definida.

Modificador	Descripción
`protected`	Válido para campos, métodos y clases. Los elementos marcados con este modificador son visibles en el paquete en el que son declarados y en las subclases.
`public`	Válido para los campos, los métodos, las interfaces y las clases. Los elementos marcados con este modificador son siempre visibles.
`static`	Válido para campos, métodos, clases internas e inicializadores. Los elementos marcados con este modificador se definen una sola vez para todas las muestras de la clase.
`strictfp`	Válido para métodos y clases. Hace que los métodos y las clases trunquen valores intermedios como las versiones más antiguas de Java.
`synchronized`	Válido para métodos. Dos threads no pueden ejecutar dos métodos simultáneamente. Bloqueos distintos para métodos estáticos y no estáticos.
`transient`	Válido para campos. El valor no se almacena al guardar la muestra de la clase.
`volatile`	Válido para variables. El valor de la variable puede cambiar desde otro thread. Indica al compilador que no realice tareas de optimización colocándolo en un almacenamiento temporal.

Tabla A.8. Conversiones

Operación	Sintaxis
`boolean to String`	String s = theBoolean ? "true" : "false";
	String s = "" + theBoolean;
`byte, short, char, int to String`	String s = Byte.toString(theByte);
	String s = Short.toString(theShort);
	String s = Integer.toString(theNumber);
`byte, short, char, int to String`	String s = Integer.toString(theNumber, base);
	String s = Integer.toBinaryString(theNumber);
	String s = Integer.toOctalString(theNumber);
	String s = Integer.toHexString(theNumber);
	String s = String.valueOf(theNumber); String s = "" + theNumber;

Operación	Sintaxis
long to String	String s = Long.toString(theLong);
	String s = Long.toString(theLong, base);
	String s = Long.toBinaryString(theLong);
	String s = Long.toOctalString(theLong);
	String s = Long.toHexString(theLong);
	String s = String.valueOf(theLong);
	String s = "" + theLong;
float to String	String s = Float.toString(theFloat);
	String s = String.valueOf(theFloat);
	String s = "" + theFloat;
double to String	String s = Double.toString(theDouble);
	String s = String.valueOf(theDouble);
	String s = "" + theDouble;
String to boolean	boolean b = Boolean.getBoolean(theString);
String to byte	byte b = Byte.parseByte(theString);
	byte b = Byte.parseByte(theString, base);
String to short	short s = Short.parseShort(theString);
	short s = Short.parseShort(theString, base);
String to int	int i = Integer.parseInt(theString);
	int i = Integer.parseInt(theString, base);
String to long	long l = Long.parseLong(theString);
	long l = Long.parseLong(theString, base);
String to char	char c = (char)Integer.parseInt(theString);
	char c = (char)Integer.parseInt(theString, base);
String to float	float f = Float.parseFloat(theString);
String to double	double d = Double.parseDouble(theString);

Tabla A.9. Tabla de colores

Color	Rojo	Verde	Azul
Agua	00	FF	FF
Aguamarina medio	66	CD	AA

Color	Rojo	Verde	Azul
Aguamarina	7F	FF	D4
Almendra blanca	FF	EB	CD
Amarillo	FF	FF	00
Amarillo claro	FF	FF	E0
Amarillo maíz	FF	F8	DC
Amarillo trigo	45	DE	B3
Amarillo verdoso	AD	FF	2F
Amarillo verdoso	9A	CD	32
Azul	00	00	FF
Azul acero claro	B0	C4	DE
Azul acero	46	82	B4
Azul aciano	64	95	ED
Azul Alice	F0	F8	FF
Azul cadete	5F	9E	A0
Azul celeste claro	87	CE	FA
Azul celeste oscuro	00	BF	FF
Azul celeste	87	CE	EB
Azul celeste	F0	FF	FF
Azul Chartreuse	7F	FF	00
Azul claro	AD	D8	E6
Azul de medianoche	19	19	70
Azul Dodger	1E	90	FF
Azul marino	00	00	80
Azul medio	00	00	CD
Azul nieve	B0	E0	E6
Azul oscuro	00	00	8B
Azul pizarra claro	84	70	FF
Azul pizarra medio	7B	68	EE
Azul pizarra oscuro	48	3D	8B
Azul pizarra	6A	5A	CD
Azul Royal	41	69	E1

Color	Rojo	Verde	Azul
Beis concha	FF	F5	EE
Beis	F5	F5	DC
Bisque	FF	E4	C4
Blanco	FF	FF	FF
Blanco encaje	FD	F5	E6
Blanco fantasma	F8	F8	FF
Blanco flor	FF	FA	F0
Blanco navajo	FF	DE	AD
Blanco nieve	FF	FA	FA
Blanco tabaco	F5	F5	F5
Blanco Viejo	FA	EB	D7
Caqui	F0	E6	8C
Caqui oscuro	BD	B7	6B
Cardo	8D	BF	D8
Carmesí	DC	14	3C
Chiffón limón	FF	FA	CD
Chocolate	D2	69	1E
Ciruela	DD	A0	DD
Coral claro	F0	80	80
Coral	FF	7F	50
Cream menta	F5	FF	FA
Cyan claro	E0	FF	FF
Cyan oscuro	00	8B	8B
Cyan	00	FF	FF
Dark goldenrod	B8	86	0B
Dorado	FF	D7	00
Fuxia	FF	00	FF
Granate	80	00	00
Gris	80	80	80
Gris claro	D3	D3	D3
Gris dim	69	69	69

Color	Rojo	Verde	Azul
Gris Gainsboro	DC	DC	DC
Gris oscuro	A9	A9	A9
Gris pizarra claro	77	88	99
Gris pizarra oscuro	2F	4F	4F
Gris pizarra	70	80	90
Índigo	4B	00	82
Lavanda	E6	E6	FA
Lavanda rosaceo	FF	F0	F5
Lino	FA	F0	E6
Madera	DE	B8	87
Magenta oscuro	8B	00	8B
Magenta	FF	00	FF
Marfil	FF	FF	F0
Marrón	A5	2A	2ª
Marrón amarillento Goldenrod	FA	FA	D2
Marrón arena	F4	A4	60
Marrón cuero	8B	45	13
Marrón claro goldenrod	EE	DD	82
Marrón Goldenrod	DA	A5	20
Marrón Goldenrod pálido	EE	E8	AA
Marrón habano	D2	B4	8C
Marrón rosado	BC	8F	8F
Melocotón	FF	DA	B9
Melón	F0	FF	F0
Mocasín	FF	E4	B5
Morado	80	00	80
Morado medio	93	70	DB
Naranja	FF	A5	00
Naranja oscuro	FF	8C	00
Negro	00	00	00
Oliva apagado	6B	8E	23

Color	Rojo	Verde	Azul
Orquídea medio	BA	55	D3
Orquídea oscuro	99	32	CC
Papaya	FF	EF	D5
Peru	CD	85	3F
Plateado	C0	C0	C0
Rojo	FF	00	00
Rojo anaranjado	FF	45	00
Rojo Indian	CD	5C	5C
Rojo ladrillo	B2	22	22
Rojo oscuro	8B	00	00
Rojo tomate	FF	63	47
Rojo violáceo medio	C7	15	85
Rojo violáceo pálido	DB	70	93
Rojo violáceo	D0	20	90
Rosa	FF	C0	CB
Rosa brillante	FF	69	B4
Rosa claro	FF	B6	C1
Rosa orquídea	DA	70	D6
Rosa oscuro	FF	14	93
Rosa pálido	FF	E4	E1
Salmón	FA	80	72
Salmón claro	FF	A0	7A
Salmón oscuro	E9	96	7A
Siena	A0	52	2D
Teja	00	80	80
Turquesa	40	E0	D0
Turquesa medio	48	D1	CC
Turquesa oscuro	00	CE	D1
Turquesa pálido	AF	EE	EE
Verde	00	FF	00
Verde bosque	22	8B	22

Color	Rojo	Verde	Azul
Verde césped	7C	FC	00
Verde claro	90	EE	90
Verde lima	32	CD	32
Verde manantial medio	00	FA	9A
Verde manantial	00	FF	7F
Verde mar claro	20	B2	AA
Verde mar oscuro	8F	BC	8F
Verde marino medio	3C	B3	71
Verde marino	2E	8B	57
Verde oliva oscuro	55	6B	2F
Verde oliva	80	80	00
Verde oscuro	00	64	00
Verde pálido	98	FB	98
Violeta	EE	82	EE
Violeta azulado	8A	2B	E2
Violeta oscuro	94	00	D3

Nota

La tabla A.10 contiene el conjunto inicial de propiedades del sistema generadas por `System.getProperties().list(System.out)`. Con API específicos, tales como JDBC JNDI, hay propiedades específicas de API que no tienen valor inicial.

Tabla A.10. Propiedades del sistema

Nombre	Descripción
`awt.toolkit`	Clase por defecto del sistema para el kit de herramientas predeterminado
`file.encoding`	Código por defecto carácter a byte y viceversa
`file.encoding.pkg`	Nombre del paquete para clases de código de carácter a byte

Nombre	Descripción
file.separator	Separador de ruta en los nombres de archivo
java.awt.graphicsenv	Clase del sistema para gráficos
java.awt.printerjob	Clase para trabajos de impresión
java.class.path	Lista de directorios y archivos JAR para clases
java.class.version	Número de la versión de las bibliotecas
java.endorsed.dirs	Lista de directorios para clases endosadas
java.ext.dirs	Directorio en el que buscar bibliotecas de extensión estándares
java.home	Lugar donde se instala Java
java.io.tmpdir	Lugar donde se crean archivos temporales
java.library.path	Lugar donde se abren las bibliotecas
java.runtime.name	Nombre del entorno de ejecución
java.runtime.version	Versión del entorno de ejecución
java.specification.name	Nombre de las especificaciones
java.specification.vendor	Distribuidor de las especificaciones
java.specification.version	Versión de la especificación
java.util.prefs.PreferencesFactory	Clase Factory para trabajar con preferencias
java.vendor	Nombre del distribuidor
java.vendor.url	URL del distribuidor
java.vendor.url.bug	URL del distribuidor para notificar errores en el código
java.version	Número de la versión
java.vm.info	Información sobre la máquina virtual
java.vm.name	Nombre de la máquina virtual
java.vm.specification.name	Nombre de las especificaciones de la máquina virtual
java.vm.specification.vendor	Distribuidor de las especificaciones de la máquina virtual
java.vm.specification.version	Versión de las especificaciones de la máquina virtual
java.vm.vendor	Distribuidor de la máquina virtual

Nombre	Descripción
java.vm.version	Versión de la máquina virtual
line.separator	Cadena especifica de la plataforma para separar líneas
os.arch	Arquitectura del Chip
os.name	Sistema operativo
os.version	Versión del sistema operativo
path.separator	Cadena utilizada para separar directorios en la ruta de clases
sun.arch.data.model	Anchura de bits de la CPU
sun.boot.class.path	Lugar donde se encuentran las clases del sistema
sun.boot.library.path	Lugar donde se encuentran las bibliotecas del sistema
sun.cpu.endian	Orden Endian
sun.cpu.isalist	Lista de las arquitecturas de CPU soportadas
sun.io.unicode.encoding	Clase del sistema para convertir caracteres Unicode en bytes y viceversa
sun.java2d.fontpath	Directorio de las fuentes
sun.os.patch.level	Nivel de ajuste del sistema operativo
user.country	Código del país del usuario
user.dir	Directorio actual
user.home	Directorio raíz del usuario
user.language	Código del idioma del usuario
user.name	Nombre de registro
user.timezone	Código de la zona horaria del usuario
user.variant	Código de variación del usuario

Tabla A.11. Tabla de flujos E/S

Nombre de la clase	Descripción
AudioInputStream	Flujo de entrada para audio
BufferedInputStream	Entrada de bytes en búfer

Nombre de la clase	Descripción
BufferedOutputStream	Salida de bytes del búfer
BufferedReader	Entrada de caracteres en búfer
BufferedWriter	Salida de caracteres del búfer
ByteArrayInputStream	Flujo de entrada para leer un array de bytes
ByteArrayOutputStream	Flujo de salida para escribir un array de bytes
CharArrayReader	Flujo de entrada para leer un array de caracteres
CharArrayWriter	Flujo de salida para escribir un array de caracteres
CheckedInputStream	Flujo de entrada para mantener un checksum al leer
CheckedOutputStream	Flujo de salida para mantener un checksum al escribir
CipherInputStream	Flujo de entrada para desencriptar bytes al leer
CipherOutputStream	Flujo de salida para encriptar bytes al escribir
DataInputStream	Flujo de entrada para leer tipos estándar de datos
DataOutputStream	Flujo de salida para escribir tipos estándar de datos
DeflaterOutputStream	Flujo de salida para comprimir datos al escribir
DigestInputStream	Flujo de entrada para mantener una recopilación de mensajes al leer
DigestOutputStream	Flujo de salida para mantener una recopilación de mensajes al escribir
FileInputStream	Flujo de entrada para leer bytes de un archivo
FileOutputStream	Flujo de salida para escribir bytes
FileReader	Flujo de entrada para leer caracteres de un archivo
FileWriter	Flujo de salida para escribir caracteres
FilterInputStream	Flujo de entrada para filtrar bytes base
FilterOutputStream	Flujo de salida para filtrar bytes base
GZIPInputStream	Flujo de entrada para descomprimir datos GZIP al leer
GZIPOutputStream	Flujo de salida para comprimir datos GZIP al escribir
InflaterInputStream	Flujo de entrada para descomprimir datos al leer
InputStream	Flujo de entrada de bytes base

Nombre de la clase	Descripción
`InputStreamReader`	Flujo de entrada para convertir bytes a caracteres y viceversa
`JarInputStream`	Flujo de entrada para descomprimir datos JAR al leer
`JarOutputStream`	Flujos de salida para comprimir datos JAR al escribir
`LineNumberReader`	Flujo de entrada para contar líneas
`ObjectInputStream`	Flujo de entrada para leer objetos
`ObjectOutputStream`	Flujo de salida para escribir objetos
`OutputStream`	Flujo de salida de bytes base
`OutputStreamWriter`	Flujo de salida para convertir bytes a caracteres y viceversa
`PipedInputStream`	Flujo de entrada para leer bytes en los threads
`PipedOutputStream`	Flujo de salida para escribir bytes en threads
`PipedReader`	Flujo de entrada para leer caracteres en threads
`PipedWriter`	Flujo de salida para escribir caracteres en los threads
`PrintStream`	Flujo de salida para convertir tipos de datos estándar en bytes
`PrintWriter`	Flujo de salida para convertir tipos de datos estándar en caracteres
`ProgressMonitorInputStream`	Flujo de entrada para gestionar la lectura. Si la lectura tarda demasiado, aparece un cuadro de diálogo.
`PushbackInputStream`	Flujo de entrada para bytes no leídos
`PushbackReader`	Flujo de entrada para caracteres no leídos
`RandomAccessFile`	Soporta archivos de E/S de bytes y tipos de datos estándar
`Reader`	Flujo de entrada de caracteres base
`SequenceInputStream`	Soporta la lectura de bytes en flujos de entrada múltiples
`StringReader`	Flujo de entrada para leer caracteres de un `String`
`StringWriter`	Flujo de salida para escribir caracteres en un `StringBuffer`
`Writer`	Flujo de salida de caracteres base

Nombre de la clase	Descripción
`ZipInputStream`	Flujo de entrada para descomprimir datos ZIP al leer
`ZipOutputStream`	Flujo de salida para comprimir datos ZIP al escribir

Tabla A.12. Sintaxis de formato de números

Patrón	Valor	Resultado	Descripción
0000	314	0314	0 - Utilizado para representar un dígito. Siempre presente.
###0	314	314	# - Utilizado para representar un dígito. Ausente si el primer dígito de todo el número es cero, o si el último dígito es cero para valores fraccionales.
##0.0#	3.10	3.1	. - Separador decimal. Se localiza a una coma en países europeos.
#,##0	3000	3,000	, - Separador de grupo. Se localiza a los caracteres locales.
##0.00%	3.1415	314.15%	% - Valor múltiple de 100 en forma de porcentaje
##0.00\u2030	3.1415	3141.50‰	\u2030 - Valor múltiple de mil en forma de millar.
\u00A4##0	314	$314	\u00A4 - Símbolo monetario. Se localiza según el símbolo local.

Tabla A.13. Sintaxis de formato de fecha

Símbolo	Significado	Resultado
`G`	Designador de Era	BC
`y`	Año	2001
`M`	Mes	Tres "Emes" para abreviatura del mes (En); cuatro o más para nombre completo (Enero); menos de tres, para el número (01).
`D`	Día del mes	01
`H`	Hora AM/PM (1-12)	11
`H`	Hour in day (0-23)	23

Símbolo	Significado	Resultado
m	Minutos	59
s	Segundos	13
S	Milisegundos	314
E	Día de la semana	Cuatro o más "Es" para el nombre completo (Domingo); menos de cuatro para abreviatura (Dom). D Día del año 14
F	Día de la semana en el mes	2
w	Semana del año	22
W	Semana en el mes	2
a	Marcador AM/PM	PM
k	Hora del día (1-24)	13
K	Hora en AM/PM (0-11)	0
z	Zona horaria	Cuatro o más "Zetas" para el nombre completo (Hora de la costa este); menos de cuatro para la abreviatura (EST)
''	Comillas sencilla	'
'	Escape para texto	o´clock

Indicadores de Applet

Tabla A.14. Parámetros de las etiquetas APPLET

Parámetro	Descripción
ALIGN	Ajustes de alineación del Applet (como IMG).
ALT	Texto que se muestra cuando la etiquetaAPPLET se entiende pero se anula.
ARCHIVE	Lista de archivos separados por comas para una carga inicial.
BOXBGCOLOR	Color de fondo del cuadro applet, que se muestra mientras se carga.
BOXFGCOLOR	Color frontal del cuadro applet, que se muestra mientras se carga.
BOXMESSAGE	Mensaje que se muestra mientras se carga el applet.

Parámetro	Descripción
CODE	OBJECT o CODE requeridos. Clase principal de applet.
CODEBASE	URL del código base
HEIGHT	Altura de los píxeles del applet.
HSPACE	Espacio extra de píxeles, que se deja a la izquierda y a la derecha del applet
MAYSCRIPT	Designa si se permite la comunicación entre el JavaScript y el applet.
NAME	Nombre del applet
PROGRESSCOLOR	Color de la barra de progreso que se muestra al cargar el applet.
OBJECT	OBJECT o CODE requeridos. Versión serializada de la clase principal del applet.
VSPACE	Espacio extra de píxeles, que se deja por encima y por debajo del applet.
WIDTH	Anchura de los píxeles del applet.

Tabla A.15. Tareas más comunes

Tarea	Descripción
Conectarse al servidor	`Socket socket = new Socket(getCodeBase().getHost(), port);`
Obtener parámetros	`String s = getParameter("name");`
Abrir nueva página	`getAppletContext().showDocument(urlString);` o `getAppletContext().showDocument(urlString, target);` Donde el destino es: `_self`: Mismo marco que `applet` `_parent`: Marco padre de `applet` `_top`: Marco superior de `applet window` `_blank`: Nuevo marco superior sin nombre `"name"`: Nuevo marco superior con marco
Abrir imágenes	`Image img = getImage(getDocumentBase(), "image.jpg");`
Marco de multitarea	`Thread runner; public void start() { if (runner == null) {`

Tarea	Descripción

	`runner = new Thread(aRunnable);`
	`}`
	`runner.start();`
	`}public void stop() {`
	`runner = null;}`
	`// dentro de Runnable`
	`public void run() {`
	`while (runner != null) {`
	`// realiza la tarea }}`

Ejecutar sonidos	`play(getDocumentBase(), "sound.wav");`
	o
	`AudioClip clip =`
	`getAudioClip(getDocumentBase(), "sound.au");`
	`clip.play(); // una vez`
	`clip.loop(); // repetidamente`
	`clip.stop(); // detiene la ejecución`

Mostrar mensaje en la barra de estado del buscador	`getAppletContent().showStatus(message);`

Resumen de los componentes de Swing

La siguiente tabla contiene una lista de componentes Swing, sus correspondientes modelos de datos (interfaz e implementación) y los eventos y propiedades más importantes. Las listas de propiedades contienen una selección de las más utilizadas.

Tabla A.16. Componentes Swing

Componente	Modelo de datos	Propiedades más importantes

`AbstractButton`	`ButtonModel`
	`DefaultButtonModel/`
	`JToggleButton.ToggleButtonModel`
		Véase `JComponent`
		`action`
		`horizontalAlignment`
		`horizontalTextPosition`
		`icon`
		`mnemonic`
		`selected`
		`text`
		`verticalAlignment`
		`verticalTextPosition`
		`model.group (ButtonGroup)`

`Box`	n/a	**Véase** `JComponent`
`JApplet`	n/a	`background`

Componente	Modelo de datos	Propiedades más importantes
		contentPane font foreground jMenuBar
Jbutton	**Véase** AbstractButton	**Véase** AbstractButton
JcheckBox	**Véase** AbstractButton	**Véase** AbstractButton borderPaintedFlat
JcheckBoxMenuItem	**Véase** AbstractButton	**Véase** JMenuItemstate
JcolorChooser	ColorSelectionModel DefaultColorSelectionModel	**Véase** Jcomponent chooserPanels color previewPanel
JcomboBox	ComboBoxModel DefaultComboBoxModel	**Véase** Jcomponent editable editor itemCount keySelectionManager lightWeightPopupEnabled maximumRowCount popupVisble renderer selectedIndex selectedItem
Jcomponent	n/a	actionMap alignmentX alignmentY background border cursor enabled font foreground height inputMap inputVerifier insets locale maximumSize minimumSize name nextFocusableComponent opaque parent preferredSize size toolkit toolTipText width x y
JDesktopPane	DesktopManager DefaultDesktopManager	**Véase** JLayeredPane allFrames dragMode selectedFrame
JDialog	n/a	**Véase** JWindow

Componente	Modelo de datos	Propiedades más importantes
		defaultCloseOperation jMenuBar
JeditorPane	**Véase** JtextComponent	**Véase** JtextComponent contentType editorKit page
JFileChooser	ListModel BasicDirectoryModel	**Véase** JComponent acceptAllFileFilter acceptAllFileFilterUsed accessory currentDirectory dialogTitle dialogType directionSelectionEnabled fileFilter fileHidingEnabled fileSelectionEnabled fileSelectionMode multiSelectionEnabled selectedFile selectedFiles
JFormattedTextField	**Véase** JTextComponent **y** AbstractFormatter DefaultFormatter/ MaskFormatter/ InternationalFormatter /DateFormatter/ NumberFormatter	**Véase** JTextField focusLostBehavior value
JFrame	n/a	**Véase** JWindow defaultCloseOperation extendedState iconImage jMenuBar resizable title undecorated
JInternalFrame	JDesktopPane	**Véase** JComponent closable closed contentPane defaultCloseOperation desktopIcon frameIcon icon iconifiable jMenuBar maximizable minimizable maximum resizable selectedtitle
JLabel	n/a	**Véase** JComponent displayedMnemonic horizontalAlignment horizontalTextPosition icon labelFor

Componente	Modelo de datos	Propiedades más importantes
		text verticalAlignment verticalTextPosition
JLayeredPane	n/a	**Véase** JComponent
JList	ListModel DefaultListModel y ListSelectionModel DefaultListSelectionModel	**Véase** JComponent anchorSelectionIndex cellRenderer firstVisibleIndex fixedCellHeight fixedCellWidth lastVisibleIndex leadSelectionIndex listData maxSelectionIndex minSelectionIndex prototypeCellValue selectedIndex selectedIndices selectedValue selectedValues selectionMode visibleRowCount
JMenu	**Véase** AbstractButton	**Véase** JMenuItem delay popupMenuVisible selected
JMenuBar	SingleSelectionModel DefaultSingleSelectionModel	**Véase** JComponent borderPainted helpMenu managingFocus selected
JMenuItem	**Véase** AbstractButton	**Véase** AbstractButton accelerator
JOptionPane	n/a	**Véase** JComponent icon initialSelectionValue initialValue inputValue maxCharactersPerLineCount message messageType options optionType selectionValues value wantsInput
JPanel	n/a	**Véase** JComponent layout
JPasswordField	**Véase** JTextComponent	**Véase** JTextField echoChar password
JPopupMenu	SingleSelectionModel DefaultSingleSelectionModel	label lightWeightPopupEnabled visible

Componente	Modelo de datos	Propiedades más importantes
JProgressBar	BoundedRangeModel DefaultBoundedRangeModel	Véase JComponent borderPainted maximum minimum orientation percentComplete string stringPainted value
JRadioButton	Véase AbstractButton	Véase AbstractButton
JRadioButtonMenuItem	Véase AbstractButton	Véase JMenuItem
JRootPane	n/a	Véase JComponent contentPane defaultButton glassPane jMenuBar layeredPane
JScrollBar	BoundedRangeModel DefaultBoundedRangeModel	Véase JComponent blockIncrement maximum minimum orientation unitIncrement value valueIsAdjusting visibleAmount
JScrollPane	n/a	Véase JComponent columnHeader columnHeaderView rowHeader rowHeaderView viewport viewportView
JSeparator	n/a	Véase JComponent orientation
JSlider	BoundedRangeModel DefaultBoundedRangeModel	Véase JComponent inverted labelTable majorTickSpacing maximum minimum minorTickSpacing orientation paintLabels paintTicks paintTrack snapToTicks value valueIsAdjusting
JSpinner	SpinnerModel SpinnerDateModel /SpinnerListModel/ SpinnerNumberModel	Véase JComponent editor value
JSplitPane	n/a	Véase JComponent

Componente	Modelo de datos	Propiedades más importantes
		bottomComponent continuousLayout dividerLocation dividerSize lastDividerLocation leftComponent maximumDividerLocation minimumDividerLocation oneTouchExpandable orientation rightComponen topComponent
JTabbedPane	SingleSelectionModel DefaultSingleSelectionModel	**Véase** JComponent selectedComponent selectedIndex tabCount tabLayoutPolicy tabPlacement
JTable	TableModel DefaultTableModel y TableColumnModel DefaultTableColumnModel y ListSelectionModel DefaultListSelectionModel	**Véase** JComponent cellEditor cellSelectionEnabled columnCount columnSelectionAllowed editing gridColor intercellSpacing rowCount rowHeight rowSelectionAllowed selectedColumn selectedColumnCount selectedColumns selectedRow selectedRowCount selectedRows selectionBackground selectionForeground selectionMode showGrid showHorizontalLines showVerticalLines tableHeader
JTextArea	**Véase** JTextComponent	**Véase** JTextComponent columns lineCount lineWrap rows tabSize wrapStyleWord
JTextComponent	Document PlainDocument/ StyledDocument/ DefaultStyledDocument/ HTMLDocument	**Véase** JComponent actions caret caretColor caretPosition document editable highlighter

Componente	Modelo de datos	Propiedades más importantes
		keymap selectedText selectedTextColor selectionColor selectionEnd selectionStart text
JTextField	**Véase** JTextComponent	**Véase** JTextComponent action columns horizontalAlignment
JTextPane	**Véase** JTextComponent	**Véase** JEditorPane characterAttributes inputAttributes logicalStyle paragraphAttributes styledDocument
JToggleButton	**Véase** AbstractButton	**Véase** AbstractButton
JToolBar	n/a	**Véase** JComponent borderPainted floatable orientation
JToolTip	ToolTipManager	**Véase** JComponent component tipText
JTree	TreeModel DefaultTreeModel**y** TreeSelectionModel DefaultTreeSelectionModel	**Véase** JComponent anchorSelectionPath cellEditor cellRenderer editable editing editingPath expandsSelectedPath fixedRowHeight largeModel lastSelectedPathComponent leadSelectionPath leadSelectionRow maxSelectionRow minSelectionRow rootVisible rowCount rowHeight scrollOnExpand selectionCount selectionPath selectionPaths selectionRow selectionRows showsRootHandles toggleClickCount visibleRowCount
JViewport	n/a	**Véase** JComponent optimizedDrawingEnabled scrollMode

Componente	Modelo de datos	Propiedades más importantes
		view viewPosition viewRect viewSize
JWindow	n/a	contentPane focusableWindow glassPane layeredPane rootPane

Asistencia para las herramientas de líneas de comandos

Tabla A.17. Opciones de javac

Opción	Descripción
`-classpath path`	Especifica el lugar donde debe buscar el compilador las clases de fuera del sistema.
`-d directory`	Especifica el destino de los archivos de clase.
`-deprecation`	Muestra los mensajes de advertencia de elementos despreciados.
`-encoding name`	Código del estado del archivo fuente.
`-g`	Genera información para reparar errores; puede utilizar `-g:none, -g:source, -g:lines, -g:vars.`
`-help`	Muestra la ayuda del compilador.
`-nowarn`	Opción para no mostrar los mensajes de advertencia.
`-source release`	Tareas de compilación disponibles con aserciones, los valores por defecto son de 1.3 (sin aserciones).
`-sourcepath path`	Especifica el lugar donde el compilador debe buscar los archivos fuentes adicionales.
`-verbose`	Compilación en modo de texto pormenorizado.
`-target release`	Genera los archivos de clases para la versión especifica; el valor por defecto es de 1.2 (en todos los lugares).
`-bootclasspath path`	Especifica el lugar donde el compilador debe buscar las clases del sistema.

Opción	Descripción
`-extdirs path`	Especifica el lugar donde los compiladores deben buscar las bibliotecas de extensiones.
`-X`	Muestra la ayuda para las opciones no estándar.
`-Xstdout file`	Redirecciona la salida estándar del compilador a un archivo.
`-Joption`	Transmite las opciones para ser ejecutadas por el compilador.
`@argsfile`	Lee los argumentos de los archivos.

Tabla A.18. Opciones de java*

Opción	Descripción
`-cp path` `-classpath path`	Especifica el lugar donde el compilador debe buscar las clases de fuera del sistema.
`-da` `-disableassertions`	Anula la comprobación de las aserciones de ejecución. Utilizada en combinación con `-ea` para anular la comprobación con algunos paquetes y/o clases.
`-Dproperty=value`	Define una propiedad del sistema.
`-dsa` `-disablesystemassertions`	Anula la comprobación de las aserciones de ejecución de las clases del sistema.
`-ea` `-enableassertions`	Habilita la comprobación de las aserciones de ejecución de las clases del usuario. Puede además especificar algunas clases o paquetes concretos para que sean comprobados.
`-esa` `-enablesystemassertions`	Habilita la comprobación de las aserciones de ejecución de las clases del sistema.
`-help` `-?`	Muestra ayuda de ejecución.
`-jar`	Encuentra la clase principal dentro del archivo JAR, como se especifica en la línea `Main-Class` del archivo manifiesto.
`-showversion`	Muestra información sobre la versión y después continúa.
`-verbose`	Ejecuta en modo verboso. Puede utilizar `-verbose:class`, `-verbose:gc`, `-verbose:jni`.
`-version`	Muestra información sobre la versión y después se detiene.
`-X`	Muestra ayuda para las opciones no estándares.

Opción	Descripción
`-Xbatch`	Anula la compilación de fondo.
`-Xbootclasspath/a:path`	Añade elementos específicos al classpath del sistema.
`-Xbootclasspath/p:path`	Devuelve elementos específicos al classpath del sistema.
`-Xbootclasspath:path`	Especifica el lugar donde el tiempo de ejecución debe buscar las clases del sistema.
`-Xcongc`	Habilita la liberación automática de memoria (requiere una CPU adicional).
`-Xdebug`	Comienza el debugger o reparador de errores de Java (jdb).
`-Xfuture`	Realiza la comprobación del formato de los archivos de clase.
`-Xincgc`	Habilita el incremento de la liberación de espacio.
`-Xint`	Opera sin HotSpot, en modo de sólo interpretación.
`-Xloggc:file`	Registra los eventos de liberación de espacio en el archivo.
`-Xmixed`	Valor por defecto de Windows. Opera con HotSpot, en modo sólo de mezclado.
`-Xmsn`	Especifica el tamaño inicial del heap; el valor por defecto es de 2 megabytes. Emplea k/K para especificar los kilobytes y m/M para los megabytes; por ejemplo, -Xms32m para 32 megabytes.
`-Xmxn`	Especifica el máximo tamaño del heap; el valor por defecto es de 64 megabytes. Utiliza k/K para especificar los kilobytes y m/M para los megabytes.
`-Xnoclassgc`	Anula la liberación de espacio de la clase.

`-Xprof`
`-Xrunhprof:help`
`-Xrunhprof:option=value`
	Genera datos del perfil. Puede especificar lo que aparece en flujos de salida informativos mediante listas de opciones y valores, separadas por comas.

`-Xrs`	Reduce el uso de las señales del sistema operativo de la máquina virtual.
`-Xssn`	Especifica el tamaño stack de un thread.
`-client` `-hotspot`	Valor por defecto de Unix. Selecciona la máquina virtual del cliente de HotSpot.
`-server`	Selecciona la máquina virtual del servidor de HotSpot.
`-d32` `-d64`	Especifica el modo de 32 o de 64 bytes.

* No todas las opciones están disponibles en todas las plataformas.

Tabla A.19. Opciones de appletviewer

Opción	Descripción
`-debug`	Inicializa el `appletviewer` de Java en el reparador de errores de Java.
`-encoding name`	Código de estado del archivo HTML.
`-J option`	Pasa la opción "`java`" al tiempo de ejecución.

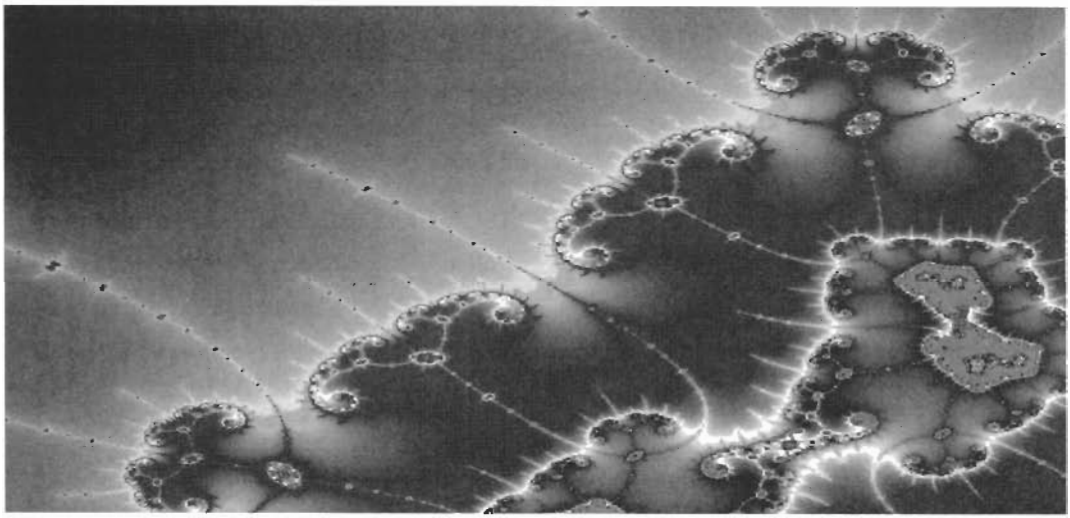
Tabla A.20. Tareas de jar

Opción	Descripción
`c`	Crea un nuevo archivo JAR.
`t`	Hace un listado del contenido de los archivos JAR.
`x file`	Extrae un archivo del archivo JAR; si no se especifica, se extrae todo.
`f file`	Se utiliza para especificar el archivo JAR que debe ejecutarse.
`v`	Genera un flujo de salida pormenorizado.
`m file`	Se emplea para especificar un archivo manifiesto.
`0`	Se emplea para no comprimir archivos añadidos.
`M`	Se emplea para no crear un archivo manifiesto.
`u`	Actualiza el archivo JAR.
`I`	Genera información sobre el índice de los archivos JAR.
`-C`	Cambia el directorio mientras otros comandos JAR se están ejecutando.

Tabla A.21. Etiquetas de comentarios de javadoc

Etiqueta	Nivel	Descripción
`@author name`	clase	Autor del código.
`@beaninfo info`	n/a	Etiqueta interna empleada por Sun para generar BeanInfo de los componentes.

Etiqueta	Nivel	Descrición
`@deprecated why`	clase, método, campo	Explicación.
`@exception class description` `@throws class description`	método	Excepción arrojada por el método.
`@param name description`	método	Parámetro del método.
`@return description`	método	Valor devuelto del método.
`@see reference` `{@link reference}`	clase, método, campo	Hipervínculo, nombre de la clase, o enlace del método para información adicional.
`@serial description option`	campo	Describe el estado de la clase serializada; puede incluir, o excluir la descripción.
`@serialData`	método	Información extra de los documentos en el estado de las clases serializadas.
`@serialField name type description`	campo	Documenta el campo `ObjectStreamField` de la clase.
`@since version`	clase, método, campo	Cuando la primera versión se añade al sistema.
`@version text`	clase	Versión de la clase.
`{@docroot}`	n/a	Representa la ruta al directorio raiz. Se emplea como la ruta relativa hacia los archivos adicionales.



Apéndice B
Recursos en la Red

Existen gran multitud de recursos en la Red sobre Java. Las mayores recopilaciones proceden de About.com y de Open Directory Project, que ofrece un directorio de recursos que contiene más de tres millones de páginas.

Nota

Para más información sobre Open Directory Project, visite http://dmoz.org/about.html.

Otros recursos de gran utilidad se encuentran en http://java.about.com/ y en http://dmoz.org/Computers/Programming/Languages/Java/. A continuación, se muestran los recursos más útiles a los que recurrir para solucionar problemas. No pierda de vista el directorio dmoz.org para estar al día de las últimas actualizaciones. Es una buena práctica incluir las dudas o respuestas en grupos de noticias, foros, o listas de correo, para que la información llegue al mayor número de personas posible. Por ejemplo en jGuru (http://www.jguru.com), están las respuestas de más de 1000 de las preguntas más frecuentes.

Cómo encontrar respuestas

Las cuatro maneras más comunes de encontrar respuestas son las siguientes: Consultar en la lista FAQ de preguntas más frecuentes, o publicar la pregunta en un

foro, una lista de correo, o en un grupo de noticias. Lo más aconsejable es encontrar el lugar más adecuado según la naturaleza de la pregunta, y antes de eso, consultar la sección de preguntas más frecuentes, para comprobar que esa misma pregunta no se ha contestado todavía. Las siguientes recomendaciones son válidas para búsqueda en foros, listas de correo y grupos de noticias.

- Intente que la pregunta sea lo más concreta posible.
- Evite enviar definiciones de clase demasiado largas. En su lugar, utilice únicamente el fragmento en cuestión.
- Asigne un título adecuado al mensaje que resuma el contenido del mismo. Evite títulos como "Urgente" o "Ayuda".

Preguntas más frecuentes

Las listas de preguntas más frecuentes se crearon por consultores que tenían que responder las mismas preguntas una y otra vez. Los grupos de noticias más populares incluyen una sección de FAQ, que actualizan una vez al mes. Otras listas FAQ provienen de compañías de producción de nuevos productos y última tecnología.

Las siguientes son algunas de las listas FAQ, y de páginas Web que las incluyen, más útiles. En algunos ejemplos, especialmente los más antiguos, se incluye la última fecha de actualización. No olvide que algunas de estas listas no contienen información sobre las últimas versiones de Java.

- **http://www.jguru.com/faq/index.jsp:** jGuru dispone de una colección de listas FAQ y productos relacionados con API. Por lo general, las preguntas más frecuentes de los foros se incluyen en estas secciones.
- **http://www.afu.com/javafaq.html:** El escritor Peter van der Linden mantiene la lista oficial de FAQ para los grupos de noticias de `comp.lang.java.*`. En estas páginas, encontrará respuestas para muchas de las dudas que se le plantearán si está comenzando en este campo. La última actualización data de Febrero de 2001.
- **http://www.mindprod.com/gloss.html:** Roedy Green mantiene esta lista, la cual presenta forma de glosario. En ella encontrará trucos e ideas novedosas para sus proyectos de programación.
- **http://www.ibiblio.org/javafaq/javafaq.html:** Es la primera lista oficial de preguntas de `comp.lang.java`. La última actualización es de 1997, pero aún ofrece información de interés.

Foros

Gracias a los foros podrá publicar una pregunta en la Red con la finalidad de que alguien la responda. Muchos de ellos le permiten buscar respuestas a preguntas

antiguas. Las siguientes direcciones corresponden a algunos de los foros más populares:

- **http://www-105.ibm.com/developerworks/java_df.nsf/AllDiscussionsByTitle?OpenForm:** Este foro de IBM cuenta con un experto residente que le ayudará a resolver sus dudas.
- **http://www.javaranch.com/:** Los foros de Java Ranch se gestionan por los mismos usuarios y contiene respuestas a preguntas de todos los niveles.
- **http://www.jguru.com/forums/index.jsp:** Tanto los usuarios como el moderador responden las preguntas que llegan hasta este foro. Puede buscar respuestas a preguntas ya emitidas o preguntar dudas nuevas.
- **http://forum.java.sun.com/:** Los foros sobre Java de Sun se controlan por los mismos usuarios. Los mejores trucos se recopilan en artículos.
- **http://news.devx.com/:** DevX proporciona foros sin moderar para tecnologías Java y relacionadas.

Listas de correo

La mayoría de las listas de correo no están dirigidas a programadores nuevos en el lenguaje de Java. Considere utilizar los foros y los grupos de noticias si no puede encontrar la solución en las listas FAQ. Las listas de correo más útiles se encuentran en las siguientes direcciones:

- **http://archives.java.sun.com/:** Sun mantiene un directorio de todas las listas de correo específicas de API con archivos accesibles a través de la Red. Muchos de los desarrolladores de los API participan en estas listas de correo.
- **http://eos.dk/mailman/listinfo/swing:** Esta es una lista de correo específica para preguntas relacionadas con Swing.
- **http://lists.xcf.berkeley.edu/mailman/listinfo/advanced-java y http://discuss.develop.com/advanced-java.html:** Para temas avanzados de Java.
- **http://groups.yahoo.com/group/jbuilder-dev:** Lista de correo específica para JBuilder.

Grupos de noticias

Las listas de correo sólo nos permiten comunicarnos con un grupo limitado de personas, mientras que los grupos de noticias o Usenet son grupos abiertos. En lugar de basarse en mensajes distribuidos mediante correo electrónico o mediante su publicación en la Web, se trata de mensajes publicados en un servidor de noticias que pueden ser compartidos entre diferentes servidores. Cuando se envía un mensaje, tarda algún tiempo en propagarse por los servidores de noticias, por lo

cual, los mensajes de un servidor difieren de los de otro. Si su ISP no dispone de un servidor de noticias, visite la página http://groups.google.com/ para leer y publicar mensajes en estos grupos.

Los siguientes grupos de noticias son los de mayor utilidad:

- **comp.lang.java.help y comp.lang.java.programmer:** Para preguntas generales. El grupo de ayuda va dirigido a temas relacionados con la configuración, mientras que el grupo de programadores atiende las preguntas relacionadas con el lenguaje en sí y con las bibliotecas.
- **comp.lang.java.3d:** Para preguntas relacionadas con los API en tres dimensiones de Java.
- **comp.lang.java.advocacy:** Más que un lugar donde buscar respuestas, es un foro de discusión sobre las ventajas de Java comparada con Microsoft.
- **comp.lang.java.beans:** Aunque se creó originariamente para responder preguntas sobre temas relacionados con JavaBeans, ahora incluye también JavaBeans Enterprise(EJB), así como otras tecnologías componentes.
- **comp.lang.java.corba y comp.object.corba:** Dirigidos a preguntas sobre cómo trabajar con Java con la arquitectura CORBA.
- **comp.lang.java.databases:** Para preguntas sobre conectividad de Java con las bases de datos. El API de Java API especializado se llama JDBC, o Conectividad de bases de datos de Java.
- **comp.lang.java.gui:** Aquí encontrará soluciones a temas relacionados con el diseño y la construcción de interfaces gráficas de usuario (GUI) con AWT y Swing.
- **comp.lang.java.machine:** Grupo dirigido a preguntas sobre JVM, o equipo virtual de Java.
- **comp.lang.java.security:** El mejor lugar para discutir temas de seguridad y encontrar soluciones a las dudas sobre los API de seguridad de Java.
- **comp.lang.java.softwaretools:** Para dudas sobre las herramientas de desarrollo existentes en el mercado.
- **alt.comp.lang.java-games:** Si lo que busca son respuestas a dudas sobre la creación de juegos con Java, este grupo le será de gran utilidad.
- comp.object: Para preguntas generales sobre programación orientada a objetos.

Los siguientes grupos son específicos de temas relacionados con el lenguaje.

cz.comp.lang.java

de.comp.lang.java

fido7.ru.java

fj.lang.java

fr.comp.lang.java

han.comp.lang.java

hun.comp.lang.java

it.comp.java

microsoft.public.de.german.java

microsoft.public.es.java

no.it.programming.java

pl.comp.lang.java

tw.bbs.comp.lang.java

Recursos generales

Algunas páginas no pertenecen a una categoría concreta pero proporcionan gran cantidad de información útil para los desarrolladores de Java. El orden en el que se presentan se basa en su utilidad, siendo las primeras las más recomendables.

- **http://java.sun.com/:** La página fundamental a la que acudir para conseguir la información oficial más actualizadas sobre Java.
- **http://java.sun.com/jdc/:** JDC o Conexión de desarrolladores de Java contiene tutoriales, trucos, así como descargables de beta.
- **http://www.ibiblio.org/javafaq/:** El escritor Elliotte Rusty Harold ofrece sus comentarios sobre las últimas novedades de Java en Cafe au Lait.
- **http://www.javaworld.com/:** JavaWorld es una revista online que ofrece tutoriales y consejos semanalmente.
- **http://www.ibm.com/java/:** Esta página de IBM contiene trucos, tutoriales y su propio foro de debate.
- **http://www.onjava.com/:** Este portal de O'Reilly & Associates contiene artículos y recursos de Java.
- **http://www.javacoffeebreak.com/:** David Reilly ofrece su colección de trucos, tutoriales, así como comentarios sobre los últimos manuales.
- **http://www.javaskyline.com/:** Colección de recursos y noticias de Rich Katz.
- **http://www.cetus-links.org/oo_java.html:** El equipo de Cetus mantiene una amplia colección de enlaces relacionados con Java.

- **http://www.javalobby.org/:** Java Lobby es una organización formada por desarrolladores con la finalidad de mantener un Java abierto.
- **http://javaboutique.internet.com/:** Java Boutique dispone de tutoriales, artículos y de una amplia colección de applets de Java, muchos de ellos con instrucciones específicas de instalación.

Grupos de usuarios

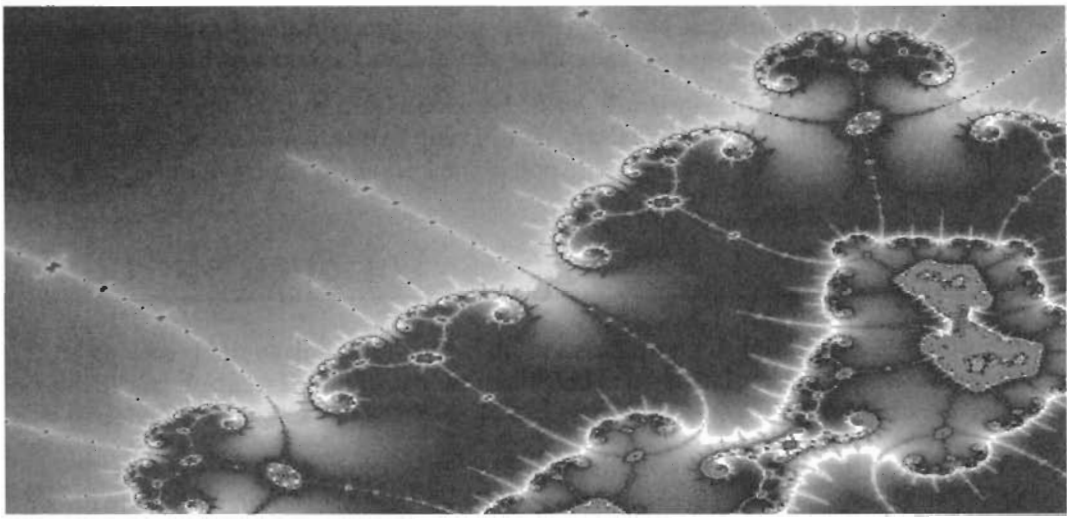
Los grupos de usuarios es el centro de reunión de desarrolladores que quieren compartir sus dudas e inquietudes. Muchos de los ofrecen listas de correo específicas y grupos de apoyo.

- **http://servlet.java.sun.com/jugs:** Sun mantiene el directorio principal de grupos de usuarios de Java e incluye solicitudes de personas interesadas en crear nuevos grupos y listas. Desafortunadamente, este directorio no se actualiza con demasiado frecuencia, por lo que algunos de los grupos que aparecen ya han desaparecido.
- **http://dmoz.org/Computers/Programming/Languages/Java/User_Groups/:** Corresponde al grupo de usuarios de Open Directory Project e incluye un listado de grupos que tienen su propia página Web.

Ayuda técnica del producto

Muchos distribuidores facilitan ayuda técnica específica para sus productos. Las cuatro páginas siguientes tratan sobre las cuatro herramientas fundamentales de IDE.

- **http://community.borland.com/java/:** Comunidad de desarrolladores de JBuilder de Borland.
- **http://www.sun.com/forte/ffj/resources/:** Forte de Sun para recursos de desarrolladores de Java.
- **http://www-4.ibm.com/software/ad/vajava/support/:** VisualAge de IBM para el centro técnico de desarrolladores de Java.
- **http://www.webgain.com/support/online_support/visual_cafe.html:** El centro de asistencia técnica de VisualCafé de WebGain.



Apéndice C
Contenido del CD-ROM

El CD-ROM que acompaña a este libro contiene elementos que han sido seleccionados especialmente para aumentar la utilidad del mismo.

La estructura de carpetas del CD-ROM es la que se muestra a continuación:

- **Código**: En este directorio encontrará los ejemplos desarrollados a lo largo del libro.

- **Software**:
  - **JBuilder 8**: Contiene la versión de prueba de treinta días de la Edición Personal del galardonado entorno de desarrollo integral (IDE) JBuilder 8. Se incluye para Windows, Linux, Solaris.

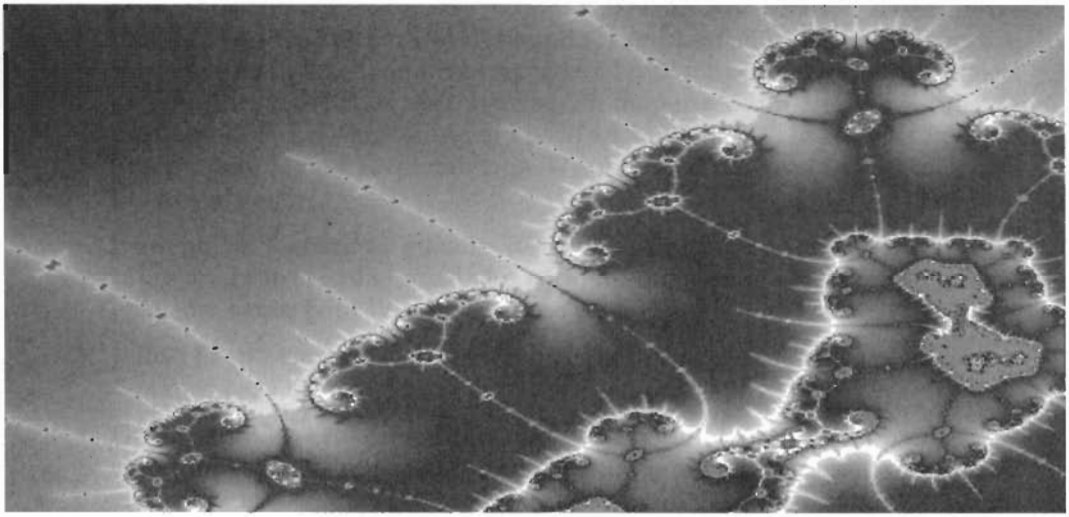
    Para poder utilizar JBuilder necesitará un número de serie, para obtenerlo deberá registrarse en la siguiente dirección: http://www.borland.com/jbuilder/offers/
  - **Ant**: Incluye la distribución binaria de Ant, herramienta de construcción de programas escrita en Java. Se incluye tanto para Windows como para Linux.
  - **BDK**: Beans Development Kit (BDK), con el que podrá desarrollar JavaBeans. Se incluye la versión para Solaris. Los usuarios de Windows podrán descargársela de la dirección:

http://java.sun.com/products/javabeans/software/bdk_download.html

- **Java_SDK**: Java 2 Platform Standard Edition, versión 1.4. Contiene la plataforma integral para el desarrollo e implantación de aplicaciones y servicios destinada al ámbito empresarial más exigente. La última versión de J2SE incorpora nuevas funciones, un mayor rendimiento y mejoras de escalabilidad que representan un paso adelante para la tecnología Java. Se incluye tanto para la plataforma Windows como para las plataformas Linux y Unix.
- **Jindent**: Se incluye la edición de prueba de Jindent, con el que podrá cambiar el formato de letra de los códigos, una vez que haya terminado de crearlos. Tanto para Windows, como para Unix.
- **JUnit**: El entorno que le permitirá ejecutar tests de clases Java. Esta incluido tanto para Linux y Unix cómo para Windows.
- **Optimizeit**: En este directorio encontrará la Suite Optimizeit, una aplicación pionera en la optimización del rendimiento en Java que proporciona características avanzadas, fáciles de usar y con gran capacidad de ampliación. Se incluye la versión de evaluación, para poder utilizarlo necesita registrarse y obtener una clave en la siguiente dirección:

  http://www.optimizeit.com/masteringjava.html
- **Tomcat**: Se incluye la última versión del software contenedor de servlets de Java, tanto para Windows como para Linux.

En el fichero leame.txt, situado en el directorio raíz del CD-ROM, encontrará una descripción más detallada de todas las utilidades y programas que se incluyen.



Glosario

A

Abstracto: Que mantiene las características esenciales de algún proceso o estructura. Lo contrario de Concreto.

ActiveX: Conjunto de tecnologías que emplean controles basados en el Modelo de objeto de componentes (COM). Este modelo se utiliza en todas las aplicaciones de escritorio de Windows para comunicación y automatización.

Administrador de diseño: Se encarga de controlar los estilos de diseño para los componentes de las interfaces gráficas de usuarios (GUI).

Alfa: Ajuste que mide la transparencia de un píxel.

Ambiente de desarrollo integrado (IDE): Programa, tal como JBuilder, que proporciona un ambiente gráfico que combina todas las herramientas requeridas para escribir código.

Ambiente de ejecución: Lugar donde se ejecutan los programas creados por la máquina virtual de Java (VM).

Ámbito: Visibilidad de una variable en un programa. Oscila entre ámbito local de variables definidas dentro de métodos a ámbito público de variables definidas con el modificador público.

Amistoso: Describe la protección de acceso por defecto cuando no existen ninguna de las claves de acceso privado, protegido, o público.

API: Véase Interfaz para la programación de aplicaciones (API).

API central de Java: Paquetes principales que constituyen el API de una versión concreta de la plataforma de Java. Contienen métodos y código para tareas de conexión, de E/S y trabajo con gráficos.

API de JavaBeans: Un API designado a crear componente reutilizables. Un bean de Java es un componente de software reutilizable.

Applet: Un programa de Java incluido en un documento Web que se ejecuta en un buscador.

Applets no seguros: Applets provenientes de una Red cuya fuente no puede ser detectada o de la que no se puede confiar.

Appletviewer: Utilidad SDK que muestra sólo los applets de Java en los documentos HTML; a diferencia de un buscador Web que muestra los applets que se encuentran dentro de documentos Web.

Archivo: Colección de archivos agrupados.

Archivo class: Archivo binario que contiene bytecodes. El compilador de Java genera un archivo de este tipo a partir del código fuente de cada clase Java.

Archivo de Java (JAR): Formato de archivo utilizado para agregar muchos archivos a uno, tal como TAR o ZIP Su creador puede permitir mayor acceso al usuario del mismo.

Archivo fuente: Archivo de texto con instrucciones legibles. Un archivo fuente de Java es un archivo de texto escrito en lenguaje Java.

Archivo JPEG (Grupo de expertos en fotografía unidos): Formato de fichero gráfico comprimido. Las imágenes pueden comprimirse con pérdida de calidad y de detalle, pero consiguiendo tamaños más pequeños, o manteniendo todos los detalles, pero con un grado de compresión menor.

Argumento: Parámetro transmitido a un método.

Array o vector: Grupo de variables ordenadas del mismo tipo a las que se puede hacer referencia con un mismo nombre. Un array es un objeto con la misma longitud que el miembro de datos públicos que contiene el tamaño del array.

Array multidimensional: Array de arrays. Puede no ser rectangular.

Aserción: Expresión booleana empleada para documentar las condiciones anteriores y posteriores a un punto específico del programa. Cuando se habilitan, se comprueban las expresiones y las aserciones falsas resultan en una condición anormal que interrumpe el flujo normal del programa (un error, no una excepción).

Atómica: Indivisible o ininterrumpible. En el contexto de multitareas, se trata de una operación (declaración, bloque de código, o un método completo) que nunca puede interrumpirse. El código que accede a estructuras de datos que se comparten con otros threads o tareas que deben proteger sus secciones más importantes haciéndolas atómicas.

AWT: Véase Kit de herramientas de ventana abstracta.

Barrido: Cuadrícula rectangular de píxeles.

Base de códigos: Lugar desde el que un applet obtiene sus clases.

BDK: Véase Kit de desarrollo de Beans.

Bean: Componente de software reutilizable que sigue una convención de nombres concreta para definir las propiedades y eventos que comprende. Puede manipularse visualmente mediante una herramienta de construcción.

BeanInfo: Clase que soporta beans y que define información sobre el mismo, con igual nombre base.

Biblioteca de clases: Colección de clases prefabricadas que puede utilizar el programador para construir aplicaciones de una manera más rápida y eficaz. La biblioteca de clases de Java es el API Core de API. Estas bibliotecas también se llaman paquetes.

Big-endian: Ordenación interna de bytes de datos en memoria, también llamada ordenación de Red que consiste en que los bytes superiores de una palabra se almacenen primero en memoria. Los procesadores RISC y Motorola utilizan este tipo de ordenación, mientras que el procesador Intel x86 utiliza la ordenación inversa, llamada little-endian.

BLIT: Transferencia de bloque de imágenes. En sistemas gráficos, corresponde a las operaciones de copiar de alta velocidad, realizadas por aceleradores. Algunos ejemplos de este tipo de transferencia son: desplazar una ventana, desplazarse por el contenido de la misma, o la creación de texto.

Bloque: Sección de código fuente que comienza con una llave abierta "{", termina con una cerrada "}", y contiene declaraciones entre ambas.

Booleano: Variable que sólo puede tomar valores de cierto y falso. Se utiliza para representar elementos de estados binarios, tales como conectado/desconectado, el mismo/diferente, o abierto/cerrado.

Búfer: Memoria intermedia utilizada para almacenar información de manera temporal. Existen búferes de disco de lectura y escritura y de marco, entre otros. En un sentido más amplio, es un sistema que desincroniza dos entidades, permitiéndolas ejecutarse a velocidades y en frecuencias diferentes.

Búferes dobles: Por lo general, se utilizan dos búferes para que uno de ellos pueda construirse mientras se emplea el primero. En tareas de animación, uno de ellos se emplea para mostrar el marco de una animación, mientras el siguiente marco desaparece de pantalla.

Buscador Web: Navegador para poder visualizar páginas Web en Internet. Algunos ejemplos son Internet Explorer, Opera, y Netscape.

Bytecodes: Código independiente de la máquina generado por el compilador de Java y ejecutado por la máquina virtual de Java (VM).

C

Campo: Miembro de una clase, generalmente no estático.

Cargador de clases: Parte de la máquina virtual de Java (VM) que obtiene clases del sistema de archivos del cliente o de la Red.

Casting: Conversión explícita de un tipo de datos a otro.

Certificación: Examen de Sun y otros distribuidores para evaluar la competencia en el tema. También se refiere a productos certificados, tales como J2EE.

Clase: Descripción de un tipo concreto de objeto de Java, que incluye las instrucciones correspondientes.

Clase abstracta: Clase que contiene uno o varios métodos abstractos y de la que, por tanto, no pueden crearse instancias.

Clase adapter: Clase que implementa todos los métodos de una interfaz. Se puede derivar esta clase para que anule sólo los métodos adecuados de la interfaz original. Se utiliza frecuentemente para escuchar eventos.

Clase base: Superclase de la que cada clase hereda sus miembros. En Java, todas las clases tienen la clase `Object` como su base principal.

Clase Padre: Véase superclase.

Clase raíz: Por lo general, cualquier clase que actúe como la superclase de una jerarquía concreta. En la jerarquía principal de Java, la clase `object` es la clase raíz.

Clases de fundación de Java (JFC): Extensión que añade bibliotecas de clase de interfaz gráfica de usuario al kit de herramientas de ventanas abstractas (AWT).

Clases identificables: Clases y applets de las que se puede averiguar su distribuidor, mediante una firma electrónica en el archivo `.class`.

Clases internas: Clases definidas dentro de otras clases, del mismo modo que los métodos se definen dentro de ellas. Tienen el mismo alcance que las variables y los métodos definidos dentro de la misma clase. Sólo las clases que las contienen son visibles.

CLASSPATH: Lugar desde donde el ambiente de ejecución abre los archivos de clase de los programas.

Clave: Palabra con un significado especial para el compilador de Java, tal como el nombre de los tipos de datos o del constructor del programa.

Clonar: Copiar un objeto mediante el método `clone()`.

Codificación: Mecanismo por el que los flujos de bytes de 8 bits se convierten en caracteres de 16 bits.

Colores aditivos: Colores primarios (rojo, verde, y azul) a partir de los cuales se pueden crear otro colores.

Comentario: Información proporcionada por el desarrollador para documentar el código fuente.

Comentarios de documentación: Bloque de comentarios que utilizan los documentos de Java para crear documentación.

Compilador: Programa que lee un archivo fuente, interpreta sus comandos y crea un nuevo archivo con los bytecodes equivalentes. Un compilador C crea un archivo con instrucciones de código de la máquina nativa.

Compilador JIT: Compilador que convierte todo el bytecode en código de la máquina nativa, de la misma forma que se ejecuta un programa Java. Como consecuencia, se mejora la velocidad en ejecución en comparación con el código interpretado por una máquina virtual Java.

Componente: Unidad de software reutilizable, tal como JavaBean o JavaBean Enterprise.

Composición: Proceso de superponer una imagen sobre otra para crear una sola imagen.

Conectividad de bases de datos de Java (JDBC): Lenguaje estándar de Java que define clases y métodos para interactuar con bases de datos.

Conexión en serie: Concepto de construir una clase E/S sencilla con algo que ofrezca mayores posibilidades.

Constructor: Método que tiene el mismo nombre que la clase que inicia.

Constructor por defecto: Constructor sin argumentos disponible de manera automática para las clases que no definen su propio constructor.

D

Debugger (reparador de errores): Utilidad para monitorizar y suspender la ejecución de un programa para examinar su estado.

Declaración: Línea de código fuente que termina en punto y coma (;). Si se necesitan colocar juntas varias declaraciones, deben formar un bloque, delimitado por llaves curvas.

Declaración break: Declaración de ruptura de flujo. Sin una etiqueta, el control se transfiere a la declaración justo después del bucle más interno o de la declaración `switch`. Con una etiqueta, el control se transmite a la declaración o bloque de declaraciones que llevan la misma etiqueta.

Declaración for: Declaración de bucle. Primero se ejecuta la parte de la inicialización seguido de la evaluación de la expresión condicional. Si esta expresión es verdadera se ejecuta el bucle seguido de la ejecución de la parte del incremento. Este ciclo se repite mientras la condicional sea verdadera.

Declaración if: Uno de los dos tipos de declaraciones condicionales. Consiste de la clave `if`, seguida de una expresión condicional entre paréntesis, y una declaración (o bloque de declaraciones) que se ejecuta cuando la condicional resulta verdadera. Puede estar seguida de la clave `else` y una declaración (o bloque de declaraciones) que se ejecuta cuando la condicional resulta verdadera

Declaración return: Declaración de ruptura de flujo. Se emplea para utilizar el control al invocador de dentro de un método o constructor. Antes de que el control se pase de nuevo al invocador, se ejecutan todas las declaraciones `try`.

Declaración switch: Declaración de multiselección. Se evalúa la integral y la primera declaración `case` cuya expresión constante tiene el mismo valor. Se ejecuta la declaración por defecto cuando no existe una declaración case con el mismo valor. La declaración `break` se utiliza como la última línea de una declaración `case` para que el control no continúe el la siguiente declaración `case`.

Declaración while: Una de las declaraciones de bucle. Se evalúa primero la condicional y si resulta verdadera, se ejecuta el bucle y se evalúa de nuevo la condicional. Este ciclo se repite hasta que la condicional resulta falsa.

Desacoplamiento: Acción de evitar un vínculo entre dos entidades al insertar una tercera entidad, que puede ser una entidad, un búfer, o todo el subsistema. Ir flexibilidad a cambio de una peor ejecución.

Deserialización: Proceso de restauración de objetos serializados.

Despreciada: Elementos de la biblioteca que se soportaban en versiones anteriores de SDK pero cuyo uso no se recomienda en nuevas versiones.

Destructor: Método llamado para destruir un objeto de la memoria. Java no soporta destructores directamente.

Disassembler: Utilidad que muestra las instrucciones de un archivo compilado. El disassembler de Java (javap) muestra lo que hará la VM cuando devuelve un archivo clase de bytecodes.

E

ECMAScript: Véase JavaScript.

Edición Enterprise de Java 2 (J2EE): Conjunto de varios API especiales de Java, tales como Enterprise JavaBeans (EJB), servlets, y Paginas de JavaServer (JSP).

Edición estándar de Java 2 (J2SE): Versión estándar de Java que se obtiene al adquirir el SDK de Java.

Edición Micro de Java 2 (J2ME): Subcategoría de Java estándar destinada a micro dispositivos tales como móviles, beepers, y asistentes digitales personales (PDA).

Encabezamiento del marco: Colección de campos al comienzo de un marco o paquete que contiene los datos necesarios para la operación correcta del protocolo. Un ejemplo, es el campo checksum que permite al destinatario comprobar si los datos llegan inalterados.

Encabezamientos de solicitud y respuesta: Términos empleados para indicar los encabezamientos multilíneales legibles de ASCII del protocolo HTTP. Este protocolo consiste de las solicitudes de un cliente (buscador) seguidas de las respuestas del servidor Web (Sitio Web).

Encapsulación: Inclusión del código y la información dentro de la misma entidad.

Estado: Modo tal como el binario de encendido o apagado.

Estático: Clave que indica que se ha definido una variable o un método en el nivel de la clase, comparado con el nivel de la muestra.

Evento: Objeto generado por el sistema. Los componentes de Java hacen de fuentes para los eventos tales como pulsaciones de teclado, clics de ratón o cambios de las propiedades de los beans.

Excepción: Condición anormal que interrumpe el flujo normal del programa.

Expresión: Combinación de términos que evalúa el único valor de un dato.

Expresión regular: Modelo de caracteres comodines que describe una cadena que buscar.

F

Factory: Modelo de diseño para crear muestras de una clase sin utilizar un constructor.

Filtro de convulsión: Filtro de procesamiento de imágenes que combina los píxeles alrededor del píxel central para formar uno nuevo con la densidad media de todos los píxeles.

Filtro de imagen: Técnica de alterar una imagen digital, generalmente para mejorarla. Puede ser tan simple como cambiar el brillo o tan complicado como aplicar correcciones ópticas para reducir los fallos en los elementos ópticos físicos del dispositivo que realizó la imagen.

Final: Modificador que indica que una clase, método o variable no es intercambiable.

Finalizer: Método llamado justo antes de la liberación de memoria.

Finally: Parte opcional de un bloque de control de excepciones que siempre se ejecuta.

Flushing: Proceso de escritura de los datos restantes en un búfer grabable. Los archivos de cierre y los flujos vacían los búferes de datos de manera automática; ésta es una de las razones por las que se cierran los archivos y los flujos.

Formato de extensiones multipropósito de correo de Internet (MIME): Extensiones del protocolo de correo de Internet que permiten incluir información adicional al simple texto. También las emplean los servidores Web para designar el tipo de datos enviados.

Formato GIF: Formato de fichero gráfico intercambiable diseñado para almacenar imágenes comprimidas.

Framework: Modelo reutilizable para crear clases comunes para las aplicaciones.

Frecuencia de marco: Frecuencia en la que se muestran las imágenes de una secuencia de animación. Las frecuencias normales van de 24Hz a 60Hz. Esta frecuencia se mide en fotogramas por segundo (fps).

Función del miembro: Véase método.

G

GIF: Véase Formato GIF.

Gráficos portables de Red PNG: Especificación diseñada para una mejor carga y calidad de los gráficos en Internet, cuya característica principal es que no reduce el número de colores de una imagen, lo que evita pérdida de calidad.

Gramática: Sintaxis utilizada para describir la composición de un programa válido de Java.

GUI: Véase Interfaz gráfica de usuario.

H

Heap: Área global dentro de la memoria donde se almacenan los objetos.

Herencia: Capacidad de una clase para extender la definición de una superclase. La nueva clase hereda las variables y funciones del miembro de dicha superclase. La nueva clase puede cancelar el comportamiento heredado.

Herencia múltiple: Práctica de derivar una clase de más de una clase base.

Herencia simple: Habilidad de escribir una clase que herede las variables y los métodos o funciones de miembro de una clase simple.

HTML: Véase lenguaje de marcado de hipertexto.

HTTP: Véase Protocolo de transferencia de hipertexto.

HTTPS: Véase Protocolo de transferencia de hipertexto seguro.

I

I18N: Abreviatura de Internacionalización; el número 18 representa los 18 caracteres que hay entre la I y la n final. Véase Internacionalización.

IDE: Véase Ambiente de desarrollo integrado.

Identificación o "signature": Reseña del método que consiste en su nombre, la lista de argumentos, y el tipo devuelto. Aunque dos métodos tengan el mismo nombre, si sus listas de argumentos son diferentes, el compilador no considera que haya ninguna relación entre ellos.

Identificador: Nombre que el programador asigna a una clase, variable o método.

Igual: Componente nativo de GUI asociado con un componente AWT. La mayoría de los componentes de Swing no tienen iguales.

Información sobre herramientas: Cuadro emergente que describe la función de un elemento y que aparece cuando el cursor se sitúa sobre un área designado con información definida.

Inicializador estático: Bloque de código declarado como un método en el nivel de clase, pero sin nombre de método.

Inmutable: Que no se puede cambiar.

Interbloqueo: Situación en la que threads múltiples no pueden continuar ya que cada uno de ellos está esperando a que otro libere el código necesario para continuar.

Interfaz: Declaraciones formales de métodos y constantes que deben definirse por las clases que las implementan.

Interfaz de programación de la aplicación (API): Conjunto de métodos, funciones, clases y bibliotecas proporcionadas por un sistema operativo que utilizan los desarrolladores a crear aplicaciones sin necesidad de crear de nuevo las funciones de bajo nivel. Todos los paquetes estándares de Java combinados forman el API Core de Java, o API principal.

Interfaz gráfica de usuario (GUI): Interfaz de iconos de los sistemas operativos actuales. Se conoce también como interfaz WIMP, de ventanas, iconos, menús y punteros.

Internacionalización: Proceso por el que un programa se adapta a los distintos idiomas y países del mundo.

Interpreter: Utilidad que lee los comandos de un archivo, para después interpretar y ejecutar cada uno de ellos por separado. **Intronspección:** Mecanismo que permite a las clases publicar las operaciones y propiedades que soporta. Mecanismo que soporta el descubrimiento de tales mecanismos.

IP: Véase Protocolo de Internet.

J

Java Web Start: Mecanismo para decodificar aplicaciones de Java a través de un buscador.

JavaScript: Lenguaje de programación ligeramente relacionado con Java. JavaScript puede codificarse directamente en un documento HTML. Su forma estandarizada recibe el nombre de ECMAScript.

JavaSoft: Antiguo nombre de la división de software de Java de Sun.

JDBC: Véase Conectividad de bases de datos de Java (JDBC).

JDK: Véase Kit de desarrollo de Java (JDK).

JFC: Véase Clases de fundación de Java (JFC)

JNLP: Véase Protocolo de Inicio en Red de Java (JNLP).

JIT: Véase compilador JIT

K

Kit de desarrollo de Beans (BDK): Herramienta de Sun que ayuda a los desarrolladores a crear componentes reutilizables (beans) incluyendo informa-

ción introductoria, un tutorial, beans de muestra, así como un contenedor de prueba.

Kit de desarrollo de Java (JDK): Nombre anterior del Kit de desarrollo de Software (SDK).

Kit de desarrollo de Software (SDK): Herramientas de desarrollo de Java. El SDK consiste en las clases Core API (incluyendo su código correspondiente), un compilador de Java (escrito en Java), así como el intérprete VM. Sun distribuye las implementaciones de referencia de SDK de manera gratuita para Linux, Microsoft Windows, y Solaris.

Kit de herramientas abstractas de Windows (AWT): Colección de clases de Java que permite implementar las interfaces GUI independientes o abstractas.

L

Lenguaje de marcado de hipertexto (HTML): Lenguaje utilizado para escribir documentos Web. Los applets de Java se encuentran dentro de estos documentos.

Lenguaje de marcado extensible (XML): Lenguaje de marcado utilizado para definir y utilizar las etiquetas necesarias para describir datos jerárquicos.

Lenguaje de modelo universal (UML): Modelo estándar utilizado para análisis y diseños orientados a objetos.

Liberación automática: También conocida como "recogida de residuos" corresponde a la detección y liberación automática de memoria que ya no está en uso. El sistema en ejecución de Java realiza la recogida de residuos, de esta forma los programadores nunca liberan los objetos de forma específica.

Línea base: Línea imaginaria en la que se escribe el texto.

Listener u oyente: Objeto que espera ser notificado cuando ocurre un evento en otro objeto.

Literal: Valor real que puede tomar la forma de un número o de una palabra.

Little-endian: Ordenación interna de bytes de datos en memoria, que consiste en que los bytes inferiores de una palabra se almacenen primero en memoria.

Llaves: Símbolos { } utilizados para delimitar bloques de código de Java.

Localizador de recursos uniforme (URL): Cadena que identifica la ubicación de un recurso y el protocolo empleado para acceder a él.

Look and feel (L&F): Apariencia y comportamiento de un GUI.

M

Main: Nombre del método en el que comienzan las aplicaciones de Java.

Manifiesto: Archivo de texto que describe el contenido de un archivo JAR.

Máquina Virtual de Java (VM): El sistema que abre, verifica y ejecuta bytecodes de Java.

Marcado: Técnica común utilizada para reducir la complejidad de la entrada de datos de texto.

Marco: En el contexto de los GUI, corresponde al contorno gráfico de una ventana. En el contexto de comunicaciones de datos, sinónimo de paquete.

Matrix: Array rectangular multidimensional.

Mensaje: Llamada a un método.

Método: Función o rutina parte de una clase, también conocido como función del miembro.

Método abstracto: Método declarado pero no implementado que se utiliza para asegurar que las subclases implementen al método. Una clase con uno o más métodos abstractos es, por definición, una clase abstracta.

Método de circuito corto: Detención de la evaluación de una expresión booleana una vez que puede determinarse su valor final y sin necesidad de evaluar las partes restantes de la expresión.

Métodos ignorados: Métodos definidos para tener implementaciones distintas en una subclase.

Método nativo: Código nativo para un procesador concreto. En Windows 2000, los métodos nativos se refieren al código escrito especialmente para procesadores x86 de Intel.

Métodos de ciclo de vida: Métodos de Applet, tal como `init()`, que son llamados por el buscador.

Métodos sobrecargados: Métodos definidos en varias ocasiones, con el mismo nombre pero con un número o tipo de parámetros diferentes. El compilador sabe el método al que llamar dependiendo del parámetro transmitido.

Midlet: Programa de Java designado para dispositivos móviles de información.

Miembro: Término genérico para entidades de datos o de código dentro de una clase.

Miembro de la clase: Variable o método de una clase disponible para su utilización. Para las variables de clase, sólo existe una copia de la misma y es única para todas las instancias de la clase dentro del programa.

Modal: Ventana emergente que bloquea la interacción con el resto del GUI.

Modelo de diseño: Método estándar para diseñar un problema. Se difiere de los algoritmos en que trata de temas de alto nivel y describe soluciones en términos de estructura y relación.

Modificador: Altera el aspecto de acceso a las clases, métodos y variables.

Muestra o instancia: Muestra de una clase, es decir, un objeto.

Multiproceso: En sistemas operativos, la habilidad de efectuar dos o más programas independientes, comúnmente en un procesador solo.

Mutable: Modificable.

N

Nivel de Sockets de seguridad (SSL): Protocolo utilizado para asegurar una comunicación segura, típicamente asociado a HTTP (HTTPS).

No un número (NaN): Resultado de una operación matemática no válida, tal como la de multiplicar infinito por cero.

Null o nulo: Valor que indica que no existe objeto, y que puede corresponder al valor de una variable.

O

Ofuscar: Alterar mecánicamente el archivo de clases de Java para que sea legal pero con nombre ilegibles para las variables y los métodos. Disminuye el tamaño de estos archivos.

Objeto: Elemento de software con características (estado) y comportamiento (métodos) propios.

Ocultación de datos: Habilidad de ocultar datos dentro de una clase. Cualquier modificación en estos datos debe realizarse desde fuera de la clase, y a través de sus métodos, si es que así se permite.

OOP: Véase programación orientada a objetos (OOP).

Operador: Símbolos especiales utilizados para combinar variables, literales, llamadas de métodos, y otros operadores.

Ordenación de la Red: Véase big-endian.

P

Paquete: Colección de clases relacionadas. También una unidad de comunicación que puede contener datos o ser un paquete de gestión de protocolo.

Paquete opcional: Un API definido para ser estándar de Java que no está implementado por J2SE. Recibe también el nombre de extensión estándar.

Parámetro: Elemento transmitido de un valor a un método mediante una llamada.

Paréntesis: Utilizados para delimitar las listas de parámetros de las llamadas de métodos.

Persistencia: Habilidad de un objeto de Java para grabar su propio estado para que pueda reproducirse en el futuro, en un ambiente distinto.

Pipe: Conexión abstracta entre dos procesadores o threads. A transfiere datos a B mediante un pipe. B lee los datos desde el pipe para recibir los datos.

Píxel: Elemento más pequeño del dibujo que aparece en una pantalla o en una página impresa.

Plantillas: Véase tipos parametrizados.

Plataforma de Java: Combinación de lenguaje de programación, bibliotecas estándares y ambiente de ejecución.

Plug-in o periférico de Java: Elemento añadido al buscador para reemplazar el ambiente de ejecución de Java del mismo.

PNG: Véase Gráficos portables de Red PNG

Polimorfismo: La habilidad de utilizar una clase derivada en lugar de su clase base.

Precedencia del operador: Define el orden en que se aplican los operadores.

Primitivo: Tipo de datos no representado por un objeto.

Procedimiento: Véase método.

Programación de eventos: Programar en respuesta a eventos externos, tales como pulsaciones de teclado, clics de ratón, resultado del input de un usuario.

Programación orientada a objetos (POO): Programación que se centra en los objetos independientes y en las relaciones entre ellos, en lugar de seguir un esquema lineal como los lenguajes más tradicionales Pascal o C).

Propiedades: Describen los atributos asociados a un componente, normalmente en un GUI, tal como el color, el tamaño, y la cadena que se va a utilizar como etiqueta.

Protocolo: Conjunto de reglas y estructuras de paquetes legales que proporcionan servicios de comunicación entre dos equipos. Algunos ejemplos son TCP/IP, Z-Modem, Kermit, SLIP, PPP, X.25, y SNA de IBM.

Protocolo de control de transmisión (TCP): Protocolo de conexión que extiende el protocolo IP. TCP garantiza la distribución de datos y soporta cantidades de datos arbitrarias.

Protocolo de datagrama de usuario (UDP): Protocolo intermedio entre IP y TCP que permite que los datagramas de estilo IP puedan enviarse al puerto de un equipo.

Protocolo de Inicio en Red de Java (JNLP): Protocolo empleado por Java Web Start para implantar la máquina JV y sus aplicaciones. Java Web Start no es más que una implementación del protocolo. Sitraka ofrece una implementación más avanzada mediante DeployDirector.

Protocolo de Internet (IP): Protocolo sobre el que se construyen todos los otros protocolos de Internet de nivel de la aplicación. Uno de los inconvenientes de IP es que no garantiza la distribución de datos y que su límite de transferencia es de 64KB.

Protocolo de transferencia de hipertexto (HTTP): Protocolo de aplicación utilizado por la World Wide Web para solicitar, transmitir y recibir documentos Web.

Protocolo de transferencia de hipertexto segura (HTTPS): Una extensión de HTTP que sirve para intercambiar datos en sesiones de encriptación de SSL (Nivel de Sockets de seguridad).

Q

Quicksort (Método de ordenación rápida): Ordenación de algoritmos utilizada por el marco de colecciones de Java.

R

Reader o lector: Similar a un flujo de entrada (utilizado en programación E/S), con la diferencia de que la unidad básica de datos es el carácter Unicode.

Rectángulo delimitador: El cuadro más pequeño que puede envolver una forma dibujable.

Recursivo: Que se llama a sí mismo repetidamente, directa o indirectamente.

Referencia: Tipo de datos representado por la dirección o el controlador del objeto.

Reflexión: Capacidad que tiene una clase de preguntarse sobre sí misma para descubrir sus métodos y propiedades.

Regex: Véase expresión regular.

Renderizar: Jerga informática que significa dibujar.

S

Sandbox o zona protegida: Ambiente en el que se ejecutan los programas de cliente de Java que restringe el acceso al ambiente del cliente.

SDK: Véase Kit de desarrollo de software (SDK).

Sección crítica: En el contexto de sistemas de multitarea, cualquier sección de código que requiera tomar precauciones para evitar que las estructuras de datos corruptos se compartan con otros threads. Si se implementan correctamente, sólo un thread puede estar dentro de una sesión crítica de código.

SecurityManager: Clase de Java que restringe el acceso a archivos, a la Red o a otras partes del sistema por razones de seguridad.

Sentencia case: Parte de la sentencia `switch`, que consiste en la palabra clave `case`, seguida de una expresión constante, un punto y coma, y una o más declaraciones.

Sentencia catch: Mecanismo para ejecutar el código de control de excepciones después de un bloque try.

Sentencia condicional: Sentencia para la ejecución selectiva de segmentos del programa, tales como las sentencias `if` y `switch`.

Sentencia continue: Una de las declaraciones de detención de flujo. Sin presencia de una etiqueta, el control se transmite al punto inmediatamente posterior a la última sentencia del bucle. Con una etiqueta, por el contrario, el control se transmite al final del bucle que tiene la misma etiqueta.

Sentencia de bucle: Sentencia para la ejecución continuada de segmentos del programa que incluyen las sentencias `for`, `while`, y `do`.

Sentencia do-while: Una de las sentencias de bucle. El bucle se ejecuta una vez para, a continuación, evaluarse la expresión condicional. Si ésta resulta verdadera, el bucle se ejecuta de nuevo y se evalúa de nuevo la condicional, que se repite hasta que arroja un resultado de falso.

Sentencia else: Sentencia opcional para una declaración `if`. Si la condicional resulta falsa, se ejecuta la sentencia o bloques de sentencia `else`.

Sentencia de control de flujo: Incluye las sentencias condicionales, de flujo y de detención de flujo.

Sentencia de detención de flujo: Sentencia para detener el flujo de la ejecución. Incluye las sentencias `break`, `continue`, y `return`.

Sentencia por defecto: Pare opcional de la sentencia `switch` que consiste de la clave por defecto seguida de un punto y coma de una o más sentencias.

Separador: Línea no seleccionable que aparece en los menús por razones decorativas.

Serialización: Proceso en el que un objeto se almacena a sí mismo, escribiendo los valores que describen su estado.

Servidor Web: Servidor de Red, como Apache y los servicios de información de Internet (IIS) de Microsoft, que transmite documentos Web a través del protocolo HTTP.

Servlet: Programa de Java del lado del servidor incluido dentro del servidor Web.

Session: Objeto utilizado para hacer un seguimiento de la interacción de la Web del usuario en solicitudes múltiples de HTTP.

Synchronized: Palabra clave de Java utilizada para restringir el acceso multitarea a bloques de código crítico, asegurando la ejecución única dentro de dicho bloque.

Sink (Unidad receptora de datos): Destino final de los datos que se desplazan por el flujo.

Sistema de nombre de dominio (DNS): Base de datos de Internet que convierte direcciones textuales de Internet en sus formas numéricas reales. El sistema DNS está organizado jerárquicamente con cada nódulo responsable de un subconjunto de nombres de direcciones host de Internet.

Sitio Web: Conjunto de documentos Web que pertenecen a una organización concreta. Un sitio Web puede compartir el mismo servidor con muchos otros sitios o puede transmitirse a través de varias máquinas.

Socket: Interfaz de software que conecta una aplicación con la Red.

Stack: Área dentro de la memoria donde se almacenan los objetos de datos locales. Se trata también de un mecanismo de almacenamiento de lista que emplea la metáfora LIFO, de la última dentro y la primera fuera.

Stream o flujo: Concepto abstracto utilizado en el contexto de programación E/S que representa un flujo lineal y secuencial de entrada y salida de datos. Un programa puede leer flujos de entrada, y escribir datos de salida, es decir, transferir datos a dicho flujo.

Streaming: Término utilizado para indicar contenido de audio, video, o de otro tipo de contenido de Internet que se distribuye en tiempo real y que no necesita descargarse.

Stub: Método sin cuerpo que satisface las necesidades del compilador y deja espacio para definir más adelante los métodos específicos.

Subclase: Una clase que hereda o extiende (en terminología de Java) elementos de otra clase, y que son más especializadas que las clases de las que heredan.

Superclase: Clase de la que hereda otra clase y que son cada vez más genéricas a medida que nos desplazamos hacia arriba en la estructura jerárquica, por lo que pueden llegar a ser abstractas.

Swing: Componente GUI de segunda generación parte de las clases de fundación de Java.

T

TCP: Véase Protocolo de control de transmisión (TCP).

This: Clave utilizada para indicar una referencia a la muestra de al clase donde se emplea.

Thread (subproceso): Unidad básica de ejecución de los programas, similar a un proceso, o un programa en ejecución, pero más sencillo de crear y de destruir ya que requiere un menor control. Cada thread debe disponer de sus propios recursos, tal como el contador del programa y el stack de ejecución. Los threads de un programa pueden compartir los mismos recursos tal como el espacio de la memoria y los archivos abiertos.

Thread daemon: Un thread que se ejecuta de manera oculta para realizar un servicio. A diferencia de los threads de usuario, este tipo de threads no evita que el sistema se cierre.

Tipo de datos: Tipos de datos almacenados como variables. Por ejemplo, las variables `int` contienen números enteros, las variables `char` contienen caracteres alfanuméricos independientes y las variables `String`, grupos de caracteres de este tipo.

Tipos parametrizados: Permiten especificar el uso de un tipo de datos concreto, también llamados templates o plantillas.

Transformación afín: Manipulación basada en mátrix para crear un mapa entre dos coordinadas bidimensionales.

Try: Comienza la definición de un bloque de control de excepción.

U

UDP: Véase Protocolo de datagrama de usuario (UDP)

UML: Véase Lenguaje de modelo Universal (UML).

Unicode: Conjunto de caracteres de 16 bits, en lugar de los 8 que soportaba ASCII. Así se pueden representar la mayor parte de los lenguajes del mundo (http://www.unicode.org/).

URL: Véase Localizador de recursos uniformes (URL).

V

Variable: Contenedor con un nombre asignado donde se almacena la información durante la ejecución del programa.

Variable de muestra: Variable dentro de una clase, cuya copia está disponible para su almacenamiento en cada muestra o instancia de esa clase. Cada objeto de esa clase dispone de su propia copia de la variable de dicha muestra (en contraposición con la variable de la clase)

Variable del miembro: Campo no estático que pertenece a la clase.

Variable local: Variable declarada dentro de un bloque de código, en contraposición de la clase o el método.

Verificador: Parte de la arquitectura de seguridad de Java que asegura la validez de los bytecodes que se cargan.

Verificador de bytecode: Una de las precauciones de seguridad de Java, según la cual el verificador de bytecode comprueba los bytecodes que pueden ejecutarse de manera segura en la máquina VM.

W

Widgets: Término genérico para referirse a los elementos de GUI, tales como barras de desplazamiento, casillas de verificación, cuadros de diálogo, etc.

WIMP: Acrónimo de los términos ingleses equivalentes a ventanas, iconos, menús y punteros. Véase la interfaz gráfica de usuario.

WORA: Del inglés "Write Once, Run Anywhere" que corresponde al objetivo de Java que consiste en la capacidad de ejecutar cualquier programa de Java en cualquier plataforma, mediante una máquina VM.

World Wide Web (WWW): Enorme colección de documentos de hipertexto interconectados en Internet.

Writer o escritor: Similar a un flujo de salida (utilizado en programación E/S), con la diferencia de que la unidad básica de datos es el carácter Unicode.

X

XML: Véase lenguaje XML.

Z

ZIP: Archivo que contiene un grupo de archivos, normalmente comprimidos.

Índice alfabético

@Etiquetas estándar de javadoc, 88-89
2Lkit, 705
3D, modelos, 751

A

AbstracListModel, 551, 552
AbstractAction, 512, 513
AbstractCellEditor, 577
AbstractSelectableChannel, 864
  interfaz LayoutManager, 378-381
  JSPlitPane, 359-360
  JtabbedPane 357-359
  métodos, 396
  SpringLayout, 368-372
Administrador
  de información sobre herramientas, 422
  de seguridad, 902, 903
ADT, 238
Agregar etiquetas, 424-425

depurar, 70
documentar, 70, 88, 87-91
ejecutar, 69
Appletviewer, 69
Árbol, 578-581
Archivo JAR, 826
Arquitectura
MVC, 509, 546, 547, 590
neutral, 41
Array, 624, 625, 640, 652, 717
de bytes, 754
ArrayList, 637, 643-645
Arrays
acceder a los elementos de, 146
buscar, 152-153
clonar, 150-152
copiar, 150-152
crear, 147-148
de kernel, 329
declarar, 147
definición, 145-146
inicializar, 147-148
multidimensionales, 153-154
ordenar, 152-153
parciales, 151
uso de la memoria, 148-150
valores de relleno por defecto, 148
Aseveraciones
Activar
el control, 183
el control en tiempo de ejecución, 184
assert, 182-183
comprobar, 181-182
estatus, 184
Atributos de impresión, 792-794
Attr, 715
Audio
formatos, 287
reproducir, 286-288
Aumentar espacio de pila, 61
Authenticator, 860
AWT, 677
AwtEventMulticaster, 510
Ayuda a discapacitados, 451-452

B

Barras
de desplazamiento, 436-439
de herramientas, 447-449
de menús, 454
BDK, 812, 813, 826
BeanBox, 814-817
BeanBox, 820, 826, 827
BeanInfo, 825, 826
Biblioteca XML, 881
Bibliotecas
de utilidades, 236-239
fundamentos, 235-236
Image I/O, 330
Math, 237
multimedia, 240-242
para programas distribuidos, 242-243
tamaños, 235
BigDec, 660, 661
BigDecimal, 615, 652, 655, 657, 658
BigInteger, 615, 652, 655, 657, 658
BinarySearch, 654
BitSet, 635-637
Bloqueo comprobado dos veces, 230-231
Bloques
de código, 98
ejecutar repetidamente, 159
sincronizado, 213
try-catch, 170, 171, 173
Borland, 829
Botones
agregar, 406-409
botones de activación, 425
casillas de verificación, 428-429
cambiar los iconos, 415-417
de opción, 429-430
definir bordes y contornos, 412-415
mnemotécnicos, 419-421
mostrar imágenes, 409-412
ubicación
de iconos, 417-419
de texto, 417-419
BOXCOLOR, 705
BOXFGCOLOR, 705
BOXMESSAGE, 705
Buffer
doble búfer, 280
operaciones de dibujo, 280-282
sobrescribir, 26
Buffer, 769
BufferDiff, 751
BufferedImage, 690
BufferedInputStream, 737, 750
BufferedOutputStream, 750, 751, 758
BufferedReader, 726, 738, 750
BufferedWriter, 750
ByteArrayInputStream, 725, 726, 729, 730
ByteArrayOuputStream, 748, 749
ByteBuffer, 770, 771

C

Caché, 689
CalcStack, 672, 682-684, 706, 707

CalculatorKeys, 673
Calendar, 615, 662-664
Cambio de foco temporal, 481
Campos
  de texto, 439-440
  definición, 75-76
  formato de campo de texto, 446
Caracter vectorial, 688
Carga sincronizada, 412
Cargador HTML, 68
Casillas de verificación, 428-429
Certificación Java, 244
ChangeEvent, 571, 572
ChangeReporter, 816
char, 769
Character, 652
CharArray, 749
CharArrayReader, 730, 748, 749
CharArrayWriter, 749
CharBuffer, 770, 771
Charset, 771
CharsetEncoder, 771
CharsetProvider, 772
Clases
  abstracción de, 77, 80-81
  abstractas, 80-81
  AbstractButton, 415-419
  ActionEvent, 469-470
  AdjustmentEvent, 472
  agregación, 82-83
  AlphaComposite, 316-317
  AncestorEvent, 474
  anónimas, 136-137
  asociación, 81-82
  BasicStroke, 297-300
  ButtonGroup, 429
  cardinalidad, 81-82
  color, 262-263
  compartir nombre, 139
  component, 343, 387-392
  ComponentEvent, 477
  concretas, 80-81
  container, 343, 345-349
  ContainerEvent, 479
  convenciones recomendadas, 142
  de excepción personalizadas, 177
  de miembro, 135
  diseñar, 77-83
  encapsulación, 77
  envolventes, 236-237, 237
  EventObject, 467
  ExitableJFrame, 250
  externas, 134
  FocusEvent, 480-481
  Frame, 249
  generalización, 78-80
  GeneralPath, 312
  graphics, 248-249
  graphics2D, 292-296
  GraphicsEnvironment, 332-333
  HyperlinkEvent, 483
  InputEvent, 491-492
  insets, 251-252
  InternalFrameEvent, 484-485
  internas, 134,135, 137
  ItemEvent, 487
  Jbutton, 407
  JComboBox, 431-432
  JComponent, 401-406
  JList, 434-436
  JPanel, 422-423
  JScrollPane, 433
  JSlider, 436-439
  JTextField, 439-440
  JToolBar, 447-449
  KeyEvent, 490-491
  LinkedList, 227
  locales, 136-137
  MediaTracker, 274-277
  MouseEvent, 496
  MouseWheelEvent, 499-500
  para curvas, 307-313
  RectangularShape, 308
  RenderingHints, 304-306
  StackTraceElement, 179
  superiores, 134
  SyncToken, 213
  TaskThread, 227
  TextLayout, 322-323
  WindowEvent, 502
class, 745
CLASSPATH, 55-57, 140
Cláusulas
  catch, 170, 173
  extends, 131
  finally, 179-181
  if, 156
  implements, 131-133, 152
  throws, 174
  try, 170
ClickClunk, 735
ClipboardOwner, 598, 600
ClipImage, 604, 610
ClipText, 600, 601
Cloneable, 643, 644
Code Warrior, 829
CodeShield, 705
Codificación de XML, 767
Código
  ASCII, 911

empaquetado, 141-142
hash, 108
CollationKey, 652
Collection, 637-640, 647, 654
Generic Collection, 641
ComboBoxModel, 551
Comentarios
anidados, 88
documentación HTML, 91
javadoc, 88-89
tipos de, 88-91
Comité ANSI, 889
COMM, Comunicaciones de Java, 924, 925
Comparable, 651, 652
Comparator, 652
Compilador JIT, 42
Compilar código, 58-59, 67
empaquetado, 141-142
Componentes
adjuntar fuentes a, 368
añadir a la lista interna del contenedor, 380
aspecto en plataformas, 341
AWT, 240, 342
BOX, 366
crear nuevos, 452
de texto, 399
definición, 38
del sistema de menú, 454
dimensiones, 373
emplear en más de una celda, 374
en un entorno distribuido, 243
especificar tamaño, 374
eventos de, 476-477
gráficos, 395
GUI, 385-386
JavaBean, 143, 239
JCheckBox, 428
JComboBox, 431
JFrame, 344
JLabel, 424
JScrollPane, 433
JSpinner, 444
JToggleButton, 425
JTree, 240
opacos, 398
para aplicaciones gráficas, 240
para orientación de idiomas, 353
resumen de componentes Swing, 458-460
reutilizables, 239
Swing, 240, 249, 342
Componentes API, 890
de Swing, 509
Comunicación, 802
Comunicaciones de Java (COMM), 924, 925
Condensity, 705
Condiciones de error irrecuperables, 171
Conector pipe, 731
Conexión UML, 133
Constructores
de control de excepciones, 169-170
de ImageIcon, 411
de JCheckBox, 428
de JCheckBoxMenuItem, 456
de JComboBox, 431
de JEditorPane, 443
de JFormattedTextField, 446
de JLabel, 424
de JList, 434
de JMenu, 455
de JMenuItem, 455
de JPanel, 422
de JPasswordField, 440
de JRadioButton, 429
de JRadioButtonMenuitem, 456
de JScrollPane, 433
de JSlider, 437
de JSpinner, 445
de JTextArea, 441-442
de JTextField, 439
de JToggleButton, 426
de JToolBar, 447
para aseveraciones, 182
para bucles, 159
por defecto, 64
Contenedor, 520
Control
de eventos
basado en delegación, 463-467
de acción, 469-471
de ajuste, 471-476
de componente, 476-478
de contenedor, 478-479
de elemento, 486-489
de hipervínculo, 482-484
de marcos internos, 484-486
de menú, 493-495
de ratón, 495-498
de rueda de ratón, 498-501
de tecla, 489-493
de ventana, 502-504
eventos
focales, 480-482
padre, 474
de flujo, 154-155
de focos, 510
de fuente actual, 265
Controlador MVC, 509, 546, 547, 590
Conversión de tipos datos
definición, 109
en C, 110

entre referencias, 123-124
entre tipos primitivos, 122-123
CORBA, 894
Coseno, 670
cosine, 678
CreateTempFile, 717
CreditManager, 898-900
Cuadro
de diálogo
confirmación, 528, 530, 531
entrada, 528, 531, 532
mensaje, 528-530
opción, 528, 531, 532
de lista
desplegable, 550
Curvas Bezier, 309

D

Daemon, 845
DashO, 705
DataFlavor, 594-596
DatagramPacket, 839, 840
DatagramSocket, 840, 841
DataInput, 719, 720
DataInputStream, 739, 740
DataOutput, 719, 739, 751
DataOutputStream, 751
Date, 615, 662, 663
DateFormat
DateFormat.FULL, 665
DateFormat.LONG, 665
DateFormat.MEDIUM, 665, 666
DateFormat.SHORT, 665
DateTimeSyntax, 791
DecimalFormat, 660
DecimalPos, 681
Declaraciones
bloque, 159, 211
condicionales, 103
de aseveraciones, 182
de asignación, 148
de interfaz, 131
de matriz, 146, 147
en bloque, 160
import, 138-139, 141
múltiples, 156
new, 128,193
package, 141
que pueden arrojar una excepción, 170
sincronizar un bloque, 212, 215
Declaraciones para el control de flujo
break, 164
condicionales, 155-159
continue, 164
de etiqueta, 165
do-while, 162-163
for, 159-160
if, 155-157
operador terciario, 156-157
para bucles, 159-163
para romper el flujo, 163-164
return, 165-166
switch, 157-159
while, 161-162
Decodificación de XML, 767
Decompilador, 706
default, 116
DefaultEditor, 601-603
DefaultEditorKit, 601
DefaultListModel, 552, 553
DefaultMutableTreeNode, 582-584
DefaultTableColumnModel, 569-571
DefaultTableModel, 567
DefaultTreeModel, 579, 580
Definición, 50-51
Delegate de botones, 548
DeleteOnExit, 717
Dennis Ritchie, 723
DeployDirector, 694
Depurar, 64
Descargar, 53
Desensamblar, 64
Deserialización, 752, 759
Desplazamiento programable, 522
DevX, 44
Diálogos de no impresión, 779
Dibujar
a pantalla completa, 334-337
arcos, 256
curvas, 307-313
imágenes, 271
líneas, 252
marcos básicos, 249-252
óvalos, 255
polígonos, 253
rectángulos, 252
texto, 257-258, 319-324
transformaciones, 313-316
usando el buffer, 280-281
Dictionary, 625, 626, 639, 646
Diferencias entre applets y aplicaciones, 49-50
Diff, 722
Diodo emisor de luz, LED, 670
DocAttributeSet, 791
DocPrintJob, 783, 784
Documentar, 61-63
DOM, 884-886
double, 769
DoubleBuffer, 770

Draw 4, 695, 698, 699
DrawNumber, 691
Duke, 36

E

Edición
  Enterprise (J2EE), 881
  Estándar (J2SE), 881
EJB, Enterprise JavaBeans, 801, 923, 924
Ejecutar
  código, 59-61, 69
    empaquetado, 141
  tareas periódicamente, 197-200
  thread, 188
Elixir Report, 799
Endian
  Big Indian, 752
  Little indian, 752
Enterprise JavaBeans (EJB), 801, 923, 924
Entorno de ejecución
  de Java, JRE, 668
  Java, 48-49
Entrada y Salida, E/S, 711
Enumeration, 616, 622, 640, 731
EnumSyntax:, 791
EOF, 730
EOFException, 739
Error estándar, 753
Errores del programador, 171
Especificar un área de recorte, 260
Etiquetas javadoc, 88-89
EventListener, 511, 806
EventListenerList, 511
EventMonitor, 816
Eventos
  adicionales, 506
  tipos de eventos, 467-503
Eventos de teclado, 510
EventTargetDialog, 818
Excepción
  InvalidClassException, 761
  InvalidObjectException, 761
  IOException, 761
  NotSerializableException, 761
  ObjectStreamException, 761
  OptionalDataException, 761
  StreamCorruptedException, 761
  StreamCorruptedException, 761
  WriteAbortedException, 761
Excepciones
  arrojarlas uno mismo, 176
  clases personalizadas, 177
  control de, 169-170
  en tiempo de ejecución, 171
  encadenar, 177-178
  jerarquía, 170-172
  que no son de tiempo de ejecución, 171-172
Expandibles con un toque, 360
Expresiones, 99-100
extends, 745
Extensión
  Criptográfica de Java (JCE), 886-888
  de Seguridad de Socket (JSSE), 887, 889
Externalizable, 756, 757

F

factories, 796
Factory, 528, 529, 877, 878
File, 712-715
FileChannel, 723, 770, 771, 862
FileFilter, 545
FileInputStream, 726, 727
FileOutputStream, 758
FileReader, 728, 729
fillPolygon, 691
Filter, 776
FilterInputStream, 726, 737
FilterOutputStream, 750
FilterReader, 726
FilterWriter, 750
findThread, 736
fireActionPerformed, 511
Firewalls, 891
float, 769
FloatBuffer, 770
Flujo
  de bucle do-while, 162
  de bucle for, 160
  de bucle while, 161
  lineal, 711
flush, 735
Fmt, 777
Foco
  Subsistema, 516
FocusEvent, 516
FocusTRansversalPolicy, 520, 521
FOP, 799
Formato
  ASCII, 751
  JAR (Archivo de Java), 810
  UTF (Unicode Transfer Format), 760
freshNumber, 681

G

Gestor de aplicaciones, 696, 697
getDateTimeInstance, 666
getFilePointer, 720

getParent, 736
getPercentInstance, 660
getPrintJob, 781
GIF, 688
goto, 163
Gradientes, 300-303
Gráficos
básicos, 247-258
configuración, 331-333
control del estado, 258-269
modo de dibujo, 264-265
modos de muestra, 333-334
Graphics, 781, 782
Graphics2D, 781
GregorianCalendar, 662, 663
GridLayout, 678
GSS-API, Servicio de Seguridad Genérica, 887
GUI
Agregar
botones, 406
campos de texto, 439-440
casillas de verificación, 428
etiquetas, 424
listas, 434-436
listas desplegables, 431-432
agrupar componentes, 347, 406
añadir
botones de opción, 429-430
paneles deslizantes, 433
cambio de tamaño de los elementos, 394
componentes
AWT, 386
Swing, 385
crear nuevos componentes, 452-453
diseño con administradores, 343
flujo de los elementos, 349
posición de Java, 341-342
proporcionar barras de desplazamiento, 436-439
superclase Swing, 401
ubicar áreas de texto, 441-442
GUI, Interfaz, 668, 703
GZIP, 742, 745
GZIPInputStream, 742
GZIPOutputStream, 754

H

Handler
ConsoleHandler, 777
FileHandler, 777
MemoryHandler, 777
SocketHandler, 777
Hash, 108
HashCode, 627
HashDocAttributeSet, 791
HashMap, 637, 638, 641, 647, 648
HashPrintJobAttributesSet, 791
HashPrintRequestAttributeSet, 784, 791
HashPrintServiceAttributeSet, 791
HashSet, 641
Hashtable, 616, 626-628
haveOilProcessed, 735
HelloServlet, 912-916
Herencia, 79
Herramientas, 51-53
BeanBuilder, 370
configuración del administrador sobre, 422
JBuilder, 381-382
SDK, 51-53
trabajar con información sobre, 421
Herramientas IDE, 806
Historia de java 101, 36
HTMLConverter, 692
HTTP, Protocolo de transferencia de hipertexto, 832, 849-851, 856-858, 908
HttpServlet, 912-914

I

IDE, 705
IDEA, 829
Identificadores, 93
IDL, Lenguaje descriptivo de la interfaz, 894
Imágenes
cargar, 270
codificación, 329-331
combinar, 316-318
deformación, 314
dibujar, 271
escala, 283
escalado, 314
formato flexible, 325-326
manipulador de, 324
métodos relacionados, 398-399
modificar la escala, 282-284
mostrar en botones, 409-412
operaciones avanzadas, 326-329
recortar, 285-286
rotación, 314
trabajar con, 270
translación, 313
ImageSelection, 604, 607, 608, 610
import, 745
InetAddress, 835-838
Infinito, 98
InflaterInputStream, 742, 754
Informe Volano, 41
InputStream, 724, 725
InputStreamReader, 737

instalar, 53
Instrucciones, 99-100
int, 769
IntBuffer, 770
IntegerSyntax:, 791
Integración con C/C++, 63
IntelliJ, 829
Interfaces
  anónimas, 126-137
  API 2D, 291-292, 331
  cloneable, 151, 377
  Comparable, 153
  con adaptadores, 505
  convenciones recomendadas, 142
  definir, 131
  Entry, 134
  LayoutManager, 378-381
  representación en UML, 133
  runnable, 131, 189
  Shape, 292
  y clases superiores, 124-135
  y métodos, 134-135
Interfaz
  Action, 512
  BeanInfo, 811
  de directorios e identificación de Java (JNDI), 921, 922
  del Usuario UI, 546
  E/S, 711
  GUI, 668
  java.awt.print.Printable, 797-799
  nativa de Java, JNI, 907
Introspección, 802
InvalidClassException, 761
InvalidObjectException, 761
Invocación de método remoto, RMI, 894, 903
Invocar métodos, 99-100
IOException, 761
IP, Protocolo de Internet, 831-833
ISO-8859-1, 772
Iterator, 637, 640, 645, 729

J

J2EE, Edición Enterprise, 881, 907
J2SE, Edición estándar, 881, 882, 907
JAAS, Servicio de Acceso y Autentificación, 887
JApplet, 672, 692
JAR (Java Archive), 700, 702, 704
JarInputStream, 742
JarOutputStream, 754
JasperReports, 799
Java
  1.1., 691
  1.4 SDK, 777
  1.5, 777
  101, 36-37
  Archive, JAR, 700
  Network Launching Protocol, JNPL, 694
  VM, máquina virtual de Java, 670
  Web Start, 668, 692, 694
JavaBean, 767, 802
javadoc, 61-63
Javadoc, 773
JavaMail, 907, 918
javax.swing.Japplet, 691
JBuilder, 705, 708, 801, 821, 829
JButton, 513, 547, 548
JCE, Extensión Criptográfica de Java, 886-888
JColorChooser, 524, 534-536, 577
JComboBox, 550, 557
JComponent, 513, 681
JDBC
  API, 817
  Select, 817
JDeveloper, 829
JDialog, 529
JDK 1.4, 775
JellyBean, 816
Jerarquía de clases
  de eventos AWT, 468
  de eventos Swing, 468
  de excepción de clases, 170
  de las superclases Component/JComponent, 386
JFC, 342
JFileChooser, 524, 536, 541-545
JFrame, 762
Jindent, 143
JInternalFrame, 528
JLabel, 555, 604
JList, 547, 550, 551, 553, 557
JMenuItem, 513
JNDI,Interfaz de directorios e identificación de Java, 921, 922
JNI, Interfaz nativa de Java, 907
JNLPTest, 701, 702
JNPL (Java Network Launching Protocol), 694
JODE, 705
JOptionPane, 524-528, 534
JPanel, 512, 677
JRE 1.1., 691
JRE, 48
JRE, Entorno de ejecución de Java, 668, 692
JReport, 799
JShrink, 705
JSP, Páginas de JavaServer, 801, 807, 916-918
JSR 51, 777
JSSE, Extensión de Seguridad de Socket, 887
JTable, 558-565, 574-576

JTextArea, 597
JTextField, 523
JTextPane, 597
JTree, 578-581, 587
Juggler, 816
JVM, máquina virtual de Java, 735

K

Kernel, 327
KeyboardFocusManager, 222, 223, 516-520
KeyCode, 515
KeyLabel, 678, 708
KeyListener, 514
KeyPanel, 671, 673-678, 708
KeyStroke, 514, 515
Kit de desarrollo de Software de Java (SDK), 881

L

Last-in-first-out (LIFO), 621
LCD, Tecnología de cristal líquido, 670
LCDDigitDisplay, 671
LCDDisplay, 673, 679-681
LED, Diodo emisor de luz, 670
Leer volcado de stack, 178-179
Legibilidad, 649
Lenguaje descriptivo de la interfaz, IDL, 894
Level
  CONFIG,776
  FINE, 776
  FINER, 776
  FINEST, 776
  INFO, 776
  SEVERE, 776
  WARNING, 776
LIFO (last-in-first-out), 621
Lineno, 741
LineNumberReader, 740, 741
LinkedHashMap, 647
LinkedList, 643
Linux
  instalación, 54-55
  Redhat, 54
List, 637, 642, 643, 654
Listas, 434-436
Listas desplegables, 431-432
ListCellRenderer, 554
ListDataEvent, 554
Listener, 515, 634
ListIterator, 645, 646
ListModel, 551
ListSelectionEvent, 556
ListSelectionModel, 555
Literales
  de punto flotante especiales, 98
  formatos de tipos de datos, 95-96
  iniciar variables con, 96-98
loader, 704
Locale, 661
Logger, 775, 776
long getFilePointer, 720
long length, 720
longBuffer, 770
Look and Feel, 549, 550

M

Manejador de eventos, 509
Manipulador de imágenes, 324
Mantisa, 670
Map, 637, 646, 647
Mapa de acción, 515
MappedByteBuffer, 770, 771
Máquina virtual de Java (JVM), 670, 735
Marco de colecciones API, 616, 803
Marcos
  anexar WindowsListener, 504
  crear básicos, 249-250
  eventos de marcos internos, 484
  internos, 449-450
markSupported, 725
Matcher, 772
Math, 615, 654-656
Memoria
  de objeto, 127
  en arrays, 148-150
  heap, 127
  stack, 127
Menú
  agregar, 454-455
    barras de menús, 454
    elementos, 455-456
    menús emergentes, 456
  componentes, 454-458
Métodos
  administración de focos, 396-397
  anular, 132
  compareTo(), 103
  consulta de geometría, 395
  consulta de ubicación, 395
  convenciones recomendadas, 142-143
  de control Thread, 189-192
  de dibujo automático, 397
  de los administradores de diseño, 396
  de menú desplegable, 399
  de representación, 395
  definición, 75-76
  equals(), 102

getClassLoader(), 184
gráficos, 395
join(), 191
modificar tamaño de componentes, 394-395
mover componentes, 394-395
para el cambio de estado, 398
para la administración de focos, 396
para la comunicación de igual a igual, 399
para solicitar el estado, 398
printAll(), 200
relacionados
con imágenes, 398-399
con los métodos de entrada, 399
schedule(), 198-200
scheduleAtFixedRate(), 198-200
throwable, 173-174
Metrowerks, 829
Microsoft, 43
MilitaryGame, 695, 699
MIME (Extensiones Multipropósito para correo en Internet), 700, 785, 796, 852, 919
Modelo de memoria de objeto, 127
Modelos 3D, 751
Modificadores
abstract, 111
almacenamiento, 110-115
aplicación, 111
de accesibilidad, 116-118
duración, 110-115
final, 115
native, 114
private, 116
protected, 116
public, 116
static, 111-113
strictfp, 115
synchronized, 114
transient, 115
volatile, 114
Molecule, 816
MORE, 731
Mostrar HTML, 443-444
MP3, 287
MQSeries, 922
MTU, 703
Multitarea
avanzada, 207-215
bases, 187-188
Multitarea, 510

N

NaN, 98
NextToken, 729
Nivel de referencia remota (RRL), 895
NNTP, Protocolo de transferencia de noticias en Internet, 832, 871
NodeChangeEvent, 635
NodeChangeListener, 634, 635
Nombres de clases
nombres de archivo de clases internas, 135
resolución, 139
nonblocking, 865, 867-869, 875, 876
NotActiveException, 761
Notepad, 695, 698
notify(), 215-222
notifyAll, 735
notifyall(), 215-222
NotSerializableException, 761
Nuevo E/S, 768
Nuevos componentes, 452-454
Null, 96
NumberFormat, 615, 655, 660

O

Oak, 36
obfuscation, 705
obfuscator, 705, 708
Object, 618, 629
ObjectInputStream, 725, 752, 766
ObjectOutput, 719
ObjectOutputStream, 758, 766
ObjectStreamException, 761
ObjectStreamField, 652, 766
Objeto UI
Interfaz del usuario, 546
Observer, 463-467
Ocultar campo de contraseñas, 440-441
ODBC, 891
Ofuscar, 705
OilRefinery, 735, 736
OOP, 74-75
Opciones de línea de comandos
-?, 60
-c, 64
-classpath, 140
con aseveraciones, 182
-cp. 140
-d, 141
-debug, 70
-ea, 182
-enableassertions, 182
-g, 64
-hotspot, 60
introducción, 60-61
no estándar, 61
-server, 60
-source, 182, 183
- target 1.1, 67

-verbose, 59, 61
-versión, 61
OPEN_DELETE, 743
OPEN_READ, 743
Operaciones
Bitwise, 105-106
De desplazamiento de bits, 106-107
E/S, 769, 771
Operadores
aritméticos101-102
bitwise, 105-107
booleanos, 104-105
conversión de datos, 109
de asignación, 108-109
de cadena, 107-108
de desplazamiento, 106-107
de desplazamiento de C, 106
de diferencia, 102
de manipulación de bits, 107
definición 100
especiales, 109-110
instance of, 110
new, 110
relacionales, 102-103
terciario, 156-157
OptimizeIt Thread Debugger, 211
OptionalDataException, 761
Oracle9i, 829
OutputStream, 746-748

P

Páginas de JavaServer (JSP), 801, 916-918
Palabras clave
assert, 182-183
break, 158
class, 131
default, 116
definición, 92-93, 116
extends, 80
final, 115
implements, 131
label, 165
new, 147-148
public, 58
static, 113, 134, 135
super, 126
synchronized, 211
this, 126
throw, 176
volatile, 114
Paneles deslizantes, 433
Paquete
"New I/O", 768
com.sybex, 633
java.awt.datatransfer, 593
java.awt.dnd, 593, 612
java.awt.print, 779
java.beans, 767, 808
java.io, 746, 769
java.io.java.nio, 711
java.io.tmpdir, 717
java.lang, 615, 616, 618, 654, 655
java.math, 657
java.net, 831, 835
java.nio.channels, 770
java.nio.charset, 771
java.text, 655, 660
java.util, 615, 637, 661
java.util.logging.Logger, 775
java.util.regex, 729, 772
java.util.zip, 742
javax.print, 782
javax.print.attribute.standard, 782
javax.print.attribute, 782, 791
javax.print.event, 782, 784
javax.swing.event, 511
javax.swing.tree, 579
Log4j, 777
Paquetes
anidados, 140
convenciones recomendadas, 142
de seguridad, 239
definición, 138, 140
Parámetros de solicitud, 915
PasswordAuthentication, 861
PATH, 54
Patrón Observer, 463-467
Pattern, 772
PatternSyntaxException, 772
Persistencia, 802
PersisTest, 762-764
Personalización, 802
Picture, 736, 737
Pila de almacenamiento, Stack, 670
pipe, conector, 731
PipedInputStream, 725, 731
PipedOutputStream, 731
PipedReader, 735
PipedWriter, 735
PLAF, Look and Feel modificable, 549
Plataforma Unix, 714
points, 689
POSTSCRIPT, 796
PreferenceChangeEvent, 635
PreferenceChangeListener, 635
Preferences, 632
Preferencias API, 632
PrinterJob, 779
PrintJob, 781-783

PrintJobAdapter, 784
PrintJobAttributesSet, 791
PrintJobListener, 784, 797
PrintRequestAttributeSet, 784, 791
PrintService, 783
PrintServiceAttributeSet, 791
PrintStream, 751-753
PrintWriter, 753, 754
Programación E/S, 710
PROGRESSCOLOR, 705
Properties, 630, 631, 751
PropertyChangeListener, 807, 808
PropertyNameDialog, 819
Propiedades, 802
  del cliente, 589
  dependientes, 807
  indexadas, 807
Protocolo
  de control de transmisiones, TCP, 831, 842-846
  de datagramas, UDT, 835
  de Internet, IP, 831
  de transferencia
    de archivos triviales, TFTP, 835
    de hipertexto, HTTP, 832, 849, 856, 910
    de noticias en Internet, NNTP, 832, 871
    simple de correo, SMPT, 832, 845, 871
  IMAP de acceso de mensajes, 919
  POP de PostOffice, 919
  TCP, 703
public, 745
Puerto
  de serie, 711
  paralelo, 711
  USB, 927
Puertos TCP, 834
PushbackInputStream, 742
PushbackReader, 742
putValue, 512

R

RandomAccess, 643
RandomAccessFile, 717-719, 721, 722, 727, 739, 751, 769
RandomAccessFile, 751
Rational Rose, 83
Reader, 724, 726
ReadExternal, 757, 759
ReadFully, 722
ReadLimit, 725
ReadLine, 720
ReadUnsignedByte, 720
ReadUnsignedShort, 720
Reflection API, 810
Registro API, 777
RegOne, 773
Relaciones
  abstracta, 80
  codificar, 82, 83
  de abstracción, 80
  de agregación, 82
  de asociación, 81
  de herencia, 80
  de implementación, 133
  dirección, 81
  generalización, 78
  generalización/herencia, 79
Rellenos de color, 300-303
remove, 511
renderDigit, 691
renderSegment, 691
Representación binaria, 751
requestFocus, 522
requestFocusWindow, 522
ResolutionSyntax, 791
RMI, Invocación de método remoto, 894, 903
Rotar campos de entrada, 444
RPN, 669
RRL, Nivel de referencia remota, 895
Ruta de Certificado de Java, 887

S

Salida estándar, 753
SavaJe, 41
SAX, 884-886
SCSI, 711
SDK
SDK, Kit de desarrollo de Software de Java, 881, 915
Secuencias de escape de carácter, 96
seek, 720
Selección
  de archivos, 536, 537
  de colores, 533, 534
SelectionKey, 866
Selector, 865, 866
Seno, 670
Separator, 714
SequenceInputStream, 725, 730
Serializable, 756
Serialización XML, 752, 812
Serialization API, 810
serialPersistentFields, 766
ServerSocket, 869-870
ServiceUI, 788, 789
Servicio
  de Acceso y Autentificación (JAAS), 887
  de Seguridad Genérica (GSS-API), 887

Servlet, 910, 911
ServletRequest, 915
Set, 637, 641
SetOfIntegerSyntax, 791
SevenSegmentDigit, 672, 682, 684-687, 689
short, 769
ShortBuffer, 770
showDialog, 536
showMessageDialog, 529
SimpleDateFormat, 666, 667
SimpleDoc, 785
Sincronización, 649
sink, 723
Sistema de Nombres de Dominio, DNS, 837
Sitraka, 694
Size2Dsyntax, 791
Skeleton del servidor, 895, 896, 901, 902
SkipBytes, 720
SmallTalk, 509
SMPT, Protocolo de transferencia simple de correo, 832, 845, 846-848, 871
Socket, 862-864, 862-864
SocketChannel, 862
Solaris, 48, 50, 55
  Unix, 813
SonicMQ, 922
Soporte para fuentes, 319-320
SORT, 731
SortedMap, 639
SortedSet, 639, 641
SorterBean, 816
SQL, 893
Stack, 616, 621, 622, 670
StackCalculator, 671, 672, 689, 694, 702
Stream, 711
  GZIP, 754
  ZIP, 754
StreamCorruptedException, 761
StreamPrintService, 795, 797
StreamTokenizer, 729
StreamTokenizer, 746
String, 515, 631, 653, 678
String value, sval, 729
StringBufferInputStream, 729
StringFlavor, 597
StringReader, 729, 730
StringSelection, 597, 598
Stub del cliente, 895, 896, 901, 902
Suavizar contornos, 303-306
Subclases, 78, 745
  thread, 188-189
Subcontenedores, 348
Subdirectorios de paquetes, 140
Sujeto, 464
Sun Microsystems, 829
Superclases, 78, 386
Superstream, 730
SuperTanker, 735, 736
sval, String value, 729
SwingSet2, 695, 697
System.err, 753
System.out, 753

T

Tablas, 558-565
TableCellEditor. 576, 577, 578
TableCellRenderer, 573, 578
TableColumnModel, 569
TableColumnModelEvent, 571
TableModel, 565, 566
TableModelEvent, 568
Tamaño
  de fuentes, 266-269
  de las clases estándar, 235-236
Tarea
  AWT, 520, 523
  de impresión, 780
Tareas periódicas
  demostración, 200-207
  detener, 200
  ejecutar, 197-200
TCP, 831-833, 842-846
Tecnología de cristal líquido, LCD, 670
TestCipher, 889
TextArea, 604
texto multilínea, 323
TextSyntax, 791
TFTP, 835
this
  para referencias a los métodos, 126
  para referencias al constructor, 124-125
Thread, 731
  agrupar, 226-230
  bloqueado, 193-194
  ciclo vital, 192
  comunicaciones, 215-216
  crear, 188
  demonio, 225-226
  ejecutable, 193
  ejecutar, 188
  eliminado, 194
  grupos, 194-195, 196-197
  monitores, 210-213
  múltiples, 523
  obtener información, 196-197
  planificación, 195-196
  prioridad, 195, 196
  sincronización, 207-215
  variables locales, 222-225

ThreadGroup, 735, 736
TickTock, 817
TimeZone, 665
Tipo MIME, 596, 597
Tipo parametrizado, 621
Tipos de datos
  boolean, 94,
  de referencia, 94
  definición, 94-95
  formatos literales, 95-96
  integrados en Java, 94,
  null, 96
  numéricos, 94, 95
  para enteros, 105
  primitivos, 94, 122
  rangos, 95
  String, 94
ToByteArray, 749
TOKEN_EXTENDS, 746
Tokenizing, 745
TomCat, 907
ToolKit, 598, 781
TrafficLight, 821-827
Transferencia
  de imágenes, 603
  de Texto, 598
TransferHandler, 604, 605, 608, 609
transient, 765
TransitionalBean, 817
TreeCellRenderer, 586, 587
TreeMap, 638, 648, 729
TreeModel, 578, 579
TreeModelEvent, 580, 581
TreeNode, 581
TreePath, 585, 586
TreeSelectionEvent, 585
TreeSelectionListener, 586
TreeSelectionModel, 579, 584
TreeSet, 637, 641
try-catch, 721

U

UDP, Protocolo de datagramas, 835, 839
UIManager, 589
UML, 73-74, 83-84, 133
Unicode, 93
  Transfer Format, UTF, 760
URISyntax, 791
URL, 853-855, 859, 860
US-ASCII, 772
UTF (Unicode Transfer Format), 760
UTF-16, 772
UTF-16BE, 772
UTF-16LE, 772
UTF-8, 772
UTF-8, 772

V

ValidateObject, 761
Variables
  de miembro, 97
  declarar, 94
  definición, 93
  iniciar con literales, 96-98
  locales, 98
  ThreadLocal, 223
  valores iniciales, 97-98
VBA6.DLL, 738
Vector, 616-620, 731, 762
VisualAge, 829
VisualCafé, 829
VisualJ#.NET, 167
void, 745
  close, 720
  seek, 720
  setLength, 720
Voter, 816

W

wait(), 215-222
WeakHashMap, 647
WebGain, 829
WebSphere, 829
Win32 EXE, 813
Window, 334
WindowEvent, 516
WingGuard, 705
wrap, 770
WriteAbortedException, 761
writeExternal, 757, 759
writeObject, 756
Writer, 724, 747, 748
WriteStreamHeader, 758

X

XML, 700, 882
XMLDecoder, 767, 768
XMLEncoder, 767, 768
XSLT, 882, 883

Z

ZIP, 742-745
ZipEntry, 743, 744
ZipFile, 742, 743
ZipInputStream, 742