PROGRAMACIÓN

Java 7

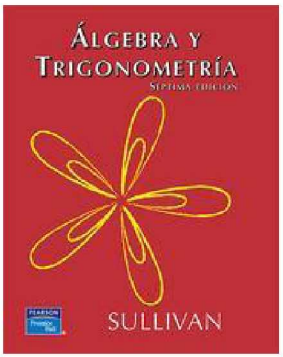
ROGERS CADENHEAD

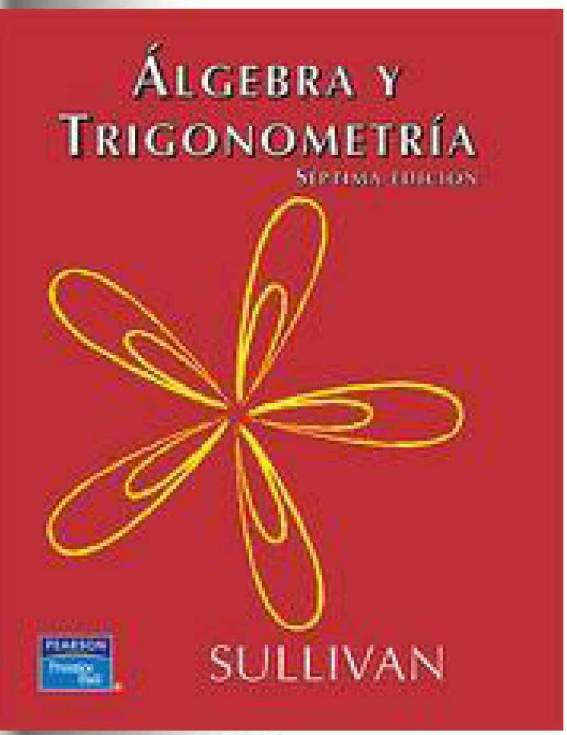
SAMS

ANAYA MULTIMEDIA

Java 7

Java 7





Rogers Cadenhead

ANAYA MULTIMEDIA

PROGRAMACIÓN

TÍTULO DE LA OBRA ORIGINAL:
Sams Teach Yourself Java in 24 Hours, Sixth Edition.

RESPONSABLE EDITORIAL:
Eugenio Tuya Feijoó

TRADUCTOR:
José Luis Gómez Celador

DISEÑO DE CUBIERTA:
Cecilia Poza Melero

Edición española:

© EDICIONES ANAYA MULTIMEDIA (GRUPO ANAYA, S.A.), 2012
Juan Ignacio Luca de Tena, 15. 28027 Madrid
Depósito legal: M-10967-2012
ISBN: 978-84-415-3178-9
Printed in Spain

En este libro me gustaría alejarme de la tradición y no dedicárselo a mi familia y a mis amigos, ya que, sinceramente, se les está subiendo a la cabeza. Me gustaría dedicar este libro a James Gosling, Mike Sheridan, Kim Polese, Bill Joy y a todos los que publicaron la primera versión de este sorprendente lenguaje de programación en 1995, un lenguaje que me sorprendió ver en una página Web y que ahora podemos encontrar en millones de teléfonos Android por todo el mundo, un ejemplo de la visión que tuvieron en Sun Microsystems. ¡Y qué dure mucho tiempo!

Agradecimientos

A todos los de la editorial, en especial a Mark Taber, Songlin Qiu, Tonya Simpson, Charlotte Kughen y Boris Minkin. Ningún autor puede escribir un libro como este sin ayuda. Su excelente trabajo sirvió para recibir todo el crédito posterior.

A mi esposa, Mary, y a mis hijos, Max, Eli y Sam. Aunque no hemos logrado el sueño de ser acróbatas del trapecio, soy el esposo y el padre más orgulloso del mundo.

También me gustaría dar las gracias a todos los lectores que me han enviado comentarios sobre correcciones, erratas y posibles mejoras del libro. La lista la componen Brian Converse, Philip B. Copp III, Wallace Edwards, M.B. Ellis, Kevin Foad, Adam Grigsby, Mark Hardy, Kelly Hoke, Donovan Kelorii, Russel Loski, Jason Saredy, Mike Savage, Peter Schrier, Gene Wines, Jim Yates y otros muchos colaboradores anónimos que me ayudaron a mejorar el libro antes de confeccionar esta lista.

Sobre el autor

Rogers Cadenhead es escritor, programador informático y desarrollador Web, y cuenta en su haber con más de 20 libros sobre temas relacionados con Internet. Se encarga del mantenimiento de Drudge Retort y otros sitios Web que reciben más de 20 millones de visitas al año.

Índice de contenidos

Introducción

Como escritor de libros de informática, dedico mucho tiempo a la sección de informática de las librerías y observo el comportamiento de los lectores mientras hago que hojeo el último número de la revista *In Touch Weekly*.

Gracias a mis investigaciones, he aprendido que si ha elegido este libro y lo abre por la introducción, tengo 12 segundos antes de que lo vuelva a colocar en la estantería y se vaya a tomar un café, así que seré breve: la programación informática con Java es más fácil de lo que parece. No debería decírselo porque miles de programadores han recurrido a sus conocimientos sobre Java para obtener suculentos sueldos en puestos de desarrollo de software, programación de aplicaciones Web y creación de aplicaciones para móviles. Lo último que quiere un programador es que su jefe sepa que cualquiera con tenacidad y tiempo libre puede aprender este lenguaje, el más utilizado en la actualidad. Si sigue paso a paso cada uno de los 24 capítulos del libro, aprenderá a programar con Java rápidamente.

Cualquiera puede aprender a crear programas informáticos, aunque no sepa ni programar un video. Java es uno de los lenguajes de programación más fáciles de aprender, al tratarse de una potente, útil y moderna tecnología utilizada por miles de programadores en todo el mundo.

Este libro está dirigido a los que no son programadores, a programadores noveles que odian el tema y a programadores con experiencia que desean ponerse a punto con Java. Usa Java 7, la última versión del lenguaje.

Java es un lenguaje de programación tremendamente popular ya que permite conseguir objetivos. Puede crear programas con una interfaz gráfica de usuario, diseñar software para Internet, leer datos XML, crear un juego para teléfonos Android y mucho más.

Este libro describe la programación con Java desde cero. Presenta los conceptos de forma sencilla y no tecnicismos, junto con ejemplos detallados de programas funcionales. Basta con dedicarle 24 horas a este libro para crear sus propios programas de Java, con la confianza de poder usar el lenguaje y ampliar sus conocimientos sobre el mismo. También adquirirá otros conocimientos importantes, como los relacionados con informática de redes, diseño de interfaces graficas de usuario y programación orientada a objetos.

Puede que ahora estos términos no le digan demasiado. De hecho, seguramente es el tipo de cosas que hacen que la programación parezca compleja. No obstante, si puede usar un ordenador para consultar el saldo de su cuenta o para crear un álbum de fotos en Facebook, podrá crear programas informáticos con este libro.

Si ahora prefiere tomarse el café y no seguir con Java, vuelva a dejar este libro en la estantería, con la portada hacia fuera en un sitio visible de la librería.

Convenciones

Para ayudarle a sacar el mayor partido al texto y saber dónde se encuentra en cada momento, a lo largo del libro utilizamos distintas convenciones:

- Las combinaciones de teclas se muestran en negrita, como por ejemplo **Control-A**. Los botones de las distintas aplicaciones también aparecen así.
- Los nombres de archivo, URL y código incluido en texto se muestran en un tipo de letra `monoespacial`.
- Los menús, submenús, opciones, cuadros de diálogo y demás elementos de la interfaz de las aplicaciones se muestran en un tipo de letra Arial.

Nota

En estos cuadros se incluye información importante directamente relacionada con el texto adjunto. Los trucos, sugerencias y comentarios afines relacionados con el tema analizado se reproducen en este formato.

Además, en la página Web del libro original en inglés, http://www.informit.com/store/product.aspx?isbn=0672335751 podrá encontrar las actualizaciones , erratas o contenidos adicionales que se actualicen.

1
Convertirse en programador

Probablemente le hayan dicho que la programación informática es increíblemente compleja, que requiere un doctorado en Informática, miles de dólares en hardware y software, una mente analítica, la paciencia del santo Job y una tendencia al consumo de bebidas con cafeína.

Menos la parte sobre la cafeína, no le han dicho la verdad. Programar es más fácil de lo que piensa, independientemente de lo que los programadores nos hayan contado durante años. Es un buen momento para aprender a programar. Existen multitud de herramientas de programación gratuitas para descargar en la red y miles de programadores distribuyen su trabajo bajo licencias de código abierto para que los usuarios puedan ver cómo escriben sus programas, puedan corregir errores y colaborar con mejoras. Incluso en época de crisis, muchas empresas contratan programadores.

Millones de dispositivos móviles utilizan Android, un sistema operativo cuyas aplicaciones se crean en Java. Si tiene un teléfono Android, habrá disfrutado del trabajo de los programadores de Java siempre que haya visto una película, consultado rutas de tráfico o disparado una ave a una fortaleza porcina.

Este libro pretende enseñar a programar con Java a dos tipos de usuarios: los que nunca antes lo han intentado y los que lo han intentado pero odian programar más que Lord Voldemort a los pobres huerfanitos británicos. Usaremos un lenguaje sencillo siempre que sea posible, en lugar de tecnicismos y enrevesados acrónimos, y explicaremos detalladamente todos los términos que aparezcan.

Si lo consigo, al finalizar el libro tendrá suficientes conocimientos de programación. Podrá crear programas, ahondar en otros manuales con más confianza y aprender nuevos lenguajes con facilidad. También tendrá conocimientos de Java, el lenguaje de programación más utilizado del mundo.

El primer capítulo del libro es una introducción a la programación seguida de instrucciones para configurar su equipo y poder crear programas de Java.

Elegir un lenguaje

Si se siente lo suficientemente cómodo con un ordenador como para redactar su CV, consultar el saldo de su cuenta o compartir las fotos de sus viajes en Facebook, puede crear programas informáticos.

La clave consiste en comenzar con el lenguaje adecuado. El que elija suele depender de las tareas que deba realizar. Cada uno tiene sus ventajas e inconvenientes. Durante muchos años, la gente aprendía a programar con alguna variante de BASIC, el lenguaje destinado a los principiantes.

Nota

El lenguaje BASIC se inventó en 1960 para que estudiantes y principiantes lo aprendieran con facilidad. El inconveniente de las variantes de BASIC es su tendencia a adoptar malos hábitos de programación.

Microsoft Visual Basic se ha usado para crear miles de sofisticados programas de uso comercial, empresarial y personal. Sin embargo, los programas creados con algunas versiones de Visual Basic pueden ralentizar los programas escritos en otros lenguajes como C# y Visual C++. Esta diferencia es especialmente evidente en programas que usan muchos gráficos, como los juegos.

Este libro analiza el lenguaje de programación Java, de Oracle Corporation. Aunque Java es más difícil de aprender que otros lenguajes como Visual Basic, es un buen punto de partida por distintos motivos. Una de sus ventajas es que se puede usar en la Web y con teléfonos móviles. Los programas de Java se pueden emplear para crear aplicaciones para teléfonos Android, juegos para navegadores y otros sectores del desarrollo de software.

Otra importante ventaja es que Java requiere un enfoque organizado para que los programas funcionen. Debe ser meticuloso al crear sus programas: Java no funciona si no sigue las reglas.

Al empezar a crear programas en Java, puede que no aprecie el comportamiento del lenguaje como una ventaja. Se cansará de crear un programa y tener que corregir varios errores antes de finalizarlo.

En los próximos capítulos, aprenderá las reglas de Java y los errores que hay que evitar. La ventaja de este esfuerzo adicional es que podrá crear programas más fiables, útiles y sin errores.

Java fue un invento del programador James Gosling como forma de crear programas informáticos. Mientras trabajaba en Sun Microsystems, Gosling no estaba conforme con el rendimiento del lenguaje C++ en un proyecto, por lo que creó un nuevo lenguaje para

solucionarlo. Un debate habitual es si Java es superior a otros lenguajes de programación, pero su éxito en la última década demuestra la estabilidad de su diseño. Tres mil millones de dispositivos en todo el mundo usan Java. Se han publicado más de 1.000 libros sobre el lenguaje desde su aparición (este es mi decimosexto...).

Independientemente de si Java es el mejor lenguaje o no, sin duda es uno de los mejores para aprender. Lo podrá comprobar en el capítulo 2.

Aprender un lenguaje de programación facilita enormemente el aprendizaje posterior de otros lenguajes. Muchos son similares, de modo que no se empieza desde cero al aprender un nuevo lenguaje. Por ejemplo, para muchos programadores de C++ y Smalltalk es fácil aprender Java, ya que usa muchas ideas de dichos lenguajes. Del mismo modo, C# adopta muchos de los conceptos de Java.

Nota

C++ es un lenguaje desarrollado por Bjarne Stroustrop de Bell Laboratories. Es una mejora del lenguaje de programación C. Seguramente se pregunte por qué no lo denominaron solo C+. El segundo signo + es un chiste que revelaremos más tarde.

Indicar al ordenador qué debe hacer

Un programa informático, también denominado software, es una forma de indicarle al ordenador lo que debe hacer. Todo lo que hace un ordenador, desde encenderse a apagarse, se realiza mediante programas. Windows 7 es un programa; Call of Duty es un programa; el software de controladores de su impresora es un programa; incluso un virus de correo electrónico es un programa.

Los programas informáticos se componen de una lista de comandos que el ordenador procesa en un orden concreto al ejecutar el programa. Cada comando se denomina instrucción.

Si tiene un mayordomo en casa, podría darle las siguientes instrucciones detalladas:

Estimado Sr. Jeeves,

Le ruego se encargue de estos recados mientras me ausento unos instantes:

Recado 1: Pasar el aspirador al salón.

Recado 2: Ir a la tienda.

Recado 3: Comprar salsa de soja, wasabi y todo el sushi que encuentre.

Recado 4: Volver a casa.

Gracias,

Bertie Wooster

Si le dice al mayordomo lo que tiene que hacer, existe cierto margen a la hora de acatar sus órdenes. Si no hay un producto en la tienda, puede traer otro.

Los ordenadores no tienen margen. Siguen las instrucciones al pie de la letra. Los programas que escriba se ejecutan con precisión, instrucción por instrucción.

El siguiente ejemplo es un sencillo programa informático escrito en BASIC. Mírelo pero no se preocupe del significado de cada línea.

```
1 PRINT "Shall we play a game?"
2 INPUT A$
```

Si lo traducimos a lenguaje normal, equivale a la siguiente lista de tareas para el ordenador:

Querido ordenador personal,

1: Mostrar la pregunta, "Shall we play a game?[1]"

2: Permitir al usuario responder la pregunta.

Saludos,

Snookie Lumps

Cada una de las líneas es una instrucción. Un ordenador procesa las instrucciones en un orden concreto, al igual que un cocinero sigue una receta o el mayordomo acata las órdenes de Bertie Wooster. En BASIC, se usan números de línea para ordenar correctamente las instrucciones. Otros lenguajes como Java no recurren a números de línea, lo que permite indicar al ordenador cómo ejecutar un programa de diversas formas.

En la figura 1.1 puede ver un sencillo programa de BASIC ejecutado en el intérprete AppleSoft BASIC de Joshua Bell. Debe ejecutar el intérprete en un navegador Web y lo puede encontrar en `www.calormen.com/Applesoft`.

Debido al funcionamiento del programa, no se puede culpar al ordenador si hay algún fallo, ya que hace exactamente lo que le decimos. La culpa de los errores es del programador. Es la mala noticia. La buena es que no hay daños permanentes. Nadie resultó herido durante la creación de este libro y tampoco habrá ordenadores heridos cuando aprenda a programar en Java.

Cómo funcionan los programas

La mayoría de programas informáticos se crean como si realizara una carta: escribiendo las instrucciones en un editor de texto. Algunas herramientas de programación incorporan un editor propio y otras se pueden usar con cualquier software de edición de texto.

1. La frase Shall we play a game? (¿Jugamos a un juego?) es de la película Juegos de Guerra (1983), en la que un joven programador informático (Matthew Broderick) salva al mundo después de estar a punto de provocar una guerra termonuclear global.

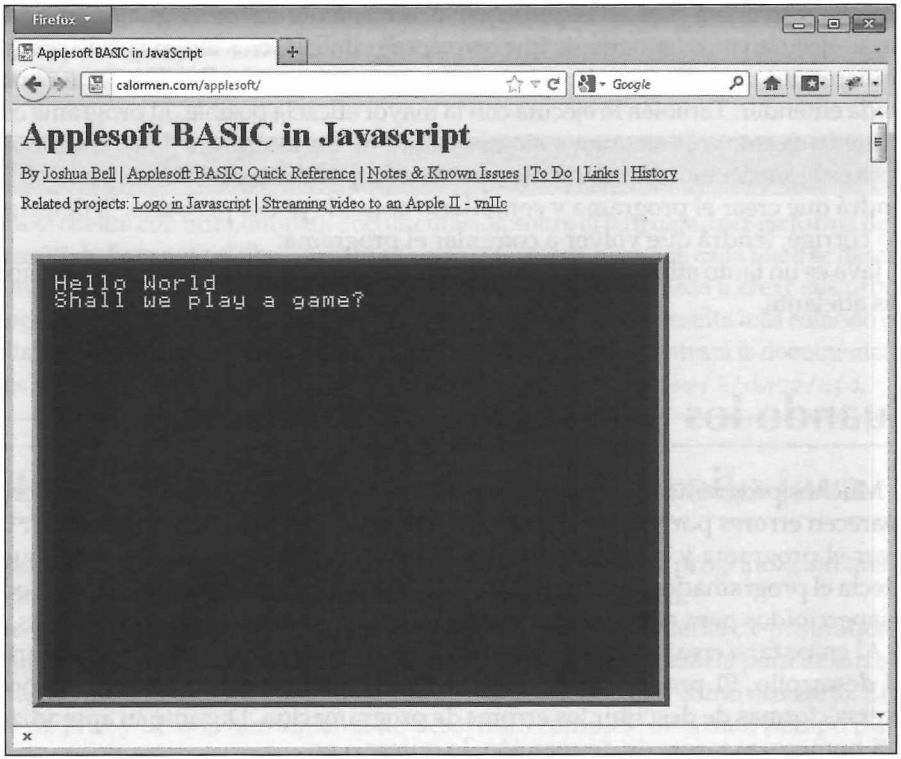


Figura 1.1. Ejemplo de un programa de BASIC.

Nota

Si su editor de texto es un programa con funciones como texto en negrita, tamaños de fuente y otros retoques estilísticos, no las utilice al escribir su programa. Los programas informáticos deben prepararse como archivos de texto sin un formato especial. El Bloc de notas, un procesador de texto incluido en Windows, guarda los archivos como texto sin formato. También puede usar el editor vi en sistemas Linux.

Cuando termine de crear un programa informático, debe guardar el archivo. Los programas suelen tener una extensión de archivo propia para indicar su naturaleza. Los programas de Java deben usar la extensión `.java`, como en `Calculator.java`.

Para ejecutar un programa guardado como archivo necesita ayuda, y el tipo de ayuda depende del lenguaje de programación que utilice. Algunos lenguajes requieren un intérprete para ejecutar sus programas. Un intérprete es un programa que interpreta las líneas de código y le dice al ordenador qué hacer. Muchas versiones de BASIC son lenguajes interpretados. La ventaja de un lenguaje interpretado es que resulta más fácil de probar. Al crear un programa de BASIC, puede probarlo inmediatamente, corregir los

errores y volverlo a probar. El principal inconveniente de los lenguajes interpretados es que se ejecutan más lentamente que otros programas. Otros lenguajes de programación requieren un compilador, el cual traduce el programa a un formato que el ordenador pueda entender. También lo ejecuta con la mayor eficacia posible. El programa compilado se puede ejecutar directamente sin necesidad de un intérprete. Los programas compilados se ejecutan más rápidamente que los interpretados pero tardan más en probarse. Tendrá que crear el programa y compilarlo antes de poder probarlo. Si detecta un error y lo corrige, tendrá que volver a compilar el programa.

Java es un tanto atípico, ya que requiere un compilador y un intérprete, como veremos más adelante.

Cuando los programas no funcionan

Muchos programadores noveles se desaniman al empezar a probar sus programas. Aparecen errores por todas partes. Algunos son sintácticos, que el ordenador identifica al ver el programa y quedar confundido por el código. Otros son lógicos, que solo los detecta el programador al probar el programa (o que puede pasar por alto). Pueden pasar desapercibidos para el ordenador pero suelen provocar acciones inesperadas.

Al empezar a crear programas, se familiarizará con los errores. Son una parte natural del desarrollo. El proceso de corrección de errores se denomina depuración. Existen diversas formas de describir los errores de programación. Durante su aprendizaje, realizará numerosas tareas de depuración, lo quiera o no.

Seleccionar una herramienta de programación de Java

Antes de poder crear sus programas de Java, necesita software de programación. Existen distintos programas para Java, como Java Development Kit, Eclipse, IntelliJ IDEA y NetBeans. Siempre que Oracle publica una nueva versión de Java, la primera herramienta que admite es el JDK (*Java Development Kit*, Kit de desarrollo de Java).

Para crear los programas de este libro, debe usar la versión 7 del JDK u otra herramienta de programación compatible con la misma. El JDK es una relación de herramientas gratuitas de línea de comandos para crear software de Java. Carece de interfaz gráfica, de modo que si nunca ha trabajado en un entorno no gráfico como DOS o Linux, le sorprenderá, y no positivamente, cuando empiece a usar el JDK.

Oracle le ofrece otra herramienta gratuita; el entorno de desarrollo integrado NetBeans, mucho más indicada para escribir código Java. NetBeans cuenta con una interfaz gráfica de usuario, un editor de código fuente, un diseñador de interfaz de usuario y un administrador de proyectos. Funciona junto al JDK, entre bastidores, de modo que debe instalar ambas herramientas en su sistema para empezar a desarrollar programas de Java.

Los programas de este libro se han creado con NetBeans, que puede descargar e instalar en un paquete con el JDK. Puede usar otras herramientas siempre que sean compatibles con JDK 7.

Nota

Oracle cuenta con una completa documentación sobre el lenguaje Java en forma de páginas Web. Para usar el libro no necesita esta información, ya que cada tema se describe detalladamente, pero las páginas son muy útiles cuando empiece a crear sus propios programas. Puede descargar la documentación completa pero resulta más cómodo consultarla cuando sea necesario desde el sitio Web de Oracle. Encontrará la documentación más actualizada en `http://download.oracle.com/javase/7/docs/api`.

Instalar una herramienta de desarrollo Java

Todos los capítulos del libro concluyen con un proyecto de programación que puede realizar para mejorar sus conocimientos sobre el tema descrito.

No podrá programar con Java si no cuenta con una herramienta de programación en su equipo. Si cuenta con una como NetBeans o el JDK, puede usarla para desarrollar los programas de los siguientes capítulos. Sin embargo, debe saber cómo utilizarla. El aprendizaje de Java y de una herramienta de desarrollo compleja al mismo tiempo puede ser complicado. Si no tiene una, puede usar NetBeans 7, que encontrará de forma gratuita en el sitio Web de Oracle: `www.netbeans.org`.

En el apéndice A encontrará instrucciones para descargar e instalar NetBeans.

Resumen

En este capítulo hemos presentado el concepto de programación en un ordenador proporcionándole una serie de instrucciones para indicarle qué debe hacer. También hemos abordado la descarga e instalación de una herramienta de desarrollo de Java para crear los programas de ejemplo del libro.

Si todavía está confuso acerca de los programas, lenguajes de programación o Java en general, no se preocupe. Todo empezará a tener sentido en el siguiente capítulo, donde analizaremos el proceso de creación de un programa de Java.

Preguntas y respuestas

P: ¿BASIC? ¿C++? ¿Smalltalk? ¿Java? ¿Qué significan los nombres de estos lenguajes?

R: BASIC toma su nombre de sus siglas: *Beginner's All Symbolic Instruction Code* (Código de instrucciones simbólicas para principiantes). C++ es un lenguaje de programación creado como mejora del lenguaje C que, a su vez, se creó como mejora del lenguaje de programación B. Smalltalk es un innovador lenguaje orientado a objetos desarrollado en los años 70 con numerosos conceptos adoptados por Java.

Java no cumple la tradición de asignar un acrónimo como nombre del lenguaje. Es el nombre que más gustaba a sus desarrolladores, por encima de WebRunner, Silk, Ruby y otros. Cuando cree mi propio lenguaje de programación, lo llamaré Salsa. A todo el mundo le gusta la salsa.

P: ¿Por qué los lenguajes interpretados son más lentos que los compilados?

R: Son más lentos por la misma razón que una persona que interpreta un discurso en directo en un idioma extranjero es más lenta que un traductor con el discurso impreso. El intérprete en directo tiene que pensar en todas las instrucciones en cuanto se producen, mientras que el traductor puede trabajar con el discurso en su totalidad y tomar atajos para acelerar el proceso. Los lenguajes compilados pueden ser mucho más rápidos que los interpretados ya que toman decisiones para que el programa se ejecute con mayor eficacia.

Ejercicios

Preguntas

Pruebe sus conocimientos sobre el material descrito en este capítulo respondiendo a las siguientes preguntas.

1. ¿Cuál de los siguientes motivos no hace pensar que la programación informática es realmente complicada?

   A. Los programadores siembran ese rumor para mejorar sus expectativas laborales.

   B. La jerga técnica y los acrónimos.

   C. La gente que cree que la programación es demasiado compleja puede obtener una fianza del gobierno.

2. ¿Qué tipo de herramienta ejecuta un programa informático interpretando línea a línea?

   A. Una herramienta lenta.

   B. Un intérprete.

   C. Un compilador.

3. ¿Por qué se encerró James Gosling en su oficina y creó Java?
   A. No le gustaba el lenguaje que estaba utilizando en un proyecto.
   B. Su grupo de rock no conseguía conciertos.
   C. Cuando en el trabajo no puedes visitar YouTube, Internet es bastante aburrida.

Respuestas

1. C. Los escritores de libros de informática tampoco consiguen fianzas.
2. B. Los compiladores interpretan las instrucciones con antelación para que el programa se ejecute más rápidamente.
3. A. Estaba desesperado con C++. En 1991, cuando Gosling creó Java, YouTube no existía.

Actividades

Si desea ampliar sus conocimientos sobre Java y la programación informática, pruebe con las siguientes actividades:

- Visite el sitio de Java de Oracle en `www.oracle.com/technetwork/topics/newtojava` y consulte las páginas introductorias.
- Con frases comunes en lugar de un lenguaje de programación, cree una serie de instrucciones para sumar 10 a un número elegido por el usuario y después multiplicar el resultado por 5. Divida las instrucciones en líneas breves de una línea.

2
Crear su primer programa

Como vimos en el capítulo anterior, un programa informático es una serie de instrucciones que indican al ordenador lo que debe hacer. Estas instrucciones se proporcionan al ordenador mediante un lenguaje de programación.

En este capítulo crearemos el primer programa de Java desde un editor de texto. Cuando termine, guarde el programa, compílelo y pruébelo.

Qué necesita para crear programas

Como explicamos antes, para crear programas de Java necesita una herramienta de programación compatible con el JDK, como el entorno de desarrollo integrado (IDE) NetBeans. Necesita una herramienta que pueda compilar y ejecutar programas de Java, y un editor de texto para escribir dichos programas. En la mayoría de lenguajes de programación, los programas se crean escribiendo texto en un editor de texto (también denominado editor de código fuente). Algunos lenguajes incorporan su propio editor de texto. NetBeans incluye uno propio para crear programas de Java. Los programas de Java son sencillos archivos de texto sin formato especial, como el texto centrado o en negrita. El editor de código fuente de NetBeans funciona como un editor de texto con una mejor muy útil. El color del texto identifica los distintos elementos del lenguaje mientras se escribe. NetBeans también sangra las líneas y ofrece documentación de ayuda.

Como los programas de Java son archivos de texto, puede abrirlos y editarlos en cualquier editor de texto. Podría crear un programa de Java con NetBeans, abrirlo en el Bloc de notas de Windows, modificarlo y volver a abrirlo en NetBeans sin problemas.

Crear el programa Saluton

El primer programa de Java que crearemos es una aplicación que muestra un saludo tradicional del mundo de la informática: "Saluton mondo!". Para preparar el nuevo proyecto de programación en NetBeans, si todavía no lo ha hecho, cree un nuevo proyecto con el nombre `Java24` y siga los pasos descritos a continuación:

1. Seleccione Archivo>Proyecto Nuevo.
2. Seleccione la categoría de proyecto Java y el tipo de proyecto Java Application (Aplicación de Java). Pulse **Siguiente**.
3. Introduzca **Java24** como nombre del proyecto. Verá un mensaje de error indicando que la carpeta del proyecto ya existe y no está vacía si ha creado antes este proyecto.
4. Anule la selección de la casilla de verificación Crear clase principal.
5. Pulse **Terminar**.

El proyecto `Java24` se crea en su propia carpeta. Puede utilizarlo para todos los programas de Java que cree en el libro.

Iniciar el programa

NetBeans agrupa los programas relacionados en un proyecto. Si no tiene abierto el proyecto `Java24`, siga estos pasos:

- Seleccione Archivo>Abrir Proyecto.
- Busque y seleccione la carpeta `NetBeansProjects` (si es necesario).
- Seleccione `Java24` y pulse **Abrir proyecto**.

El proyecto `Java24` aparece en el panel Proyectos. Para añadir un nuevo programa al proyecto actual, seleccione Archivo>Archivo Nuevo. Se abrirá el asistente Archivo Nuevo (véase la figura 2.1).

El panel Categorías enumera los distintos tipos de programas de Java que puede crear. Haga clic en la carpeta Java del panel para ver los tipos de archivos de esa categoría. Para este primer proyecto, seleccione Empty Java File (Archivo de Java vacío) y pulse **Siguiente**.

En el campo Nombre de Clase introduzca **Saluton** y pulse **Terminar** para crear el nuevo programa de Java. El archivo vacío `Saluton.java` se abre en el editor de código fuente.

Con el editor de código fuente abierto, debe introducir las líneas del listado 2.1. Estas instrucciones forman el código fuente del programa.

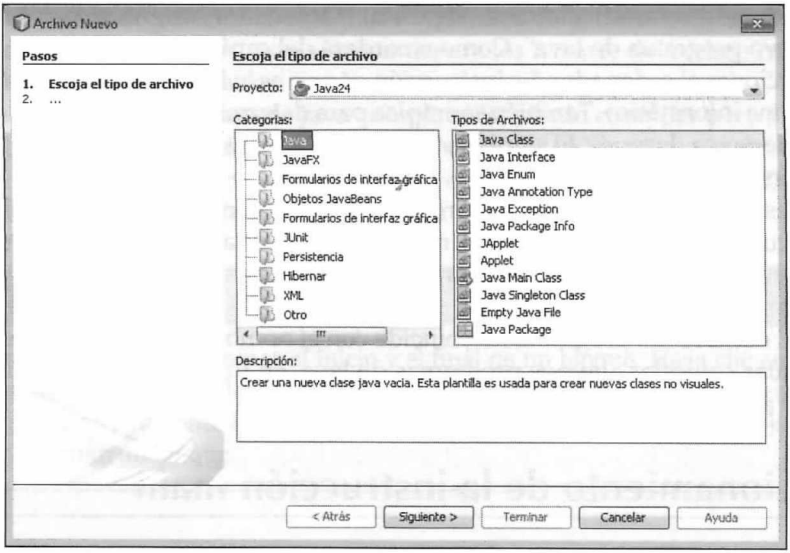


Figura 2.1. El asistente Archivo Nuevo.

Listado 2.1. El programa Saluton.

```
1: public class Saluton {
2:    public static void main(String[] arguments) {
3:       // Mi primer programa de Java
4:    }
5: }
```

Advertencia

No añada los números de línea y los dos puntos; se usan en el libro para hacer referencia a números de línea concretos.

Introduzca el texto tal y como se muestra en el libro y use espacios o tabulaciones para añadir los espacios en blanco de las líneas 2-4. Cuando termine, seleccione Archivo>Guardar o pulse el botón **Guardar todo** para almacenar el archivo. Por el momento, `Saluton.java` contiene la estructura de un programa de Java. Crearemos varios programas idénticos a éste, a excepción de la palabra `Saluton` en la línea 1. Esta palabra representa el nombre del programa y cambia en los nuevos programas que cree. La línea 3 es una frase en lenguaje estándar. El resto seguramente le sea nuevo.

La instrucción class

La primera línea del programa es la siguiente:

```
class Saluton {
```

Traducido a nuestro idioma, significa: "Ordenador, asigna el nombre `Saluton` a mi nuevo programa de Java". Como recordará del capítulo anterior, debe proporcionar instrucciones al ordenador. La instrucción `class` es la forma de asignar un nombre a un programa informático. También se emplea para determinar otros aspectos del programa, como veremos después. El significado del término `class` es que los programas de Java también se denominan clases.

En este ejemplo, el nombre del programa `Saluton` coincide con el nombre de archivo del documento, `Saluton.java`. Un programa de Java debe tener un nombre que coincida con la primera parte de su nombre de archivo, y debe mostrar las mismas mayúsculas y minúsculas.

Si el nombre del programa no coincide con el nombre de archivo, se genera un error al intentar compilar algunos programas de Java, en función de cómo se use la instrucción `class` para configurar el programa.

Funcionamiento de la instrucción main

La siguiente línea del programa:

```
public static void main(String[] arguments) {
```

indica al ordenador que es el inicio de la parte principal del programa. Los programas de Java se organizan en diferentes secciones, por lo que debe haber una forma de identificar la parte que se procesa primero.

La instrucción `main` es el punto de entrada a la mayoría de programas de Java. Las principales excepciones son los applet, programas que se ejecutan como parte de una página Web, y los servlet, programas ejecutados por un servidor Web. La mayor parte de los programas que crearemos en los siguientes capítulos usan `main` como punto de partida.

Llaves

En el programa Saluton, todas las líneas menos la 3 contienen una llave: `{` o `}`. Estos signos permiten agrupar partes de un programa (al igual que se emplean los paréntesis en una frase para agrupar palabras). Todo lo contenido entre la llave de inicio `{` y la de cierre `}` forma parte del mismo grupo. Estas agrupaciones se denominan bloques. En el listado 2.1, la llave de inicio de la línea 1 se asocia a la llave de cierre de la línea 5, lo que convierte a todo el programa en un bloque. Puede usar las llaves de esta forma para indicar el inicio y el final de un programa.

Los bloques se pueden ubicar dentro de otros bloques (al igual que los paréntesis de esta frase (y otro conjunto más en ésta)). El programa `Saluton` tiene llaves en las líneas 2 y 4 para definir otro bloque. Este bloque comienza con la instrucción `main`. Todo lo incluido dentro del bloque de `main` es un comando que el ordenador debe procesar al ejecutar el programa. La siguiente instrucción es lo único ubicado dentro del bloque:

```
// Mi primer programa de Java
```

Es un marcador de posición. Los signos `//` al inicio de la línea indican al ordenador que la ignore, ya que se ha incluido solamente para los lectores del código fuente. Estas líneas se denominan comentarios.

Hemos creado un programa de Java completo. Se puede compilar pero si se ejecuta no sucede nada, por que todavía no le hemos dicho al ordenador qué debe hacer. El bloque de la instrucción `main` solo contiene un comentario, que se ignora. Debe añadir más instrucciones entre las llaves del bloque `main`.

Nota

NetBeans le permite determinar el inicio y el final de un bloque. Haga clic en una de las llaves del código fuente del programa `Saluton`. La llave se vuelve de color amarillo junto a su correspondiente llave. Las instrucciones de Java incluidas entre las llaves amarillas forman un bloque.

Almacenar información en una variable

En los programas que cree, necesitará un lugar para almacenar información durante un breve periodo de tiempo. Para ello puede emplear una variable, un almacén para guardar información como números enteros, de coma flotante, valores `true` y `false`, caracteres y líneas de texto. La información almacenada en una variable puede cambiar, por ello se denomina variable.

En `Saluton.java`, cambie la línea 3 por lo siguiente:

```
String greeting = "Saluton mondo!";
```

Esta instrucción indica al ordenador que almacena la línea de texto `"Saluton mondo!"` en la variable `greeting`.

En un programa de Java, debe indicar al ordenador qué tipo de información almacena una variable. En este ejemplo, `greeting` es una cadena, una línea de texto que puede incluir letras, números, signos de puntuación y otros caracteres. Al incluir `String` en la instrucción se configura la variable para almacenar valores de cadena.

Al añadir esta instrucción a un programa, debe incluir punto y coma al final de la línea, para finalizar la instrucción; son como los puntos al final de una clase. El ordenador los usa para determinar cuándo finaliza una instrucción y comienza la siguiente. Si incluye una instrucción por línea, el programa resulta más legible (para los lectores humanos).

Mostrar el contenido de una variable

Si ahora ejecuta el programa, no mostrará nada. La orden de almacenar una línea de texto en la variable `greeting` se produce entre bastidores. Para que el ordenador muestre lo que hace, puede mostrar el contenido de la variable.

Añada otra línea vacía en el programa `Saluton` tras la cadena `greeting = "Saluton mondo!"`. Use ese espacio vacío para introducir la siguiente instrucción:

```
System.out.println(greeting);
```

Esta instrucción indica al ordenador que muestre el valor almacenado en la variable `greeting`. La instrucción `System.out.println` indica al ordenador que muestre una línea en el dispositivo de salida de su equipo: su monitor.

Guardar el producto terminado

Su programa debería parecerse al ilustrado en el listado 2.2, aunque puede haber utilizado un espaciado diferente en las líneas 3-4. Realice las correspondientes correcciones y guarde el archivo (por medio de Archivo>Guardar o el botón **Guardar todo**).

Listado 2.2. Versión final del programa Saluton.

```
1: class Saluton {
2:   public static void main(String[] args) {
3:     String greeting = "Saluton mondo!";
4:     System.out.println(greeting);
5:   }
6: }
```

Al ejecutar el programa, se ejecutan las instrucciones del bloque `main` en las líneas 3 y 4. El listado 2.3 muestra el programa si se escribiera en lenguaje normal, no en Java.

Listado 2.3. Descripción línea a línea del program Saluton.

```
1: Aquí empieza el programa Saluton:
2:   Aquí empieza la parte principal del programa:
3:     Almacenar el texto "Saluton mondo!" en la variable de cadena greeting
4:     Mostrar el contenido de la variable greeting
5:   Aquí termina la parte principal del programa.
6: Aquí termina el programa Saluton.
```

Compilar el programa en un archivo de clase

Antes de poder ejecutar un programa de Java, debe compilarlo. Al hacerlo, las instrucciones asignadas al ordenador se convierten a un formato que pueda comprender.

NetBeans compila automáticamente los programas al guardarlos. Si ha escrito el listado 2.2, el programa se compilará correctamente y se crea una versión compilada del mismo, el archivo `Saluton.class`. Todos los archivos de Java se compilan en archivos de clase y se les asigna la extensión `.class`. Un programa de Java puede estar formado por varias clases pero en un programa sencillo como `Saluton` solo se necesita una.

Nota

El compilador de Java solo se manifiesta cuando hay un error. Si compila correctamente un programa sin errores, no hay respuesta. Es un tanto extraño. Cuando empecé a programar en Java, esperaba que una compilación satisfactoria me saludara con una gran banda de cornetas.

Corregir errores

Al crear un programa en el editor de código fuente de NetBeans, los errores se indican con un icono rojo a la izquierda del panel del editor (véase la figura 2.2).

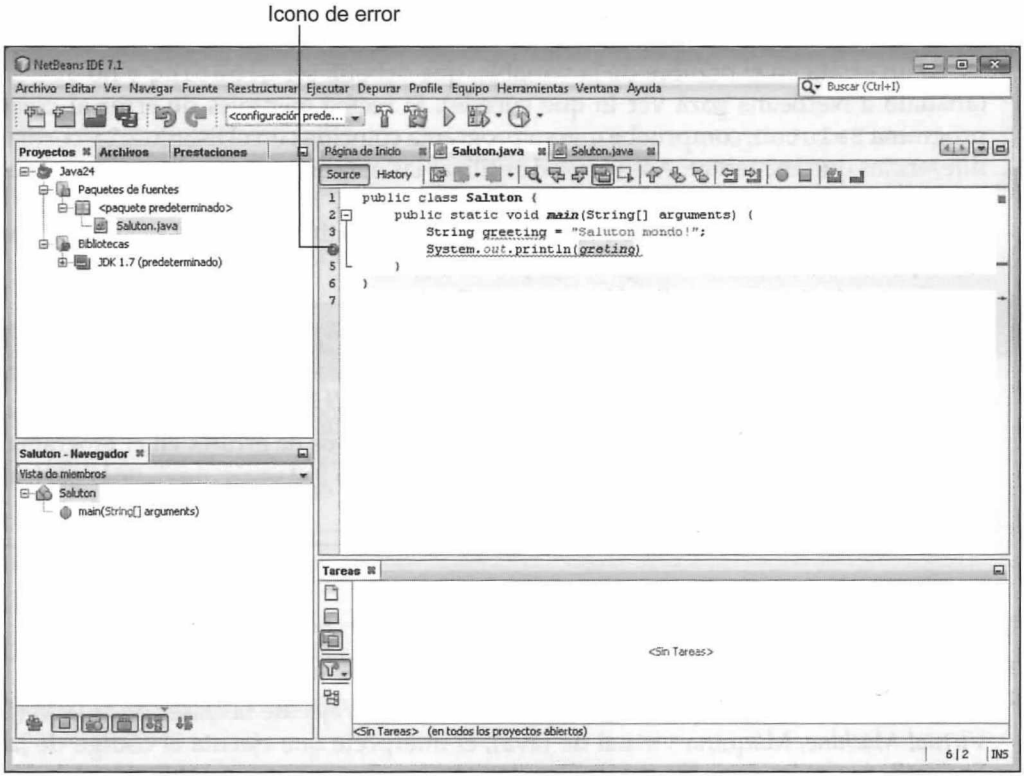


Figura 2.2. Detección de errores en el editor de código fuente.

El icono aparece en la línea que ha desencadenado el error. Puede hacer clic en él para mostrar un mensaje que explique el error de compilación con los siguientes detalles:

- El nombre del programa de Java.
- El tipo de error.
- La línea en que se produce el error.

Veamos un ejemplo de un mensaje de error que puede aparecer al compilar el programa `Saluton`:

```
cannot find symbol.
symbol : variable greting
location: class Saluton
```

El error es la primera línea del mensaje: `cannot find symbol`. Estos mensajes pueden resultar confusos para los programadores noveles. Si el error no tiene sentido, no dedique demasiado tiempo a descifrarlo. Fíjese en la línea en la que aparece y busque los motivos más obvios. Por ejemplo, ¿qué sucede en la siguiente instrucción?

```
System.out.println(greting);
```

Es un errata en el nombre de la variable, que debería ser `greeting` y no `greting` (añádalo a NetBeans para ver lo que sucede). Si recibe mensajes de error al crear el programa `Saluton`, compruebe que su programa coincida con el listado 2.2 y corrija las diferencias que encuentre. Asegúrese de utilizar las mayúsculas y minúsculas correctas, y de incluir todos los signos de puntuación, como `{`, `}` y `;`.

Por lo general, un detallado examen de la línea identificada por el mensaje de error basta para revelar los errores que hay que corregir.

Truco

En la página Web del libro encontrará los archivos de código de todos los programas que vamos a crear. Si no detecta erratas ni otros motivos de errores en el programa `Saluton` pero sigue sin funcionar, descargue el archivo `Saluton.java` de la página y pruébelo.

Ejecutar un programa de Java

Para comprobar si el programa `Saluton` funciona, ejecute la clase con la JVM (*Java Virtual Machine*, Máquina virtual de Java), el intérprete que ejecuta el código de Java. En NetBeans, seleccione Ejecutar>Ejecutar archivo. Se abrirá el panel Salida debajo del editor de código. Si no hay errores, el programa muestra su resultado en este panel (véase la figura 2.3). Si ve el texto `"Saluton Mondo!"` significa que ha escrito su primer programa de Java. Su ordenador acaba de saludar al mundo, una tradición en el mundo de la programación tan importante para muchos de nosotros como la cafeína, las camisas de manga corta y el Call of Duty.

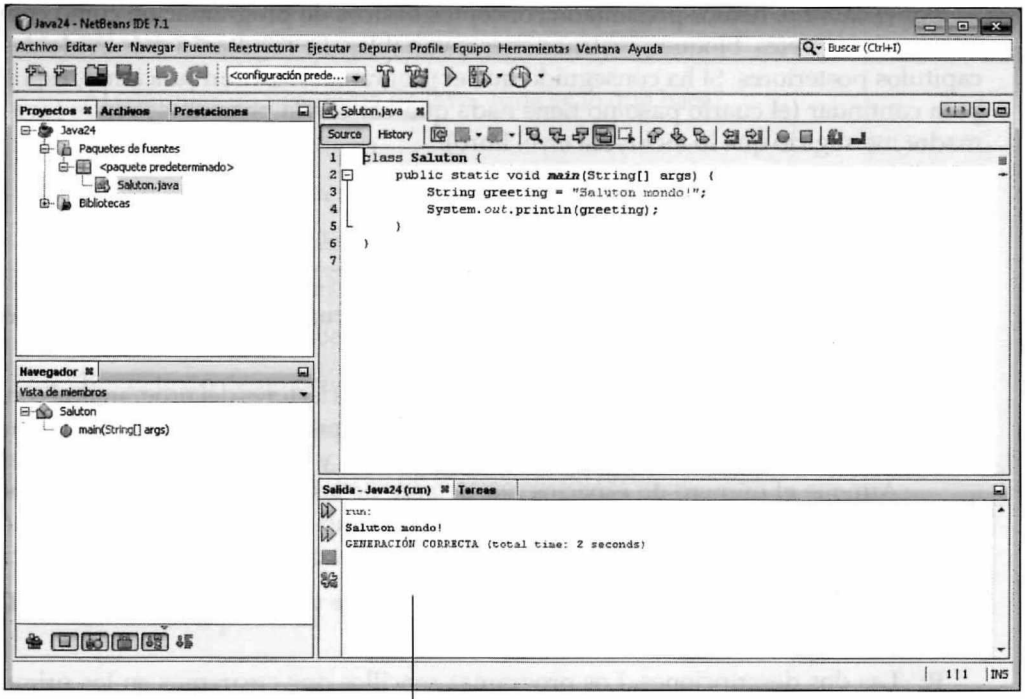


Figura 2.3. Ejecución de su primer programa de Java.

Nota

Seguramente se pregunte porqué "`Saluton mondo!`" es un saludo tradicional. Significa "¡Hola mundo!" en esperanto, un idioma artificial creado por Ludwig Zamenhof en 1887 para facilitar la comunicación internacional. Es tradicional en el sentido de que voy a intentar instaurarlo como tal.

Resumen

En este capítulo hemos creado un programa de Java con el IDE NetBeans. Hemos aprendido que para desarrollar un programa de Java es necesario completar cuatro pasos básicos:

1. Crear el programa en un editor de texto.
2. Compilar el programa en un archivo de clase.
3. Indicar a la MVJ que ejecute la clase.
4. Llamar a su madre.

Por el camino, hemos presentado conceptos básicos de programación como compiladores, intérpretes, bloques, instrucciones y variables. Le resultarán más evidentes en capítulos posteriores. Si ha conseguido que el programa `Saluton` funciones, está listo para continuar (el cuarto paso no tiene nada que ver con la programación de Java. Mi madre me sugirió que lo incluyera en el libro).

Preguntas y respuestas

P: ¿Qué importancia tiene añadir el número adecuado de espacios en blanco en una línea de un programa de Java?

R: Es totalmente irrelevante. El espaciado ayuda a los lectores del programa; al compilador de Java le da igual. Podría haber escrito el programa `Saluton` sin espacios o emplear tabulaciones para sangrar las líneas y se compilaría correctamente. Aunque el número de espacios por delante de las líneas es irrelevante, debería ser coherente en sus programas, ya que el espaciado le permite saber cómo se organiza el programa y a qué bloque de programación pertenece una instrucción.

P: Un programa de Java se ha descrito como clase y como grupo de clases. ¿Qué es lo correcto?

R: Las dos descripciones. Los programas sencillos que crearemos en los primeros capítulos se compilan en un mismo archivo con la extensión `.class`. Puede ejecutarlos con la MVJ. Los programas también pueden estar formados por un grupo de clases, como veremos en el capítulo 10.

P: Si se necesita punto y coma al final de cada instrucción, ¿por qué la línea `// Mi primer programa de Java` no termina en punto y coma?

R: El compilador ignora los comentarios. Si añade `//` en una línea de su programa, indica al compilador que ignore todo lo que aparece a la derecha de las barras. El siguiente ejemplo muestra un comentario en la misma línea que una instrucción:

```
System.out.println(greeting); // ¡Hola, mundo!
```

P: No puedo encontrar errores en la línea indicada por el compilador. ¿Qué debo hacer?

R: El número de línea mostrado con el mensaje de error no siempre es donde debe corregirlo. Examine las instrucciones situadas directamente encima del mensaje de error. Éste suele encontrarse en el mismo bloque de programación.

Ejercicios

Pruebe sus conocimientos sobre el material descrito en este capítulo respondiendo a las siguientes preguntas.

Preguntas

1. Al compilar un programa de Java, ¿qué es lo que hace?
   A. Lo guarda en el disco.
   B. Lo convierte a un formato que el ordenador entienda mejor.
   C. Lo añade a su colección de programas.
2. ¿Qué es una variable?
   A. Algo que se tambalea pero no se cae.
   B. El texto de un programa que el compilador ignora.
   C. Un lugar para almacenar información en un programa.
3. ¿Cómo se llama el proceso de corrección de errores?
   A. Descongelación.
   B. Depuración.
   C. Descomposición.

Respuestas

1. B. Al compilar un programa se convierte un archivo `.java` en un archivo o grupo de archivos `.class`.
2. C. Las variables son un punto para almacenar información; más adelante veremos otros como las matrices y las constantes.
3. B. El proceso se denomina depuración. Algunas herramientas de programación incluyen un depurador para corregir errores.

Actividades

Si desea ampliar sus conocimientos sobre los temas descritos en el capítulo, pruebe con las siguientes actividades:

- Puede traducir la frase "Hello world!" en inglés a otros idiomas con ayuda de Babelfish de Yahoo! (`http://babelfish.yahoo.com`). Cree un programa que permita a su ordenador saludar al mundo en francés, italiano o portugués.
- En el programa `Saluton`, añada uno o dos errores. Por ejemplo, elimine el punto y coma al final de una línea o cambie `println` por `println` (con un número `1` en lugar de la letra `L`). Guarde el programa e intente compilarlo. Compare los mensajes de error.

3
Vacaciones en Java

Antes de adentrarnos en la programación con Java, conviene saber más respecto al lenguaje y cómo lo usan los programadores en la actualidad. Aunque Java ha superado sus orígenes como lenguaje orientado a programas de navegador Web, encontrará interesantes ejemplos de su aplicación en la red.

En este capítulo, veremos sitios con programas de Java y analizaremos la historia y el desarrollo del lenguaje. Para emprender este viaje, necesita un navegador Web configurado para ejecutar programas de Java.

Abra su navegador preferido, póngase su camisa de flores y prepárese para sus vacaciones. No tendrá que salir de casa y no disfrutará de las simples placeres del turismo: taxistas kamikazes, comida exótica, lugareños exóticos, lugareños exóticos con comida y demás. Pero mírelo por el lado bueno: sin problemas de cambio de moneda, sin pasaportes y sin la venganza de Montezuma.

Primera parada: Oracle

Las vacaciones en Java comienza en `http://www.java.com/es/`, un sitio creado por Oracle, la empresa propietaria del lenguaje Java. Un programa Java que se ejecuta como parte de una página Web se denomina applet. Se añaden a una página como cualquier otro elemento. El lenguaje de marcado HTML define dónde debe mostrarse el programa, su tamaño y cuándo se ejecuta. Java también mejora la Web de otras formas: los programas de escritorio creados en Java se pueden iniciar desde un servidor Web y los servlet de Java se ejecutan en servidores Web para ofrecer aplicaciones Web.

La división Java de Oracle lidera el desarrollo del lenguaje y del software relacionado. La sección Java en acción del sitio Web muestra el uso de Java en sitios Web, teléfonos Android y otras plataformas. Existen millones de dispositivos que ejecutan programas creados con Java. En la figura 3.1 puede ver RuneScape, un famoso juego *online* multijugador basado en Java. Puede jugar de forma gratuita desde cualquier navegador Web (`www.runescape.com`).

Figura 3.1. El juego online RuneScape, basado en Java.

En `Java.com` podrá saber cómo se emplea Java. Oracle también dispone de un sitio Web más técnico para programadores de Java (`http://www.oracle.com/technetwork/java`). Aquí encontrará las últimas versiones publicadas de NetBeans y el JDK, además de otros recursos de programación.

Breve historia de Java

Bill Joy, uno de los ejecutivos de Sun Microsystems cuando la empresa creó Java, denominó al lenguaje "el resultado final de 15 años de trabajo en busca de una forma más fiable y mejorada de crear programas informáticos". La creación de Java fue más complicada.

James Gosling desarrolló Java en 1990 como lenguaje para aparatos inteligentes (televisores interactivos, hornos omniscientes, satélites militares SkyNet para dominar a la humanidad, etc.). Gosling no estaba convencido con los resultados obtenidos con el lenguaje C++. En un arrebato de inspiración, se encerró en su oficina y creó un nuevo lenguaje más adecuado a sus necesidades.

Gosling bautizó al nuevo lenguaje Oak (Roble), por el árbol que veía a través de su ventana. El lenguaje formaba parte de la estrategia de su empresa para hacerse ricos cuando la televisión interactiva se convirtiera en un sector multimillonario. Eso todavía no ha ocurrido (aunque Netflix, TiVo y otros lo intenten), pero pasó algo totalmente diferente. Justo cuando Oak empezaba a tomar forma, la Web se hizo popular.

En una afortunada coincidencia, muchas de las cualidades del lenguaje de Gosling hacían que se pudiera adaptar a la Web. Su equipo diseñó una forma de ejecutar programas desde páginas Web y eligieron un nuevo nombre para esta nueva realidad: Java.

Aunque Java se podía usar para otras muchas cosas, la Web le proporcionó el escaparate que necesitaba. Cuando el lenguaje se hizo más conocido, había que estar aislado en una estación espacial para no oír hablar del mismo.

Ha habido ocho versiones principales del lenguaje:

- **Otoño de 1995:** Java 1.0, la versión original.
- **Primavera de 1997:** Java 1.1, una actualización con compatibilidad mejorada con interfaces gráficas de usuario.
- **Verano de 1998:** Java 2 versión 1.2; un importante ampliación que convirtió al lenguaje en un lenguaje de programación de propósito general.
- **Otoño de 2000:** Java 2 versión 1.3; versión para multimedia.
- **Primavera de 2002:** Java 2 versión 1.4; actualización de compatibilidad con Internet, funciones XML y procesamiento de texto.
- **Primavera de 2004:** Java 2 versión 5, una versión con mayor fiabilidad y conversión automática de datos.
- **Invierno de 2006:** Java 6; actualización con una base de datos integrada y compatibilidad con servicios Web.
- **Verano de 2011:** Java 7; la versión actual, con mejoras del lenguaje, administración de memoria y la interfaz gráfica de usuario Nimbus.

Nota

Seguramente haya oído hablar del significado de Java y también de la pasión de Gosling por el café. La historia del nombre de Java no oculta mensajes secretos ni declaraciones de amor líquido. Eligieron el nombre Java por la misma razón que al cómico Jerry Seinfeld le gusta decir la palabra salsa: porque suena bien.

Ir a clase con Java

La Web incluye numerosos recursos para educadores y estudiantes. Como los programas de Java pueden ofrecer una mayor experiencia interactiva que las páginas Web estándar, algunos programadores emplean el lenguaje para crear programas educativos para Internet.

Por ejemplo, en `http://www.cs.ubc.ca/~van/sssjava` puede acceder a un simulador de saltos de esquí creado por Michiel van de Panne, profesor de Informática en la University of British Columbia. El programa usa Java para demostrar animaciones basadas en la física de distintos saltos de esquí. El movimiento del esquiador se controla mediante el ratón en ocho direcciones, y cada una afecta al éxito del salto. En la figura 3.2 puede ver un ejemplo.

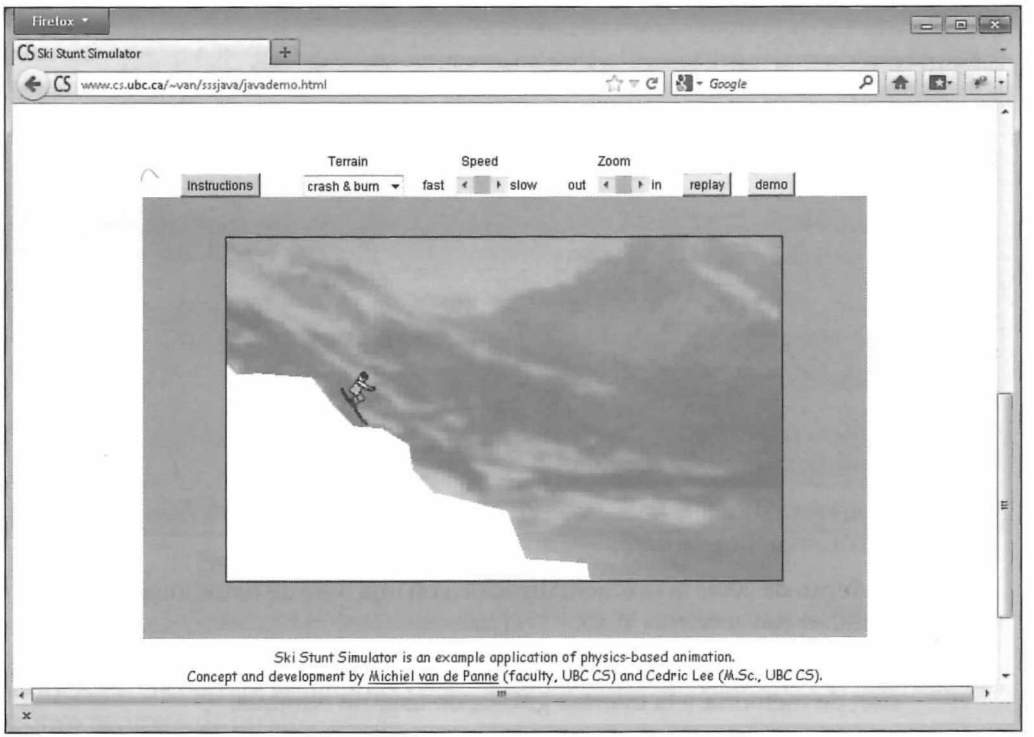


Figura 3.2. Un simulador de saltos de esquí que emplea un programa de Java.

Existen numerosos programas educativos para distintos sistemas operativos, pero éste destaca por su disponibilidad. El simulador se ejecuta directamente desde una página Web. No se necesita una instalación especial y, al contrario de lo que sucede con muchos programas de escritorio, no se limita a un sistema operativo concreto. Puede ejecutar programas de Java en cualquier equipo que cuente con una máquina virtual de Java (MVJ).

La MVJ que carga el navegador es la misma que usamos para ejecutar el programa `Saluton` del capítulo anterior. La MVJ del navegador solo puede ejecutar programas de Java configurados para su ejecución en páginas Web y no admite programas configurados para su ejecución en otra parte, como en una carpeta de archivos.

Los primeros navegadores compatibles con Java incluían una MVJ integrada. En la actualidad, los navegadores admiten Java a través del complemento Java, una MVJ que funciona como mejora del navegador.

Nota

Oracle incluye el complemento Java en el JDK y otros productos, de modo que puede que ya esté instalado en su ordenador. Para comprobarlo, visite `http://www.java.com/es/`. Por medio del enlace ¿Tengo Java? puede detectarlo.

No es necesario crear los programas de Java para un sistema operativo concreto. Como los sistemas operativos como Windows también se denominan plataformas, esta ventaja se denomina independencia de la plataforma. Java se creó para funcionar en múltiples sistemas. Originalmente, los desarrolladores de Java creyeron que tenía que ser multiplataforma ya que se emplearía en distintos aparatos y dispositivos electrónicos.

Los usuarios pueden ejecutar los programas que cree en Java desde diferentes sistemas sin necesidad de esfuerzos adicionales por su parte. En determinados casos, Java hace que no sea necesario crear versiones concretas de un programa para los distintos sistemas operativos y dispositivos.

Almuerzo en JavaWorld

Tras despertar el apetito esquiando, es hora de almorzar con JavaWorld, una revista *online* para programadores de Java (`www.javaworld.com`).

JavaWorld ofrece artículos prácticos, noticias y análisis de los temas más candentes sobre el desarrollo con Java. Una de las ventajas del formato Web de la publicación es que puede mostrar programas funcionales de Java junto a los artículos. En la figura 3.3 puede ver un ejemplo.

Nota

En ocasiones, JavaWorld cambia la ubicación de sus materiales, pero al cierre de esta edición, el ejemplo del tablón magnético se podía encontrar en `www.cadenhead.org/poetry`. Si la página no está disponible, introduzca **poetry** en el motor de búsqueda del sitio.

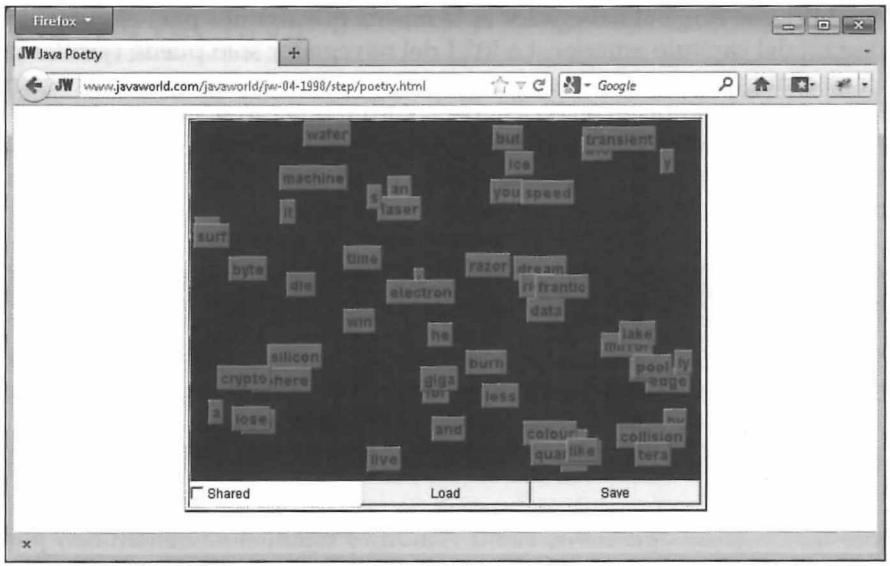


Figura 3.3. Artículo de JavaWorld para crear un mural magnético, con un ejemplo funcional del programa.

JavaWorld publica artículos y comentarios sobre el lenguaje y su desarrollo. Un tema muy debatido desde la aparición de Java es la seguridad del lenguaje.

La seguridad es importante debido al funcionamiento de los programas de Java al añadirlos a una página Web. Los programas descritos en este capítulo se descargan a su ordenador. Cuando terminan de descargarse, se ejecutan.

A menos que tenga muchos amigos, la mayoría de páginas Web que visita son publicaciones de desconocidos. En términos de seguridad, ejecutar sus programas no es muy diferente a que cualquiera pueda usar su ordenador. Si el lenguaje Java no contara con medidas para evitar los abusos, sus programas podrían añadir virus a un sistema, borrar archivos, reproducir las obras completas de Justin Bieber y otras cosas inimaginables. Java incluye diversos tipos de seguridad para garantizar que sus programas son seguros al ejecutarlos desde páginas Web. La seguridad principal se obtiene a través de restricciones sobre los programas ejecutados en la Web:

- Los programas no pueden abrir, leer, escribir ni eliminar archivos del sistema del usuario.
- Los programas no pueden ejecutar otros programas en el sistema del usuario.
- Todas las ventanas creadas por el programa se identifican claramente como ventanas de Java.
- Los programas no pueden conectarse a sitios Web que no sean de los que provienen.
- Se comprueba que todos los programas no modifican nada una vez compilados.

Aunque no existan garantías, el lenguaje ha demostrado contar con la seguridad adecuada para poder utilizarse en la Web.

El lenguaje Java también ofrece una política de seguridad más flexible para los programas ejecutados en un navegador. Puede designar a determinadas empresas o programadores como desarrolladores de confianza, lo que permite ejecutar sus programas de Java en su navegador sin las restricciones habituales.

Este sistema de confianza se establece a través del uso de los applet con firmas digitales, archivos que identifican al autor del programa. Estas firmas se crean en colaboración con grupos de verificación independientes como VeriSign.

Si alguna vez ha autorizado la ejecución de un programa en un navegador como Internet Explorer o Google Chrome, habrá trabajado con un sistema similar de confianza y verificación de identidad.

Los applet siguen siendo útiles, pero con el tiempo se han utilizado otras tecnologías como Flash, Silverlight y HTML5 para programas basados en la Web. Java es más habitual entre aplicaciones móviles, programas de servidor y software de escritorio.

Observar los cielos en la NASA

La primera parada en nuestro viaje por Java es la NASA, una agencia gubernamental estadounidense que emplea Java. Uno de los ejemplos más conocidos es SkyWatch, un applet que permite observar satélites en órbita. Puede abrirlo en su navegador desde `www.cadenhead.org/nasa`; accederá directamente al sitio SkyWatch de la NASA.

SkyWatch superpone la ubicación actual y el recorrido de ocho satélites diferentes, que puede añadir o excluir de la vista, sobre la esfera terrestre. En la figura 3.4 puede ver el applet en funcionamiento.

El applet actualiza la posición de cada satélite al ejecutarse. Esta acción en tiempo real es posible gracias al carácter de multiprocesamiento de Java, el cual permite al ordenador hacer varias cosas al mismo tiempo. Una parte del programa se encarga de una tarea y otra parte de otra diferente, ambas independientes. En este ejemplo, cada parte del programa se denomina proceso.

En un programa como SkyWatch, cada satélite podría ejecutarse en su propio proceso. Si usa un sistema operativo como Windows 7, empleará este comportamiento al ejecutar varios programas a la vez. Si está en la oficina jugando al Solitario en otra ventana mientras calcula un informe de ventas en otra y hablando por teléfono a la vez, enhorabuena, eso es el multiprocesamiento.

Negocios

Llegados a este punto del viaje, puede que tenga la impresión de que Java es básicamente para fanáticos del espacio, poetas de tercera y aspirantes a esquiador. La siguiente parada de nuestro viaje nos muestra la aplicación empresarial de Java.

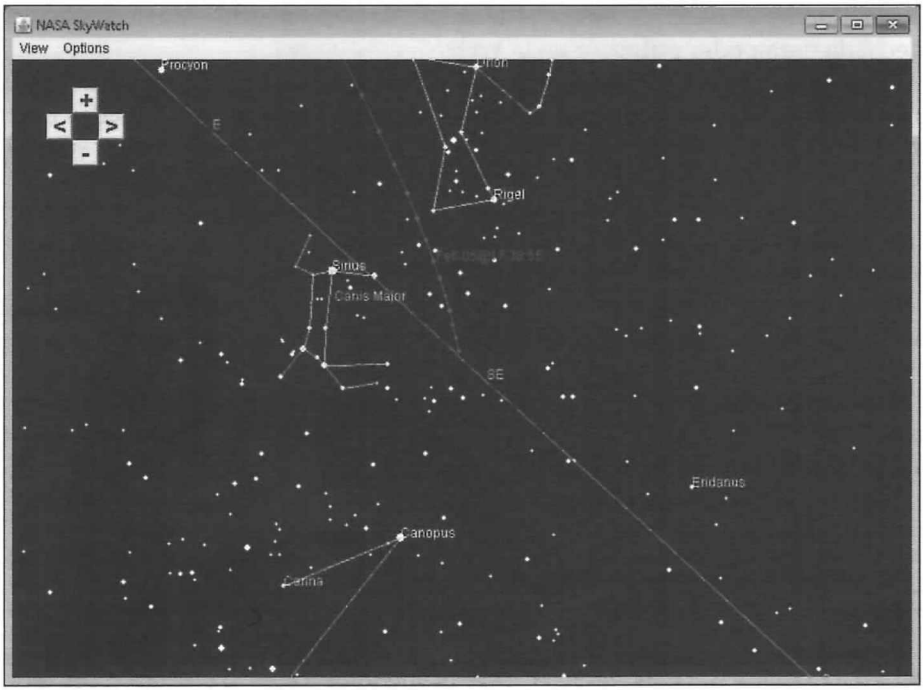


Figura 3.4. El applet SkyWatch de la NASA controla la ubicación y el recorrido de los satélites en órbita.

Diríjase al sitio Web de JTicker, en `www.jticker.com`.

El editor de JTicker, la empresa Stock Applets, desarrolla programas de Java que muestran titulares de noticias financieras y la cotización de la bolsa para usar en otros sitios Web (véase la figura 3.5).

Al contrario de lo que sucede con otros programas de análisis bursátil que requieren la instalación de software en los equipos de todos los empleados, el uso de Java permite a los clientes de Stock Applets disponer de los programas desde cualquier navegador Web. Basta con acceder al sitio Web de la empresa.

Puede pensar en este applet de varias formas. Puede pensar que es un objeto, algo que existe en el mundo, que ocupa un espacio y que realiza ciertas funciones. La programación orientada a objetos (POO), empleada por Java (como veremos en el capítulo 10), es una forma de crear programas como grupos de objetos. Cada objeto se encarga de una tarea concreta y se comunica con el resto. Por ejemplo, un programa de información bursátil podría configurarse en el siguiente grupo de objetos:

- Un objeto `quote`, que represente un acción concreta.
- Un objeto `portfolio`, que contenga una serie de acciones concretas.
- Un objeto `ticker`, que muestre un objeto `portfolio`.
- Un objeto `Internet`, un objeto `user` y muchos otros.

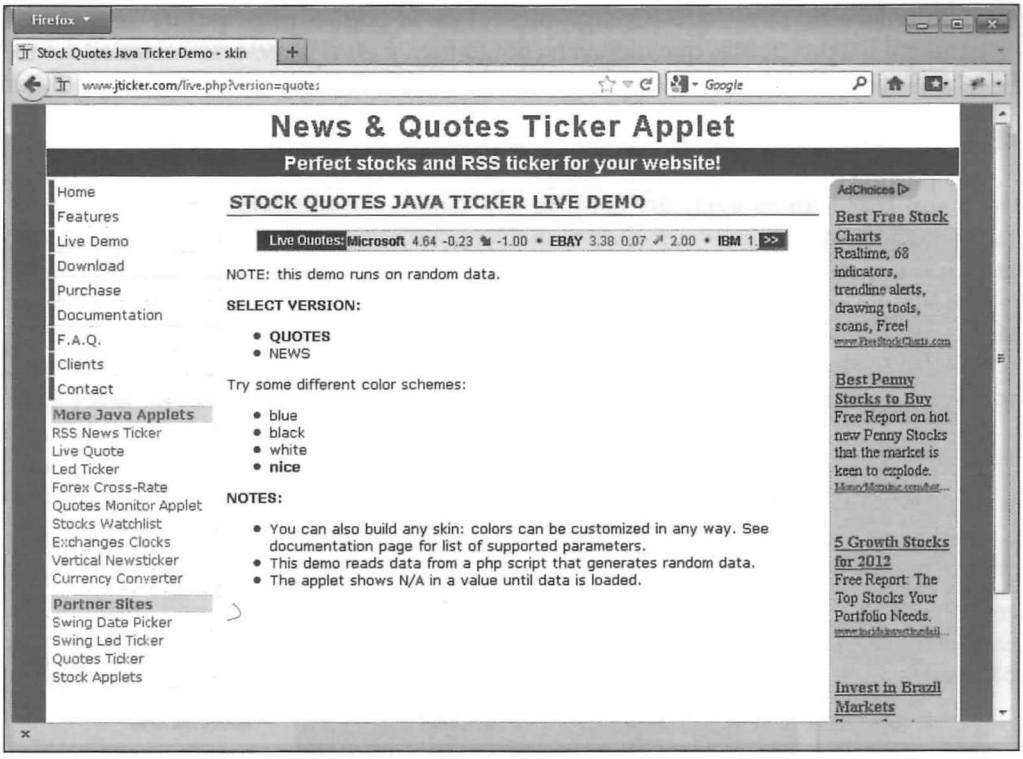


Figura 3.5. Los programas Java de Stock Applets informan de la cotización bursátil.

En este modelo, el software de información bursátil es una colección de los objetos necesarios para realizar el trabajo.

POO es una potente forma de crear programas y aumenta la utilidad de sus creaciones. Piense en el software bursátil. Si el programador desea usar las prestaciones del programa en otro software, el objeto de cotización podría emplearse en el nuevo programa, sin necesidad de cambios.

Detenerse en Java Boutique

Este recorrido mundial por programas de Java está dirigido por un profesional versado en los peligros de los viajes por la Web. Como pronto emprenderá sus propios viajes, es recomendable detenerse por una de las mejores guías para el turista que quiera verlo todo: Java Boutique en `http://javaboutique.internet.com`.

Java Boutique ofrece un directorio de programas de Java y recursos de programación relacionados con el lenguaje. Uno de los principales atractivos del sitio es ver los programas disponibles con código fuente. Si desconoce este término, el código fuente es otro nombre que reciben los archivos de texto usados para crear programas informáticos. El archivo `Saluton.java` del capítulo 2 es un ejemplo de código fuente.

El enlace Source Code (Código fuente) de la página principal de Java Boutique enumera los programas que incluyen código fuente en el directorio del sitio.

Uno de estos es Absolute de Aleksey Udovydchenko, un videojuego espacial en el que controla una nave a través de un campo de asteroides (véase la figura 3.6). El juego cuenta con desplazamientos animados, gráficos, control de teclado y sonido. Puede jugar una partida desde `http://javaboutique.internet.com/Absolute`.

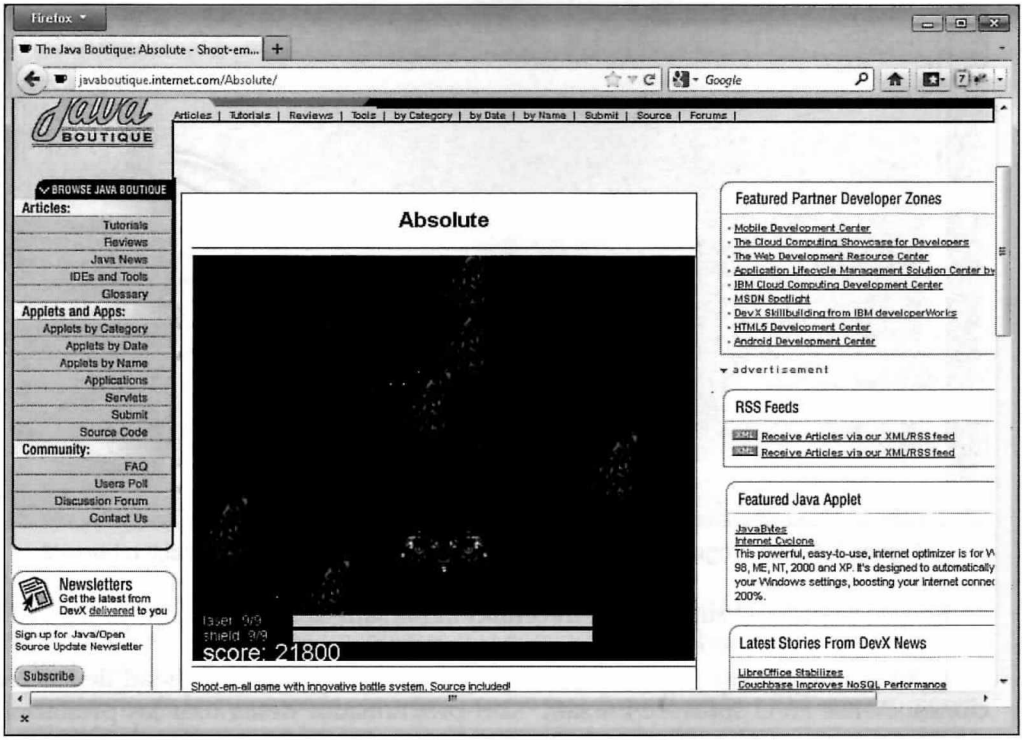


Figura 3.6. El código fuente de programas de Java, como el juego Absolute de Aleksey Udovydchenko, se puede encontrar con Java Boutique.

Nota

JARS (*Java Applet Ratings Service*, Servicio de clasificación de applet de Java) de Gamelan, un directorio de programas Java para navegadores y otros recursos, disponible en `www.jars.com`, suele incluir programas junto al código fuente empleado para crearlos. El lenguaje ha sido adoptado por miles de programadores en todo el mundo en parte debido a su simplicidad.

El programa Absolute se creó con apenas 700 líneas de código. Una cantidad reducida, si tenemos en cuenta todo lo que hace. Java incluye una completa biblioteca de clases que puede usar en sus programas. Udovydchenko emplea la clase `Image` para

mostrar gráficos como los asteroides y la clase `AudioClip` para reproducir sonidos de láser y explosiones. Un objetivo del diseño de Java era que fuera más fácil de aprender que C++, el lenguaje con el que Gosling tenia problemas en su primer proyecto. Gran parte de Java se basa en C++, por lo que los programadores que saben usarlo aprenderán Java con facilidad. Sin embargo, algunos elementos de C++ difíciles de aprender y de emplear no se incluyen en Java.

Para los que aprenden a programar por primera vez, Java es más fácil que C++. Algunos lenguajes se crean para que los programadores experimentados puedan dominar las prestaciones del ordenador en sus programas. Estos lenguajes incluyen trucos y otras funciones que los veteranos comprenden con facilidad.

Java no usa estas opciones y prefiere que el lenguaje sea tan sencillo como permite la programación orientada a objetos. Java se creó para ser fácil de aprender, de depurar y de emplear. Incluye numerosas mejoras que lo convierten en un meritorio competidor frente a otros lenguajes.

Ejecutar Java en su teléfono

La última parada de este viaje es el teléfono Google Android más cercano. Todos los programas que se ejecutan en Android se han diseñado con Java. Estos programas móviles, que amplían la funcionalidad de los teléfonos, se denominan aplicaciones (o *apps*). Una de las más conocidas es Angry Birds (véase la figura 3.7).

Figura 3.7. Angry Birds y otras aplicaciones para Android se han creado con Java.

Puede aprender más sobre este juego en la dirección `www.angrybirds.com` (pero no lo haga. El juego impedirá que sea productivo lo que queda del día, la semana o del mes, en función de su odio hacia esos cerdos). Android es la parada final de este viaje por Java debido a la tremenda popularidad del lenguaje. Una vez que aprenda Java,

puede aplicar sus conocimientos en el desarrollo de sus propias aplicaciones con el SDK de Android, una herramienta de programación gratuita para Windows, MacOS y Linux. Existen más de 250.000 aplicaciones para teléfonos Android y otros dispositivos que ejecutan el sistema operativo móvil, como veremos en el capítulo 24.

Resumen

Terminadas las vacaciones, es hora de deshacer las maletas y prepararse para volver a la programación de Java. En los siguientes capítulos analizaremos las piezas básicas del lenguaje, aprenderemos a crear objetos propios para realizar tareas en programación orientada a objetos, diseñaremos interfaces graficas de usuario y mucho más.

A menos que haya dejado de leer el libro para jugar a Angry Birds.

Preguntas y respuestas

P: ¿Por qué los applet de Java ya no son tan populares?

R: Cuando apareció Java a mediados de los 90, la gran mayoría de los usuarios aprendió el lenguaje para crear los applet. Java era la única forma de crear programas interactivos para los navegadores Web.

Con los años, surgieron otras alternativas. Macromedia Flash, Microsoft Silverlight y el nuevo estándar de publicación Web HTML5 ofrecen formas de añadir programas a páginas Web. Los applet sufrieron por el tiempo de carga y la lenta compatibilidad con las nuevas versiones de Java por parte de los desarrolladores de navegadores. Apareció un complemento Java que podía ejecutar la versión actual de Java en los navegadores, pero cuando sucedió, Java ya había superado sus orígenes y se había convertido es un sofisticado lenguaje de programación de propósito general.

Ejercicios

Si su cabeza todavía no se ha tomado vacaciones, pruebe sus conocimientos por medio de las siguientes preguntas.

Preguntas

1. ¿De dónde recibe su nombre la programación orientada a objetos?

   A. Los programas se consideran un grupo de objetos que funcionan de forma conjunta.

B. Por ser muy difícil de dominar para los usuarios.

C. Sus padres la bautizaron así.

2. De lo siguiente; ¿qué no forma parte de la seguridad de Java?

   A. Los programas Web no pueden ejecutar programas en el ordenador del usuario.

   B. La identidad del creador del programa siempre se verifica.

   C. Las ventanas de Java se etiquetan como ventanas de Java.

3. ¿Cómo se denomina la capacidad de un programa para procesar más de una tarea?

   A. Esquizofrenia.

   B. Multiculturalismo.

   C. Multiprocesamiento.

Respuestas

1. A. También se abrevia como POO.
2. B. En Java, los programadores pueden usar firmas digitales y una empresa de verificación de identidad como VeriSign, pero no es obligatorio.
3. C. También se denomina multitarea pero el término multiprocesamiento se emplea en Java ya que una parte en ejecución de un programa se denomina proceso.

Actividades

Antes de deshacer las maletas, puede aumentar sus conocimientos sobre los temas descritos en este capítulo si realiza las siguientes actividades:

- Use el sitio Java Boutique (`http://javaboutique.internet.com`) para buscar juegos de cartas desarrollados con Java.
- Visite el sitio Web de Oracle para usuarios de Java (`http://www.java.com/es/`) y haga clic en el enlace ¿Tengo Java?. Siga las instrucciones para comprobar si tiene Java en su ordenador. Descargue e instale la versión más actualizada.

4
Funcionamiento de los programas de Java

Una importante distinción de la programación con Java es dónde ejecutar su programa. Algunos programas se orientan al ordenador y otros para ejecutarse como parte de una página Web.

Los programas de Java que se ejecutan localmente en su ordenador se denominan aplicaciones. Los que se ejecutan en páginas Web se llaman applet. En este capítulo veremos la importancia de esta distinción.

Crear una aplicación

El programa `Saluton` creado en el capítulo 2 es un ejemplo de aplicación de Java. La siguiente que crearemos calcula la raíz cuadrada de un número y muestra el valor.

Con el proyecto `Java24` abierto en NetBeans, inicie una nueva aplicación:

1. Seleccione Archivo>Archivo Nuevo. Se abrirá el asistente Archivo Nuevo.
2. Seleccione la categoría Java y el tipo de archivo Empty Java File. Pulse **Siguiente**.
3. Introduzca el nombre de clase **Root** y pulse **Terminar**.

NetBeans crea `Root.java` y abre el archivo vacío en el editor de código. Introduzca el código del listado 4.1, sin los números de línea y los dos puntos del margen izquierdo. Los números permiten describir más fácilmente partes del programa en el libro. Cuando termine, pulse el botón **Guardar todo** de la barra de herramientas para guardar el archivo.

Listado 4.1. Texto completo de Root.java.

```
 1: class Root {
 2:   public static void main(String[] arguments) {
 3:     int number = 225;
 4:     System.out.println("The square root of "
 5:       + number
 6:       + " is "
 7:       + Math.sqrt(number)
 8:     );
 9:   }
10: }
```

La aplicación `Root` realiza las siguientes tareas:

- **Línea 3:** El valor entero 225 se almacena en la variable `number`.
- **Líneas 4-8:** Se muestran este entero y su raíz cuadrada, La instrucción `Math.sqrt(number)` de la línea 7 muestra la raíz cuadrada.

Si ha introducido sin errores el listado 4.1, con todos los signos de puntuación y las mismas mayúsculas y minúsculas, puede ejecutar el archivo en NetBeans por medio de Ejecutar>Ejecutar archivo. El resultado del programa se muestra en el panel Salida (véase la figura 4.1).

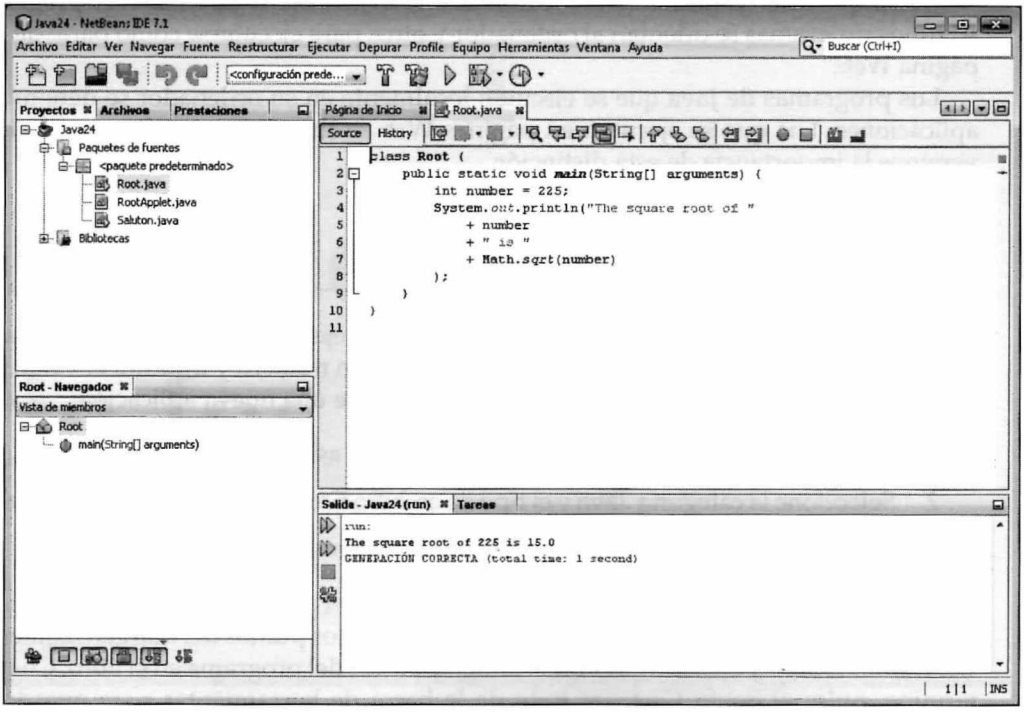


Figura 4.1. El resultado de la aplicación Root.

Al ejecutar una aplicación de Java, la MVJ busca un bloque `main()` y comienza a procesar las instrucciones de Java de ese bloque. Si el programa carece de bloque `main()`, la MVJ responde con un error.

Enviar argumentos a aplicaciones

Puede ejecutar aplicaciones de Java desde la línea de comandos por medio de `java`, un programa que invoca la MVJ. NetBeans usa este programa entre bastidores al ejecutar sus programas. Al ejecutar un programa de Java como comando, la MVJ carga la aplicación. El comando puede incluir información adicional, como en este ejemplo:

```
java TextDisplayer readme.txt /p
```

La información adicional enviada a un programa se denomina argumento. El primer argumento, si existe, se incluye un espacio por detrás del nombre de la aplicación. Los argumentos adicionales se separan mediante un espacio. En el ejemplo anterior, los argumentos son `readme.txt` y `/p`.

Si desea incluir un espacio dentro de un argumento, debe incluirlo entre comillas:

```
java TextDisplayer readme.txt /p "Page Title"
```

Este ejemplo ejecuta el programa `TextDisplayer` con tres argumentos: `readme.txt`, `/p` y `"Page Title"`. Las comillas evitan que `Page` y `Title` se procesen como argumentos independientes.

Puede enviar todos los argumentos que desee a una aplicación de Java (dentro de unos límites razonables). Para utilizarlos, debe crear instrucciones que los procesen.

Para ver cómo funcionan los argumentos en una aplicación, cree una nueva clase en el proyecto `Java24`:

1. Seleccione Archivo>Archivo Nuevo.
2. En el asistente Archivo Nuevo, seleccione la categoría Java y el tipo de archivo Empty Java File.
3. Asigne el nombre **BlankFiller** a la clase y pulse **Terminar**.

Introduzca el texto del listado 4.2 en el editor de código y guárdelo cuando termine. Compile el programa y corrija los errores identificados en el editor.

Listado 4.2. Texto completo de BlankFiller.java.

```
1: class BlankFiller {
2:   public static void main(String[] arguments) {
3:     System.out.println("The " + arguments[0]
4:       + " " + arguments[1] + " fox "
5:       + "jumped over the "
6:       + arguments[2] + " dog."
7:     );
8:   }
9: }
```

Esta aplicación se compila correctamente y se puede ejecutar, pero si lo intenta por medio de Ejecutar>Ejecutar archivo, obtendrá un extraño error:

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 0
   at BlankFiller.main(BlankFiller.java:3)
```

Este error se produce porque el programa espera recibir tres argumentos al ejecutarse Puede especificarlos si personaliza el proyecto en NetBeans:

1. Seleccione Ejecutar>Establecer la configuración del proyecto>Personalizar. Se abrirá el cuadro de diálogo Propiedades del proyecto.
2. Introduzca **BlankFiller** en el campo de texto Main Class (Clase principal).
3. En el campo Argumentos, introduzca **retromingent purple lactose-intolerant** y pulse **Aceptar**.

Como ha personalizado el proyecto, debe ejecutarlo de forma diferente. Seleccione Ejecutar>Ejecutar Proyecto Principal. La aplicación emplea los argumentos especificados como adjetivos para completar la frase y genera el siguiente resultado:

```
The retromingent purple fox jumped over the lactose-intolerant dog.
```

Vuelva al cuadro de diálogo Propiedades del proyecto y designe como argumentos tres adjetivos que elija, recordando incluir al menos tres en todos los casos. Los argumentos son una forma sencilla de personalizar el comportamiento de un programa. Se almacenan en un tipo de variable denominada matriz, como veremos en el capítulo 9.

Crear un applet

Cuando apareció el lenguaje Java, su característica más atractiva fueron los applet, programas de Java que se ejecutan en páginas Web. Puede ejecutarlos en cualquier navegador Web que admita programas de Java y probarlos con `appletviewer`, una herramienta incluida en el JDK y compatible con NetBeans.

La estructura de los applet es diferente a la de las aplicaciones. Al contrario que éstas, los applet carecen de un bloque `main()`. En su lugar tienen diversas secciones que se procesan en función de lo que suceda en el applet. Dos de estas secciones son `init()` y el bloque `paint()`. `init()` es la abreviatura de inicialización y se usa para configurar los elementos necesarios para la primera ejecución del applet. El bloque `paint()` se emplea para mostrar los elementos necesarios. Para ver una versión de applet de la aplicación `Root`, cree un nuevo archivo vacío de Java con el nombre de clase `RootApplet`. Introduzca el código del listado 4.3 y guárdelo cuando termine.

Listado 4.3. Texto completo de RootApplet.java.

```
1: import java.awt.*;
2:
3: public class RootApplet extends javax.swing.JApplet {
4:    int number;
```

```
 5:
 6:    public void init() {
 7:       number = 225;
 8:    }
 9:
10:    public void paint(Graphics screen) {
11:       Graphics2D screen2D = (Graphics2D) screen;
12:       screen2D.drawString("The square root of " +
13:          number +
14:          " is " +
15:          Math.sqrt(number), 5, 50);
16:    }
17: }
```

Este programa contiene muchas de las instrucciones de la aplicación `Root`. La principal diferencia es su organización. El bloque `main()` se ha sustituido por un bloque `init()` y un bloque `paint()`.

Al ejecutar el programa en NetBeans (Ejecutar>Ejecutar archivo), el applet carga la herramienta `appletviewer` (véase la figura 4.2).

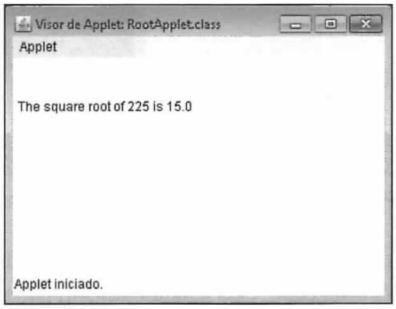


Figura 4.2. Applet RootApplet ejecutado en appletviewer.

Los applet son ligeramente más complicados que las aplicaciones ya que deben poder ejecutarse en una página Web y coexistir con otros elementos de la página en el navegador. Encontrará más información al respecto en el capítulo 17.

Nota

El programa de ejemplo de este capítulo simplemente presenta la estructura de los programas Java. El objetivo de este capítulo es que los programas se compilen y vea cómo funcionan al ejecutarlos. Algunos aspectos de los programas se describirán detalladamente en capítulos posteriores.

La herramienta `appletviewer` es muy útil para realizar pruebas, pero deja una impresión equivocada sobre los applet. No se ejecutan en sus propias ventanas como las aplicaciones de Java, sino que se añaden a las páginas Web como si fueran texto, fotos o gráficos. El applet se presenta conjuntamente al resto de la página.

En la figura 4.3 puede ver `RootApplet` en una página Web. La ventana del applet es el cuadro blanco que muestra el resultado del programa: la raíz cuadrada de 225. El título, los párrafos de texto y la foto de la bombilla son elementos estándar de una página Web.

Figura 4.3. RootApplet en una página Web.

Los applet de Java pueden ser estáticos como el resultado de este proyecto, pero es una forma de malgastar el lenguaje. Los applet suelen mostrar contenido dinámico como en un visor de información bursátil, un cliente de sala de chat o videojuegos.

Resumen

En este capítulo hemos creado una aplicación y un applet de Java. Estos dos tipos de programas tienen diferencias de funcionamiento y en la forma de crearse.

En los siguientes capítulos nos centraremos en las aplicaciones y ampliará su experiencia como programador de Java. Las aplicaciones son más rápidas de probar ya que no requieren la creación de una página Web para verlas; también suelen ser más fáciles de crear y más potentes.

Preguntas y respuestas

P: ¿Todos los argumentos enviados a una aplicación de Java deben ser cadenas?

R: Java almacena todos los argumentos como cadenas al ejecutar una aplicación. Si desea usar uno de estos argumentos como entero o como otro tipo que no sea una cadena, tendrá que convertirlo. Encontrará más información al respecto en el capítulo 11.

P: Si los applet se ejecutan en páginas Web y las aplicaciones en otros soportes, ¿qué son los programas ejecutados por Java Web Start?

R: Java Web Start es una forma de iniciar aplicaciones de Java desde un navegador Web. El usuario hace clic en un enlace de una página Web para ejecutar el programa, más fácil que descargarlo, ejecutar un asistente de instalación e iniciarlo como otro programa de escritorio. Aunque se ejecuten desde un navegador Web, los programas Java Web Start son aplicaciones, no applet. La aplicación siempre está actualizada ya que se recupera sobre le Web desde el proveedor del programa siempre que se ejecuta. Google Web Toolkit (GWT), una serie de herramientas de código abierto para programación Web, puede convertir un programa de Java a JavaScript, lo que hace que se ejecute más rápidamente y con mayor fiabilidad en navegadores Web sin necesidad de una máquina virtual de Java.

Ejercicios

Pruebe sus conocimientos sobre el material descrito en este capítulo respondiendo a las siguientes preguntas.

Preguntas

1. ¿Qué tipo de programa de Java se puede ejecutar desde un navegador?
   A. Applet.
   B. Aplicaciones.
   C. Ninguno.
2. ¿Qué significa MVJ?
   A. Memoria Verdad y Justicia.
   B. Maquillaje Verónica Jara.
   C. Máquina virtual de Java.
3. Si discute con alguien sobre la forma de enviar información a una aplicación de Java, ¿qué es lo que hace?
   A. Discutir sobre cadenas.
   B. Debatir sobre argumentos.
   C. Luchar por la funcionalidad.

Respuestas

1. A. Los applet se ejecutan como parte de una página Web, mientras que las aplicaciones se ejecutan en otros soportes.
2. A, B o C. Una pregunta con trampa. Las siglas equivalen a las tres respuestas, pero Máquina virtual de Java es lo que tiene que recordar para los próximos 24 capítulos.
3. B. Las aplicaciones reciben información en forma de argumentos. Vamos a llevarnos bien todos.

Actividades

Si desea aplicar sus conocimientos sobre applet y aplicaciones, pruebe con las siguientes actividades:

- Con la aplicación `Root` como guía, cree una nueva aplicación `NewRoot` que muestre la raíz cuadrada de 625.
- Con la aplicación `Root` como guía, cree una nueva aplicación `NewRoot` que muestre la raíz cuadrada de un número proporcionado como argumento.

5
Almacenar y modificar información en un programa

En el capítulo 2 usamos una variable, un almacén especial diseñado para guardar información. La información almacenada en variables puede cambiar durante la ejecución de un programa. Su primer programa guardaba una cadena de texto en una variable. Las cadenas son uno de los tipos de información que puede almacenar en variables; también puede guardar caracteres, enteros, números de coma flotante y objetos.

En este capítulo ampliaremos el uso de variables en sus programas de Java.

Instrucciones y expresiones

Los programas informáticos son una serie de instrucciones que indican al ordenador lo que debe hacer. El siguiente ejemplo de un programa de Java es una instrucción:

```
int highScore = 450000;
```

Puede usar llaves para agrupar una serie de instrucciones en un programa de Java. Estos grupos se denominan instrucciones de bloque. Fíjese en el siguiente fragmento de código:

```
1: public static void main(String[] args) {
2:    int a = 3;
3:    int b = 4;
4:    int c = 8 * 5;
5: }
```

Las líneas 2-4 forman una instrucción de bloque. La llave de apertura en la línea 1 indica el inicio del bloque y la llave de cierre de la línea 5, el final.

Algunas instrucciones se denominan expresiones ya que cuentan con una expresión matemática y generan un resultado. La línea 4 del ejemplo anterior es una expresión ya que establece el valor de la variable `c` en `8` multiplicado por `5`. En este capítulo trabajaremos con expresiones.

Asignar tipos de variables

Las variables son la principal forma que tiene el ordenador de recordar algo al ejecutar un programa. El programa `Saluton` del capítulo 2 usaba la variable `greeting` para almacenar `"Saluton mondo!"`. El ordenador tenía que recordar el texto para poder mostrar el mensaje.

En un programa de Java, las variables se crean con una instrucción que debe incluir dos elementos:

- El nombre de la variable.
- El tipo de información que almacenará la variable.

Las variables también pueden incluir el valor de la información almacenada.

Para ver los distintos tipos de variables y cómo se crean, ejecute NetBeans y cree un nuevo archivo vacío de Java con el nombre de clase `Variable`. Introduzca las siguientes líneas:

```
class Variable {
   public static void main(String[] args) {
      // Próximamente: variables
   }
}
```

Guarde estas líneas antes de realizar cambios.

Números enteros y de coma flotante

Hasta el momento, el programa `Variable` tiene un bloque `main()` con una sola instrucción: el comentario `// Próximamente: variables`. Elimínelo y añada:

```
int tops;
```

Esta instrucción crea la variable `tops`. No especifica su valor, de modo que es un espacio de almacenamiento vacío. El texto `int` al inicio de la instrucción designa `tops` como variable usada para almacenar números enteros. Puede emplear el tipo `int` para guardar la mayoría de números no decimales que necesite en sus programas informáticos. Puede almacenar cualquier entre comprendido entre -2.14 mil millones y 2.14 mil millones.

Cree una nueva línea tras la instrucción `tops` y añada la siguiente instrucción:

```
float gradePointAverage;
```

Esta instrucción crea la variable `gradePointAverage`. `float` equivale a números de coma flotante; las variables de coma flotante se usan para almacenar números con decimales.

El tipo de variable `float` contiene números decimales de hasta 38 cifras. El tipo `double` puede almacenar números decimales de hasta 300 cifras.

Nota

Puede usar una variable de coma flotante para almacenar una nota media como 2.25 (la mía en la University of North Texas; saludos, profesor Wells). También puede emplearla para almacenar un número como 0, las probabilidades de acceder a un master con mi nota media, motivo por el que estaba libre en 1996 cuando esta editorial buscaba autores de libros de informática.

Caracteres y cadenas

Como las variables vistas hasta ahora son numéricas, puede pensar que todas se usan para almacenar números. Piénselo bien. También puede recurrir a variables para guardar texto, en concreto de dos tipos: caracteres y cadenas. Un carácter es una letra, número, signo de puntuación o símbolo. Una cadena es un grupo de caracteres.

El siguiente paso es crear una variable `char` y otra `String`. Añada las dos siguientes instrucciones tras la línea `float gradePointAverage`:

```
char key = 'C';
String productName = "Larvets";
```

Al emplear valores de carácter en un programa, debe añadir comillas simples a ambos lados del valor de carácter asignado a una variable. Con los valores de cadena debe usar comillas dobles, ya que evitan que el carácter o la cadena se confundan con un nombre de variable u otra parte de una instrucción. Fíjese en la siguiente instrucción:

```
String productName = Larvets;
```

Esta instrucción parece que indica al ordenador que cree la variable de cadena `productName` con el valor `Larvets`. Sin embargo, como `Larvets` no está entre comillas, se le indica al ordenador que establezca el valor `productName` en el mismo valor que la variable `Larvets`.

Tras añadir las instrucciones `char` y `String`, su programa será similar al del listado 5.1. Realice los cambios necesarios y guarde el archivo.

Listado 5.1. El programa Variable.

```
1: class Variable {
2:   public static void main(String[] args) {
3:     int tops;
```

```
4:      float gradePointAverage;
5:      char key = 'C';
6:      String productName = "Larvets";
7:   }
8: }
```

Las dos últimas variables del programa usan = para asignar un valor inicial al crear las variables. Puede emplear esta opción para las variables que cree en sus programas de Java, como veremos más adelante.

Puede ejecutar este programa aunque no genera resultado alguno.

Nota

Aunque los demás tipos de variable son letras minúsculas (`int`, `float` y `char`), es necesario una mayúscula en `String` al crear variables de cadena. En un programa de Java, una cadena es distinta a los demás tipos de información usados en variables, como veremos en el siguiente capítulo.

Otros tipos de variables numéricas

Los tipos de variables presentadas hasta ahora son los principales que usará en la mayoría de sus programas. Puede recurrir a otros tipos en circunstancias especiales.

El primero, `byte`, se usa con números enteros comprendidos entre -128 y 127. La siguiente instrucción crea la variable `escapeKey` con el valor inicial `27`:

```
byte escapeKey = 27;
```

El segundo, `short`, se puede emplear con enteros de menor tamaño que `int`. Un entero `short` está comprendido entre -32,768 y 32,767, como en el siguiente ejemplo:

```
short roomNumber = 222;
```

El último tipo de variable numérica, `long`, suele usarse con enteros demasiado elevados para el tipo `int`. Un entero `long` puede estar comprendido entre -9.22 y 9.22 quintillones, lo suficientemente elevado para abarcarlo todo menos el gasto del gobierno. Al trabajar con números de gran tamaño en Java, puede resultar complicado ver el valor del número a simple vista, como en esta instrucción:

```
long salary = 264400000;
```

A menos que cuente los ceros, seguramente no vea que corresponde a 264.4 millones de dólares. Java 7 permite organizar números elevados por medio de guiones bajos (_). Veamos un ejemplo:

```
long salary = 264_400_000;
```

Los guiones bajos se ignoran, de modo que la variable sigue teniendo el mismo valor. Es una forma de facilitar la legibilidad de los números.

Advertencia

Todas las mejoras de Java 7, incluidos los guiones bajos en números, aparecen como errores en el editor de código fuente de NetBeans a menos que haya configurado el IDE para reconocer Java 7. Encontrará más información al respecto en el capítulo 7.

El tipo de variable boolean

Java cuenta con el tipo de variable `boolean` que solo puede emplearse para almacenar los valores `true` o `false`. A primera vista, una variable `boolean` no parece especialmente útil a menos que desee crear preguntas `true/false`. Sin embargo, estas variables se usarán en distintas situaciones. En las siguientes preguntas se pueden emplear variables `boolean` como respuesta:

- ¿El usuario ha pulsado una tecla?
- ¿Ha terminado la partida?
- ¿Estoy en números rojos en mi cuenta?
- ¿Estos pantalones me hacen más gordo?

La siguiente instrucción crea la variable `boolean gameOver`:

```
boolean gameOver = false;
```

El valor inicial de esta variable es `false`, de modo que esta instrucción podría indicar en un juego que la partida no ha terminado. Después, si sucede algo para finalizar la partida, la variable `gameOver` se puede establecer en `true`.

Aunque los dos posibles valores `boolean` parezcan cadenas, no debe incluirlos entre comillas. En el capítulo 7 encontrará más información sobre las variables `boolean`.

Nota

Los números Booleanos toman su nombre de George Boole (1815-1864), un matemático autodidacta que inventó el álgebra Booleana, que se ha convertido en una parte fundamental de la programación informática, la electrónica digital y la lógica. Imaginamos que de niño se le dieron muy bien los cuestionarios de verdadero/falso.

Asignar nombres a sus variables

En Java, los nombres de variables pueden comenzar por una letra, un guión bajo o el símbolo del dólar (`$`). El resto del nombre puede ser cualquier combinación de letras y números. Puede asignar los nombres que desee a sus variables pero debe intentar

mantener la coherencia. En este apartado describiremos el método de asignación de nombres recomendado para usar con variables. Java distingue entre mayúsculas y minúsculas en los nombres de variables, de modo que siempre debe emplear el mismo formato. Por ejemplo, si en un punto del programa se hace referencia a la variable `gameOver` como `GameOver`, se producirá un error de compilación.

El nombre de una variable debe describir su función. La primera letra siempre debe ser minúscula y si el nombre tiene más de una palabra, la primera letra de cada palabra debe ser mayúscula. Por ejemplo, si quiere crear una variable entera para almacenar la máxima puntuación de un juego, puede usar la siguiente instrucción:

```
int allTimeHighScore;
```

No puede emplear signos de puntuación ni espacios en el nombre de una variable, por lo que esto no es correcto:

```
int all-TimeHigh Score;
int all Time High Score;
```

Si usa estos nombres de variable en un programa, NetBeans identifica el error con el icono rojo de alerta en el editor de código.

Almacenar información en variables

Puede almacenar un valor en una variable al momento de crearla en un programa de Java. También puede añadir un valor a una variable después.

Para establecer un valor inicial para una variable al crearla, use el signo igual (=). El siguiente ejemplo crea la variable de coma flotante `pi` con el valor inicial `3.14`:

```
double pi = 3.14;
```

Todas las variables que almacenan números pueden configurarse de esta forma. Si crea una variable de carácter o de cadena, debe incluir el valor entre comillas, como indicamos antes.

También puede establecer una variable en el valor de otra si ambas son del mismo tipo. Por ejemplo:

```
int mileage = 300;
int totalMileage = mileage;
```

Primero, se crea la variable entera `mileage` con el valor inicial `300`. Tras ello, se crea la variable entera `totalMileage` con el mismo valor que `mileage`. Ambas tienen el valor inicial `300`. En capítulos posteriores convertiremos el valor de una variable al tipo de otra.

Como hemos visto, Java cuenta con variables numéricas similares para almacenar valores de distintos tamaños. Tanto `int` como `long` guardan enteros pero `long` almacena un intervalo mayor de posibles valores. Tanto `float` como `double` admiten números de coma flotante, pero `double` es de mayor tamaño.

Puede añadir una letra a un valor numérico para indicar el tipo del valor, como en esta instrucción:

```
float pi = 3.14F;
```

La `F` tras el valor `3.14` indica que es un valor `float`. Si omite la letra, Java asume que `3.14` es un valor `double`. La letra `L` se usa para enteros `long` y `D` para valores de coma flotante.

Otra convención en Java es usar mayúsculas con los nombres de variables que no cambian de valor. Estas variables se denominan constantes. El siguiente código crea tres constantes:

```
final int TOUCHDOWN = 6;
final int FIELDGOAL = 3;
final int PAT = 1;
```

Como las constantes no cambian de valor, se preguntará para qué sirven, si bastaría con emplear su valor. Una ventaja de usar constantes es que facilitan la compresión de los programas.

En las tres instrucciones anteriores, el nombre de la constante aparece en mayúsculas. No es obligatorio en Java, pero se ha convertido en un estándar entre los programadores para distinguir las constantes de otras variables.

Advertencia

Si no asigna a su variable un valor inicial, tendrá que hacerlo antes de poder emplearla en otra instrucción. En caso contrario, al compilar el programa, se genera un error indicando que la variable puede no haberse inicializado.

Operadores

Las instrucciones pueden usar expresiones matemáticas por medio de los operadores `+`, `-`, `*`, `/` y `%`. Estos operadores se usan en cálculos numéricos en sus programas de Java. En Java, una expresión de suma usa el operador `+`, como en estas instrucciones:

```
double weight = 205;
weight = weight + 10;
```

La segunda instrucción usa el operador + para establecer la variable `weight` en su valor actual más `10`. Las expresiones de resta usan el operador `-`:

```
weight = weight - 15;
```

Esta expresión establece la variable `weight` en su valor actual menos `15`. Las expresiones de división usan el signo `/`:

```
weight = weight / 3;
```

Establece la variable `weight` en el valor actual dividido por `3`. Para hallar el resto de una división, use el operador `%` (de módulo). La siguiente expresión calcula el resto de 245 entre 3:

```
int remainder = 245 % 3;
```

Las expresiones de multiplicación usan el signo `*`. La siguiente instrucción usa una expresión de multiplicación como parte de otra instrucción más compleja:

```
int total = 500 + (score * 12);
```

`score * 12` multiplica `score` por `12`. La instrucción completa multiplica `score` por `12` y suma `500` al resultado. Si `score` equivale a `20`, el resultado total es `740`: `500 + (20 * 12)`.

Incrementar y reducir un valor

Una tarea habitual en programación es cambiar el valor de una variable en una unidad. Puede aumentarlo, operación denominada de incremento, o reducirlo, operación de decremento. Ambas cuentan con sus correspondientes operadores.

Para incrementar el valor de una variable en una unidad, use el operador `++`:

```
x++;
```

Esta instrucción suma uno al valor almacenado en la variable `x`.

Para reducir el valor de una variable en una unidad, use el operador `--`:

```
y--;
```

Esta instrucción reduce `y` en uno.

También puede situar los operadores de incremento y decremento por delante del nombre de la variable, como en las siguientes instrucciones:

```
++x;
--y;
```

El operador situado delante del nombre de una variable se denomina prefijo y, si se sitúa detrás, sufijo.

> **Nota**
>
> ¿Le resulta confuso? Es más fácil de lo que parece, si recuerda lo que aprendió en el colegio sobre prefijos. Al igual que un prefijo como sub- se añade delante de una palabra, un operador de prefijo se sitúa por delante del nombre de una variable. Los operadores de sufijo se sitúan al final.

La diferencia entre operadores de prefijo o sufijo es relevante al emplear los operadores de incremento o decremento en una expresión.

Fíjese en las siguientes instrucciones:

```
int x = 3;
int answer = x++ * 10;
```

¿A qué equivale la variable `answer` tras procesar estas instrucciones? Seguramente piense que a `40`, si `3` se incrementara en una unidad, lo que daría `4`, y después se multplicara por 10. Pero la respuesta es el valor `30`, ya que se ha utilizado el operador de sufijo en lugar del de prefijo.

Al usar un operador de sufijo en una variable dentro de una expresión, el valor de la variable no cambia hasta que la expresión se evalúa totalmente. La instrucción `int answer = x++ *` hace lo mismo en el mismo orden en las dos siguientes instrucciones:

```
int answer = x * 10;
x++;
```

En el caso de los operadores de prefijo sucede lo contrario. Si se usan en una variable dentro de una expresión, el valor de la variable cambia antes de evaluar la expresión.

Fíjese en estas dos instrucciones:

```
int x = 3;
int answer = ++x * 10;
```

Como resultado, la variable `answer` es igual a `40`. El operador de prefijo hace que el valor de la variable `x` cambie antes de evaluar la expresión. La instrucción `int answer = ++x * 10` hace lo mismo en el mismo orden que estas instrucciones:

```
x++;
int answer = x * 10;
```

Puede que los operadores ++ y -- le desesperen, ya que no son tan sencillos como otros de los conceptos del libro. Espero que si le digo que no es necesario emplear estos operadores en sus programas no incumpla ningún código secreto de los programadores de Java. Puede obtener los mismos resultados por medio de los operadores + y -:

```
x = x + 1;
y = y - 1;
```

El incremento y el decremento son atajos muy útiles, pero en ocasiones no pasa nada por tomar el camino más largo en una expresión.

Nota

En el capítulo 1 dijimos que el nombre del lenguaje de programación C++ era un chiste que explicaríamos después. Ahora que ya conoce el operador de incremento ++, tiene la información necesaria para saber por qué incluye dos signos en el nombre y no solo uno. Como C++ añade nuevas características y funciones al lenguaje C, se puede considerar un incremento de C, de ahí el nombre C++. Cuando haya completado los 24 capítulos del libro, también podrá contar chistes como esté, incomprensibles para más del 99 por 100 de la población.

Precedencia de operadores

Al usar una expresión con más de un operador, tendrá que saber el orden que usa el ordenador para procesar la expresión. Fíjese en las siguientes instrucciones:

```
int y = 10;
x = y * 3 + 5;
```

A menos que sepa qué orden usa el ordenador para resolverlas, no puede estar seguro del valor de la variable `x`. Podría ser `35` o `80`, en función de si `y * 3` se evalúa primero que `3 + 5`. El siguiente orden es el empleado al desarrollar una expresión:

1. Primero las operaciones de incremento y decremento.
2. Después, la multiplicación, división y módulo.
3. Seguidamente, sumas y restas.
4. Tras ello, las comparaciones.
5. El signo = se usa para establecer el valor de una variable.

Como la multiplicación tiene lugar antes que la suma, ahora puede determinar la respuesta del ejemplo anterior: primero se multiplica `y` por `3`, lo que genera `30`, y después se suma `5`. La variable `x` se establece en `35`.

Veremos las comparaciones en el capítulo 7. El resto ya lo hemos descrito en este capítulo, de modo que podrá determinar el resultado de las siguientes instrucciones:

```
int x = 5;
int number = x++ * 6 + 4 * 10 / 2;
```

Estas instrucciones establecen la variable numérica `equal` en `50`.

¿Cómo calcula el ordenador este total? Primero, se procesa el operador de incremento y `x++` establece el valor de la variable `x` en `6`. Sin embargo, el operador `++` aparece como sufijo tras `x` en la expresión, lo que significa que la expresión se evalúa con el valor original de `x`. Como se usa el valor original de `x` antes de incrementar la variable, la expresión es la siguiente:

```
int number = 5 * 6 + 4 * 10 / 2;
```

Ahora, la multiplicación y la división se calculan de izquierda a derecha. Primero se multiplica `5` por `6`, `4` por `10`, y el resultado se divide por `2` (`4 * 10 / 2`). La expresión es la siguiente:

```
int number = 30 + 20;
```

Como resultado, la variable `number` se establece en `50`.

Si desea evaluar la expresión en otro orden diferente, puede emplear paréntesis para agrupar las partes de la expresión que evaluar primero. Por ejemplo, la expresión `x = 5 * 3 + 2;` haría que `x` fuera `17` ya que la multiplicación se calcula antes que la suma. Pero fíjese en la expresión modificada:

```
x = 5 * (3 + 2);
```

En este caso, primero se calcula la expresión entre paréntesis, de modo que el resultado es `25`. Puede usar todos los paréntesis que desee en una expresión.

Emplear expresiones

En el colegio, cuando tenía que resolver un problema de matemáticas complejo, ¿se quejaba ante algún ser superior de que nunca usaría esos conocimientos en la vida real? Siento el inciso, pero sus profesores tenían razón, sus conocimientos matemáticos son muy útiles para la programación informática. Una mala noticia.

La buena es que el ordenador hace lo que le pidamos. Las expresiones suelen emplearse en programas informáticos para realizar tareas como las siguientes:

- Cambiar el valor de una variable.
- Contar el número de veces que algo sucede en un programa.
- Usar una fórmula matemática en un programa.

Al crear programas informáticos, tendrá que recurrir a sus conocimientos matemáticos para trabajar con expresiones. Las expresiones pueden emplear suma, resta, multiplicación, división y módulo.

Para ver una expresión en funcionamiento, vuelva a NetBeans y cree un nuevo archivo de Java con el nombre de clase `PlanetWeight`. Este programa controla la variación de peso de una persona mientras recorre otros planetas del sistema solar. Introduzca el texto del listado 5.2 en el editor de código. Analizaremos cada una de las partes del programa.

Listado 5.2. El programa PlanetWeight.

```
 1: class PlanetWeight {
 2:   public static void main(String[] args) {
 3:     System.out.print("Your weight on Earth is ");
 4:     double weight = 205;
 5:     System.out.println(weight);
 6:
 7:     System.out.print("Your weight on Mercury is ");
 8:     double mercury = weight * .378;
 9:     System.out.println(mercury);
10:
11:     System.out.print("Your weight on the Moon is ");
12:     double moon = weight * .166;
13:     System.out.println(moon);
14:
15:     System.out.print("Your weight on Jupiter is ");
16:     double jupiter = weight * 2.364;
17:     System.out.println(jupiter);
18:   }
19: }
```

Cuando termine, guarde el archivo. Se compilará automáticamente. Seleccione Ejecutar>Ejecutar archivo para ejecutar el archivo (véase la figura 5.1).

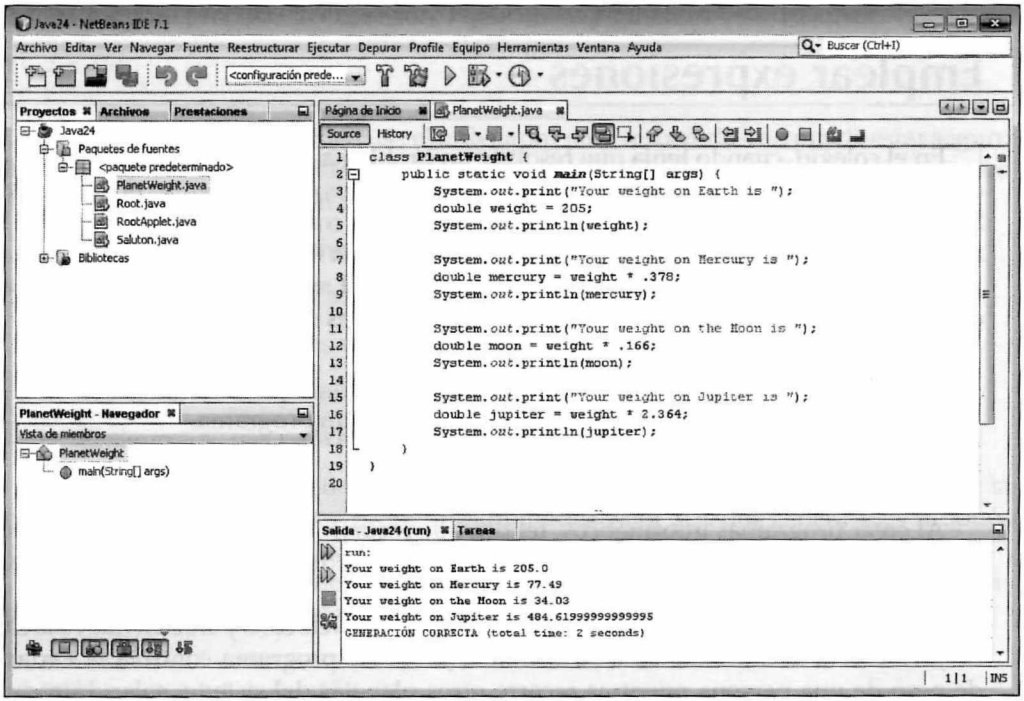


Figura 5.1. Resultado del programa PlanetWeight.

Como en otros programas, `PlanetWeight` usa un bloque `main()` para desempeñar su función. Esta instrucción se puede dividir en cuatro secciones:

1. **Líneas 3-5:** El peso de la persona se establece inicialmente en 205.
2. **Líneas 7-9:** Se calcula la pérdida de peso en Mercurio.
3. **Líneas 11-13:** Se calcula la pérdida de peso en la Luna.
4. **Líneas 15-17:** Se calcula el aumento de peso en Júpiter.

La línea 4 crea la variable `weight` como variable de tipo `double`. Se le asigna el valor inicial `205`, que se usa a lo largo del programa para controlar el peso de la persona.

La siguiente línea es similar a otras instrucciones del programa:

```
System.out.println(weight);
```

El comando `System.out.println()` muestra una cadena incluida entre sus paréntesis. En la línea 3, el comando `System.out.print()` muestra el texto `"Your weight on Earth is"`. El programa incluye varias instrucciones `System.out.print()` y `System.out.println()`. La diferencia entre ellas es que `print()` no inicia una nueva línea tras mostrar el texto, y `println()` sí.

Resumen

Una vez presentadas las variables y las expresiones, puede proporcionar diferentes instrucciones a su ordenador mediante un programa. Con los conocimientos desarrollados durante este capítulo, podrá crear programas que realicen las mismas tareas que una calculadora y resuelvan sofisticadas ecuaciones matemáticas con facilidad. También hemos visto que un viaje a la Luna es muy eficaz para perder peso.

Los números son uno de los tipos de información que se pueden almacenar en una variable. También puede guardar caracteres, cadenas de caracteres y valores especiales `true` o `false` denominados variables `boolean`. En el siguiente capítulo veremos las variables `String` y cómo se usan.

Preguntas y respuestas

P: ¿En un programa de Java una línea es lo mismo que una instrucción?

R: No. Los programas que cree en este libro muestran cada instrucción en una línea para facilitar su legibilidad, pero no es obligatorio. El compilador de Java no tiene en cuenta líneas, espacios ni otros formatos al compilar un programa. Simplemente necesita puntos y comas al final de cada instrucción. La siguiente línea es correcta:

```
int x = 12; x = x + 1;
```

Si incluye más de una instrucción por línea, el programa es más difícil de leer para los humanos, motivo por el que se desaconseja esta práctica.

P: ¿Por qué se usa la minúscula en la primera letra del nombre de una variable, como en `gameOver`?

R: Es una convención que mejora la programación de dos formas. Por un lado, hace que las variables destaquen entre otros elementos de un programa de Java. Por otra parte, al utilizar un sistema de nombres coherente para variables, se eliminan errores que pueden surgir al usar una variable en distintos puntos de un programa. El estilo usado en este libro lo emplean los programadores de Java desde hace años.

P: ¿Puedo especificar enteros como valores binarios en Java?

R: Desde Java 7 sí. Añada `0b` por delante del número y después los bits del valor. Como 1101 es el formato binario del número 13, la siguiente instrucción establece un entero en 13:

```
int z = 0b0000_1101;
```

El guión bajo simplemente mejora la legibilidad del número. El compilador lo ignora. NetBeans considera esta función como un error a menos que configure su proyecto para emplear Java 7, como veremos en el capítulo 7.

Ejercicios

Pruebe sus conocimientos sobre variables, expresiones y demás materiales descritos en este capítulo respondiendo a las siguientes preguntas.

Preguntas

1. ¿Qué nombre recibe un grupo de instrucciones incluidas entre llaves?
   A. Instrucción de bloque.
   B. Agrupación.
   C. Instrucciones entre llaves.
2. Una variable `boolean` se usa para almacenar valores `true` o `false`.
   A. Verdadero.
   B. Falso
   C. No, gracias. Ya he comido.
3. ¿Qué carácter no se puede usar para iniciar el nombre de una variable?
   A. El símbolo del dólar.
   B. Dos barras (//).
   C. Una letra.

Respuestas

1. A. Las instrucciones agrupadas se denominan bloque de instrucción o bloque.
2. A. `true` y `false` son las únicas respuestas que puede almacenar una variable `boolean`.
3. B. Las variables pueden empezar con una letra, el símbolo del dólar (`$`) o un guión bajo (_). Si inicia un nombre de variable con dos barras, el resto de la línea se ignora, por tratarse de un comentario.

Actividades

Puede aumentar sus conocimientos sobre los temas descritos en este capítulo si realiza las siguientes actividades:

- Amplíe el programa `PlanetWeight` para calcular el peso de una persona en Venus (90.7% del peso en la Tierra) y en Urano (88.9%), y no haga bromas con el nombre de este planeta.
- Cree un breve programa de Java que use un entero `x` y otro `y`, y muestre el resultado de `x` al cuadrado más `y` al cuadrado.

6
Usar cadenas para comunicarse

En la película *El Piano*, Holly Hunter interpreta a Ada, una joven escocesa muda que solo puede expresarse a través de su piano.

Al igual que Ada, sus programas son capaces de desempeñar sus funciones y no pararse a hablar con los humanos. Pero si algo nos enseña *El Piano* es que la comunicación está al mismo nivel que la comida, el agua y un lugar para vivir como necesidades esenciales (también nos enseña que el actor Harvey Keitel confía mucho en su cuerpo, pero eso sería el tema de otro libro).

Los programas de Java emplean las cadenas como medio principal para comunicarse con los usuarios. Las cadenas son colecciones de texto: letras, números, puntuación y otros caracteres. En este capítulo veremos cómo trabajar con cadenas en sus programas de Java.

Almacenar texto en cadenas

Las cadenas almacenan texto y lo muestran a los usuarios. El elemento más básico de una cadena es el carácter, que puede ser una letra, un número, un signo de puntuación u otro símbolo.

En los programas de Java, un carácter es uno de los tipos de información que se pueden guardar en una variable. Las variables de carácter se crean con el tipo `char` en una instrucción como la siguiente:

```
char keyPressed;
```

Esta instrucción crea la variable `keyPressed`, que puede almacenar un carácter. Al crear variables de carácter, puede configurarlas con un valor inicial:

```
char quitKey = '@';
```

El valor del carácter debe incluirse entre comillas simples. Una cadena es una colección de caracteres.

Puede crear una variable para almacenar un valor de cadena si usa `String` con el nombre de la variable:

```
String fullName = "Ada McGrath Stewart";
```

Esta instrucción crea la variable de cadena `fullName` con el texto `"Ada McGrath Stewart"`, el nombre completo del personaje de Hunter. Las cadenas se delimitan por comillas dobles, que no se incluyen en la propia cadena.

Al contrario de lo que sucede con otros tipos de variables, como `int`, `float`, `char`, `boolean` y demás, el nombre del tipo `String` empieza con mayúscula.

Las cadenas son un tipo especial de información denominada objeto y todos los objetos en Java emplean mayúsculas. Encontrará más información al respecto en el capítulo 10. Lo que debe recordar es que las cadenas son distintas a otras variables y por esa diferencia `String` usa mayúscula.

Mostrar cadenas en programas

La forma más básica de mostrar una cadena en un programa de Java consiste en emplear la instrucción `System.out.println()`. Esta instrucción acepta cadenas y otras variables entre los paréntesis y muestra sus valores en el dispositivo de salida del sistema: el monitor del ordenador.

Veamos un ejemplo:

```
System.out.println("Silence affects everyone in the end.");
```

Esta instrucción muestra el siguiente texto:

```
Silence affects everyone in the end.
```

La acción de mostrar un texto en pantalla se denomina imprimir, motivo por el que `println()` equivale a *print line* (imprimir línea). Puede usar `System.out.println()` para mostrar el texto entre comillas dobles y también variables, como veremos más adelante. Incluya entre los paréntesis el material que desee mostrar.

Otra forma de enseñar un texto consiste en invocar `System.out.print()`. Esta instrucción muestra cadenas y otras variables dentro de los paréntesis, pero al contrario de lo que sucede con `System.out.println()`, permite que instrucciones posteriores muestren el texto en la misma línea.

Puede emplear `System.out.print()` repetidamente para que aparezcan varios elementos en la misma línea, como en este ejemplo:

```
System.out.print("She ");
System.out.print("never ");
System.out.print("said ");
System.out.print("another ");
System.out.println("word.");
```

Estas instrucciones muestran el siguiente texto:

```
She never said another word.
```

Usar caracteres especiales en cadenas

Al crear o mostrar una cadena, su texto debe incluirse entre comillas dobles, las cuales no se muestran, lo que genera una duda: qué hacer para que aparezcan.

Para ello, Java ha creado un código especial que puede incluir en una cadena: \". Siempre que aparece este código en una cadena, se cambia por una comilla doble. En el siguiente ejemplo:

```
System.out.println("Jane Campion directed \"The Piano\" in 1993.");
```

El código se muestra de esta forma:

```
Jane Campion directed "The Piano" in 1993.
```

Puede añadir varios caracteres de especiales en una cadena de esta forma. La siguiente lista muestra los caracteres especiales, todos ellos precedidos de una barra invertida (\).

Caracteres especiales	Muestran
\'	Comilla simple
\"	Comilla doble
\\	Barra invertida
\t	Tabulación
\b	Retroceso
\r	Retorno del carro
\f	Nueva página
\n	Nueva línea

El carácter de nueva línea hace que el texto siguiente se muestre al inicio de la siguiente línea. Fíjese en este ejemplo:

```
System.out.println("Music by\nMichael Nyman");
```

La instrucción se mostraría de esta forma:

```
Music by
Michael Nyman
```

Pegar cadenas

Al emplear `System.out.println()` y trabajar con cadenas, puede pegar dos cadenas por medio de +, el mismo operador usado para sumar números.

El operador + tiene un sentido diferente al aplicarlo a cadenas. En lugar de realizar operaciones matemáticas, pega dos cadenas. Esta acción permite mostrar dos cadenas juntas o generar una cadena más extensa a partir de otras dos más breves. Para describir esta acción se emplea la palabra concatenación, que significa vincular dos elementos. La siguiente instrucción usa el operador + para mostrar una cadena extensa:

```
System.out.println("\"\'The Piano\' is as peculiar and haunting as any" +
   " film I've seen.\"\n\t- Roger Ebert, Chicago Sun-Times");
```

En lugar de incluir toda la cadena en la misma línea, lo que dificultaría la legibilidad del programa, se emplea el operador + para dividir el texto en dos líneas de código. Al mostrar esta instrucción, aparece de esta forma:

```
"'The Piano' is as peculiar and haunting as any film I've seen."
   - Roger Ebert, Chicago Sun-Times
```

En la cadena se usan varios caracteres especiales: `\"`, `\'`, `\n` y `\t`. Para familiarizarse con ellos, compare el resultado con la instrucción `System.out.println()` que lo ha creado.

Nota

Seguramente haya leído el término concatenación en otros libros de programación. Sin embargo, aquí empleamos pegar al hablar de combinación de cadenas. Pegar suena divertido. Concatenar parece algo que nunca debe hacerse si hay objetos inflamables cercanos.

Usar otras variables con cadenas

Aunque puede emplear el operador + para pegar dos cadenas, generalmente se usa para vincular cadenas y variables. Fíjese en este caso:

```
int length = 121;
char rating = 'R';
System.out.println("Running time: " + length + " minutes");
System.out.println("Rated " + rating);
```

Este código se muestra de esta forma:

```
Running time: 121 minutes
Rated R
```

El ejemplo ilustra una faceta exclusiva del funcionamiento del operador + con cadenas. Puede hacer que variables que no sean cadenas se procesen como cadenas al mostrarse. La variable `length` es un entero establecido en el valor `121`. Se muestra entre las cadenas `Running time:` y `minutes`. Se le pide a la instrucción `System.out.println()` que muestre una cadena más un entero más otra cadena. Funciona porque al menos una parte del grupo es una cadena. Java ofrece esta funcionalidad para facilitar la forma de mostrar información. También puede necesitar pegar algo repetidamente a una cadena, como en el siguiente ejemplo:

```
String searchKeywords = "";
searchKeywords = searchKeywords + "drama ";
searchKeywords = searchKeywords + "romance ";
searchKeywords = searchKeywords + "New Zealand";
```

Este código establece la variable `searchKeywords` en `"drama romance New Zealand"`. La primera línea crea la variable `searchKeywords` y la establece en una cadena vacía porque no hay nada entre las comillas dobles. La segunda línea establece `searchKeywords` en su cadena actual más la cadena `"drama"` añadida al final. Las dos siguientes líneas añaden `"romance"` y `"New Zealand"` de la misma forma.

Como puede apreciar, al pegar más texto al final de una variable, el nombre debe aparecer dos veces. Java ofrece un atajo para simplificar este proceso: el operador +=. Este operador combina las funciones de = y +. Con cadenas, se emplea para añadir algo al final de una cadena existente. El ejemplo `searchKeywords` se puede simplificar con +=, como muestran las siguientes instrucciones:

```
String searchKeywords = "";
searchKeywords += "drama ";
searchKeywords += "romance ";
searchKeywords += "New Zealand";
```

Este código genera el mismo resultado: `searchKeywords` se establece en `"drama romance New Zealand"`.

Procesamiento avanzado de cadenas

Existen otras formas de examinar una variable de cadena y cambiar su valor. Estas funciones avanzadas son posibles porque en Java las cadenas son objetos. Al trabajar con cadenas desarrollará habilidades que después usará con los objetos.

Comparar dos cadenas

En sus programas, tendrá que comprobar si una cadena es igual a otra. Para ello, puede emplear `equals()` en una instrucción con las dos cadenas, como en este ejemplo:

```
String favorite = "piano";
String guess = "ukulele";
System.out.println("Is Ada's favorite instrument a " + guess + "?");
System.out.println("Answer: " + favorite.equals(guess));
```

Este ejemplo usa dos variables de cadena diferentes. Una, `favorite`, almacena el nombre del instrumento favorito de Ada: el piano. La otra, `guess`, almacena cuál podría ser su favorito (en este caso, un ukulele). La tercera línea muestra el texto `"Is Ada's favorite instrument a"` seguido del valor de la variable `guess` y un signo de interrogación. La cuarta línea muestra el texto `"Answer:"` y después contiene algo nuevo:

```
favorite.equals(guess)
```

Esta parte de la instrucción emplea un método. Un método es una forma de realizar una tarea en un programa de Java. La tarea de este método es determinar si una cadena tiene el mismo valor que otra. Si las dos variables de cadena tienen el mismo valor, se muestra el texto `true`. En caso contrario, se muestra `false`. El resultado de este ejemplo es el siguiente:

```
Is Ada's favorite instrument a ukulele?
Answer: false
```

Determinar la longitud de una cadena

También puede necesitar determinar la longitud de una cadena en caracteres. Para ello cuenta con el método `length()`, el cual funciona igual que `equals()` pero solo con una variable de cadena. Fíjese en el siguiente ejemplo:

```
String cinematographer = "Stuart Dryburgh";
int nameLength = cinematographer.length();
```

Establece `nameLength`, una variable entera, en `15`. El método `cinematographer.length()` cuenta el número de caracteres de la variable de cadena `cinematographer` y lo almacena en la variable entera `nameLength`.

Cambiar las mayúsculas y minúsculas de una cadena

Como los ordenadores se toman todo literalmente, es habitual confundirlos. Aunque un humano reconocería que el texto Harvey Keitel y el texto HARVEY KEITEL se refieren a lo mismo, muchos ordenadores no estarían de acuerdo. El método `equals()` mencionado antes afirmaría sin lugar a dudas que `Harvey Keitel` no es igual a `HARVEY KEITEL`. Para solventar estos obstáculos, Java dispone de métodos para mostrar una variable de cadena en mayúsculas y minúsculas, `toUpperCase()` y `toLowerCase()`, respectivamente. El siguiente ejemplo muestra el método `toUpperCase()` en funcionamiento:

```
String baines = "Harvey Keitel";
String change = baines.toUpperCase();
```

Este código establece la variable de cadena `change` en la variable de cadena `baines` convertida a mayúsculas: `"HARVEY KEITEL"`. El método `toLowerCase()` funciona igual pero devuelve un valor de cadena en minúsculas.

El método `toUpperCase()` no cambia las mayúsculas y minúsculas de la variable de cadena en la que se invoca. En el ejemplo anterior, la variable `baines` sigue siendo `"Harvey Keitel"`.

Buscar una cadena

Otra tarea habitual al trabajar con cadenas es comprobar si una cadena aparece dentro de otra. Para buscar dentro de una cadena, use su método `indexOf()`. Incluya la cadena que busca entre los paréntesis. Si no se encuentra, `indexOf()` devuelve `-1`. Si se encuentra, `indexOf()` devuelve un entero que representa la posición en la que comienza la cadena. En una cadena, las posiciones se numeran de 0 en adelante, desde el primer carácter de la misma. En la cadena `"The Piano"`, el texto `"Piano"` comienza en la posición 4.

Un posible uso del método `indexOf()` sería buscar en el guión de *El Piano* la parte en la que el marido de Ada le dice a su hija Flora: "You are greatly shamed and you have shamed those trunks".

Si el guión completo de *El Piano* se almacenara en la cadena `script`, podría buscar esa frase con la siguiente instrucción:

```
int position = script.indexOf("you have shamed those trunks");
```

Si el texto se puede encontrar en la cadena `script`, `position` equivale a la posición en la que comienza el texto "you have shamed those trunks". En caso contrario, será `-1`.

Advertencia

El método `indexOf()` distingue entre mayúsculas y minúsculas, lo que significa que solo buscará el texto exacto que aparezca en la cadena de búsqueda. Si la cadena contiene el mismo texto pero con diferentes mayúsculas y minúsculas, `indexOf()` genera el valor `-1`.

Presentar títulos de crédito

En *El Piano*, Ada McGrath Stewart deambula en un territorio extraño cuando viaja de Escocia a Nueva Zelanda para casarse con un desconocido que no valora su música. Seguramente se haya sentido perdido con los conceptos presentados en este capítulo.

A continuación, para reforzar las funciones de procesamiento de cadenas, crearemos un programa de Java para mostrar los títulos de crédito de una película, que seguramente adivine cuál es.

Vuelva al proyecto `Java24` en NetBeans y cree una nueva clase de Java con el nombre `Credits`. Introduzca el texto del listado 6.1 en el editor de código y guarde el archivo cuando termine.

Listado 6.1. El programa Credits.

```
 1: class Credits {
 2:   public static void main(String[] args) {
 3:     // definir información de la película
 4:     String title = "The Piano";
 5:     int year = 1993;
 6:     String director = "Jane Campion";
 7:     String role1 = "Ada";
 8:     String actor1 = "Holly Hunter";
 9:     String role2 = "Baines";
10:     String actor2 = "Harvey Keitel";
11:     String role3 = "Stewart";
12:     String actor3 = "Sam Neill";
13:     String role4 = "Flora";
14:     String actor4 = "Anna Paquin";
15:     // mostrar información
16:     System.out.println(title + " (" + year + ")\n" +
17:       "A " + director + " film.\n\n" +
18:       role1 + "\t" + actor1 + "\n" +
19:       role2 + "\t" + actor2 + "\n" +
20:       role3 + "\t" + actor3 + "\n" +
21:       role4 + "\t" + actor4);
22:   }
23: }
```

Fíjese en el programa e intente determinar qué sucede en cada fase. Veamos su descripción:

- La línea 1 asigna el nombre `Credits` al programa.
- La línea 2 inicia el bloque `main()` en el que se desarrolla la actividad del programa.
- Las líneas 4-14 definen variables para almacenar información sobre la película, su director y los actores. Una de las variables, `year`, es un entero. El resto son variables de cadena.
- Las líneas 16-21 forman una extensa instrucción `System.out.println()`. Todo lo comprendido entre el primer paréntesis de la línea 16 y el último de la línea 21 se muestra en pantalla. El carácter de nueva línea (`\n`) hace que el texto posterior se muestre al inicio de una nueva línea. El carácter de tabulación (`\t`) añade tabulaciones al resultado. El resto es texto o variables de cadena para mostrar.
- La línea 22 finaliza el bloque `main()`.
- La línea 23 finaliza el programa.

Si encuentra mensajes de error, corríjalos y guarde el programa. NetBeans lo compila automáticamente. Al ejecutarlo, verá una ventana de resultado como la mostrada en la figura 6.1.

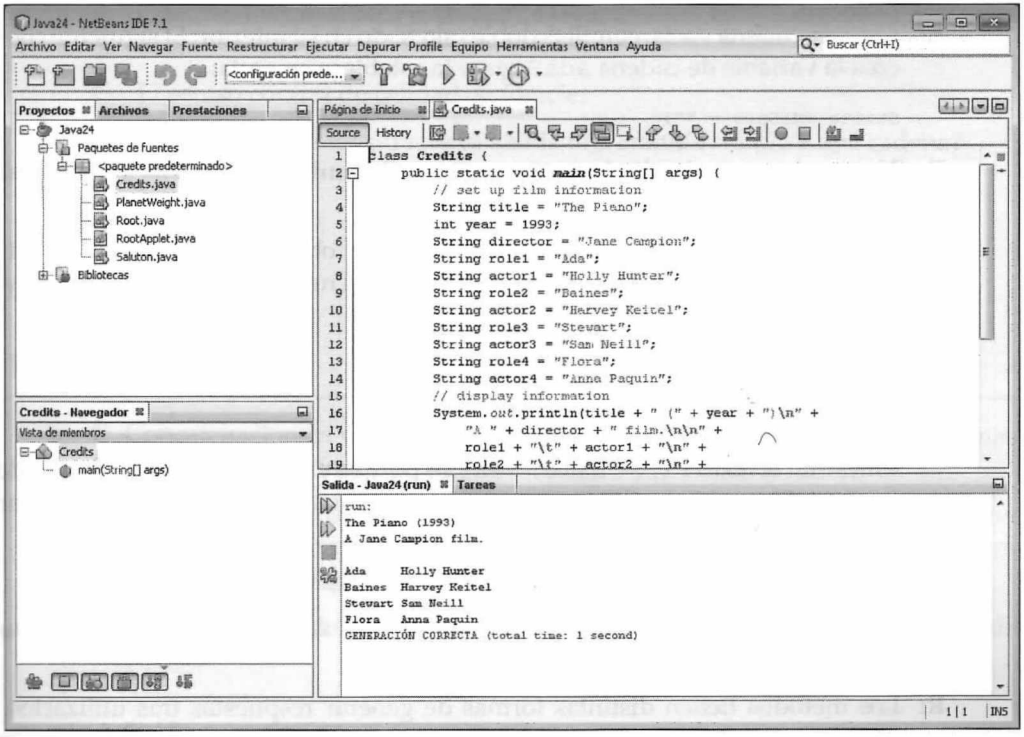


Figura 6.1. Resultado del programa Credits.

Resumen

Si su versión de `Credits` se parece a la figura 6.1, enhorabuena. Seis capítulos después, ya sabe crear programas de Java más extensos y enfrentarse a problemas más sofisticados. Las cadenas son elementos que usará siempre que escriba un programa. Al inicio de *El Piano*, el personaje de Holly Hunter, Ada, pierde su piano cuando su nuevo esposo se niega a que sus portadores maoris lo lleven a casa. Nadie le puede arrebatar sus cadenas. Las empleará de diversas formas para comunicarse con los usuarios.

Nota

Si este capítulo con curiosidades sobre *El Piano* y las películas de Jane Campion ha despertado su curiosidad o si le atraen las damas tímidas, visite el sitio Web sobre *El Piano* de Magnus Hjelstuen (`www.cadenhead.org/piano`).

Preguntas y respuestas

P: ¿Cómo puedo establecer en blanco el valor de una variable de cadena?

R: Use una cadena vacía, comillas dobles sin texto en su interior. El siguiente código crea la variable de cadena `adaSays` y la establece en nada:

```
String adaSays = "";
```

P: No consigo que el método `toUpperCase()` cambie una cadena a mayúsculas. ¿Cuál es el problema?

R: Al invocar el método `toUpperCase()` de un objeto `String`, en realidad no cambia el objeto sobre el que se invoca. Crea una nueva cadena con letras mayúsculas. Fíjese en las siguientes instrucciones:

```
String firstName = "Nessie";
String changeName = firstName.toUpperCase();
System.out.println("First Name: " + firstName);
```

Muestran el texto `First Name: Nessie` porque `firstName` contiene la cadena original. Si cambia la última instrucción para mostrar la variable `changeName`, devolvería `First Name: NESSIE`.

En Java, las cadenas no cambian de valor una vez creadas.

P: ¿Todos los métodos de Java muestran `true` o `false` al igual que el método `equals()` hace con cadenas?

R: Los métodos tienen distintas formas de generar respuestas tras utilizarlos. El método `equals()` está configurado para devolver un valor Booleano. Otros métodos pueden devolver una cadena, un entero, otro tipo de variable o nada (representado por `void`).

Ejercicios

Las siguientes preguntas comprueban sus conocimientos sobre cadenas.

Preguntas

1. Un amigo mió concatena. ¿Debo denunciarle a la policía?

   A. No. Solo es ilegal durante los meses de invierno.

   B. Sí, pero solo cuando haya vendido la exclusiva primero.

   C No. Lo único que hace es pegar dos cadenas en un programa.

2. ¿Por qué la palabra `String` usa mayúscula mientras que `int` y otras no lo hacen?

   A. `String` es una palabra completa pero `int` no.

   B. Como todos los objetos estándar de Java, `String` tiene un nombre en mayúscula.

   C. El escaso control de calidad en Oracle.

3. ¿Cuál de los siguientes caracteres añade una comilla simple a una cadena?

   A. `<quote>`

   B. `\'`

   C. `'`

Respuestas

1. C. Concatenación es un sinónimo de pegar, combinar o conectar dos cadenas. Emplea los operadores + y +=.
2. B. Los tipos de objetos disponibles en Java empiezan todos por mayúscula, motivo por el que los nombres de variables tienen la primera letra en minúscula. Así no se confunden con objetos.
3. B. La barra invertida es el inicio de los caracteres especiales que se pueden añadir a cadenas.

Actividades

Si desea ampliar sus conocimientos sobre los temas descritos en el capítulo, pruebe con las siguientes actividades:

- Cree un breve programa de Java con el nombre `Favorite` que incluya el código del apartado sobre comparación de dos cadenas en el bloque `main()`. Compruebe que funciona y que dice que el instrumento favorito de Ada no es el ukulele. Cuando termine, cambie el valor inicial de la variable `guess` de `ukulele` a `piano`. Compruebe el resultado.
- Modifique el programa `Credits` para que los nombres del director y los actores se muestren en letras mayúsculas.

7

Usar pruebas condicionales para tomar decisiones

Al crear un programa, debe proporcionar al ordenador una lista de instrucciones, que se siguen al pie de la letra. Puede decirle al ordenador que calcule complejas fórmulas matemáticas y las resuelva. Dígale que muestre información y lo hará sin rechistar.

Pero en ocasiones el ordenador debe ser selectivo. Por ejemplo, si ha creado un programa para su cuenta bancaria, puede que le interese ver un mensaje de advertencia cuando su saldo se agote.

Para realizar esta tarea en un programa de Java debe usar una instrucción condicional que haga que suceda algo en un programa solo si se cumple una condición específica. En este capítulo aprenderemos a emplear las instrucciones condicionales `if`, `else` y `switch`.

Cuando un programa de Java toma una decisión, para ello usa una instrucción condicional. En este capítulo comprobaremos la condición de varios elementos de sus programas de Java por medio de las palabras clave `if`, `else`, `switch`, `case` y `break`. También emplearemos los operadores condicionales ==, !=, <, >, <=, >= y ?, junto con variables `boolean`.

Instrucciones if

La forma más básica de probar una condición en Java consiste en usar una instrucción `if`, la cual comprueba si una condición es `true` o `false` y realiza una acción solamente si es `true`. Puede emplear `if` con la condición que probar, como en la siguiente instrucción:

```
if (account < 0) {
   System.out.println("Account overdrawn; you need a bailout");
}
```

La instrucción `if` comprueba si la variable `account` se encuentra por debajo de `0` por medio del operador menor que <. En caso afirmativo, se ejecuta el bloque de la instrucción y se muestra el texto. El bloque solo se ejecuta si la condición es `true`. En el ejemplo anterior, si la variable `account` tiene el valor `0` o superior, se ignora la instrucción `println`. La condición que hay que probar debe incluirse entre paréntesis, como en `(account < 0)`. El operador menor que < es uno de los distintos operadores que puede usar con instrucciones condicionales.

Comparaciones menor que y mayor que

En el apartado anterior, se emplea el operador < como en clase de Matemáticas. También existe un operador condicional mayor que, >, que se usa en las siguientes instrucciones:

```
int elephantWeight = 900;
int elephantTotal = 13;
int cleaningExpense = 200;

if (elephantWeight > 780) {
   System.out.println("Elephant too fat for tightrope act");
}

if (elephantTotal > 12) {
   cleaningExpense = cleaningExpense + 150;
}
```

La primera instrucción `if` comprueba si el valor de la variable `elephantWeight` es mayor que `780`. La segunda comprueba si `elephantTotal` es mayor que `12`.

Si las dos instrucciones anteriores se emplean en un programa en el que `elephantWeight` es igual a `600` y `elephantTotal` es igual a `10`, se ignoran las instrucciones de cada bloque. Puede determinar si algo es menor o igual que otro elemento por medio del operador <=, como en este ejemplo:

```
if (account <= 0) {
   System.out.println("You are flat broke");
}
```

También existe un operador >= para comparaciones menor o igual que.

Comparaciones de igualdad y no igualdad

Otra condición que se puede comprobar en un programa es la igualdad, por ejemplo si una variable es igual a un valor concreto o igual al valor de otra variable. Estas preguntas se pueden responder con el operador ==, como en las siguientes instrucciones:

```
if (answer == rightAnswer) {
   studentGrade = studentGrade + 10;
}

if (studentGrade == 100) {
   System.out.println("Show off!");
}
```

También puede comprobar si algo no es igual a algo, con el operador !=:

```
if (answer != rightAnswer) {
   score = score - 5;
}
```

Puede usar los operadores == y != con todos los tipos de variables menos con cadenas, que son objetos.

Advertencia

El operador empleado para realizar pruebas de igualdad tiene dos signos igual: ==. Puede confundirse con el operador =, que se usa para asignar un valor a una variable. Siempre debe emplear dos signos igual en una instrucción condicional.

Organizar un programa con instrucciones de bloque

Hasta el momento, las instrucciones `if` de este capítulo contaban con un bloque incluido entre llaves.

Anteriormente, vimos el uso de las instrucciones de bloque para marcar el inicio y el final del bloque `main()` de un programa de Java. Cada instrucción del bloque `main()` se procesa al ejecutar el programa.

Una instrucción `if` no requiere una instrucción de bloque. Puede ocupar una sola línea:

```
if (account <= 0) System.out.println("No more money");
```

La instrucción que aparece tras `if` solo se ejecuta si la condicional es `true`.

El listado 7.1 es un ejemplo de un programa de Java con una instrucción de bloque para indicar el bloque `main()`. La instrucción de bloque comienza con la llave de apertura { de la línea 2 y termina con la llave de cierre de la línea 13. Cree un nuevo archivo vacío de Java con el nombre `Game` en NetBeans e introduzca el texto del listado 7.1.

Listado 7.1. El programa Game.

```
1: class Game {
2:   public static void main(String[] arguments) {
3:      int total = 0;
4:      int score = 7;
5:      if (score == 7) {
6:         System.out.println("You score a touchdown!");
7:      }
```

```
 8:       if (score == 3) {
 9:          System.out.println("You kick a field goal!");
10:       }
11:       total = total + score;
12:       System.out.println("Total score: " + total);
13:    }
14: }
```

Al ejecutar el programa obtendrá el resultado mostrado en la figura 7.1.

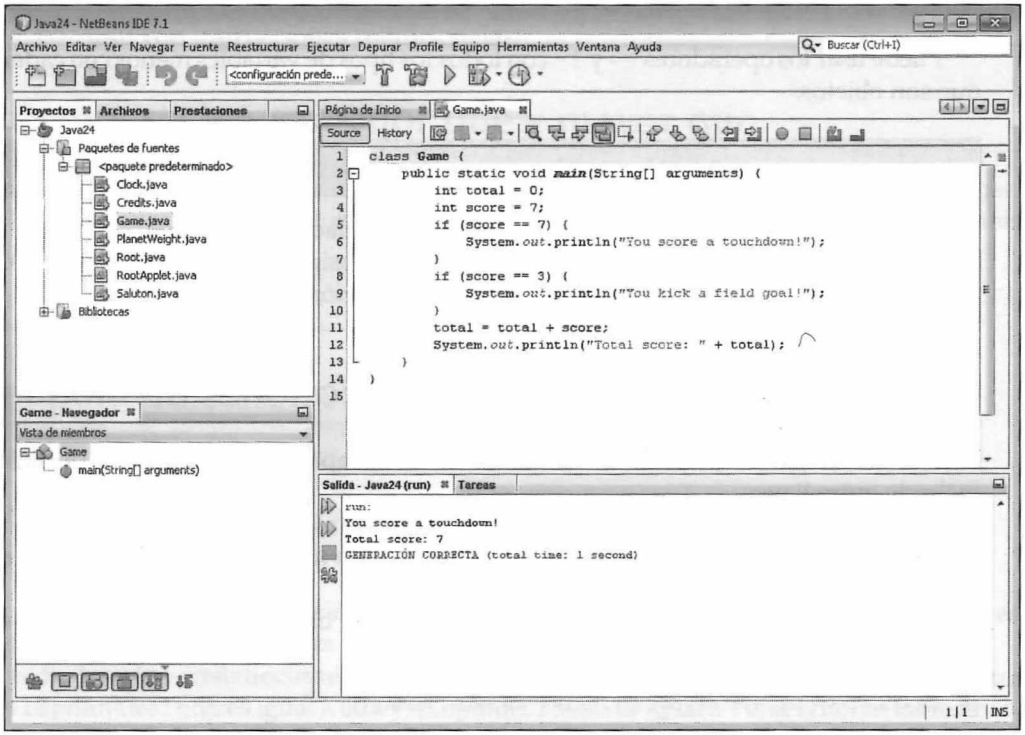


Figura 7.1. Resultado del programa Game.

Puede usar instrucciones de bloque en las instrucciones `if` para que el ordenador realice más de una acción si la condición es `true`. El siguiente ejemplo muestra una instrucción `if` que incluye una instrucción de bloque:

```
int playerScore = 12000;
int playerLives = 3;
int difficultyLevel = 10;

if (playerScore > 9999) {
   playerLives++;
   System.out.println("Extra life!");
   difficultyLevel = difficultyLevel + 5;
}
```

Las llaves se emplean para agrupar todas las instrucciones que forman parte de la instrucción `if`.

Si la variable `playerScore` es mayor de `9,999`, pasan tres cosas:

- El valor de `playerLives` aumenta en una unidad (debido al operador de incremento ++).
- Se muestra el texto `"Extra life!"`.
- El valor de la variable `difficultyLevel` se incrementa en `5`.

Si la variable `playerScore` no es mayor de `9,999`, no sucede nada, se ignoran las tres instrucciones del bloque `if`.

Instrucciones if-else

En ocasiones tendrá que realizar una operación si una condición es `true` y otra acción diferente si es `false`. Para ello puede usar la instrucción `else` junto a la instrucción `if`, como en el siguiente ejemplo:

```
int answer = 17;
int correctAnswer = 13;

if (answer == correctAnswer) {
   score += 10;
   System.out.println("That's right. You get 10 points");
} else {
   score -= 5;
   System.out.println("Sorry, that's wrong. You lose 5 points");
}
```

La instrucción `else` no tiene una condición, al contrario de lo que sucede con `if`. `else` se compara con la instrucción `if` inmediatamente anterior.

También puede emplear `else` para encadenar varias instrucciones `if`, como en el siguiente ejemplo:

```
if (grade == 'A') {
   System.out.println("You got an A. Great job!");
} else if (grade == 'B') {
   System.out.println("You got a B. Good work!");
} else if (grade == 'C') {
   System.out.println("You got a C. What went wrong?");
} else {
   System.out.println("You got an F. You'll do well in Congress!");
}
```

Al combinar varias instrucciones `if` y `else`, puede controlar distintas condiciones. El ejemplo anterior envía un mensaje concreto a los alumnos con una nota A, B, C y F (futuros legisladores).

Instrucciones switch

Las instrucciones `if` y `else` son muy útiles para casos con dos posibles condiciones, pero en ocasiones puede haber más de dos.

Con el ejemplo anterior, vimos que las instrucciones `if` y `else` se pueden combinar para procesar distintas condiciones.

Otra forma de hacerlo consiste en usar la instrucción `switch`, que prueba distintas condiciones y responde según corresponda. En el siguiente código, se ha modificado el ejemplo anterior con una instrucción `switch`:

```
switch (grade) {
   case 'A':
      System.out.println("You got an A. Great job!");
      break;
   case 'B':
      System.out.println("You got a B. Good work!");
      break;
   case 'C':
      System.out.println("You got a C. What went wrong?");
      break;
   default:
      System.out.println("You got an F. You'll do well in Congress!");
}
```

La primera línea de la instrucción `switch` especifica la variable que probar: `grade`. Tras ello, la instrucción `switch` emplea llaves `{` y `}` para crear una instrucción de bloque.

Cada instrucción `case` comprueba la variable de prueba de la instrucción `switch` con respecto a un valor concreto. El valor usado en una instrucción `case` puede ser un carácter, un entero o una cadena. En el ejemplo anterior hay instrucciones `case` para los caracteres A, B y C. Cada uno tiene una o dos instrucciones posteriores. Cuando una de estas instrucciones `case` coincide con la variable de `switch`, el ordenador procesa las instrucciones tras `case` hasta que detecta una instrucción `break`.

Por ejemplo, si la variable `grade` tiene el valor `B`, se muestra el texto `"You got a B. Good work!"`. La siguiente instrucción es `break`, de modo que no se ejecuta nada más de la instrucción `switch`. La instrucción `break` indica al ordenador que salga de la instrucción `switch`.

La instrucción `default` se emplea como comodín si ninguna de las instrucciones `case` anteriores es `true`. En este ejemplo, se produce si la variable `grade` no es igual a `A`, `B` o `C`. No es necesario usar `default` en todos los bloques `switch` de sus programas. Si la omite, no pasa nada si ninguna de las instrucciones `case` tiene el valor correcto.

Java 7 añade compatibilidad con el uso de cadenas como variable de prueba en una instrucción `switch-case`. La clase `Commodity` del listado 7.2 emplea esta instrucción para comprar o vender un producto indeterminado. El producto vale 20 dólares cuando se compra y genera 15 dólares cuando se vende. Una instrucción `switch-case` comprueba el valor de la cadena `command` y ejecuta un bloque si es igual a `"BUY"` y otro si es igual a `"SELL"`.

Listado 7.2. El programa Commodity.

```
 1: public class Commodity {
 2:   public static void main(String[] arguments) {
 3:      String command = "BUY";
 4:      int balance = 550;
 5:      int quantity = 42;
 6:
 7:      switch (command) {
 8:         case "BUY":
 9:            quantity += 5;
10:            balance -= 20;
11:            break;
12:         case "SELL":
13:            quantity -= 5;
14:            balance += 15;
15:      }
16:      System.out.println("Balance: " + balance + "\n"
17:      + "Quantity: " + quantity);
18:      }
19: }
```

Esta aplicación establece la cadena `command` en `"BUY"` en la línea 3. Cuando se prueba `switch`, se ejecuta el bloque `case` en las líneas 9-11. La cantidad del producto aumenta en 5 y el saldo se reduce en 20 dólares.

Al crear este programa puede encontrar un error que impida su compilación y ejecución. Puede que NetBeans no esté configurado para usar las funciones del lenguaje añadidas en Java 7.

Si emplea una cadena en una instrucción `switch`, puede ver el icono rojo de alerta en la parte izquierda del editor de código en la línea 7, para indicar que se necesita cierta configuración.

Las funciones de Java 7 se habilitan en función de cada proyecto en NetBeans. Siga los pasos descritos a continuación para hacerlo:

1. En el panel **Proyectos**, haga clic con el botón derecho del ratón sobre el elemento **Java24** (o el nombre que haya asignado a su proyecto) y seleccione **Propiedades** en el menú contextual. Se abrirá el cuadro de diálogo **Propiedades del proyecto**.
2. En el panel **Categorías**, haga clic en **Fuentes**. El cuadro de diálogo muestra las propiedades del código fuente (véase la figura 7.2).
3. En la lista desplegable **Formatos de fuentes/binarios**, seleccione **JDK 7** y pulse **Aceptar**.

De este modo configura el editor de código fuente para que todos los programas del proyecto funcionen con Java 7.

Al ejecutar el programa `Commodity`, genera el siguiente resultado:

```
Balance: 530
Quantity: 47
```

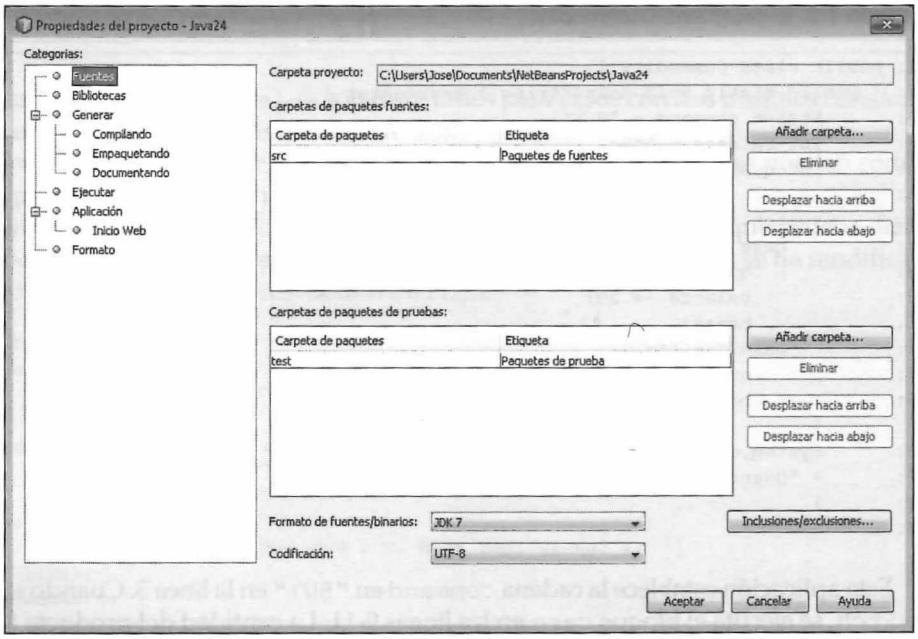


Figura 7.2. Configuración del editor de NetBeans para Java 7.

El operador condicional

La instrucción condicional más complicada de Java es el operador ternario `?`.

Puede usarlo para asignar un valor o mostrarlo en función de una condición. Por ejemplo, imagine un videojuego que establece la variable `numberOfEnemies` en función de si el valor de la variable `skillLevel` es mayor que `5`. Puede emplear una instrucción `if-else`:

```
if (skillLevel > 5) {
   numberOfEnemies = 10;
} else {
   numberOfEnemies = 5;
}
```

Una forma más breve de hacerlo consiste en usar el operador ternario, que tiene cinco partes:

- La condición que probar, entre paréntesis, como en `(skillLevel > 5)`.
- Un signo de interrogación (`?`).
- El valor que emplear si la condición es `true`.
- Dos puntos (`:`).
- El valor que usar si la condición es `false`.

Para emplear el operador ternario para establecer `numberOfEnemies` en función de `skillLevel`, use la siguiente instrucción:

```
int numberOfEnemies = (skillLevel > 5) ? 10 : 5;
```

También puede usar el operador ternario para determinar qué información mostrar. Imagine un programa que muestra el texto `"Mr."` o `"Ms."` en función del valor de la variable `gender`. Podría emplear esta instrucción:

```
System.out.print( (gender.equals("male")) ? "Mr." : "Ms." );
```

El operador ternario puede ser muy útil, pero también es el más complicado de entender. Cuando aumente sus conocimientos de Java, no verá demasiados casos en los que el operador ternario se usa en lugar de instrucciones `if-else`.

Mirar el reloj

El siguiente proyecto analiza las distintas pruebas condicionales que puede emplear en sus programas. Para este proyecto, usaremos la función de reloj de Java, que controla la fecha y hora actual, y presenta esta información en forma de frases.

Ejecute NetBeans (u otro programa para crear programas de Java) y cree una nueva clase con el nombre `Clock`. Es un programa extenso pero en su gran mayoría está formado por instrucciones condicionales. Introduzca el texto completo del listado 7.3 en el editor de código y guarde el archivo.

Listado 7.3. El programa Clock.

```
 1: import java.util.*;
 2:
 3: class Clock {
 4:   public static void main(String[] arguments) {
 5:      // obtener la hora y fechas actuales
 6:      Calendar now = Calendar.getInstance();
 7:      int hour = now.get(Calendar.HOUR_OF_DAY);
 8:      int minute = now.get(Calendar.MINUTE);
 9:      int month = now.get(Calendar.MONTH) + 1;
10:      int day = now.get(Calendar.DAY_OF_MONTH);
11:      int year = now.get(Calendar.YEAR);
12:
13:      // mostrar saludo
14:      if (hour < 12) {
15:         System.out.println("Good morning.\n");
16:      } else if (hour < 17) {
17:         System.out.println("Good afternoon.\n");
18:      } else {
19:         System.out.println("Good evening.\n");
20:      }
21:
22:      // iniciar mensaje de hora mostrando los minutos
```

```
23:       System.out.print("It's");
24:       if (minute != 0) {
25:          System.out.print(" " + minute + " ");
26:          System.out.print( (minute != 1) ? "minutes" :
27:             "minute");
28:          System.out.print(" past");
29:       }
30:
31:       // mostrar la hora
32:       System.out.print(" ");
33:       System.out.print( (hour > 12) ? (hour - 12) : hour );
34:       System.out.print(" o'clock on ");
35:
36:       // mostrar el nombre del mes
37:       switch (month) {
38:          case 1:
39:             System.out.print("January");
40:             break;
41:          case 2:
42:             System.out.print("February");
43:             break;
44:          case 3:
45:             System.out.print("March");
46:             break;
47:          case 4:
48:             System.out.print("April");
49:             break;
50:          case 5:
51:             System.out.print("May");
52:             break;
53:          case 6:
54:             System.out.print("June");
55:             break;
56:          case 7:
57:             System.out.print("July");
58:             break;
59:          case 8:
60:             System.out.print("August");
61:             break;
62:          case 9:
63:             System.out.print("September");
64:             break;
65:          case 10:
66:             System.out.print("October");
67:             break;
68:          case 11:
69:             System.out.print("November");
70:             break;
71:          case 12:
72:             System.out.print("December");
73:       }
74:
75:       // mostrar la fecha y el año
76:       System.out.println(" " + day + ", " + year + ".");
77:    }
78: }
```

Una vez compilado el programa, lo examinaremos para ver cómo se emplean las pruebas condicionales. A excepción de la línea 1 y las líneas 6-11, el programa `Clock` contiene material que ya hemos descrito en el libro. Tras configurar una serie de variables para guardar la fecha y la hora actuales, se utilizan instrucciones `if` o `switch` para determinar la información que se va a mostrar. El programa contiene varios usos de `System.out.println()` y `System.out.print()` para mostrar cadenas.

Las líneas 6-11 hacen referencia a la variable `Calendar now`. El tipo de variable `Calendar` se escribe con mayúscula por tratarse de un objeto.

En el capítulo 10 aprenderemos a crear y a emplear objetos. Por el momento, nos centraremos en lo que sucede en esas líneas y no en cómo sucede.

El programa `Clock` se divide en las siguientes secciones:

- La línea 1 permite al programa usar una clase necesaria para controlar la fecha y la hora actuales: `java.util.Calendar`.
- Las líneas 3-4 inician el programa `Clock` y su bloque `main()`.
- La línea 6 crea un objeto `Calendar` con el nombre `now` que contiene la fecha y hora actuales del sistema. El objeto `now` cambia cada vez que ejecuta el programa (a menos que se alteren las leyes físicas del universo y el tiempo se detenga).
- Las líneas 7-11 crean variables para guardar la hora, el minuto, el mes, el día y el año. Los valores de estas variables se obtienen del objeto `Calendar`, un almacén de esta información.
- Las líneas 14-20 muestran uno de tres saludos posibles: `"Good morning."`, `"Good afternoon."` o `"Good evening."`. El saludo que se va a mostrar se selecciona en función del valor de la variable `hour`.
- Las líneas 23-29 muestran el minuto actual junto a un texto. Primero aparece el texto `"It's"` en la línea 23. Si el valor de `minute` es `0`, las líneas 25-28 se ignoran debido a la instrucción `if` de la línea 24. Esta instrucción es necesaria ya que no tendría sentido que el programa mostrara que son los 0 minutos de una hora. La línea 25 muestra el valor actual de la variable `minute`. En las líneas 26-27 se emplea el operador ternario para mostrar el texto `"minutes"` o `"minute"` en función de si `minute` es igual a `1`. Por último, en la línea 28 se muestra el texto `past`.
- Las líneas 32-34 muestran la hora actual por medio de otro operador ternario. La instrucción condicional ternaria de la línea 33 muestra la hora de forma diferente si es mayor que `12`, lo que impide que el ordenador muestre las horas como `"15 o'clock"`.
- Las líneas 37-73, prácticamente la mitad del programa, son una extensa instrucción `switch` que muestra un nombre del mes en función del valor entero almacenado en la variable `month`.
- La línea 76 muestra la fecha y el año actuales.
- Las líneas 77-78 cierran el bloque `main()` y finalizan el programa `Clock`.

Al ejecutar este programa, el resultado mostrará una frase en función de la fecha y hora actuales. En la figura 7.3 puede ve el resultado de la aplicación. Ejecute varias veces el programa para comprobar si muestra correctamente el reloj.

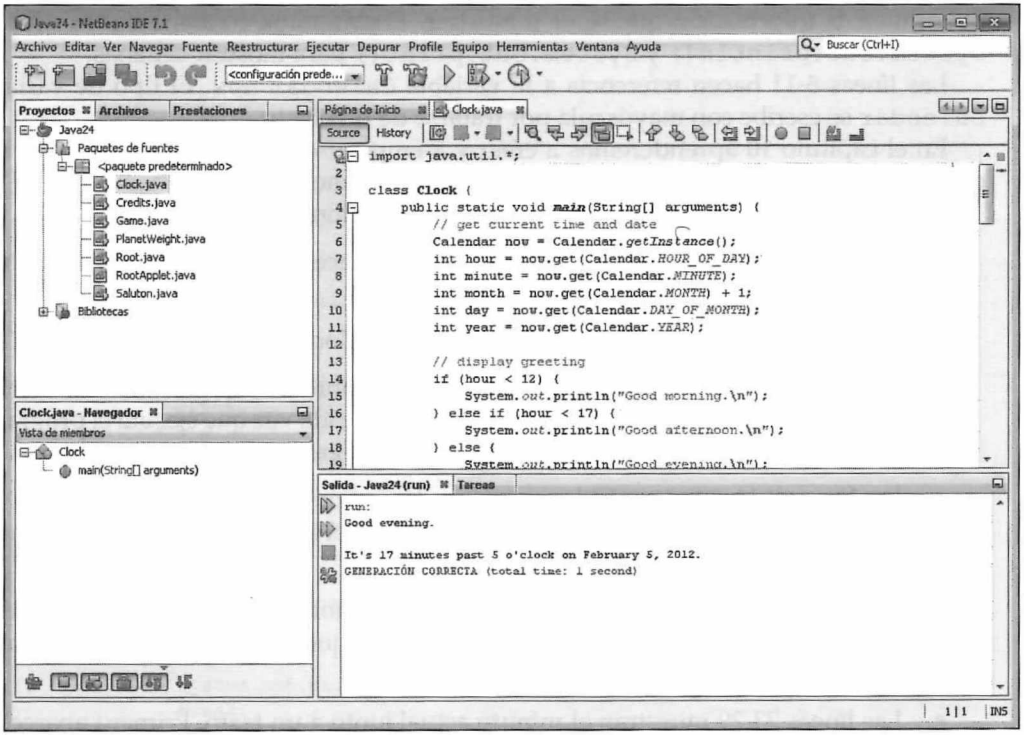


Figura 7.3. Resultado del programa Clock.

Resumen

Ahora que ya sabe cómo usar las instrucciones condicionales, la inteligencia de sus programas de Java mejorará considerablemente. Sus programas podrán evaluar información y emplearla para reaccionar de acuerdo a cada situación, aunque la información cambie con el programa en ejecución. Podrán optar entre dos o más alternativas en función de condiciones concretas.

La programación de un ordenador hace que tenga que dividir las tareas en una serie de pasos lógicos que realizar y de decisiones que adoptar. Con la instrucción `if` y las demás condicionales, se impulsa una especie de pensamiento lógico que puede aplicar a otros aspectos de la vida:

- Si resulta elegido presidente en noviembre, buscaré un puesto en su gabinete; en caso contrario, emigro a Canadá.

- Si mi cita a ciegas es atractiva, pagaré la cena en un restaurante caro, sino iremos a Pizza Hut.
- Si incumplo mi fianza, el único equipo que me contratará serán los Philadelphia Eagles.

Preguntas y respuestas

P: La instrucción `if` parece la más útil. ¿Se pueden usar solo instrucciones if en un programa y no las demás?

R: Se puede sobrevivir sin `else` o `switch`, y muchos programadores nunca emplean el operador ternario `?`. Sin embargo, `else` y `switch` son muy útiles para sus programas ya que facilitan su comprensión. Un conjunto de instrucciones `if` encadenadas puede resultar imposible de entender.

P: En el programa `Clock`, ¿por qué se suma `1` a `Calendar.MONTH` para obtener el valor del mes actual?

R: Es necesario debido a la forma de representar los meses de la clase `Calendar`. En lugar de numerarlos del 1 al 12, `Calendar` empieza a numerar por `0` para enero y termina con `11` para diciembre. Al sumar `1` los meses se representan numéricamente de forma más comprensible.

P: En este capítulo no se usan llaves en instrucciones `if` si éstas se emplean junto a otra instrucción. ¿Es obligatorio usar llaves?

R: No. Las llaves se pueden emplear como parte de cualquier instrucción `if` para incluir la parte del programa que depende de la prueba condicional. El uso de llaves es una práctica recomendada ya que evita un error habitual en la revisión de programas. Si añade otra instrucción tras una condición `if` sin llaves, pueden producirse errores inesperados al ejecutar el programa.

P: ¿Es necesario usar `break` en todas las secciones de las instrucciones posteriores a `case`?

R: No. Si no la emplea al final de un grupo de instrucciones, las instrucciones restantes del bloque `switch` se procesan, independientemente del valor `case` con el que se prueben. Sin embargo, en la mayoría de los casos necesitará una instrucción `break` al final de cada grupo.

Ejercicios

Las siguientes preguntas ponen a prueba sus conocimientos sobre instrucciones condicionales de Java.

Preguntas

1. Las pruebas condicionales generan un valor `true` o `false`. ¿A qué tipo de variable se parecen?
   A. A ninguna. No me moleste más con todas estas preguntas.
   B. A la variable `long`.
   C. Al tipo `boolean`.
2. ¿Qué instrucción se usa como categoría comodín en un bloque `switch`?
   A. `default`
   B. `otherwise`
   C. `onTheOtherHand`
3. ¿Qué es una condicional?
   A. Esa cosa que repara las puntas abiertas después de darnos champú.
   B. Un elemento de un programa que comprueba si una condición es verdadera o no.
   C. El lugar donde confesamos nuestros pecados a una autoridad religiosa.

Respuestas

1. C. El tipo de variable `boolean` puede ser igual a `true` o `false`, de modo que es similar a las pruebas condicionales. Si ha respondido A, lo siento, pero solo quedan 17 capítulos y tenemos mucho que ver. Java no se enseña solo.
2. A. Las instrucciones `default` se procesan si ninguna de las demás instrucciones `case` coincide con la variable `switch`.
3. B. Las otras dos preguntas describen acondicionador y confesionario.

Actividades

Para mejorar sus conocimientos sobre las condicionales de Java, realice las siguientes actividades:

- Añada `//` por delante de la una instrucción `break` en una de las líneas del programa `Clock` para convertirla en un comentario. Compile el programa y observe qué sucede al ejecutarlo. Vuelva a probar eliminando algunas de las instrucciones `break`.

- Cree un breve programa que almacene un valor entre 1 y 100 en la variable entera `grade`. Use la variable con una instrucción condicional para mostrar un mensaje diferente a alumnos con las notas A, B, C, D y F. Pruebe primero con una instrucción `if` y después con `switch`.

8
Repetir una acción con bucles

Uno de los peores castigos para un alumno es tener que copiar algo repetidamente en la pizarra, como le sucede a Bart Simpson. Este castigo puede funcionar con los alumnos, pero un ordenador puede repetir operaciones sin dificultad alguna.

Los programas informáticos están especialmente ideados para repetir operaciones gracias a los bucles. Un bucle es una instrucción o un bloque que se repite en un programa. Algunos bucles se ejecutan un número concreto de veces y otros de forma indefinida.

En Java existen tres instrucciones de bucle: `for`, `do` y `while`. Su funcionamiento es similar pero es aconsejable aprender a utilizar las tres variantes. Por lo general, podrá simplificar una sección de bucle de un programa si elige la instrucción adecuada.

Bucles for

En sus programas, encontrará números casos en los que aplicar un bucle. Puede usarlo para realizar una operación varias veces, como en un programa antivirus que se abra al recibir un correo y busque virus. También puede emplear bucles para que el ordenador no haga nada durante un periodo de tiempo concreto, como por ejemplo un reloj animado que muestre la hora actual una vez por minuto.

Una instrucción de bucle hace que el programa informático vuelva al mismo lugar más de una vez, como un avión realizando una acrobacia.

La instrucción de bucle más compleja de Java es `for`. Un bucle `for` repite una sección de un programa un número concreto de veces. Veamos un ejemplo:

```
for (int dex = 0; dex < 1000; dex++) {
   if (dex % 12 == 0) {
      System.out.println("#: " + dex);
   }
}
```

Este bucle muestra todos los números del 0 al 999 divisibles por 12.

Cada bucle `for` tiene una variable que determina cuándo debe comenzar y finalizar el bucle. Esta variable se denomina contador (o índice). El contador del bucle anterior es la variable `dex`. El ejemplo ilustra las tres partes de una instrucción `for`:

- **La sección de inicialización:** En la primera parte, se asigna el valor inicial `0` a la variable dex.
- **La sección condicional:** En la segunda parte, se incluye una prueba condicional como las usadas en instrucciones `if`: `dex < 1000`.
- **La sección de cambio:** La tercera parte es una instrucción que cambia el valor de la variable `dex`, en este ejemplo, por medio del operador de incremento.

En la sección de inicialización, se define la variable de contador. Puede crear la variable en la instrucción `for`, como en el ejemplo anterior con `dex`. También puede crear la variable en otro punto del programa. En cualquier caso, debe asignar a la variable un valor inicial en esta sección de la instrucción `for`. La variable tendrá este valor al iniciarse el bucle. La sección condicional contiene una prueba que debe ser `true` para que el bucle siga ejecutándose. Cuando la prueba es `false`, el bucle termina. En este ejemplo, el bucle termina cuando la variable `dex` es mayor o igual que `1000`.

La última sección de la instrucción for contiene una instrucción que cambia el valor de la variable de contador. Esta instrucción se procesa cada vez que se ejecuta el bucle. La variable de contador tiene que cambiar de algún modo o el bucle nunca termina. En el ejemplo, `dex` se incrementa en uno en la sección de cambio. Si `dex` no cambiara, conservaría su valor original `0` y `dex < 1000` siempre sería `true`.

El bloque de la instrucción `for` se ejecuta en cada iteración del bucle.

El ejemplo anterior contiene las siguientes instrucciones en el bucle:

```
if (dex % 12 == 0) {
   System.out.println("#: " + dex);
}
```

Estas instrucciones se ejecutan mil veces. El bucle comienza estableciendo la variable `dex` en `0`. Tras ello, añade 1 a `dex` en cada pasada del bucle y finaliza cuando `dex` ya no es menor que `1000`.

Nota

Un término poco habitual relacionado con los bucles es la iteración. Una iteración es un recorrido por el bucle. La variable de contador empleada para controlar el bucle se denomina iterador.

Como vimos con las instrucciones `if`, un bucle `for` no requiere llaves si solo contiene una instrucción, como en el siguiente ejemplo:

```
for (int p = 0; p < 500; p++)
   System.out.println("I will not sell miracle cures");
```

Este bucle muestra el texto `"I will not sell miracle cures"` 500 veces. Aunque las llaves no son necesarias para un bucle con una sola instrucción, puede usarlas para que el bloque se detecte más rápidamente.

El primer programa que crearemos en este capítulo muestra los 200 primeros múltiplos de 9: 9 × 1, 9 × 2, 9 × 3, y así sucesivamente hasta 9 × 200. En NetBeans, cree un nuevo proyecto de Java vacío con el nombre `Nines` e introduzca el texto del listado 8.1. Guárdelo como `Nines.java`.

Listado 8.1. Texto completo de Nines.java.

```
1: class Nines {
2:    public static void main(String[] arguments) {
3:       for (int dex = 1; dex <= 200; dex++) {
4:          int multiple = 9 * dex;
5:          System.out.print(multiple + " ");
6:       }
7:    }
8: }
```

El programa `Nines` contiene una instrucción `for` en la línea 3, dividida en tres partes:

- **Inicialización:** `int dex = 1`, que crea la variable entera `dex` con el valor inicial `1`.
- **Condicional:** `dex <= 200`, que debe ser `true` en cada iteración del bucle. Cuando no lo sea, el bucle termina.
- **Cambio:** `dex++`, que incrementa la variable `dex` una unidad en cada iteración del bucle.

Seleccione Run>Run File en NetBeans. El programa genera el siguiente resultado:

```
9 18 27 36 45 54 63 72 81 90 99 108 117 126 135 144 153 162 171
180 189 198 207 216 225 234 243 252 261 270 279 288 297 306 315
324 333 342 351 360 369 378 387 396 405 414 423 432 441 450 459
468 477 486 495 504 513 522 531 540 549 558 567 576 585 594 603
612 621 630 639 648 657 666 675 684 693 702 711 720 729 738 747
756 765 774 783 792 801 810 819 828 837 846 855 864 873 882 891
900 909 918 927 936 945 954 963 972 981 990 999 1008 1017 1026
1035 1044 1053 1062 1071 1080 1089 1098 1107 1116 1125 1134 1143
1152 1161 1170 1179 1188 1197 1206 1215 1224 1233 1242 1251 1260
1269 1278 1287 1296 1305 1314 1323 1332 1341 1350 1359 1368 1377
1386 1395 1404 1413 1422 1431 1440 1449 1458 1467 1476 1485 1494
1503 1512 1521 1530 1539 1548 1557 1566 1575 1584 1593 1602 1611
1620 1629 1638 1647 1656 1665 1674 1683 1692 1701 1710 1719 1728
1737 1746 1755 1764 1773 1782 1791 1800
```

La ventana de resultado de NetBeans muestra todos los números en una misma línea. Para que el texto se ajuste, haga clic con el botón derecho del ratón sobre el panel y seleccione `Wrap text` en el menú contextual.

Bucles while

El bucle `while` no tiene tantas secciones diferentes como `for`. Lo único que necesita es una prueba condicional junto a la instrucción `while`. Veamos un ejemplo:

```
while (gameLives > 0) {
   // añada aquí las instrucciones internas del bucle
}
```

Este bucle se repite hasta que la variable `gameLives` ya no es mayor que `0`.

La instrucción `while` prueba la condición al inicio del bucle antes de procesar sus instrucciones. Si la condición probada es `false` cuando el programa llega por primera vez a la instrucción `while`, se ignoran la instrucciones dentro del bucle.

Si la condición `while` es `true`, el bucle se ejecuta una vez y vuelve a probar la condición `while`. Si la condición probada nunca cambia dentro del bucle, se ejecutará indefinidamente.

Las siguientes instrucciones hacen que el bucle `while` muestre la misma línea varias veces:

```
do-while Loops 99
int limit = 5;
int count = 1;
while (count < limit) {
   System.out.println("Pork is not a verb");
   count++;
}
```

Un bucle `while` emplea una o más variables definidas antes de la instrucción de bucle. En este ejemplo, se crean dos variables enteras: `limit`, con el valor `5`, y `count`, con el valor `1`.

El bucle `while` muestra el texto `"Pork is not a verb"` cuatro veces. Si asigna a la variable `count` el valor inicial `6` en lugar de `1`, el texto nunca se muestra.

Bucles do-while

El bucle `do-while` es similar a `while`, pero la prueba condicional se ubica en un punto distinto. Veamos un ejemplo:

```
do {
   // añada aquí las instrucciones internas del bucle
} while (gameLives > 0);
```

Como sucede con `while`, este bucle se ejecuta continuamente hasta que la variable `gameLives` ya no es mayor que `0`. El bucle `do-while` es distinto ya que la prueba condicional se realiza tras las instrucciones del bucle, no antes.

Al alcanzar el bucle `do` en la primera ejecución del programa, las instrucciones entre `do` y `while` se procesan automáticamente, y se prueba la condición `while` para determinar si el bucle debe repetirse o no. Si la condición `while` es `true`, el bucle se ejecuta una vez más. Si la condición es `false`, el bucle finaliza. Debe suceder algo entre las instrucciones `do` y `while` que cambie la condición probada en `while` o el bucle se ejecutará indefinidamente. Las instrucciones de un bucle `do-while` siempre se procesan al menos una vez.

Las siguientes instrucciones hacen que un bucle `do-while` muestre repetidamente la misma línea de texto:

```
int limit = 5;
int count = 1;
do {
   System.out.println("I will not Xerox my butt");
   count++;
} while (count < limit);
```

Como sucede con `while`, un bucle `do-while` usa una o varias variables definidas antes de la instrucción de bucle.

El bucle muestra el texto `"I will not Xerox my butt"` cuatro veces. Si asigna a la variable `count` el valor inicial `6` en lugar de `1`, el texto se mostraría una vez, aunque `count` no sea nunca inferior a `limit`.

En un bucle `do-while`, las instrucciones del bucle se ejecutan al menos una vez aunque la condición del bucle sea `false` en la primera iteración.

Salir de un bucle

La forma habitual de salir de un bucle es que la condición probada sea `false`. Así funciona en los tres tipos de bucle de Java. Pero en ocasiones tendrá que salir inmediatamente de un bucle, aunque la condición probada siga siendo `true`. Para ello, puede emplear una instrucción `break`, como en el siguiente código:

```
int index = 0;
while (index <= 1000) {
   index = index + 5;
   if (index == 400) {
      break;
   }
}
```

Una instrucción `break` finaliza el bucle que contiene la instrucción.

En este ejemplo, el bucle `while` itera hasta que la variable `index` es mayor que `1000`. Sin embargo, un caso especial hace que el bucle termine antes: si `index` es igual a `400`, se ejecuta la instrucción `break` y el bucle termina de forma inmediata.

Otra instrucción especial que puede usar en un bucle es `continue`. Hace que el bucle salga de su iteración actual y se reinicie en la primera instrucción. Veamos un ejemplo:

```
int index = 0;
while (index <= 1000) {
   index = index + 5;
   if (index == 400)
      continue;
   System.out.println("The index is " + index);
}
```

En este bucle, las instrucciones se procesan normalmente hasta que el valor de `index` es `400`. En ese caso, la instrucción `continue` hace que el bucle vuelva a la instrucción `while` en lugar de continuar a `System.out.println()`. Debido a la instrucción `continue`, el bucle nunca muestra el siguiente texto:

```
The index is 400
```

Puede emplear las instrucciones `break` y `continue` con los tres tipos de bucle. `break` permite crear un bucle en el programa que se ejecuta indefinidamente, como en este ejemplo:

```
while (true) {
   if (quitKeyPressed == true) {
      break;
   }
}
```

Asignar un nombre a un bucle

Como sucede con otras instrucciones de los programas de Java, puede incluir un bucle dentro de otro. El siguiente ejemplo muestra un bucle `for` dentro de un bucle `while`:

```
int points = 0;
int target = 100;
while (target <= 100) {
   for (int i = 0; i < target; i++) {
      if (points > 50)
         break;
      points = points + i;
   }
}
```

En este ejemplo, la instrucción `break` finaliza el bucle `for` si la variable `points` es mayor de `50`. Sin embargo, el bucle `while` no termina nunca, ya que `target` nunca es mayor que `100`.

En determinados casos, puede salir de los dos bucles. Para ello, debe asignar un nombre al bucle exterior, en este caso la instrucción `while`. Para asignar un nombre, inclúyalo en la línea anterior al inicio del bucle, seguido de dos puntos (`:`). Cuando el

bucle tiene nombre, puede usarlo tras las instrucciones `break` o `continue` para indicar a qué bucle se aplica la instrucción `break` o `continue`. El siguiente ejemplo repite el anterior pero si la variable `points` es mayor que `50`, se finalizan ambos bucles.

```
int points = 0;
int target = 100;
targetLoop:
while (target <= 100) {
   for (int i = 0; i < target; i++) {
      if (points > 50)
         break targetLoop;
      points = points + i;
   }
}
```

Al emplear el nombre de un bucle en una instrucción `break` o `continue`, el nombre no incluye dos puntos.

Bucles for complejos

Un bucle `for` puede ser más complejo e incluir más de una variable en las secciones de inicialización, condicional y cambio. Cada sección de un bucle `for` se separa de las demás mediante punto y coma (`;`). Un bucle `for` puede tener más de una variable durante la sección de inicialización y más de una instrucción en la sección de cambio, como en el siguiente código:

```
int i, j;
for (i = 0, j = 0; i * j < 1000; i++, j += 2) {
   System.out.println(i + " * " + j + " = " + (i * j));
}
```

En cada sección del bucle `for` se usan comas para separar las variables, como en `i = 0, j = 0`. El ejemplo muestra una lista de ecuaciones en las que se multiplican las variables `i` y `j`. La variable `i` aumenta en uno y la variable `j` en dos en cada iteración del bucle. Cuando `i` multiplicado por `j` es igual o mayor que `1000`, el bucle termina. Las secciones de un bucle `for` también pueden estar vacías, por ejemplo, cuando la variable de contador de un bucle ya se ha creado con un valor inicial en otra parte del programa:

```
for ( ; displayCount < endValue; displayCount++) {
   // añada aquí las instrucciones internas del bucle
}
```

Probar la velocidad de su ordenador

El ejercicio de este capítulo es un programa de Java que realiza una prueba de la velocidad del hardware o software del ordenador. El programa `Benchmark` emplea un bucle para ejecutar repetidamente la siguiente expresión matemática:

```
double x = Math.sqrt(index);
```

Esta instrucción invoca el método `Math.sqrt()` para calcular la raíz cuadrada de un número. En el capítulo 11 aprenderemos a trabajar con métodos.

El programa que crearemos muestra cuántas veces por minuto puede calcular una raíz cuadrada un programa de Java.

En NetBeans cree un nuevo archivo vacío de Java con el nombre `Benchmark`. Introduzca el texto del listado 8.2 y guarde el programa cuando termine.

Listado 8.2. Código fuente completo de Benchmark.java.

```
1: class Benchmark {
2:   public static void main(String[] arguments) {
3:      long startTime = System.currentTimeMillis();
4:      long endTime = startTime + 60000;
5:      long index = 0;
6:      while (true) {
7:         double x = Math.sqrt(index);
8:         long now = System.currentTimeMillis();
9:         if (now > endTime) {
10:           break;
11:        }
12:        index++;
13:     }
14:     System.out.println(index + " loops in one minute.");
15:  }
16: }
```

El programa realiza las siguientes operaciones:

- **Líneas 1-2:** Se declara la clase `Benchmark` y se inicia el bloque `main()` del programa.
- **Línea 3:** Se crea la variable `startTime` con la hora actual en milisegundos como valor, obtenida tras invocar el método `currentTimeMillis()` de la clase `System` de Java.
- **Línea 4:** Se crea la variable `endTime` con un valor 60.000 veces mayor que `startTime`. Como un minuto equivale a 60.000 milisegundos, establece la variable un minuto después que `startTime`.
- **Línea 5:** Se crea `index`, de tipo `long`, con el valor inicial `0`.
- **Línea 6:** La instrucción `while` inicia un bucle con `true` como condicional, lo que hace que el bucle se ejecute indefinidamente (es decir, hasta que algo lo detenga).
- **Línea 7:** Se calcula la raíz cuadrada de `index` y se almacena en la variable `x`.
- **Línea 8:** Con `currentTimeMillis()`, se crea la variable `now` con la hora actual.
- **Líneas 9-11:** Si `now` es mayor que `endTime`, significa que el bucle se ha ejecutado durante un minuto y `break` finaliza el bucle `while`. En caso contrario, la ejecución continúa.

- **Línea 12:** La variable `index` se incrementa en `1` en cada iteración del bucle.
- **Líneas 14:** Fuera del bucle, el programa muestra el número de veces que ha calculado la raíz cuadrada.

En la figura 8.1 puede ver el resultado de la aplicación.

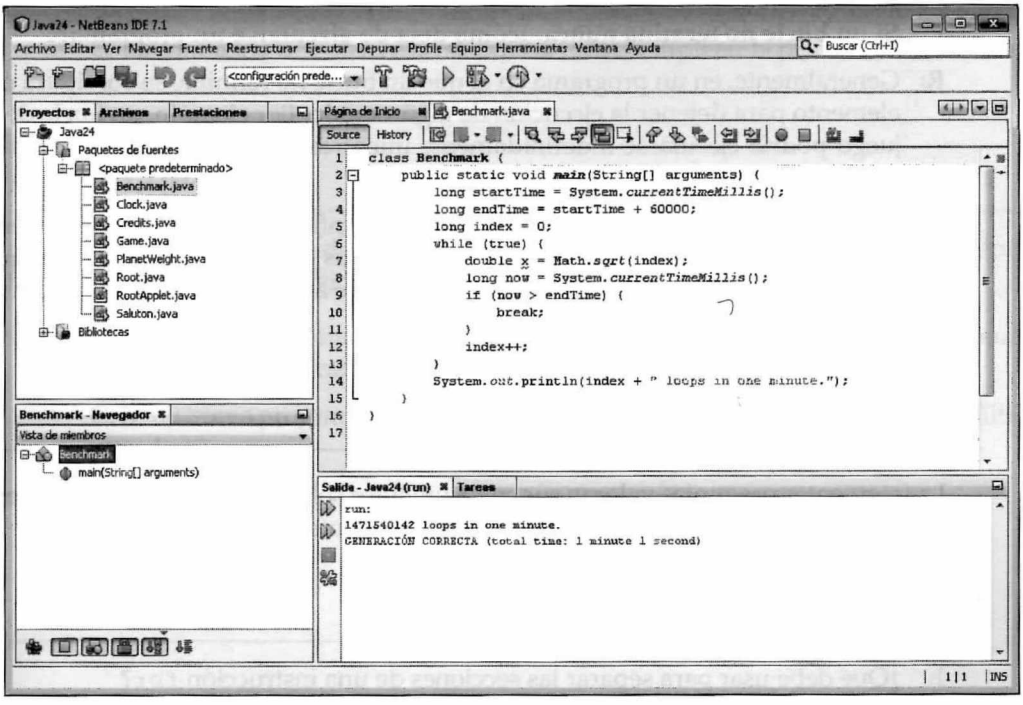


Figura 8.1. Resultado del programa Benchmark.

El programa `Benchmark` es una forma excelente de comprobar si su ordenador es más rápido que el mío. Al probar el programa, mi ordenador realizó unos 4.5 mil millones de cálculos. Si su ordenador ofrece mejores resultados, no se compadezca de mí. Compre más de mis libros para que pueda cambiar de equipo.

Resumen

Los bucles son una parte fundamental de la mayoría de lenguajes de programación. La animación creada mostrando varios gráficos secuencialmente es una de las muchas tareas que no podría realizar en Java ni en otros lenguajes sin ayuda de los bucles.

En la Web se documentan todas las veces que han castigado a Bart Simpson a escribir en la pizarra. Visite `www.snpp.com/guides/chalkboard.openings.html` para ver la lista y un programa de Java que muestra lo que escribe Bart en una pizarra verde.

Preguntas y respuestas

P: Se ha utilizado varias veces el término inicialización. ¿Qué significa?

R: Significa asignar a algo un valor inicial y configurarlo. Al crear una variable y asignarle un valor inicial, se inicializa la variable.

P: Si un bucle no termina nunca, ¿cómo deja de ejecutarse un programa?

R: Generalmente, en un programa en el que un bucle no termina, se configura otro elemento para detener la ejecución de algún modo. Por ejemplo, un bucle de un juego podría ejecutarse indefinidamente mientras el jugador tenga vidas suficientes.

Un problema que suele surgir son los bucles infinitos, que nunca se detienen debido a un error de programación. Si uno de sus programas de Java se bloquea en un bucle infinito, haga clic en el icono rojo de alerta de la parte izquierda del panel Output.

Ejercicios

Las siguientes preguntas valoran sus conocimientos sobre bucles. De acuerdo al tema analizado, repítalas hasta que las responda correctamente.

Preguntas

1. ¿Qué debe usar para separar las secciones de una instrucción `for`?

   A. Comas.

   B. Punto y coma.

   C. Policías de paisano.

2. ¿Qué instrucción hace que un programa vuelva a la instrucción que inicia el bucle y se ejecute desde ahí?

   A. `continue`

   B. `next`

   C. `skip`

3. ¿Qué instrucción de bucle de Java se ejecuta al menos una vez?

   A. `for`

   B. `while`

   C. `do-while`

Respuestas

1. B. Se emplean comas para separar elementos de una sección y punto y coma para separar secciones.
2. A. La instrucción `break` finaliza un bucle y `continue` pasa a la siguiente iteración del mismo.
3. C. La condición `do-while` no se evalúa hasta completar la primera iteración del bucle.

Actividades

Si no se siente mareado después de todos estos bucles, repase los temas de este capítulo con las siguientes actividades.

- Modifique el programa `Benchmark` para probar sencillas operaciones matemáticas como multiplicaciones o divisiones.
- Cree un breve programa con bucles que busque los primeros 400 números múltiplos de 13.

9
Almacenar información con matrices

Nadie se ha beneficiado más con el desarrollo de los ordenadores que Santa Claus. Durante años, la humanidad le ha presionado enormemente para que obtuviera y procesara información. El viejo Santa Claus tiene que controlar todos estos datos:

- Niños que se han portado mal.
- Niños que se han portado bien.
- Los regalos que se piden.
- Hogares con chimeneas infranqueables.
- Mujeres que quieran más de lo que la Sra. Claus está dispuesta a permitir.
- Países que primero derriban objetos volantes sin identificar y preguntan después.

Los ordenadores fueron una bendición para el Polo Norte. Resultan perfectos para almacenar, categorizar y analizar información.

La forma de almacenamiento más básica en un ordenador es guardar la información en una variable. La lista de niños que se han portado mal es un ejemplo de colección de información similar. Para controlar este tipo de listas, puede usar matrices.

Las matrices son grupos de variables relacionadas que comparten el mismo tipo. Cualquier tipo de información que se pueda almacenar como variable puede guardarse en una matriz. Las matrices se pueden emplear para controlar tipos de información más sofisticada pero son tan fáciles de crear y manipular que las variables.

Crear matrices

Las matrices son variables agrupadas bajo un nombre común. Como las variables, las matrices se crean indicando el tipo de variable organizada en la matriz y el nombre de ésta. Tras el tipo se usan corchetes (`[]`) para distinguir las matrices de las variables.

Puede crear matrices de cualquier tipo de información que pueda almacenarse como variable. Por ejemplo, la siguiente instrucción crea una matriz de variables de cadena:

```
String[] naughtyChild;
```

Veamos otros dos ejemplos:

```
int[] reindeerWeight;
boolean[] hostileAirTravelNations;
```

Los ejemplos anteriores crean matrices, pero no almacenan valores en su interior. Para ello, puede emplear la palabra clave `new` junto con el tipo de variable o almacenar los valores mediante llaves. Al usar `new` debe especificar la cantidad de elementos almacenados en la matriz. La siguiente instrucción crea una matriz y reserva espacio para los valores que contiene:

```
int[] elfSeniority = new int[250];
```

Este ejemplo crea la matriz de enteros `elfSeniority`. Tiene 250 elementos que pueden almacenar los meses que cada uno de los elfos de Santa Claus lleva trabajando en el Polo. Si Santa Claus liderase un sindicato, esta información sería tremendamente importante. Al crear una matriz con la instrucción `new` debe especificar el número de elementos. Cada elemento de la matriz recibe un valor inicial que depende del tipo de la matriz. Todas las matrices numéricas tienen el valor inicial `0`, las matrices `char '\0'` y las de tipo `boolean` el valor `false`. Una matriz `String` y los demás objetos se crean con el valor inicial `null`. En matrices sin un tamaño excesivo, puede configurar sus valores iniciales al mismo tiempo que las crea. Los siguientes ejemplos crean una matriz de cadenas y asignan sus valores iniciales:

```
String[] reindeerNames = { "Dasher", "Dancer", "Prancer", "Vixen",
   "Comet", "Cupid", "Donder", "Blitzen" };
```

Nota

Java es flexible sobre la posición de los corchetes al crear una matriz. Puede ubicarlos tras el nombre de la variable en lugar del tipo:

```
String niceChild[];
```

Para que las matrices se detecten más fácilmente en sus programas, debe aplicar el mismo estilo. En los ejemplos del libro siempre verá los corchetes tras la variable o el tipo de clase.

La información que se va a almacenar en los elementos de una matriz se incluye entre llaves y se emplean comas para separarla . El número de elementos de una matriz se establece con el número de elementos de la lista separada por comas. Cada elemento de la matriz debe ser del mismo tipo. En el ejemplo anterior, se usa una cadena para almacenar los nombres de los renos.

Tras crear la matriz, no puede contar con más espacio para otros elementos. Aunque se acuerde del reno más famoso de todos, no puede añadir `"Rudolph"` como noveno elemento de la matriz `reindeerNames`. El compilador de Java no permitirá que el pobre Rudolph forme parte de `reindeerNames`.

Utilizar matrices

En sus programas, las matrices se emplean como si fueran variables, a excepción del número de elemento entre corchetes junto al nombre de la matriz. Puede usar un elemento de matriz en los mismos casos que emplearía una variable. Las siguientes instrucciones usan matrices que ya se han definido en ejemplos anteriores:

```
elfSeniority[193] += 1;
niceChild[9428] = "Eli";
if (hostileAirTravelNations[currentNation] == true) {
   sendGiftByMail();
}
```

El primer elemento de una matriz se numera como `0`, no `1`, lo que significa que el número más alto es una unidad inferior al que se espera. Fíjese en esta instrucción:

```
String[] topGifts = new String[10];
```

Crea una matriz de variables de cadena numeradas del 0 al 9. Si hace referencia a `topGifts[10]` en otra parte del programa, se genera un mensaje de error de tipo `ArrayIndexOutOfBoundsException`.

Las excepciones son errores en programas de Java. Esta excepción es un error de índice de matriz fuera de límite, lo que significa que un programa ha intentado emplear un elemento de matriz que no existe entre sus límites definidos. Encontrará más información sobre errores en el capítulo 18.

Si desea comprobar el límite superior de una matriz para evitar superarlo, se asocia la variable `length` al crear cada matriz. Esta variable es un entero que contiene el número de elementos que contiene una matriz. El siguiente ejemplo crea una matriz y después informa de su longitud:

```
String[] reindeerNames = { "Dasher", "Dancer", "Prancer", "Vixen",
   "Comet", "Cupid", "Donder", "Blitzen", "Rudolph" };
System.out.println("There are " + reindeerNames.length + " reindeer.");
```

En este ejemplo, el valor de `reindeerNames.length` es `9`, lo que significa que el número de elemento mayor que puede especificar es `8`.

En Java, puede trabajar con texto en forma de cadena o como matriz de caracteres. Al trabajar con cadenas, una técnica muy útil consiste en incluir cada carácter de la cadena como elemento independiente de una matriz de caracteres. Para ello, invoque el método `toCharArray()` de la cadena, que genera una matriz `char` con el mismo número de elementos que la longitud de la cadena.

El primer proyecto de este capítulo usa las dos técnicas presentadas en este apartado. El programa `SpaceRemover` muestra una cadena con todos los espacios sustituidos por puntos (`.`).

Para empezar, abra el proyecto `Java24` en NetBeans, seleccione Archivo>Archivo Nuevo y cree un nuevo archivo vacío de Java con el nombre `SpaceRemover`. Introduzca el texto del listado 9.1 y guarde el archivo cuando termine.

Listado 9.1. Texto completo de SpaceRemover.java

```
 1: class SpaceRemover {
 2:   public static void main(String[] args) {
 3:     String mostFamous = "Rudolph the Red-Nosed Reindeer";
 4:     char[] mfl = mostFamous.toCharArray();
 5:     for (int dex = 0; dex < mfl.length; dex++) {
 6:       char current = mfl[dex];
 7:       if (current != ' ') {
 8:         System.out.print(current);
 9:       } else {
10:         System.out.print('.');
11:       }
12:     }
13:     System.out.println();
14:   }
15: }
```

Ejecute el programa por medio de Ejecutar>Ejecutar archivo para obtener el resultado de la figura 9.1.

La aplicación `SpaceRemover` almacena el texto `Rudolph the Red-Nosed Reindeer` en dos puntos: la cadena `mostFamous` y la matriz `char mfl`. La matriz se crea en la línea 4 mediante la invocación del método `toCharArray()` de `mostFamous`, que completa la matriz con un elemento por cada uno de los caracteres del texto. El carácter `R` se corresponde al elemento 0, `u` al elemento 1 y así sucesivamente hasta `r` en el elemento 29.

El bucle `for` de las líneas 5-12 busca los caracteres de la matriz `mfl`. Si el carácter no es un espacio, se muestra. Si es un espacio, se muestra el carácter `.` (un punto).

Matrices multidimensionales

Hasta ahora, las matrices del capítulo tienen una sola dimensión, de modo que puede recuperar un elemento con un número. Algunos tipos de información requieren más dimensiones para almacenarse como matrices, como los puntos de un sistema de coordenadas (x,y). Una dimensión de la matriz podría almacenar la coordenada `x` y otra la coordenada `y`.

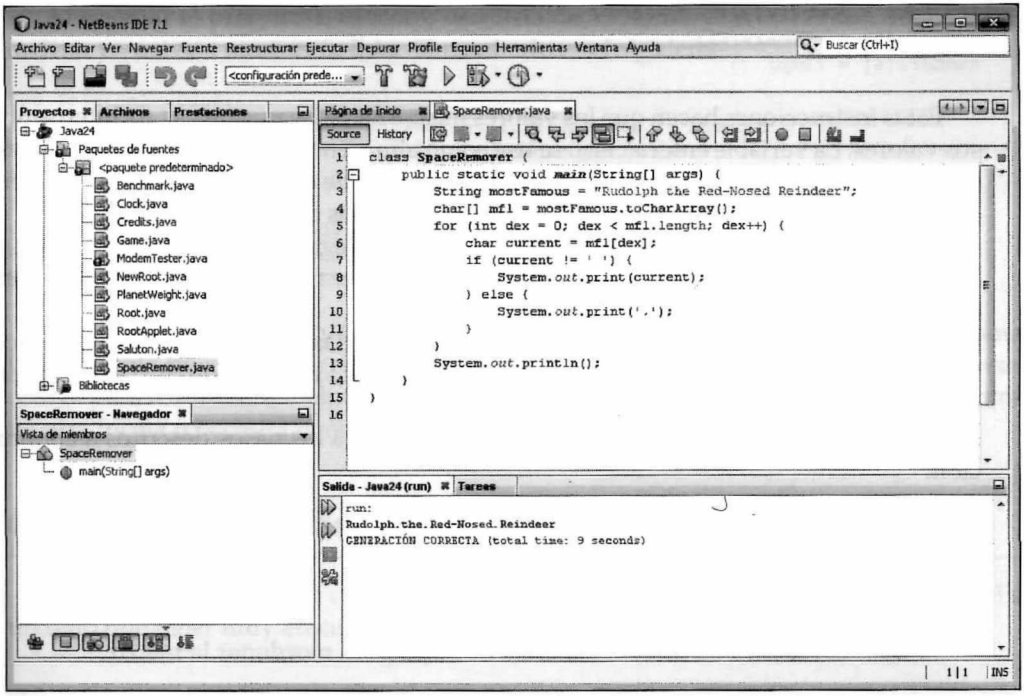


Figura 9.1. Resultado del programa SpaceRemover.

Para crear una matriz de dos dimensiones, debe emplear un grupo adicional de corchetes, como en las siguientes instrucciones:

```
boolean[][] selectedPoint = new boolean[50][50];
selectedPoint[4][13] = true;
selectedPoint[7][6] = true;
selectedPoint[11][22] = true;
```

Este ejemplo crea la matriz de valores Booleanos `selectedPoint`. Tiene 50 elementos en la primera dimensión y otros 50 en la segunda, de modo que 2.500 elementos de matriz pueden almacenar valores (50 por 50). Al crear la matriz, cada elemento recibe el valor predeterminado `false`. Tres elementos reciben el valor `true`: un punto en las posiciones (x,y) 4.13, 7.6 y 11.22. Las matrices pueden tener todas las dimensiones que necesite pero recuerde que ocupan mucha memoria si son de gran tamaño. La creación de la matriz `selectedPoint` de 50 por 50 equivale a crear 2.500 variables.

Ordenar una matriz

Si agrupa un conjunto de elementos similares en una matriz, puede reorganizar dichos elementos. Las siguientes instrucciones intercambian los valores de dos elementos de la matriz `numbers`:

```
int temp = numbers[7];
numbers[5] = numbers[6];
numbers[6] = temp;
```

Estas instrucciones hacen que los valores `numbers[5]` y `numbers[6]` intercambien sus valores. La variable entera `temp` se usa como almacén temporal de uno de los valores intercambiados. Ordenar permite organizar una lista de elementos relacionados en un orden concreto, como por ejemplo una lista de números de menor a mayor.

Santa Claus podría ordenar los destinatarios de sus regalos por apellido, de forma que Willie Aames y Hank Aaron serían los primeros y Dweezil Zappa y Jim Zorn ocuparían un desafortunado último lugar.

En Java es muy fácil ordenar una matriz ya que la clase `Arrays` se encarga de todo. `Arrays`, que forma parte del grupo de clases `java.util`, puede organizar matrices de todos los tipos de variables.

Para emplear la clase `Arrays` en un programa, siga los pasos descritos a continuación:

1. Use la instrucción `import java.util.*` para obtener todas las clases de `java.util`.
2. Cree la matriz.
3. Use el método `sort()` de la clase `Arrays` para reordenar la matriz.

Una matriz de variables ordenada por la clase `Arrays` se organiza en orden numérico ascendente. Los caracteres y las cadenas se organizan en orden alfabético.

Para verlo en funcionamiento, cree un nuevo archivo vacío de Java con el nombre `Name` e introduzca el texto del listado 9.2, un breve programa para ordenar nombres.

Listado 9.2. Código completo de Name.java.

```
 1: import java.util.*;
 2:
 3: class Name {
 4:   public static void main(String[] args) {
 5:     String names[] = { "Lauren", "Audrina", "Heidi", "Whitney",
 6:       "Stephanie", "Spencer", "Lisa", "Brody", "Frankie",
 7:       "Holly", "Jordan", "Brian", "Jason" };
 8:     System.out.println("The original order:");
 9:     for (int i = 0; i < names.length; i++) {
10:       System.out.print(i + ": " + names[i] + " ");
11:     }
12:     Arrays.sort(names);
13:     System.out.println("\nThe new order:");
14:     for (int i = 0; i < names.length; i++) {
15:       System.out.print(i + ": " + names[i] + " ");
16:     }
17:     System.out.println();
18:   }
19: }
```

Al ejecutar el programa, muestra una lista de 13 nombres en su orden original, los ordena y vuelve a mostrar la lista. El resultado es el siguiente:

```
The original order:
0: Lauren 1: Audrina 2: Heidi 3: Whitney 4: Stephanie 5: Spencer
6: Lisa 7: Brody 8: Frankie 9: Holly 10: Jordan 11: Brian
12: Jason
The new order:
0: Audrina 1: Brian 2: Brody 3: Frankie 4: Heidi 5: Holly
6: Jason 7: Jordan 8: Lauren 9: Lisa 10: Spencer 11: Stephanie 12:
Whitney
```

Al trabajar con cadenas y tipos básicos de variables como números enteros y de coma flotante, solo puede ordenarlos de forma ascendente con la clase `Arrays`. Puede crear código para realizar manualmente su propia ordenación si quiere organizar los elementos de otra forma o necesita mayor eficacia que la proporcionada por la clase `Arrays`.

Contar caracteres de cadenas

Las letras más comunes en inglés son E, R, S, T, L, N, C, D, M y O, en este orden. Es un dato que tener muy en cuenta si alguna vez participa en la Ruleta de la Fortuna.

Nota

Si no conoce el programa, la Ruleta de la Fortuna es un juego en el que tres concursantes deben adivinar las letras de una frase, un nombre o una cita. Si adivinan una consonante, ganan la cantidad de dinero indicada al girar una ruleta. Si quiere recrear la experiencia, puede jugar al ahorcado con sus amigos delante de numeroso público, entregar cantidades aleatorias de dinero cuando alguien adivine una letra y darle un premio al ganador.

El siguiente programa cuenta la frecuencia de letras en distintas frases y expresiones. Se usa una matriz para contar las veces que aparece cada letra. Cuando termine, el programa muestra el número de veces que ha aparecido cada letra en las frases.

Cree un nuevo archivo vacío de Java en NetBeans con el nombre `Wheel.java`, añada el código del listado 9.3 y guarde el archivo cuando termine. Puede añadir las frases que desee entre las líneas 17 y 18, con el mismo formato que la línea 17.

Listado 9.3. Código completo de Wheel.java.

```
1: class Wheel {
2:   public static void main(String[] args) {
3:     String phrase[] = {
4:       "A STITCH IN TIME SAVES NINE",
5:       "DON'T EAT YELLOW SNOW",
6:       "JUST DO IT",
```

```
 7:          "EVERY GOOD BOY DOES FINE",
 8:          "I WANT MY MTV",
 9:          "I LIKE IKE",
10:          "PLAY IT AGAIN, SAM",
11:          "FROSTY THE SNOWMAN",
12:          "ONE MORE FOR THE ROAD",
13:          "HOME FIELD ADVANTAGE",
14:          "VALENTINE'S DAY MASSACRE",
15:          "GROVER CLEVELAND OHIO",
16:          "SPAGHETTI WESTERN",
17:          "AQUA TEEN HUNGER FORCE",
18:          "IT'S A WONDERFUL LIFE"
19:       };
20:       int[] letterCount = new int[26];
21:       for (int count = 0; count < phrase.length; count++) {
22:          String current = phrase[count];
23:          char[] letters = current.toCharArray();
24:          for (int count2 = 0; count2 < letters.length; count2++) {
25:             char lett = letters[count2];
26:             if ( (lett >= 'A') & (lett <= 'Z') ) {
27:                letterCount[lett - 'A']++;
28:             }
29:          }
30:       }
31:       for (char count = 'A'; count <= 'Z'; count++) {
32:          System.out.print(count + ": " +
33:             letterCount[count - 'A'] +
34:             " ");
35:       }
36:       System.out.println();
37:    }
38: }
```

Si ejecuta el programa sin añadir frases propias, el resultado será similar al ilustrado en el listado 9.4.

Listado 9.4. Resultado del programa Wheel.

```
A: 22 B: 1 C: 4 D: 10 E: 33 F: 7 G: 6 H: 7 I: 18
J: 1 K: 2 L: 10 M: 8 N: 19 O: 20 P: 2 Q: 1 R: 12
S: 15 T: 20 U: 4 V: 7 W: 6 X: 0 Y: 7 Z: 0
```

Veamos qué sucede en el programa `Wheel`:

- **Líneas 3-19:** Las frases se almacenan en la matriz de cadena `phrase`.
- **Línea 20:** Se crea la matriz entera `letterCount` con 26 elementos. Se emplea para almacenar cuántas veces aparece cada letra. El orden de los elementos es de A a Z. `letterCount[0]` almacena la cantidad de la letra A, `letterCount[1]` la cantidad de B y así sucesivamente hasta `letterCount[25]` para la Z.
- **Línea 21:** Un bucle `for` itera por las frases almacenadas en la matriz `phrase`. Se usa la variable `phrase.length` para finalizar el bucle una vez alcanzada la última frase.

- **Línea 22:** Se establece la variable de cadena `current` con el valor del elemento actual de la matriz `phrase`.
- **Línea 23:** Se crea una matriz de caracteres que almacena todos los caracteres de la frase actual.
- **Línea 24:** Un bucle `for` itera por las letras de la frase actual. Se emplea la variable `letters.length` para finalizar el bucle una vez alcanzada la última letra.
- **Línea 25:** Se crea la variable de carácter `lett` con el valor de la letra actual. Además de su valor de texto, los caracteres tienen un valor numérico. Como los elementos de una matriz están numerados, el valor de cada carácter se usa para determinar su número de elemento.
- **Líneas 26-28:** Una instrucción `if` descarta todos los caracteres no alfabéticos, como signos de puntuación y espacios. Un elemento de la matriz `letterCount` se incrementa en `1` en función del valor numérico del carácter actual, almacenado en `lett`. Los valores numéricos del alfabeto oscilan entre `65` para `'A'` y `90` para `'Z'`. Como la matriz `letterCount` comienza en `0` y termina en `25`, `'A' (65)` se resta de `lett` para determinar qué elemento de la matriz incrementar.
- **Línea 31:** Un bucle `for` itera por el alfabeto de `'A'` a `'Z'`.
- **Líneas 32-34:** Se muestra la letra actual más punto y coma, y el número de veces que aparece en las frases almacenada en la matriz `phrase`.

Este proyecto muestra cómo emplear dos bucles `for` anidados para iterar por un grupo de frases letra a letra. Java añade un valor numérico a cada carácter, valor que es más fácil de usar que el carácter interno de las matrices.

Nota

Los valores numéricos asociados a cada uno de los caracteres de la A a la Z son los empleados por el conjunto de caracteres ASCII. Forma parte de Unicode, el conjunto de caracteres completo admitido por el lenguaje Java. Unicode admite más de 60.000 caracteres distintos usados en los lenguajes escritos mundiales. ASCII se limita a 256.

Resumen

Las matrices permiten almacenar tipos complicados de información en un programa y manipularla. Son perfectas para todo lo que tenga que reorganizar en una lista y permiten el acceso mediante instrucciones de bucle.

Para ser honestos, las necesidades de procesamiento de información de Santa Claus han superado las matrices. Cada año se manufacturan más niños y los regalos que piden son cada vez más complejos y caros.

Seguramente sus programas usen matrices para almacenar información que resultaría incontrolable con tan solo variables, aunque no confeccione listas ni las compruebe dos veces.

Preguntas y respuestas

P: ¿El intervalo numérico del alfabeto, de 65 a 90, forma parte de Java? En caso afirmativo, ¿para qué se reserva del 1 al 64?

R: Los números del 1 al 64 incluyen numerales, signos de puntuación y caracteres que no se imprimen, como saltos de línea, nuevas líneas y retroceso. Se asocia un número a cada carácter imprimible que se puede emplear en un programa de Java, y también a algunos que no se pueden imprimir. Java usa el sistema de numeración Unicode. Los 127 primeros caracteres se corresponden al conjunto de caracteres ASCII, que seguramente haya empleado en otros lenguajes de programación.

P: ¿Por qué algunos errores se denominan excepciones?

R: El significado del término es que un programa se suele ejecutar sin problemas y la excepción indica una circunstancia especial que solucionar. Las excepciones son mensajes de advertencia enviados desde un programa de Java. En el lenguaje Java, el término error suele limitarse a problemas en el intérprete que ejecuta el programa. Encontrará más información al respecto en el capítulo 18.

P: En una matriz multidimensional, ¿se puede usar la variable `length` para medir otras dimensiones que no sean la primera?

R: Puede probar cualquier dimensión de una matriz. Para la primera, use `length` con el nombre de la matriz, como en `x.length`. Las siguientes dimensiones se pueden medir por medio de `length` con el elemento `[0]` de la correspondiente dimensión. Imagine la matriz `data` creada con la siguiente instrucción:

```
int[][][] data = new int[12][13][14];
```

Las dimensiones de esta matriz se pueden medir con la variable `data.length` para la primera dimensión, `data[0].length` para la segunda y `data[0][0].length` para la tercera.

Ejercicios

Si su mente fuera una matriz, podría probar su longitud respondiendo a las siguientes preguntas sobre matrices.

Preguntas

1. ¿Para qué tipo de información puede emplear una matriz?
   A. Listas.
   B. Pares de información relacionada.
   C. Preguntas.
2. ¿Qué variable puede usar para comprobar el límite superior de una matriz?
   A. top
   B. length
   C. limit
3. ¿Cuántos renos tiene Santa Claus, incluido Rudolph?
   A. 8
   B. 9
   C. 10

Respuestas

1. A. Las listas que contienen el mismo tipo de información, como cadenas, números y demás, resultan muy adecuadas para almacenar en matrices.
2. B. La variable length contiene el número de elementos de una matriz.
3. B. Santa Claus tenía "ocho pequeños renos", según *A Visit from St. Nicholas* de Clement Clarke Moore, de modo que con Rudolph son nueve.

Actividades

Si desea ampliar sus conocimientos sobre matrices, pruebe con las siguientes actividades:

- Cree un programa que use una matriz multidimensional para almacenar notas de alumnos. La primera dimensión debe ser un número para cada alumno y la segunda la nota. Muestre la media de todas las notas de cada alumno y la media general.
- Cree un programa que almacene en una matriz los 400 primeros números múltiplos de 13.

10
Crear el primer objeto

Uno de los términos más temibles que encontrará en el libro es POO (Programación orientada a objetos). Describe, de forma elegante, qué es un programa informático y cómo funciona.

Antes de la POO, un programa informático se solía describir con la sencilla definición que hemos visto en el libro: una serie de instrucciones enumeradas en un archivo y procesadas de forma fiable.

Al pensar en un programa como en una colección de objetos, puede determinar las tareas que debe realizar y asignarlas a los objetos más indicados.

Funcionamiento de la programación orientada a objetos

Puede pensar que los programas de Java que cree son objetos, similares a los objetos físicos del mundo real. Los objetos existen de forma independiente a resto, interactúan de forma concreta y se pueden combinar con otros objetos para crear algo mayor. Si piensa en un programa informático como en un grupo de objetos que interactúan entre sí, puede diseñar un programa más fiable, fácil de entender y reutilizable en otros proyectos.

En el capítulo 23, crearemos un programa de Java que muestre gráficos circulares, con porciones de diferentes colores para representar datos (véase la figura 10.1). Un gráfico circular es un objeto formado por otros más pequeños: porciones individuales de distintos colores, una leyenda que identifica cada porción y un título.

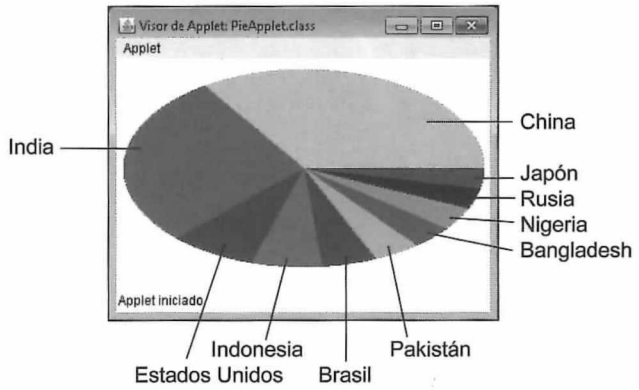


Figura 10.1. Programa de Java que muestra un gráfico circular.

Cada objeto cuenta con rasgos que lo diferencian de otros objetos. Los gráficos circulares son redondos, mientras que los gráficos de barras representan datos en forma de rectángulo. Si diseccionara un programa informático como si fuera un gráfico circular, estará recurriendo a la POO. En POO, un objeto tiene dos rasgos: atributos y comportamiento. Los atributos describen el objeto y muestran las diferencias con respecto a otros objetos. El comportamiento es lo que hace el objeto.

En Java, los objetos se crean mediante una clase como plantilla. Una clase es una copia maestra del objeto que determina los atributos y el comportamiento que debe tener el objeto. El término clase ya debería resultarle familiar porque los programas de Java se denominan clases.

Todos los programas que cree en Java son una clase que puede usar como plantilla para crear nuevos objetos. Por ejemplo, un programa que use cadenas emplea objetos creados con la clase `String`. Esta clase contiene atributos que determinan qué es un objeto `String` y un comportamiento que controla lo que los objetos `String` pueden hacer.

Con la POO, un programa informático es un grupo de objetos que funcionan conjuntamente para realizar una acción. Algunos programas sencillos pueden parecer que solo tengan un objeto: el archivo de clase. Sin embargo, incluso esos programas usan otros objetos para realizar sus acciones.

Objetos en funcionamiento

Imagine un programa que muestra un gráfico circular. Un objeto `PieChart` podría estar formado por lo siguiente:

- Un comportamiento para calcular el tamaño de cada porción.
- Un comportamiento para dibujar el gráfico.
- Un atributo para mostrar el título del grafico.

Le parecerá extraño pedirle al objeto `PieChart` que se dibuje ya que en el mundo real los gráficos no se dibujan solos. En POO; los objetos funcionan de forma independiente siempre que es posible. Esta cualidad facilita su incorporación a otros programas. Si un objeto `PieChart` no supiera dibujarse, cada vez que lo empleara en otro programa, tendría que crear un comportamiento para dibujarlo.

Veamos otro ejemplo. Imagine el programa autollamador que el personaje de Matthew Broderick usaba en la película *Juegos de Guerra* para localizar ordenadores a los que acceder.

Hoy en día, un autollamador atraería la atención de las operadoras de telefonía o de la policía. En 1980, era una forma de rebeldía sin salir de casa. David Lightman (el personaje de Broderick) empleaba el autollamador para localizar el sistema informático privado de una empresa de videojuegos (quería jugar al nuevo juego de la empresa antes de que lo comercializaran). Pero lo que encontró fue un ordenador secreto del gobierno que permitía jugar a todo, desde el ajedrez a la guerra termonuclear global.

Un autollamador, como cualquier programa informático, se puede imaginar como un grupo de objetos que funcionan conjuntamente. Se podría dividir en los siguientes elementos:

- Un objeto `Modem`, que conoce sus atributos como velocidad y que tiene un comportamiento, por ejemplo, puede marcar un número y detectar el sistema informático que ha respondido a la llamada.
- Un objeto `Monitor`, que controla los números a los que se llama y cuáles responden.

Cada objeto es independiente del otro.

Nota

Un autollamador es software que usa un módem para marcar secuencialmente varios números de teléfono. El objetivo del programa es buscar otros ordenadores que respondan a la llamada, para después llamarlos y comprobar qué son.

Una de las ventajas de diseñar un objeto `Modem` totalmente independiente es que se puede emplear en otros programas que necesiten la misma funcionalidad.

Otro motivo para usar objetos independientes es que son más fáciles de depurar. Los programas informáticos suelen aumentar rápidamente de tamaño. Si tiene que depurar algo como un objeto `Modem` y sabe que no depende de nada más, puede centrarse en garantizar que el objeto `Modem` realiza la tarea que debe y que contiene la información adecuada.

Aprender un lenguaje orientado a objetos como Java como primer lenguaje de programación puede ser una ventaja, ya que no afecta a los hábitos de otros estilos de programación.

Qué son los objetos

Los objetos se crean por medio de una clase de objetos como plantilla. La siguiente instrucción crea una clase:

```
public class Modem {
}
```

Un objeto creado a partir de esta clase no hace nada ya que carece de atributos y comportamiento. Tendrá que añadirlos para que la clase sea útil, como en las siguientes instrucciones:

```
public class Modem {
   int speed;

   public void displaySpeed() {
      System.out.println("Speed: " + speed);
   }
}
```

La clase `Modem` empieza a parecerse a las que hemos creado en capítulos anteriores. Comienza con una instrucción `class` pero incluye `public`, lo que significa que está disponible públicamente, es decir, para cualquier programa que quiera emplear objetos `Modem`. La primera parte de la clase `Modem` crea la variable entera `speed`, que es un atributo del objeto. La segunda parte de la clase es el método `displaySpeed()`. Este método forma parte del comportamiento del objeto. Contiene una instrucción, `System.out.println()`, que muestra el valor `speed` del módem.

Las variables de un objeto suelen denominarse variables de instancia o variables miembro. Si desea usar un objeto `Modem` en su programa, debe crearlo con:

```
Modem device = new Modem();
```

Esta instrucción crea un objeto `Modem` con el nombre `device`. Una vez creado el objeto, puede establecer sus variables e invocar sus métodos. Veamos cómo establecer el valor de la variable `speed` del objeto `device`:

```
device.speed = 28800;
```

Para que este módem muestre su velocidad invocando el método `displaySpeed()`, debe invocar el método `device.displaySpeed()`.

El objeto `device` responderá a esta instrucción mostrando el texto `Speed: 28800`.

Herencia

Una de las principales ventajas de la POO es la herencia, que permite a un objeto heredar los atributos y el comportamiento de otro. Al empezar a crear objetos, en ocasiones el que necesita es muy similar a otro que ya tiene.

Imagine que David Lightman necesitara un objeto que pudiera corregir errores y tuviera modernas funciones de módem inimaginables en 1983, cuando apareció *Juegos de Guerra*. Lightman podría crear un nuevo objeto `ErrorCorrectionModem` copiando y revisando las instrucciones del objeto `Modem`. Pero si la mayor parte del comportamiento y los atributos de `ErrorCorrectionModem` coinciden con las de `Modem`, es un esfuerzo innecesario. También significa que Lightman tendría que actualizar dos programas diferentes para realizar cambios.

Gracias a la herencia, un programador puede crear una nueva clase de objetos si define sus diferencias con respecto a una clase existente. Lightman podría haber heredado `ErrorCorrectionModem` de `Modem`, y solo tendría que crear los elementos que distinguen a los módem con corrección de errores de los módem estándar. Una clase de objetos se hereda de otra por medio de la instrucción `extends`. El siguiente ejemplo muestra el esqueleto de una clase `ErrorCorrectionModem` heredada de la clase `Modem`:

```
public class ErrorCorrectionModem extends Modem {
   // añadir aquí el programa
}
```

Crear una jerarquía de herencia

La herencia, que permite desarrollar diferentes clases relacionadas sin duplicar esfuerzos, posibilita pasar código de una clase a otra. Esta agrupación de clases se denomina jerarquía de clases y todas las clases estándar que puede emplear en sus programas de Java forman parte de una jerarquía. Le resultará más fácil comprender las jerarquías si entiende los conceptos de subclases y superclases. Una clase heredada de otra se denomina subclase. La clase de la que se hereda se denomina superclase.

En el ejemplo anterior, la clase `Modem` es la superclase de `ErrorCorrectionModem` que, a su vez, es la subclase de `Modem`.

Una clase puede tener más de una clase de herencia en la jerarquía; otra subclase de `Modem` podría ser `ISDNModem`, ya que los módem ISDN tienen comportamientos y atributos que los diferencian de los módem de corrección de errores. Si existiera una subclase de `ErrorCorrectionModem` como `InternalErrorCorrectionModem`, heredaría todas las clases superiores de la jerarquía, tanto `ErrorCorrectionModem` como `Modem`. En la figura 10.2 puede ver estas jerarquías. Las clases que forman el lenguaje Java se basan en la herencia, por lo que es fundamental entender este concepto. En el capítulo 12 encontrará más información al respecto.

Nota

Cuando un método como `System.out.println()` requiere un argumento de cadena, puede usar el operador + para combinar distintos tipos de información en el argumento. Mientras uno de los elementos combinados sea una cadena, el argumento combinado se convertirá en cadena.

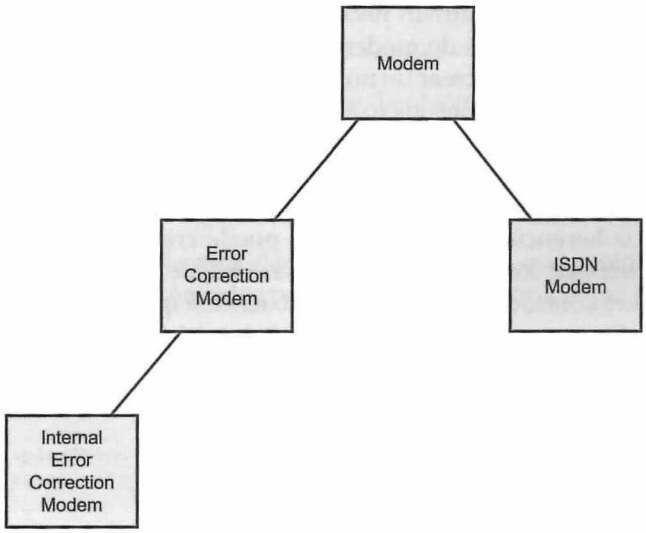


Figura 10.2. Ejemplo de jerarquía de clases.

Convertir objetos y variables sencillas

Una de las tareas más habituales en Java es convertir información de un formato a otro. Puede realizar distintas conversiones, como las siguientes:

- Convertir un objeto en otro.
- Convertir una variable sencilla en otro tipo de variable.
- Emplear un objeto para crear una variable sencilla.
- Usar una variable sencilla para crear un objeto.

Las variables sencillas son los tipos de datos básicos que vimos en el capítulo 5, como `int`, `float`, `char`, `long` y `double`.

Al emplear un método o una expresión en un programa, debe usar el tipo de información correcto que esperan dichos métodos y expresiones. Un método que espera un objeto `Calendar` debe recibir un objeto `Calendar`. Si emplea un método que adopta un único argumento entero y envía un número de coma flotante, se producirá un error al intentar compilar el programa.

Esta operación se denomina conversión. El proceso genera un nuevo valor de un tipo de variable u objeto diferente al original. Al convertir no se cambia el valor, sino que se crea una nueva variable u objeto en el formato necesario.

Los términos origen y destino son muy útiles para describir el concepto de conversión. El origen es un tipo de información en formato original, ya sea una variable o un objeto. El destino es la versión convertida del origen al nuevo formato.

Convertir variables sencillas

Con las variables sencillas, la conversión se suele realizar entre variables numéricas como números enteros y de coma flotante. Un tipo de variable que no se puede usar en conversiones son los valores Booleanos.

Para convertir información a un nuevo formato, debe precederla con el nuevo formato entre paréntesis. Por ejemplo, si desea convertir algo en una variable `long`, debe precederlo con `(long)`. Las siguientes instrucciones convierten un valor `float` en `int`:

```
float source = 7.06F;
int destination = (int) source;
```

En la conversión de variables, si el destino tiene valores mayores que el origen, el valor se convierte fácilmente, como al convertir `byte` en `int`. Un `byte` contiene valores entre -128 y 127, mientras que un `int` contiene valores entre -2.1 y 2.1 mil millones. Independientemente del contenido de la variable `byte`, la nueva variable `int` tendrá capacidad suficiente para almacenarlo.

En ocasiones puede emplear una variable en otro formato sin necesidad de convertirla. Por ejemplo, puede usar variables `char` como si fueran variables `int`. Asimismo, puede emplear variables `int` como si fueran variables `long`, y puede usar lo que desee como `double`.

En muchos casos, como el destino ofrece más espacio que el origen, la información se convierte sin cambiar el valor. Las principales excepciones se producen al convertir una variable `int` o `long` a `float`, o `long` a `double`.

Al convertir información de un tipo de variable mayor a otro menor, debe hacerlo explícitamente, como en las siguientes instrucciones:

```
int xNum = 103;
byte val = (byte) xNum;
```

Aquí, se convierte el valor entero `xNum` a la variable `byte val`. Es un ejemplo en el que la variable de destino contiene un menor intervalo de valores que la de origen. `byte` contiene valores enteros entre -128 y 127, e `int` contiene un intervalo mucho mayor de valores enteros.

Cuando la variable de origen de una operación de conversión tiene un valor no permitido en la variable de destino, Java cambia el valor para que se pueda realizar la conversión. Esto puede producir resultados inesperados.

Convertir objetos

Puede convertir un objeto en otro cuando el origen y el destino se relacionan mediante herencia. Una clase debe ser una subclase de la otra. Algunos objetos no requieren conversión. Puede emplear un objeto donde se espere cualquiera de sus superclases. Todos los objetos de Java son subclases de la clase `Object`, de modo que puede usar cualquier objeto como argumento siempre que se espere `Object`.

También puede emplear un objeto cuando se espere una de sus subclases. Sin embargo, como las subclases suelen contener más información que sus superclases, puede perder parte de esta información. Si el objeto carece de un método que la subclase debería incluir, se genera un error si el método que falta se usa en el programa.

Para emplear un objeto en lugar de una de sus subclases, debe convertirlo explícitamente con instrucciones como la siguiente:

```
Graphics comp = new Graphics();
Graphics2D comp2D = (Graphics2D) comp;
```

Se convierte el objeto `Graphics comp` a un objeto `Graphics2D`. No se pierde información en la conversión y se obtienen todos los métodos y variables que define la subclase.

Convertir variables sencillas en objetos y viceversa

No se puede convertir un objeto en una variable sencilla o una variable sencilla en un objeto. Java cuenta con clases para cada uno de los tipos de variables sencillas, como `Boolean`, `Byte`, `Character`, `Double`, `Float`, `Integer`, `Long` y `Short`. Usan mayúscula porque son objetos, no variables.

Con los métodos definidos en estas clases puede crear un objeto empleando el valor de la variable como argumento. La siguiente instrucción crea un objeto `Integer` con el valor `5309`:

```
Integer suffix = new Integer(5309);
```

Una vez creado este objeto, puede usarlo normalmente. Si desea volver a emplear el valor como variable sencilla, la clase tiene métodos para realizar la conversión. Para obtener un valor `int` del objeto `suffix` anterior, puede usar la siguiente instrucción:

```
int newSuffix = suffix.intValue();
```

Esta instrucción hace que la variable `newSuffix` tenga el valor `5309`, expresado como `int`. Una conversión habitual de objeto a variable consiste en emplear una cadena en una expresión numérica. Cuando el valor de la cadena puede ser un entero, se puede realizar por medio del método `parseInt()` de la clase `Integer`, como en este ejemplo:

```
String count = "25";
int myCount = Integer.parseInt(count);
```

De esta forma se convierte una cadena con el texto `"25"` a un entero con el valor `25`. Si el valor de cadena no es un entero válido, la conversión no funciona.

El siguiente proyecto es una aplicación que convierte un valor de cadena de un argumento de línea de comandos en un valor numérico, una técnica muy habitual al aceptar entradas del usuario a través de la línea de comandos.

En el proyecto `Java24` de NetBeans, seleccione Archivo>Archivo Nuevo y cree un nuevo archivo vacío de Java con el nombre `NewRoot`. Introduzca el texto del listado 10.1 en el editor de código fuente y guarde el archivo.

Listado 10.1. Texto completo de NewRoot.java.

```
1: class NewRoot {
2:   public static void main(String[] args) {
3:      int number = 100;
4:      if (args.length > 0) {
5:         number = Integer.parseInt(args[0]);
6:      }
7:      System.out.println("The square root of "
8:         + number
9:         + " is "
10:        + Math.sqrt(number) );
11:  }
12: }
```

Antes de ejecutar el programa, debe configurar NetBeans para ejecutarlo con un argumento de línea de comandos. Seleccione Ejecutar>Establecer la configuración del proyecto>Personalizar. Se abrirá la ventana Propiedades del proyecto. Introduzca **NewRoot** en Main Class (Clase principal) y **169** en el campo Argumentos. Pulse **Aceptar** para cerrar el cuadro de diálogo.

Para ejecutar el programa, seleccione Ejecutar>Ejecutar Proyecto Principal (en lugar de Ejecutar>Ejecutar archivo). El programa muestra el número y su raíz cuadrada (véase la figura 10.3).

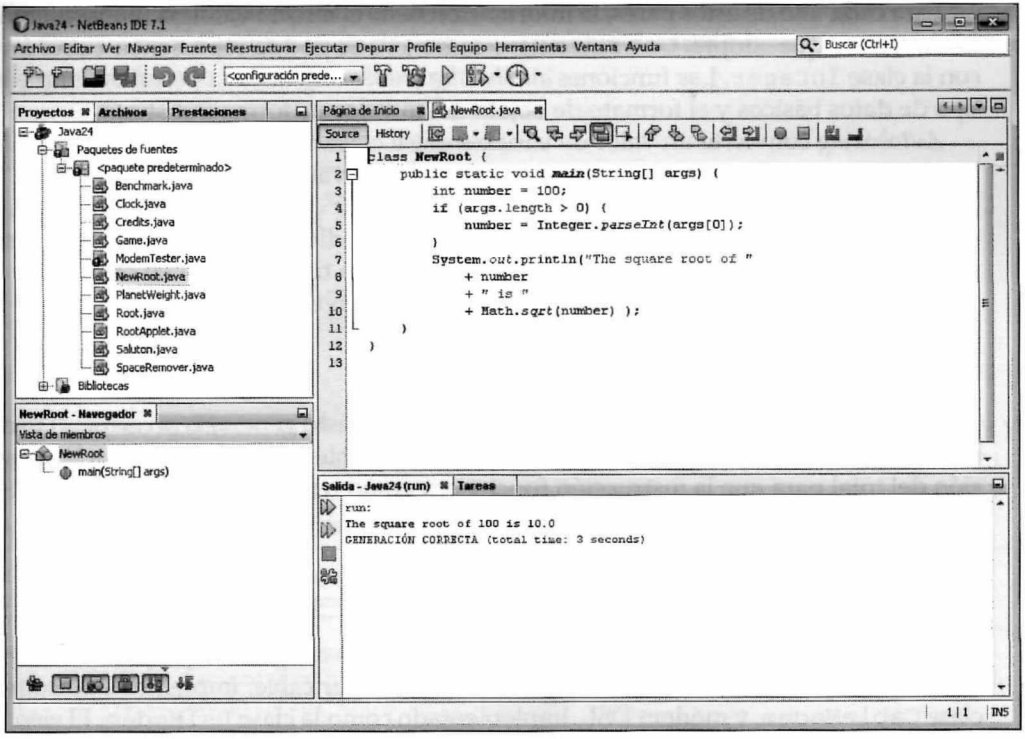


Figura 10.3. Resultado del programa NewRoot.

La aplicación `NewRoot` es una ampliación del ejercicio del capítulo 4 que mostraba la raíz cuadrada del entero 225.

El programa hubiera sido más útil si aceptara un número proporcionado por el usuario y mostrara su raíz cuadrada. Para ello hay que convertir una cadena en un entero. Todos los argumentos de línea de comandos se almacenan como elementos de una matriz `String`, de modo que debe convertirlos a números antes de poder usarlos en expresiones matemáticas.

Para crear un valor entero basado en los contenidos de una cadena, se invoca el método `Integer.parseInt()` con la cadena como único argumento, como en la línea 5:

```
number = Integer.parseInt(args[0]);
```

El elemento `args[0]` contiene el primer argumento de línea de comandos enviado al ejecutar la aplicación. Al ejecutar el programa con `169` como argumento, la cadena `"169"` se convierte al entero `169`.

Autoboxing y unboxing

Todos los tipos de datos básicos de Java cuentan con la correspondiente clase de objeto: `boolean` (`Boolean`), `byte` (`Byte`), `char` (`Character`), `double` (`Double`), `float` (`Float`), `int` (`Integer`), `long` (`Long`) y `short` (`Short`).

Para cada uno de estos pares, la información tiene el mismo valor; la única diferencia es el formato que adopta. Un valor entero como `413` se podría representar como `int` o con la clase `Integer`. Las funciones *autoboxing* y *unboxing* de Java permiten emplear el tipo de datos básicos y el formato de objeto de un valor de forma indistinta.

Autoboxing convierte un valor de variable en su clase correspondiente.

Unboxing convierte un objeto a su correspondiente valor sencillo.

Estas funciones se realizan entre bastidores, para que cuando espere un tipo de datos sencillo como `float`, se convierta un objeto al correspondiente tipo de datos con el mismo valor. Si espera un objeto como `Float`, se convierte un tipo de datos en objeto.

Las siguientes instrucciones muestran estas dos conversiones:

```
Float total = new Float(1.3F);
float sum = total / 5;
```

En versiones anteriores de Java (antes de Java 1.5), esto generaría un error; el uso de un objeto `Float` en la segunda instrucción no era posible. Ahora Java realiza la conversión del total para que la instrucción funcione, y como resultado `sum` es igual a `0.26`.

Crear un objeto

Para ver un ejemplo funcional de clases y herencia, en el siguiente proyecto crearemos clases que representan dos tipos de objetos: módem por cable, implementado como la clase `CableModem`, y módem DSL, implementado como la clase `DslModem`. El ejercicio se centra en los atributos y el comportamiento de estos objetos:

- Cada objeto debe tener una velocidad que mostrar.
- Cada objeto debe poder conectarse a Internet.

Un rasgo común de ambos objetos es que tienen velocidad. Al compartir este rasgo, se puede incluir en una clase que sea la superclase de las clases `CableModem` y `DslModem`. Asigne el nombre `Modem` a esta clase. En NetBeans, cree un nuevo archivo vacío de Java con el nombre `Modem`. Introduzca el texto del listado 10.2 y guarde el archivo.

Listado 10.2. Texto completo de Modem.java.

```
1: public class Modem {
2:    int speed;
3:
4:    public void displaySpeed() {
5:       System.out.println("Speed: " + speed);
6:    }
7: }
```

Este archivo se compila automáticamente como `Modem.class`. No puede ejecutar directamente el programa pero puede usarlo en otras clases. La clase `Modem` puede procesar un elemento que las clases `CableModem` y `DslModem` tienen en común. Al emplear la instrucción `extends` al crear las clases `CableModem` y `DslModem`, puede convertirlas en subclases de `Modem`.

Cree un nuevo archivo vacío de Java con el nombre de clase `CableModem`, introduzca el texto del listado 10.3 y guarde el archivo.

Listado 10.3. Texto completo de CableModem.java.

```
1: public class CableModem extends Modem {
2:    String method = "cable connection";
3:
4:    public void connect() {
5:       System.out.println("Connecting to the Internet ...");
6:       System.out.println("Using a " + method);
7:    }
8: }
```

Cree otro archivo con el nombre `DslModem`, introduzca el texto del listado 10.4 y guarde el archivo.

Listado 10.4. Texto completo de DslModem.java.

```
1: public class DslModem extends Modem {
2:    String method = "DSL phone connection";
3:
4:    public void connect() {
5:       System.out.println("Connecting to the Internet ...");
6:       System.out.println("Using a " + method);
7:    }
8: }
```

Si no se producen errores, tendrá tres archivos de clase: `Modem.class`, `CableModem.class` y `DslModem.class`. Sin embargo, no podrá ejecutarlos ya que carecen de bloques `main()` como los de otros programas que ha creado. Tendrá que crear una aplicación para probar esta jerarquía de clases. En NetBeans, cree un nuevo archivo vacío de Java con el nombre de clase `ModemTester`. Introduzca el texto del listado 10.5.

Listado 10.5. Texto completo de ModemTester.java

```
 1: public class ModemTester {
 2:   public static void main(String[] args) {
 3:      CableModem surfBoard = new CableModem();
 4:      DslModem gateway = new DslModem();
 5:      surfBoard.speed = 500000;
 6:      gateway.speed = 400000;
 7:      System.out.println("Trying the cable modem:");
 8:      surfBoard.displaySpeed();
 9:      surfBoard.connect();
10:      System.out.println("Trying the DSL modem:");
11:      gateway.displaySpeed();
12:      gateway.connect();
13:   }
14: }
```

Al ejecutar el programa obtendrá el resultado mostrado en la figura 10.4.
En el listado 10.5 se producen las siguientes acciones:

- **Líneas 3-4:** Se crean dos objetos: un objeto `CableModem` con el nombre `surfBoard` y un objeto `DslModem` con el nombre `gateway`.
- **Línea 5:** La variable `speed` del objeto `CableModem` `surfBoard` se establece en `500000`.
- **Línea 6:** La variable `speed` del objeto `DslModem` `gateway` se establece en `400000`.
- **Línea 8:** Se invoca el método `displaySpeed()` del objeto `surfBoard`. Este método se hereda de `Modem`, aunque no se encuentre en la clase `CableModem` puede invocarlo.
- **Línea 9:** Se invoca el método `connect()` del objeto `surfBoard`.
- **Línea 11:** Se invoca el método `displaySpeed()` del objeto `gateway`.
- **Línea 12:** Se invoca el método `connect()` del objeto `gateway`.

Resumen

Tras crear su primera clase de objetos y organizar jerárquicamente diversas clases, se sentirá más cómodo con la POO. En los dos siguientes capítulos aprenderá más sobre comportamientos y atributos de objetos, y creará objetos más sofisticados. Términos

como programa, clase y objeto cobran sentido al ampliar su experiencia en este tipo de desarrollo. La POO es un concepto que lleva tiempo. Cuando lo domine, dispondrá de una forma eficaz de diseñar, desarrollar y depurar programas informáticos.

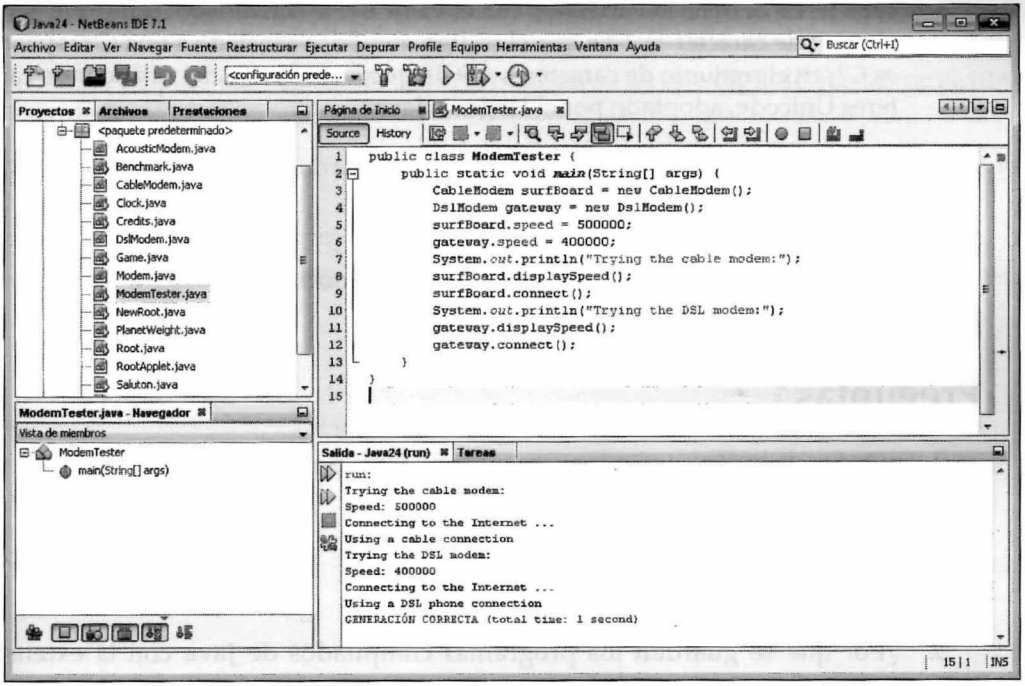


Figura 10.4. Resultado del programa ModemTester.

Preguntas y respuestas

P: ¿Las clases se pueden heredar de más de una clase?

R: En algunos lenguajes de programación (como C++) es posible, pero no en Java. La herencia múltiple es una función muy potente, pero también dificulta el aprendizaje y el uso de la POO. Los programadores de Java decidieron limitar la herencia a una superclase por clase, aunque una clase puede tener varias subclases. Una forma de compensar esta limitación consiste en heredar métodos de un tipo especial de clase denominada interfaz, como veremos en el capítulo 19.

P: ¿Cuándo se debe crear un método que no sea `public`?

R: Normalmente se evita que un método esté disponible para otros programas cuando se usa estrictamente en el programa que está creando. Si crea un juego y su método `shootRayGun()` es específico del mismo, podría ser privado. Para evitar que un método sea público, omita la instrucción `public` del nombre del método.

P: ¿Se pueden usar valores `char` como si fueran `int`?

R: Un carácter se puede emplear como variable `int` ya que cada carácter cuenta con el correspondiente código numérico que representa su posición en el conjunto de caracteres. Si tiene la variable `k` con el valor `67`, la conversión `(char) k` genera el valor de carácter `'C'` ya que el código numérico asociado a una C mayúscula es `67`, en el conjunto de caracteres ASCII, que forma parte del estándar de caracteres Unicode, adoptado por el lenguaje Java.

Ejercicios

Las siguientes preguntas ponen a prueba sus conocimientos sobre objetos y los programas en los que se usan.

Preguntas

1. ¿Qué instrucción se emplea para que una clase pueda heredarse de otra?
   A. `inherits`
   B. `extends`
   C. `handItOverAndNobodyGetsHurt`
2. ¿Por qué se guardan los programas compilados de Java con la extensión `.class`?
   A. Los programadores de Java lo prefieren.
   B. Es un sutil atributo de los profesores del mundo.
   C. Todo programa de Java es una clase.
3. ¿Qué dos componentes forman un objeto?
   A. Atributos y comportamiento.
   B. Comandos y archivos de datos.
   C. Aceite y vinagre.

Respuestas

1. B. La instrucción `extends` se usa ya que la subclase es una extensión de los atributos y del comportamiento de la superclase y de las superclases anteriores a ésta en la jerarquía de clases.
2. C. Sus programas siempre tendrán al menos una clase principal y otras secundarias si las necesita.

3. A. En cierto modo, B también es válido, ya que los comandos son comparables al comportamiento y los archivos de datos son similares a los atributos.

Actividades

Puede ampliar sus conocimientos sobre los temas de este capítulo con las siguientes actividades:

- Cree la clase `AcousticModem` con un valor `speed` 300 y su propio método `connect()`.
- Añada un método `disconnect()` a una de las clases del proyecto `Modem` y determine su posición para admitir la desconexión en módem por cable, ADSL y acústicos.

11
Describir un objeto

Como vimos en el capítulo anterior, un objeto es una forma de organizar un programa para que disponga de todo lo necesario para realizar una tarea. Los objetos están formados por atributos y comportamientos.

Los atributos son la información almacenada en un objeto. Pueden ser variables como enteros, caracteres o valores Booleanos, y objetos como `String` y `Calendar`. El comportamiento es el grupo de instrucciones usadas para realizar tareas concretas en el objeto. Cada uno de estos métodos se denomina método. Hasta ahora hemos trabajado con métodos y variables de objetos sin saberlo. Siempre que una instrucción incluye un punto que no forma parte de un decimal o de una cadena, hay un objeto implicado.

Crear variables

En este capítulo veremos la clase de objetos `Virus` cuya única función en la vida es reproducirse siempre que pueda (como mucha gente que conocí en la universidad). `Virus` cuenta con distintos elementos para realizar su trabajo, elementos que se implementan como comportamiento de la clase. La información necesaria para los métodos se almacena como atributos. Los atributos de un objeto representan las variables que necesita para funcionar. Dichas variables pueden ser tipos de datos sencillos como enteros, caracteres y números de coma flotante, o matrices u objetos de clases como `String` o `Calendar`. Puede emplear las variables de un objeto en su clase, en cualquiera de los métodos que contenga. Por convención, las variables se crean inmediatamente después de la instrucción de clase que crea la clase y antes de los métodos.

Uno de los elementos que necesita el objeto `Virus` es una forma de indicar que un objeto ya se ha infectado. Algunos virus informáticos cambian el campo que almacena la hora de la última modificación del archivo; por ejemplo, un virus puede cambiar la hora de 13:41:20 a 13:41:61. Como ningún archivo normal se guardaría en el segundo 61 de un minuto, esto indica que el archivo está infectado. El objeto `Virus` usa el valor `86`, imposible para segundos, en la variable entera `newSeconds`.

Las siguientes instrucciones inician la clase `Virus` con el atributo `newSeconds` y otros dos atributos:

```
public class Virus {
   public int newSeconds = 86;
   public String author = "Sam Snett";
   int maxFileSize = 30000;
}
```

Las tres variables son atributos de la clase: `newSeconds`, `maxFileSize` y `author`.

La inclusión de una instrucción como `public` en la declaración de una variable se denomina control de acceso, ya que determina cómo otros objetos de otras clases pueden emplear esta variable, o si pueden usarla. Cuando una variable es `public` se puede modificar desde otro programa que use el objeto `Virus`.

Si el otro programa añade un significado especial al número `92`, por ejemplo, puede cambiar `newSeconds` por ese valor. Las siguientes instrucciones crean un el objeto `Virus influenza` y establecen su variable `newSeconds`:

```
Virus influenza = new Virus();
influenza.newSeconds = 92;
```

En la clase `Virus`, la variable `author` también es pública, de modo que puede cambiarse en otros programas. La otra variable, `maxFileSize`, solo se puede emplear en la clase.

Cuando una variable de una clase es `public`, la clase pierde el control de su uso en otros programas. En algunos casos, no es un problema. Por ejemplo, la variable `author` se puede cambiar por otro nombre o seudónimo que identifique al autor del virus. El nombre podría usarse en documentos de un tribunal si el autor es llevado a juicio, de modo que no conviene elegir un nombre entupido. El Estado de Ohio contra LoveHandles no es lo mismo que Ohio contra MafiaBoy.

Al limitar el acceso a una variable se evitan errores si la variable se establece de forma incorrecta en otro programa. Con la clase `Virus`, si `newSeconds` se establece en `60` o menos, no es una forma fiable de saber si el archivo está infectado. Algunos archivos podrían guardarse con ese número de segundos independientemente del virus. Si la clase `Virus` debe evitar este problema, siga estos pasos:

- Cambie la variable de `public` a `protected` o `private`, otras dos instrucciones con acceso más restrictivo.
- Añada comportamiento para cambiar el valor de la variable e informar del mismo a otros programas.

Solo puede emplear una variable `protected` en la misma clase de la variable, en subclases de dicha clase o en clases del mismo paquete. Un paquete es un grupo de clases relacionadas con una función común. Por ejemplo, el paquete `java.util` contiene clases que ofrecen funciones de programación de fecha y hora, y almacenamiento de archivos. Al usar la instrucción `import` en un programa de Java con un asterisco, como en `import java.util.*`, es más fácil hacer referencia a clases de ese paquete en su programa.

Una variable `private` es incluso más restringida que una variable `protected`; solo puede emplearse en la misma clase. A menos que sepa que una variable se puede modificar sin que afecte al funcionamiento de su clase, las variables deben ser `private` o `protected`.

La siguiente instrucción convierte `newSeconds` en una variable `private`:

```
private int newSeconds = 86;
```

Si desea que otros programas usen la variable `newSeconds` de algún modo, tendrá que crear el comportamiento que lo posibilite, como veremos más adelante.

Existe otro tipo de control de acceso: la ausencia de una instrucción `public`, `private` o `protected` al crear la variable.

En muchos de los programas creados hasta ahora, no hemos especificado control de acceso, lo que significa que la variable solo está disponible para las clases del mismo paquete. Este tipo de acceso se denomina predeterminado o de paquete.

Crear variables de clase

Al crear un objeto, dispone de su propia versión de todas las variables que forman parte de su clase. Cada objeto creado a partir de la clase `Virus` tiene su propia versión de las variables `newSeconds`, `maxFileSize` y `author`. Si cambia una de estas variables en un objeto, no afecta a la misma variable de otro objeto `Virus`.

En ocasiones, un atributo debe describir toda una clase de objetos y no un objeto concreto. Son las llamadas variables de clase. Si desea controlar la cantidad de objetos `Virus` usados en un programa, puede emplear una variable de clase para almacenar esta información. Solo existe una copia de la variable para toda la clase. Las variables creadas para objetos hasta el momento son variables de objeto, ya que están asociadas a un objeto concreto.

Ambos tipos de variables se crean y usan de la misma forma, pero `static` forma parte de la instrucción que crea las variables de clase. La siguiente instrucción crea una variable de clase para el ejemplo `Virus`:

```
static int virusCount = 0;
```

El cambio de una variable de clase es similar al cambio de variables de objeto. Si tiene un objeto `Virus` con el nombre `tuberculosis`, podría cambiar la variable de clase `virusCount` con la siguiente instrucción:

```
tuberculosis.virusCount++;
```

Como las variables de clase se aplican a la totalidad de la clase, también puede emplear el nombre de la clase:

```
Virus.virusCount++;
```

Ambas instrucciones realizan la misma acción pero la ventaja de usar el nombre de la clase al trabajar con variables de clase es que muestra inmediatamente que `virusCount` es una variable de clase y no de objeto. Si siempre emplea nombres de objeto al trabajar con variables de clase, no podrá saber si son variables de clase o de objeto si no se fija atentamente en el código fuente. Las variables de clase también se denominan variables estáticas.

Advertencia

Aunque las variables de clase son muy útiles, debe evitar un uso excesivo de las mismas. Existen mientras se ejecute la clase. Si se almacena una matriz de objetos de gran tamaño en ellas, ocupará gran parte de la memoria y no la liberará.

Crear comportamiento con métodos

Los atributos son la forma de controlar la información sobre una clase de objetos, pero para que una clase realice las acciones que debe, tendrá que crear comportamientos. Un comportamiento describe las partes de una clase que realizan tareas concretas. Cada una de estas secciones se denomina método. Hemos usado métodos en los programas anteriores, incluido uno en concreto: `println()`. Este método muestra un texto en la pantalla. Como las variables, los métodos se emplean junto a un objeto o una clase. Tras el nombre del objeto o la clase se añade un punto y el nombre del método, como en `screen2D.drawString()` o `Integer.parseInt()`.

Nota

El método `System.out.println()` puede parecer confuso ya que tiene dos puntos en lugar de uno. Se debe a que la instrucción incluye dos clases: `System` y `PrintStream`. La clase `System` tiene la variable `out` que es un objeto `PrintStream`. `println()` es un método de la clase `PrintStream`. La instrucción `System.out.println()` significa: usar el método `println()` de la clase `System`. Es la forma de encadenar referencias.

Declarar un método

Los métodos se crean con una instrucción similar a la que inicia una clase. En ambos casos se aceptan argumentos en los paréntesis que aparecen tras el nombre, y se emplean llaves para indicar el inicio y el final. La diferencia es que los métodos pueden devolver

un valor tras ser procesados, valor que puede ser uno de los tipos simples como enteros o valores Booleanos, o una clase de objetos. El siguiente ejemplo muestra un método de la clase `Virus` para infectar archivos:

```
public boolean infectFile(String filename) {
   boolean success = false;
   // añadir aquí las instrucciones de virus
   return success;
}
```

Este método acepta un único argumento, la cadena `filename`, una variable que representa el archivo que atacar. Si el ataque es satisfactorio, la variable `success` se establece en `true`. En la instrucción que inicia el método, `boolean` precede al nombre del método, `infectFile`. Esta instrucción indica que se devuelve un valor `boolean` tras procesar el método. La instrucción `return` es la que devuelve el valor. En este método, se devuelve el valor de `success`. Si un método no debe devolver un valor, use la palabra clave `void`.

Cuando un método devuelve un valor, puede usar el método como parte de una expresión. Por ejemplo, si ha creado el objeto `Virus malaria`, puede emplear instrucciones como estas:

```
if (malaria.infectFile(currentFile)) {
   System.out.println(currentFile + " has been infected!");
} else {
   System.out.println("Curses! Foiled again!");
}
```

Puede usar los métodos que devuelvan valores en cualquier punto de un programa en el que emplearía una variable. Anteriormente, cambiamos la variable `newSeconds` a `private` para evitar que otros programas la leyeran o modificaran. Pero hay una forma de poder usarla en otros puntos: crear métodos `public` en la clase `Virus` para obtener el valor de `newSeconds` y establecerla en un nuevo valor. Estos nuevos métodos deben ser `public`, al contrario que la variable `newSeconds`, para poder invocarse en otros programas. Fíjese en los siguientes métodos:

```
public int getSeconds() {
   return newSeconds;
}

public void setSeconds(int newValue) {
   if (newValue > 60) {
      newSeconds = newValue;
   }
}
```

Se denominan métodos de acceso ya que permiten el acceso a la variable `newSeconds` desde otros objetos. El método `getSeconds()` se emplea para recuperar el valor actual de `newSeconds`. Carece de argumentos pero debe tener paréntesis después del nombre. El método `setSeconds()` acepta un argumento, el entero `newValue`, que será el nuevo valor de `newSeconds`. Si `newValue` es mayor de `60`, se realiza el cambio.

En este ejemplo, la clase `Virus` controla cómo usar la variable `newSeconds` en otras clases. Este proceso se denomina encapsulación y es un concepto fundamental de la programación orientada a objetos. Cuanto mejor puedan protegerse sus objetos de usos incorrectos, más útiles serán al emplearlo en otros programas.

Aunque `newSeconds` es `private`, los nuevos métodos `getSeconds()` y `setSeconds()` pueden funcionar con `newSeconds` por ser de la misma clase.

Métodos similares con argumentos diferentes

Como hemos visto con el método `setSeconds()`, puede enviar argumentos a un método que afecten a su funcionamiento. Los métodos de una clase pueden tener diferentes nombres pero también el mismo nombre si tienen argumentos diferentes.

Dos métodos pueden tener el mismo nombre si tienen un número distinto de argumentos o si los argumentos son de diferente tipo de variable. Por ejemplo, sería muy útil para la clase `Virus` tener dos métodos `tauntUser()`: uno sin argumentos que realizara una amenaza genérica, y otro que especificara la amenaza como argumento de cadena:

```
void tauntUser() {
   System.out.println("That has gotta hurt!");
}

void tauntUser(String taunt) {
   System.out.println(taunt);
}
```

Los métodos tienen el mismo nombre pero argumentos distintos: uno carece de argumento y el otro tiene un único argumento `String`. Los argumentos de un método se denominan su firma. Una clase puede tener diferentes métodos con el mismo nombre siempre que cada uno tenga una firma distinta.

Métodos constructores

Para crear un objeto en un programa, se usa la instrucción `new`:

```
Virus typhoid = new Virus();
```

Esta instrucción crea un nuevo objeto `Virus` con el nombre `typhoid`. Al emplear la instrucción `new`, se invoca un método especial de la clase de ese objeto. Este método se denomina constructor, ya que procesa las tareas necesarias para crear el objeto. Su función es la de establecer variables e invocar los métodos necesarios para que el objeto funcione correctamente.

Los constructores se definen como otros métodos pero no devuelven valores. A continuación se muestran dos constructores para la clase `Virus`:

```
public Virus() {
   String author = "Ignoto";
   int maxFileSize = 30000;
```

```
}

public Virus(String name, int size) {
   author = name;
   maxFileSize = size;
}
```

Como sucede con otros métodos, los constructores pueden usar los argumentos que reciben para definir más de un constructor en una clase. En este ejemplo, el primer constructor se invocaría al emplear la siguiente instrucción `new`:

```
Virus mumps = new Virus();
```

El otro constructor solo se usa si se envían como argumentos una cadena y un entero junto a `new`:

```
Virus rubella = new Virus("April Mayhem", 60000);
```

Si no incluye constructores en una clase, heredará un solo constructor sin argumentos de su superclase. También puede heredar otros constructores, en función de la superclase empleada.

En una clase debe haber un método constructor con el mismo número y tipo de argumentos que la instrucción `new` usada para crear objetos de dicha clase. En el ejemplo de la clase `Virus`, que tiene los constructores `Virus()` y `Virus(String name, int size)`, solo podría crear objetos `Virus` con dos tipos diferentes de instrucciones `new`: una con argumentos y otra con una cadena y un entero como únicos argumentos.

Métodos de clase

Como las variables de clase, los métodos de clase proporcionan funcionalidad asociada a una clase completa y no a un objeto concreto. Puede emplear un método de clase cuando el método no haga nada que afecte a un objeto concreto de la clase. En el capítulo anterior usamos el método `parseInt()` de la clase `Integer` para convertir una cadena en una variable de tipo `int`:

```
int fontSize = Integer.parseInt(fontText);
```

Es un método de clase. Para convertir un método en método de clase, use `static` por delante del nombre del método:

```
static void showVirusCount() {
   System.out.println("There are " + virusCount + " viruses");
}
```

La variable de clase `virusCount` se usó anteriormente para controlar cuántos objetos `Virus` creaba el programa. `showVirusCount()` es un método de clase que muestra este total y puede invocarlo con la siguiente instrucción:

```
Virus.showVirusCount();
```

Ámbito de variables con métodos

Al crear una variable o un objeto dentro de un método en una de sus clases, solo puede emplearse dentro de dicho método. Se debe al concepto de ámbito de variables. El ámbito es el bloque en el que existe una variable en un programa. Si sale de la parte del programa definida por el ámbito, ya no podrá usar la variable.

Las llaves { y } de un programa definen los límites del ámbito de una variable. Cualquier variable creada en dichos límites no se puede emplear fuera de los mismos. Fíjese en las siguientes instrucciones:

```
if (numFiles < 1) {
   String warning = "No files remaining.";
}
System.out.println(warning);
```

Este código no funciona y no se compila en NetBeans ya que la variable `warning` se ha creado dentro de las llaves del bloque `if`, que definen el ámbito de la variable. La variable `warning` no existe fuera de las llaves, de modo que el método `System.out.println()` no puede usarla como argumento.

Al emplear llaves dentro de otro grupo de llaves, debe fijarse especialmente en el ámbito de las variables. Veamos un ejemplo:

```
if (infectedFiles < 5) {
   int status = 1;
   if (infectedFiles < 1) {
      boolean firstVirus = true;
      status = 0;
   } else {
      firstVirus = false;
   }
}
```

¿Detecta algún problema? En este ejemplo, la variable `status` se puede usar en cualquier parte pero la instrucción que asigna un valor a la variable `firstVirus` genera un error de compilación. Como `firstVirus` se crea en el ámbito de la instrucción `if (infectedFiles < 1)`, no existe en el ámbito de la instrucción `else` posterior.

Para solucionar el problema, debe crear `firstVirus` fuera de estos bloques para que su ámbito los incluya. Una solución consiste en crear `firstVirus` una línea después a la creación de `status`.

Las reglas de aplicación de ámbito facilitan la depuración de los programas, ya que el ámbito limita la zona en la que emplear una variable. Esto minimiza uno de los problemas más habituales en programación: usar la misma variable de dos formas distintas en diferentes partes de un programa.

El concepto de ámbito también se aplica a métodos, ya que se definen mediante corchetes de apertura y cierre. Una variable creada dentro de un método no se puede emplear en otros métodos. Solo se puede usar en más de un método si se ha creado como variable de objeto o de clase.

Incluir una clase en otra

Aunque un programa de Java se denomine clase, en ocasiones necesita más de una clase para funcionar. Estos programas se componen de una clase principal y las clases de ayuda que necesite.

Al dividirlos en varias clases, hay dos formas de definir las clases de ayudas. Una consiste en definir cada clase de modo independiente, como en este ejemplo:

```
public class Wrecker {
   String author = "Ignoto";

   public void infectFile() {
      VirusCode vic = new VirusCode(1024);
   }
}

class VirusCode {
   int vSize;

   VirusCode(int size) {
      vSize = size;
   }
}
```

Como puede ver, la clase `VirusCode` es una clase de ayuda para la clase `Wrecker`. Las clases de ayuda suelen definirse en el mismo archivo de código fuente que la clase a la que asisten. Al compilar el archivo, se generan varios archivos de clase. El ejemplo anterior genera los archivos `Wrecker.class` y `VirusCode.class`.

Al crear una clase principal y una clase de ayuda, también puede incluir la clase de ayuda dentro de la principal, lo que se denomina clase interna.

Advertencia

Si se define más de una clase en el mismo archivo de código fuente, solo una puede ser `public`. Las demás no pueden serlo. El nombre del archivo de código fuente debe coincidir con la clase `public` que define.

Puede incluir una clase interna dentro de las llaves de otra clase.

```
public class Wrecker {
   String author = "Ignoto";

   public void infectFile() {
      VirusCode vic = new VirusCode(1024);
   }

   class VirusCode {
      int vSize;
```

```
        VirusCode(int size) {
            vSize = size;
        }
    }
}
```

Puede emplear una clase interna de la misma manera que otro tipo de clase de ayuda. La principal diferencia, a pesar de su ubicación, es lo que sucede al compilar las clases. Las clases internas no reciben el nombre indicado por su instrucción de clase, sino que el compilador les asigna un nombre que incluye el de la clase principal. En el ejemplo anterior, el compilador genera `Wrecker.class` y `Wrecker$VirusCode.class`.

Utilizar la palabra clave this

Antes de poder hacer referencia a variables y métodos en otras clases junto a las variables y métodos de sus propias clases, la variable a la que hace referencia puede resultar confusa en determinados casos. Una forma de aclararlo consiste en usar la instrucción `this`, que permite hacer referencia al objeto de un programa desde el propio programa.

Al emplear métodos y variables de un objeto, se incluye su nombre delante del nombre del método o la variable, separado por un punto, como en los siguientes ejemplos:

```
Virus chickenpox = new Virus();
chickenpox.name = "LoveHandles";
chickenpox.setSeconds(75);
```

Estas instrucciones crean un nuevo objeto `Virus` con el nombre `chickenpox`, establecen la variable `name` de `chickenpox` e invocan el método `setSeconds()` de `chickenpox`. En ocasiones, un programa tendrá que hacer referencia al objeto actual, es decir, al objeto representado por el propio programa. Por ejemplo, en la clase `Virus`, puede tener un método con su propia variable `author`:

```
public void checkAuthor() {
    String author = null;
}
```

La variable `author` existe en el ámbito del método `checkAuthor()` pero no es la misma variable que la variable de objeto `author`. Si desea hacer referencia a la variable `author` del objeto actual, debe usar la instrucción `this`:

```
System.out.println(this.author);
```

Al utilizar `this`, indica claramente a qué variable o método se refiere. Puede emplear `this` en cualquier punto de una clase para hacer referencia a un objeto por su nombre. Si desea enviar el objeto actual como argumento de un método, puede usar la siguiente instrucción:

```
verifyData(this);
```

En muchos casos, no se necesita la instrucción `this` para indicar que se está haciendo referencia a las variables y métodos de un objeto. Sin embargo, no pasa nada por emplear `this`.

La palabra clave `this` es muy útil en un constructor al establecer el valor de las variables de instancia de un objeto. Imagine un objeto `Virus` con las variables `author` y `maxFileSize`. Este constructor las define:

```
public Virus(String author, int maxFileSize) {
   this.author = author;
   this.maxFileSize = maxFileSize;
}
```

Usar métodos y variables de clase

A petición de nuestro abogado, el siguiente proyecto no es la creación de un virus. Crearemos un objeto `Virus` que cuente el número de objetos `Virus` creados por el programa y muestre el total.

En NetBeans, seleccione Archivo>Archivo Nuevo y cree un nuevo proyecto vacío de Java con el nombre `Virus`. Introduzca el código del listado 11.1.

Listado 11.1. Texto completo de Virus.java.

```
 1: public class Virus {
 2:    static int virusCount = 0;
 3:
 4:   public Virus() {
 5:      virusCount++;
 6:   }
 7:
 8:   static int getVirusCount() {
 9:      return virusCount;
10:   }
11: }
```

Guarde el archivo y NetBeans lo compila automáticamente. Esta clase carece de método `main()` y por ello no se puede ejecutar directamente. Para probar la clase `Virus` debe crear una segunda clase que permita crear objetos `Virus`.

La clase `VirusLab` es una sencilla aplicación que crea objetos `Virus` y cuenta el número de objetos creados con el método de clase `getVirusCount()` de la clase `Virus`. Abra un nuevo archivo en su procesador de texto e introduzca el código del listado 11.2. Guárdelo como `VirusLab.java`.

Listado 11.2. Texto completo de VirusLab.java.

```
1: public class VirusLab {
2:   public static void main(String[] args) {
3:      int numViruses = Integer.parseInt(args[0]);
4:      if (numViruses > 0) {
```

```
 5:          Virus[] virii = new Virus[numViruses];
 6:          for (int i = 0; i < numViruses; i++) {
 7:             virii[i] = new Virus();
 8:          }
 9:          System.out.println("There are " + Virus.getVirusCount()
10:             + " viruses.");
11:       }
12:    }
13: }
```

La clase `VirusLab` es una aplicación que acepta un argumento al ejecutarla desde la línea de comandos: el número de objetos `Virus` que se van a crear. Para especificar el argumento de línea de comandos en NetBeans, siga estos pasos:

1. Seleccione Ejecutar>Establecer la configuración del proyecto>Personalizar. Se abre el cuadro de diálogo Propiedades del proyecto.
2. Introduzca **VirusLab** en el campo Main Class (Clase principal) y el número de objetos `Virus` que se van a crear en el campo Argumentos.
3. Pulse **Aceptar** para cerrar el cuadro de diálogo.

Para ejecutar un programa configurado de esta forma, seleccione Ejecutar>Ejecutar Proyecto Principal en NetBeans.

Los argumentos se leen en una aplicación por medio de una matriz de cadenas enviada al método `main()`, como sucede en la línea 2 de la clase `VirusLab`.

Para trabajar con un argumento como entero, debe convertirlo de objeto `String` a entero. Para ello necesita el método `parseInt()` de la clase `Integer`. En la línea 3, se crea la variable `int numViruses` a partir del primer argumento enviado al programa en la línea de comandos. Si la variable `numViruses` es mayor que `0`, se realizan las siguientes acciones en la aplicación `VirusLab`:

- **Línea 5:** Se crea una matriz de objetos `Virus` y la variable `numViruses` determina el número de objetos de la matriz.
- **Líneas 6-8:** Se emplea un bucle `for` para invocar el constructor de cada objeto `Virus` de la matriz.
- **Líneas 9-10:** Una vez creados todos los objetos `Virus`, se usa el método de clase `getVirusCount()` de la clase `Virus` para contar el número de objetos creados. Debe coincidir con el argumento enviado al ejecutar la aplicación `VirusLab`.

Si la variable `numViruses` no es mayor que 0, no sucede nada en la aplicación `VirusLab`.

Una vez compilado el archivo `VirusLab.java`, pruébelo con el argumento de línea de comandos que desee. El número de objetos `Virus` creados dependerá de la memoria disponible en su sistema al ejecutar la aplicación `VirusLab`. En mi sistema, cualquier cantidad mayor de 5.5 millones de virus hace que el programa se bloquee y muestre un mensaje `OutOfMemoryError`. Si no especifica más objetos `Virus` de lo que el sistema acepte, el resultado será similar al ilustrado en la figura 11.1.

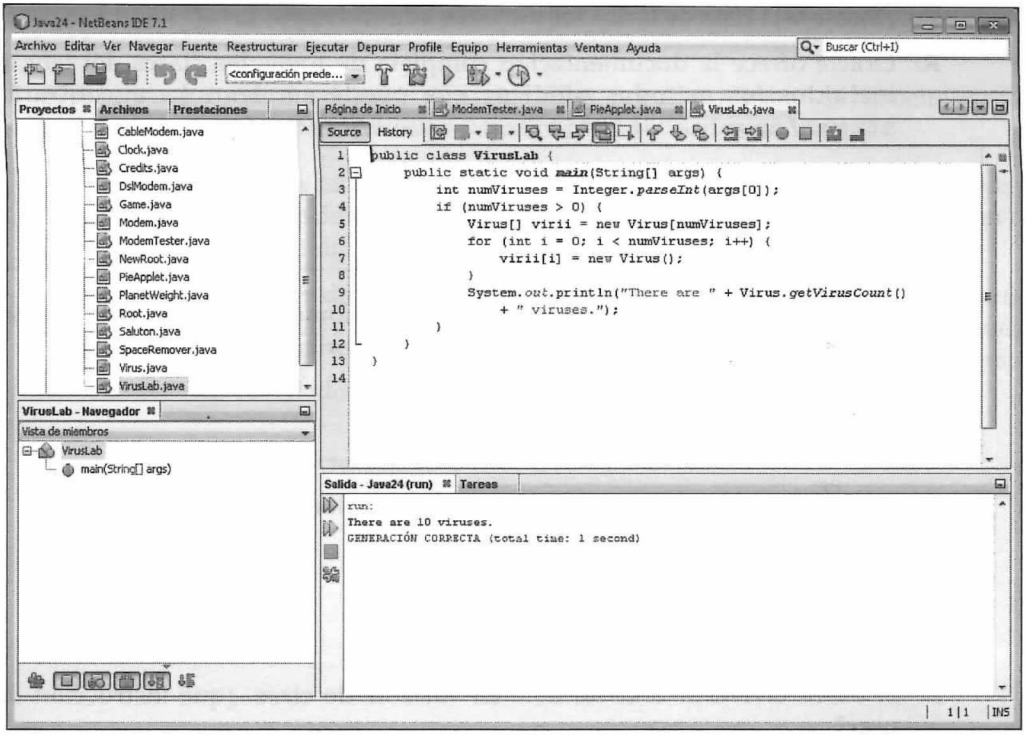


Figura 11.1. Resultado del programa VirusLab.

Resumen

Ha completado dos de los tres capítulos del libro dedicados a conceptos orientados a objetos. Ha aprendido a crear un objeto, a asignar comportamientos y atributos a él y a su clase, y a convertir objetos y variables mediante la técnica de conversión. Pensar en términos de objetos es uno de los principales desafíos del lenguaje de programación Java. Pero cuando empiece a entenderlo, comprobará que la totalidad del lenguaje emplea objetos y clases. En el siguiente capítulo veremos objetos principales y secundarios.

Preguntas y respuestas

P: ¿Es necesario crear un objeto para usar variables o métodos de clase?

R: Como las variables y los métodos no se asocian a un objeto en concreto, no es necesario crear un objeto exclusivamente para usar estos elementos. El uso del método `Integer.parseInt()` es un ejemplo, ya que no tiene que crear un nuevo objeto `Integer` para convertir una cadena en un valor `int`.

P: ¿Existe una lista de todos los métodos admitidos por Java?

R: Oracle ofrece la documentación completa de todas las clases del lenguaje, incluidos los métodos públicos que puede emplear. La encontrará en `http://download.oracle.com/javase/7/docs/api`.

Ejercicios

Los siguientes ejercicios comprueban si tiene los atributos y el comportamiento necesarios para comprender las técnicas de programación orientada a objetos.

Preguntas

1. En una clase de Java, ¿de qué es ejemplo un método?
   A. Atributos.
   B. Instrucciones.
   C. Comportamiento.
2. Si desea convertir una variable en variable de clase. ¿qué instrucción debe usar?
   A. `new`
   B. `public`
   C. `static`
3. ¿Qué nombre recibe la parte de un programa en la que se encuentra una variable?
   A. Su nido.
   B. El ámbito.
   C. Valle de las variables.

Respuestas

1 C. Un método está formado por instrucciones pero es un ejemplo de comportamiento.

2. C. Si se omite la instrucción `static`, la variable será una variable de objeto y no de clase.

3. B. El compilador falla si se emplea una variable fuera de su ámbito.

Actividades

Si se ha puesto enfermo después de tantos virus, puede aumentar sus conocimientos sobre los conceptos de este capítulo por medio de las siguientes actividades:

- Añada una variable `private` a la clase `Virus` que almacene el entero `newSeconds`. Cree métodos para devolver el valor de `newSeconds` y cámbielo solo si el nuevo valor está comprendido entre 60 y 100.
- Cree una aplicación de Java que acepte como argumento una cadena, lo convierta en una variable `float`, transforme dicha variable en un objeto `Float` y después en una variable `int`. Ejecute la aplicación con distintos argumentos para ver cómo cambian los resultados.

12
Maximizar los objetos existentes

Los objetos están especialmente preparados para la procreación. Al crear un objeto, un conjunto de atributos y el comportamiento, diseña algo que puede transmitir estas cualidades a su descendencia. Estos objetos secundarios adoptan muchos de los atributos y el comportamiento de su principal, y también pueden realizar tareas diferentes.

Este sistema se denomina herencia y es algo que toda superclase (principal) pasa a sus subclases (secundarias). La herencia es uno de los aspectos más útiles de la programación orientada a objetos (POO), como veremos en este capítulo. Otro aspecto muy útil de la POO es la capacidad de crear un objeto para usarlo con diferentes programas. La reutilización facilita el desarrollo de programas fiables y sin errores.

El poder de la herencia

Hemos empleado la herencia siempre que hemos trabajado con una de las clases estándar de Java como `String` o `Integer`. Las clases de Java se organizan en una jerarquía piramidal en la que todas las clases descienden de la clase `Object`.

Una clase de objetos se hereda de todas las superclases situadas por encima. Para hacerse una idea, imagine la clase `JApplet`. Es la superclase de todos los applet, programas de Java basados en el navegador que usan una estructura de interfaz grafica de usuario denominada Swing. La clase `JApplet` es una subclase de `Applet`.

En la figura 12.1 puede ver un árbol genealógico parcial de `JApplet`. Cada uno de los recuadros es una clase y las subclases se conectan a la superclase mediante las líneas. En la cima se encuentra la clase `Object`. `JApplet` tiene cinco superclases por encima en la jerarquía: `Applet`, `Panel`, `Container`, `Component` y `Object`.

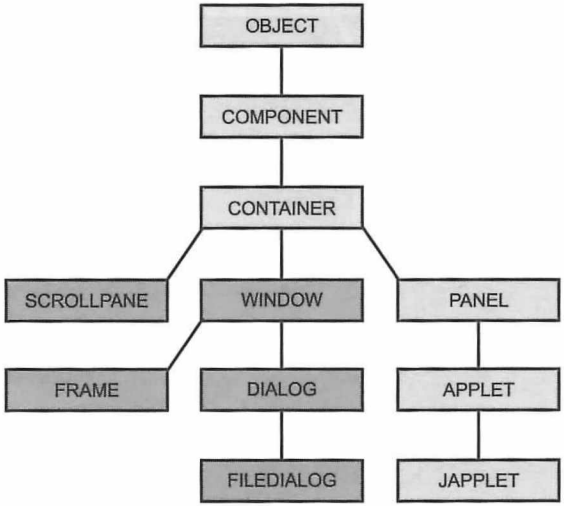


Figura 12.1. Árbol genealógico de la clase JApplet.

La clase `JApplet` hereda los atributos y el comportamiento de todas estas clases por encontrarse por encima en la jerarquía de superclases. `JApplet` no hereda nada de las cinco clases mostradas en color oscuro en la imagen, como `Dialog` y `Frame`, ya que no se encuentran más arriba en la jerarquía.

Si le parece confuso, imagine que `JApplet` hereda de su padre, del padre de su padre y así sucesivamente. Incluso puede heredar rasgos de su tátara tátara-abuelo, `Object`. La clase `JApplet` no hereda nada de sus primos, por ejemplo.

La creación de una nueva clase se reduce a definir las formas en que se diferencia de otra clase existente. El resto del proceso es automático.

Heredar el comportamiento y los atributos

El comportamiento y los atributos de una clase son una combinación de dos elementos: sus propios atributos y el comportamiento, y el comportamiento y los atributos que hereda de sus superclases.

A continuación se muestra parte del comportamiento y los atributos de `JApplet`:

- El método `equals()` determina si un objeto `JApplet` tiene el mismo valor que otro.
- El método `setBackground()` establece el color de fondo de la ventana del applet.
- El método `add()` añade componentes de interfaz de usuario como botones y campos de texto al applet.
- El método `setLayout()` define la organización de la interfaz gráfica de usuario del applet.

La clase `JApplet` puede emplear todos estos métodos, aunque `setLayout()` es el único que no hereda de otra clase. El método `equals()` se define en `Object`, `setBackground()` proviene de `Component` y `add()` de `Container`.

Reemplazar métodos

Algunos de los métodos definidos en la clase `JApplet` también se definen en una de sus superclases. Por ejemplo, el método `update()` forma parte de las clases `JApplet` y `Component`. Al definir un método en una subclase y su superclase, se usa el de la subclase. De este modo, una subclase puede cambiar, reemplazar o eliminar parte del comportamiento o los atributos de sus superclases. En el caso de `update()`, la función es eliminar parte del comportamiento de su superclase.

La creación de un método en una subclase para cambiar el comportamiento heredado de su superclase se denomina reemplazar el método. Tendrá que reemplazar un método siempre que el comportamiento heredado genere un resultado inesperado.

Establecer la herencia

Una clase se define como subclase de otra por medio de la instrucción `extends`:

```
class AnimatedLogo extends JApplet {
   // añadir aquí comportamiento y atributos
}
```

La instrucción `extends` establece la clase `AnimatedLogo` como subclase de `JApplet`. Como veremos en el capítulo 17, todos los applet de Swing deben ser subclases de `JApplet`. Necesitan la funcionalidad que ofrece esta clase para ejecutarse al aparecer en una página Web.

Un método que `AnimatedLogo` debe reemplazar es `paint()`, que se emplea para dibujar en la ventana del programa. El método `paint()`, implementado por la clase `Component`, se pasa hasta `AnimatedLogo`. Sin embargo, no hace nada. Existe para que las subclases de `Component` tengan un método que usar cuando deban mostrar algo.

Para reemplazar un método, debe declararlo como se declare en la superclase de la que se hereda. Un método `public` debe ser `public`, el valor devuelto por el método debe ser el mismo y el número y tipo de argumentos no deben cambiar.

El método `paint()` de la clase `Component` comienza de esta forma:

```
public void paint(Graphics g) {
```

Cuando `AnimatedLogo` reemplaza este método, debe comenzar con la siguiente instrucción:

```
public void paint(Graphics screen) {
```

La única diferencia es el nombre del objeto `Graphics`, que es irrelevante al determinar si los métodos se crean de la misma forma. Estas dos instrucciones coinciden por lo siguiente:

- Ambos métodos `paint()` son `public`.
- Ambos métodos no devuelven valores, debido al uso de `void`.
- Ambos tienen un objeto `Graphics` como único argumento.

Emplear this y super en una subclase

Dos palabras clave muy útiles en una subclase son `this` y `super`.

Como vimos en el capítulo anterior, la palabra clave `this` se usa para hacer referencia al objeto actual. Si al crear una clase tiene que hacer referencia al objeto concreto creado a partir de la misma, puede emplear `this`, como en la siguiente instrucción:

```
this.title = "Cagney";
```

Esta instrucción establece la variable `title` del objeto en el texto `"Cagney"`.

La palabra clave `super` desempeña una función similar: hace referencia a la superclase inmediata del objeto. Puede usar `super` de varias formas:

- Para hacer referencia a un constructor de la superclase, como en `super("Adam", 12);`.
- Para hacer referencia a una variable de la superclase, como en `super.hawaii = 50`.
- Para hacer referencia a un método de la superclase, como en `super.dragnet()`.

Una forma de emplear la palabra clave `super` es en el constructor de una subclase. Como una subclase hereda el comportamiento y los atributos de su superclase, tiene que asociar cada constructor de la subclase con el constructor de su superclase. En caso contrario, puede que el comportamiento y los atributos no se configuren correctamente y la subclase no funcione. Para asociar los constructores, el primer elemento de un constructor de subclase debe ser una invocación de un constructor de la superclase. Para ello necesita la palabra clave `super`, como en estas instrucciones:

```
public readFiles(String name, int length) {
   super(name, length);
}
```

Este ejemplo es el constructor de una subclase, que usa `super(name, length)` para invocar un constructor comparable de su superclase.

Si no emplea `super` para invocar un constructor de la superclase, Java invoca automáticamente `super()` sin argumentos al iniciar el constructor de la subclase. Si el constructor de esta superclase no existe o proporciona un comportamiento inesperado, se generan errores, de modo que conviene invocar el constructor personalmente.

Trabajar con objetos existentes

La POO aboga por la reutilización. Si desarrolla un objeto para usarlo en un proyecto de Java, debe poder incorporarlo a otro proyecto sin cambios.

Si una clase de Java está bien diseñada, puede emplearse en otros programas. Cuantos más objetos disponibles pueda usar en sus programas, menos tendrá que trabajar al crear su propio software. Si existe un excelente objeto de corrección ortográfica perfecto para sus necesidades, puede emplearlo en lugar de tener que crear uno propio. En teoría, incluso puede hacer creer a su jefe que necesitaba mucho tiempo para añadir la función de corrección ortográfica a un proyecto, y usar ese tiempo adicional para llamar a sus amigos desde la oficina.

Nota

El autor de este libro, como otros muchos del sector, es autónomo y trabaja desde casa. No lo olvide cuando lea sus consejos de comportamiento en el lugar de trabajo.

Cuando apareció Java, el sistema para compartir objetos era muy informal. Los programadores desarrollaban sus objetos de forma independiente y los protegían a través de variables privadas y métodos públicos para leer y escribir dichas variables. Compartir objetos es ahora mucho más potente gracias a un enfoque estándar para desarrollar objetos reutilizables. Entre las ventajas de un estándar destacan las siguientes:

- La necesidad de documentar el funcionamiento de un objeto es menor ya que todo el que conoce el estándar sabe cómo funciona.
- Puede diseñar herramientas de desarrollo que sigan el estándar, lo que le permite trabajar más fácilmente con estos objetos.
- Dos objetos que cumplan el estándar pueden interactuar entre ellos sin necesidad de una programación especial.

Almacenar objetos de la misma clase en vectores

Una importante decisión que adoptar al crear un programa informático es dónde almacenar los datos. En la primera mitad del libro hemos visto tres puntos básicos para guardar información: tipos de datos básicos como `int` y `char`, matriz y objetos `String`.

Cualquier clase de Java puede almacenar datos. Una de las más útiles es `Vector`, una estructura de datos que almacena objetos de la misma clase.

Los vectores son como las matrices; contienen elementos de datos relacionados pero pueden aumentar o reducir su tamaño en cualquier momento.

La clase `Vector` pertenece al paquete de clases `java.util`, uno de los más útiles de la biblioteca de clases de Java. Para emplearlo en su programa, use una instrucción `import`:

```
import java.util.Vector;
```

Un vector contiene objetos que pertenecen a la misma clase o comparten la misma superclase. Se crean haciendo referencia a dos clases: a la clase `Vector` y a la que contiene el vector. El nombre de la clase contenida por el vector se incluye entre caracteres < y >:

```
Vector<String> victor = new Vector<String>();
```

La instrucción anterior crea un vector que contiene cadenas. Este tipo de identificación de la clase de un vector usa genéricos, una forma de indicar el tipo de objetos que contiene una estructura de datos como un vector. Si emplea vectores con una versión de Java más antigua, tendría que crear este constructor:

```
Vector victor = new Vector();
```

Aunque puede usarlo, los genéricos hacen que el código sea más fiable, ya que permiten al compilador evitar el uso incorrecto de un vector. Si intenta incluir un objeto `Integer` en un vector que deba almacenar objetos `String`, se producirá un error de compilación.

Al contrario de lo que sucede con las matrices, los vectores no se crean con un número fijo de elementos. El vector se crea con 10 elementos. Si sabe que va a almacenar más, puede especificar el tamaño como argumento del constructor. La siguiente instrucción crea un vector de 300 elementos:

```
Vector<String> victoria = new Vector<String>(300);
```

Puede añadir un objeto a un vector mediante la invocación de su método `add()` empleando el objeto como único argumento:

```
victoria.add("Vance");
victoria.add("Vernon");
victoria.add("Velma");
```

Los objetos se añaden en orden, de forma que si estos son los tres primeros objetos añadidos a `victoria`, el elemento 0 es `"Vance"`, el elemento 1 es `"Vernon"` y el elemento 2 es `"Velma"`. Para recuperar elementos de vectores, debe invocar su método `get()` con el número de índice del elemento como argumento:

```
String name = victoria.get(1);
```

Esta instrucción almacena `"Vernon"` en la misma cadena.

Para ver si un vector contiene un objeto como uno de sus elementos, invoque su método `contains()` con ese objeto como argumento:

```
if (victoria.contains("Velma")) {
   System.out.println("Velma found");
}
```

Puede eliminar un objeto de un vector si usa el vector o un número de índice:

```
victoria.remove(0);
victoria.remove("Vernon");
```

Estas dos instrucciones conservan `"Velma"` como única cadena del vector.

Iterar por un vector

Java incluye un bucle `for` especial para facilitar la carga de un vector y el examen individual de sus elementos.

Este bucle tiene dos partes, una menos que los bucles `for` que vimos en el capítulo 8. La primera parte es la sección de inicialización: la clase y el nombre de la variable que contiene cada objeto recuperado del vector. Este objeto debe pertenecer a la misma clase que contiene el vector. La segunda parte identifica al vector.

El siguiente código itera por el vector `victoria` y muestra los nombres en la pantalla:

```
for (String name : victoria) {
   System.out.println(name);
}
```

El primer proyecto de este capítulo emplea vectores y el bucle `for` especial para presentar una lista de cadenas en orden alfabético. La lista proviene de una matriz y de argumentos de línea de comandos.

Abra el proyecto `Java24` en NetBeans, seleccione Archivo>Archivo Nuevo y cree un nuevo archivo vacío de Java con el nombre `StringLister`. Introduzca el código del listado 12.1 en el editor de código fuente y guarde el archivo.

Listado 12.1. Texto completo de StringLister.java.

```
 1: import java.util.*;
 2:
 3: public class StringLister {
 4:    String[] names = { "Spanky", "Alfalfa", "Buckwheat", "Daria",
 5:       "Stymie", "Marianne", "Scotty", "Tommy", "Chubby" };
 6:
 7:    public StringLister(String[] moreNames) {
 8:       Vector<String> list = new Vector<String>();
 9:       for (int i = 0; i < names.length; i++) {
10:          list.add(names[i]);
11:       }
12:       for (int i = 0; i < moreNames.length; i++) {
13: list.add(moreNames[i]);
14:       }
15:       Collections.sort(list);
16:       for (String name : list) {
17:          System.out.println(name);
18:       }
19:    }
```

```
20:
21:    public static void main(String[] args) {
22:       StringLister lister = new StringLister(args);
23:    }
24: }
```

Antes de ejecutar la aplicación (Ejecutar>Ejecutar archivo), debe ejecutar el comando Ejecutar>Establecer la configuración del proyecto>Personalizar para establecer la clase principal en `StringLister` y el argumento en uno o varios nombres separados por espacios, como `Jackie Mickey Farina Woim`. Los nombres especificados en la línea de comandos se añaden a los almacenados en la matriz en las líneas 4-5. Como el número total de nombres se desconoce hasta la ejecución del programa, un vector es un medio de almacenamiento mejor para estos elementos que una matriz. Las cadenas del vector se ordenan alfabéticamente con un método de la clase `Collections`:

```
Collections.sort(list);
```

Esta clase, como `Vector`, pertenece al paquete `java.util`. Los vectores y otras estructuras de datos se denominan colecciones en Java.

Al ejecutar el programa, el resultado será una lista de nombres en orden alfabético (véase la figura 12.2). El tamaño flexible de los vectores le permite añadir nuevos nombres a la estructura de datos y ordenarlos junto al resto.

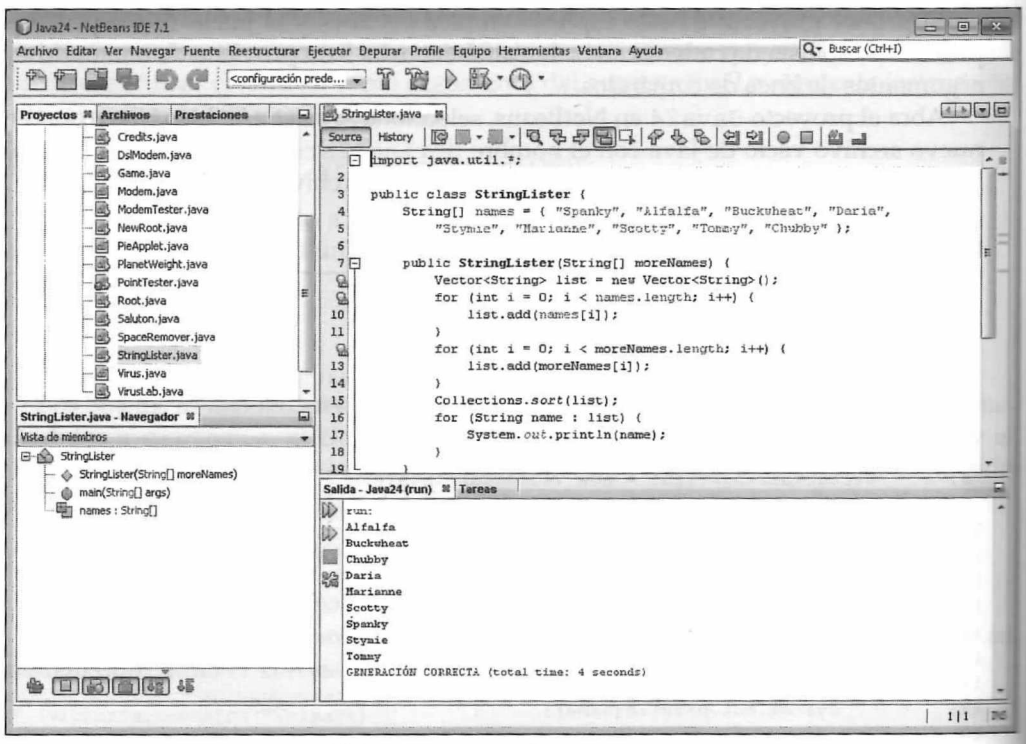


Figura 12.2. Resultado del programa StringLister.

Crear una subclase

Para ver un ejemplo de herencia en funcionamiento, en el siguiente proyecto crearemos la clase `Point3D`, que representa un punto en el espacio tridimensional. Puede expresar un punto en dos dimensiones con una coordenada (x,y). Los applet usan un sistema de coordenadas (x,y) para determinar dónde mostrar el texto y los gráficos. El espacio tridimensional añade una tercera coordinada: z. La clase `Point3D` realiza tres acciones:

- Controla la coordenada (x,y,z) de un objeto.
- Mueve un objeto a una nueva coordenada (x,y,z) cuando es necesario.
- Mueve un objeto una cantidad concreta de valores x, y y z según corresponda.

Java ya cuenta con una clase estándar para representar puntos en dos dimensiones: `Point`. Dispone de dos variables enteras, `x` e `y`, que almacenan la ubicación (x,y) de un objeto `Point`. También cuenta con un método `move()` para colocar un punto en la ubicación especificada y un método `translate()` para desplazar un objeto en una cantidad de valores x e y.

En el proyecto `Java24` de NetBeans, cree un nuevo archivo vacío con el nombre `Point3D` e introduzca el texto del listado 12.2. Guarde el archivo cuando termine.

Listado 12.2. Texto completo de Point3D.java.

```
 1: import java.awt.*;
 2:
 3: public class Point3D extends Point {
 4:   public int z;
 5:
 6:   public Point3D(int x, int y, int z) {
 7:      super(x,y);
 8:      this.z = z;
 9:   }
10:
11:   public void move(int x, int y, int z) {
12:      this.z = z;
13:      super.move(x, y);
14:   }
15:
16:   public void translate(int x, int y, int z) {
17:      this.z += z;
18:      super.translate(x, y);
19:   }
20: }
```

La clase `Point3D` carece de un bloque `main()`, de modo que no puede ejecutarlo con el intérprete de Java pero sí utilizarla en programas de Java cuando necesite un punto tridimensional. La clase `Point3D` solo tiene que realizar tareas que no se realicen en su superclase, `Point`. Básicamente se encarga de controlar la variable entera `z` y recibirla como argumento de los métodos `move()`, `translate()` y `Point3D()`.

Todos los métodos emplean las palabras clave `super` y `this`. La instrucción `this` se usa para hacer referencia al objeto `Point3D` actual, por lo que `this.z = z;` en la línea 8 establece la variable de objeto `z` en el valor `z` enviado como argumento al método de la línea 6.

La instrucción `super` hace referencia a la superclase del objeto actual, `Point`. Se emplea para establecer variables e invocar métodos heredados por `Point3D`. La instrucción `super(x,y)` de la línea 7 invoca el constructor `Point(x,y)` de la superclase, que establece las coordenadas (x,y) del objeto `Point3D`. Como `Point` ya puede procesar los ejes x e y, sería redundante que la clase `Point3D` hiciera lo mismo. Para probar la clase Point3D compilada, cree un programa que use objetos `Point` y `Point3D` y los mueva. En NetBeans, cree un nuevo archivo con el nombre `PointTester` e introduzca el código del listado 12.3. El archivo se compila automáticamente al guardarlo.

Listado 12.3. Texto completo de PointTester.java.

```
 1: import java.awt.*;
 2:
 3: class PointTester {
 4:   public static void main(String[] args) {
 5:      Point object1 = new Point(11,22);
 6:      Point3D object2 = new Point3D(7,6,64);
 7:
 8:      System.out.println("The 2D point is located at (" + object1.x
 9:         + ", " + object1.y + ")");
10:      System.out.println("\tIt's being moved to (4, 13)");
11:      object1.move(4,13);
12:      System.out.println("The 2D point is now at (" + object1.x
13:         + ", " + object1.y + ")");
14:      System.out.println("\tIt's being moved -10 units on both the x "
15:         + "and y axes");
16:      object1.translate(-10,-10);
17:      System.out.println("The 2D point ends up at (" + object1.x
18:         + ", " + object1.y + ")\n");
19:
20:      System.out.println("The 3D point is located at (" + object2.x
21:         + ", " + object2.y + ", " + object2.z + ")");
22:      System.out.println("\tIt's being moved to (10, 22, 71)");
23:      object2.move(10,22,71);
24:      System.out.println("The 3D point is now at (" + object2.x
25:         + ", " + object2.y + ", " + object2.z + ")");
26:      System.out.println("\tIt's being moved -20 units on the x, y "
27:         + "and z axes");
28:      object2.translate(-20,-20,-20);
29:      System.out.println("The 3D point ends up at (" + object2.x
30:         + ", " + object2.y + ", " + object2.z + ")");
31:   }
32: }
```

Al ejecutar el archivo por medio de Ejecutar>Ejecutar archivo, se genera el resultado mostrado en la figura 12.3 si el programa se ha compilado correctamente. En caso contrario, busque el icono de color rojo en el editor de código para identificar la línea que ha provocado el error.

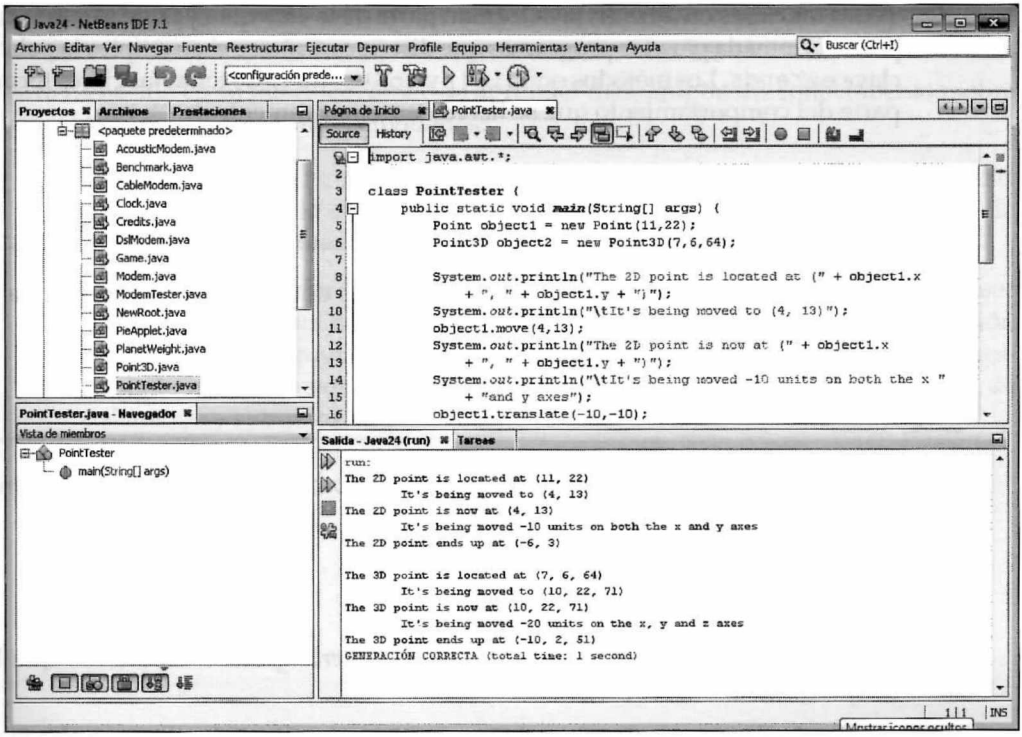


Figura 12.3. Resultado del programa PointTester.

Resumen

Cuando se habla del milagro de la vida, seguramente nadie se refiera a cómo una superclase de Java puede engendrar subclases ni a cómo el comportamiento y los atributos se heredan en una jerarquía de clases. Si el mundo real funcionara igual que la POO, todos los descendientes de Mozart podrían elegir ser brillantes compositores y todos los de Mark Twain podrían escribir sobre la vida en el río Mississippi. Todos los rasgos que tuvieran sus antecesores se transmitirían inmediatamente.

En la escala de milagros, la herencia no está al nivel de la existencia de las especies, pero es una forma eficaz de diseñar software sin esfuerzos innecesarios.

Preguntas y respuestas

P: Muchos de los programas de Java creados hasta el momento no usan `extends` para heredar de una superclase. ¿Significa que existen fuera de la jerarquía de clases?

R: Todas las clases creadas en Java forman parte de la jerarquía ya que la superclase predeterminada para los programas que diseñe es `Object` si no emplea la palabra clave `extends`. Los métodos `equals()` y `toString()` de todas las clases forman parte del comportamiento que se hereda automáticamente de `Object`.

Ejercicios

Para determinar los conocimientos que ha heredado de este capítulo, responda a las siguientes preguntas.

Preguntas

1. Si una superclase procesa un método de una forma que prefiere no usar en la subclase, ¿qué debe hacer?
   A. Borrar el método en la superclase.
   B. Reemplazar el método en la subclase.
   C. Escribir una carta al editor del *San Jose Mercury News* esperando que los programadores de Java la lean.
2. ¿Qué métodos puede emplear para recuperar un elemento almacenado en un vector?
   A. `get()`
   B. `read()`
   C. `elementAt()`
3. ¿Qué instrucción puede usar para hacer referencia a los métodos y variables del objeto actual?
   A. `this`
   B. `that`
   C. `theOther`

Respuestas

1. B. Como puede reemplazar un método, no es necesario cambiar ningún aspecto de la superclase ni su funcionamiento.
2. A. El método `get()` tiene un argumento, el número de índice del elemento.
3. A. La palabra clave `this` hace referencia al objeto en el que aparece.

Actividades

Si ha nacido en su interior el deseo de aprender más, amplíe sus conocimientos sobre la herencia con las siguientes actividades:

- Cree una clase `Point4D` que añada una coordenada `t` al sistema de coordenadas (x,y,z) creado por la clase `Point3D`. La coordenada `t` representa el tiempo, de modo que nunca puede establecerse en un valor negativo.
- Piense en los jugadores de un equipo de fútbol. Diseñe una jerarquía de clases que representen sus habilidades e incluya los rasgos comunes en una posición superior de la jerarquía. Por ejemplo, el bloqueo es un comportamiento supuestamente heredado por centrales y laterales, y la velocidad un rasgo propio de delanteros y extremos.

13
Crear una sencilla interfaz de usuario

Durante este capítulo, las cosas se van a complicar bastante. Crearemos nuestra primera interfaz gráfica de usuario (IGU) con Java.

Los usuarios de ordenadores esperan sin duda que sus programas incluyan una IGU, acepten entradas del ratón y funcionen como otros programas. Aunque algunos usuarios siguen trabajando en entornos de línea de comandos como MS DOS o Linux, la mayoría se confundirán si no tienen una interfaz gráfica para pulsar, arrastrar y soltar como la Microsoft Windows o MacOS.

Java admite este tipo de software gracias a Swing, una colección de clases de Java que representan los distintos botones, campos de texto, reguladores y otros componentes habituales en una IGU, así como las clases necesarias para aceptar entradas del usuario a través de dichos componentes.

En este capítulo y el siguiente crearemos y organizaremos interfaces gráficas de usuario en Java. En el capítulo 15, las habilitaremos para recibir clics del ratón y otras entradas del usuario.

Swing y AWT

Como Java es un lenguaje multiplataforma que le permite crear programas para distintos sistemas operativos, su software de interfaz gráfica de usuario debe ser flexible. En lugar de limitarse al estilo de Windows o Mac, también debe adecuarse a otras plataformas. Con Java, el desarrollo de la interfaz de usuario de un programa se basa en Swing y en un conjunto de clases anterior denominado AWT (*Abstract Windowing Toolkit*, Kit

de herramientas de ventana abstracta). Estas clases le permiten crear una IGU y recibir entradas del usuario. Swing incluye todo lo necesario para crear programas que usen una IGU. Con las clases de interfaz de usuario de Java puede crear una IGU que incluya los siguientes elementos y otros muchos:

- Botones, casillas de verificación, etiquetas y otros componentes sencillos.
- Campos de texto, reguladores y otros componentes más complejos.
- Menús desplegables y emergentes.
- Ventanas, marcos, cuadros de diálogo, paneles y ventanas de applet.

Usar componentes

En Java, todas las partes de una IGU se representan por una clase del paquete Swing. Existe una clase `JButton` para botones, una clase `JWindow` para ventanas, una clase `JTextField` para campos de texto, etc.

Para crear y mostrar una interfaz, debe crear objetos, establecer sus variables e invocar sus métodos. Las técnicas son las mismas que las empleadas en los tres últimos capítulos sobre programación orientada a objetos.

Al confeccionar una IGU, se trabaja con dos tipos de objetos: componentes y contenedores. Un componente es un elemento concreto de una interfaz de usuario, como un botón o un regulador. Un contenedor es un componente que puede emplear para almacenar otros componentes.

El primer paso para diseñar una interfaz consiste en crear un contenedor para almacenar componentes. En una aplicación, este contenedor suele ser una ventana o un marco.

Ventanas y marcos

Las ventanas y los marcos son contenedores que se pueden mostrar en una interfaz de usuario y que albergan otros componentes. Las ventanas son contenedores sencillos que no tienen una barra de título ni otros botones de la parte superior de una IGU. Los marcos son ventanas que incluyen las funciones de ventana habituales que los usuarios esperan al ejecutar software, como botones para cerrar, ampliar o reducir la ventana.

Estos contenedores se crean con las clases `JWindow` y `JFrame` de Swing. Para que el paquete de clases Swing esté disponible en un programa de Java, use esta instrucción:

```
import javax.swing.*;
```

Una forma de usar un marco en una aplicación de Java consiste en convertir a la aplicación en una subclase de `JFrame`. Su programa hereda el comportamiento que necesita para funcionar como marco. Las siguientes instrucciones crean una subclase de `JFrame`:

```
import javax.swing.*;

public class MainFrame extends JFrame {
   public MainFrame() {
      // definir el marco
   }
}
```

Esta clase crea un marco pero no lo configura totalmente. En el constructor, debe realizar varias acciones al crear un marco:

- Invocar un constructor de la superclase, `JFrame`.
- Establecer el título del marco.
- Definir el tamaño del marco.
- Establecer el aspecto operativo y visual del marco.
- Definir qué sucede cuando el usuario cierra el marco.

El marco también debe ser visible, a menos que no deba mostrarse al ejecutar la aplicación. Estos aspectos se pueden procesar en el constructor del marco. Lo primero que debe incluir el método es la invocación de uno de los constructores de `JFrame`, por medio de la instrucción `super`:

```
super();
```

Esta instrucción invoca el constructor de `JFrame` sin argumentos. También puede invocarlo con el título del marco como argumento:

```
super("Main Frame");
```

De este modo se establece el título del marco, que aparece en la barra de título del borde superior, en la cadena especificada, en este caso en el saludo `"Main Frame"`.

Si no define el título de esta forma, puede invocar el método `setTitle()` del marco con una cadena como argumento:

```
setTitle("Main Frame");
```

El tamaño del marco se puede establecer invocando su método `setSize()` con dos argumentos: la anchura y la altura. La siguiente instrucción define un marco de 350 píxeles de ancho por 125 píxeles de alto:

```
setSize(350, 125);
```

Otra forma de establecer el tamaño consiste en rellenarlo con componentes y después invocar el método `pack()` del marco sin argumentos:

```
pack();
```

El método `pack()` establece un tamaño suficiente para albergar los componentes dentro del marco a su tamaño preferido. Todos los componentes de interfaz tienen un tamaño preferido, aunque en ocasiones se descarta, en función de cómo se organicen los

componentes en la interfaz. No es necesario definir explícitamente el tamaño de un marco antes de invocar `pack();` el método lo establece antes de mostrar el marco. Todo marco se muestra con un botón junto a la barra de título que sirve para cerrarlo. En Windows, este botón es una X en la esquina superior derecha del marco. Para definir lo que sucede al pulsar este botón, invoque el método `setDefaultCloseOperation()` del marco con una de las siguientes variables de clase `JFrame` como argumento:

- **setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE):** Salir del programa al pulsar el botón.
- **setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE):** Cerrar el marco y continuar la aplicación en ejecución.
- **setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE):** Mantener abierto el marco y continuar la ejecución.
- **setDefaultCloseOperation(JFrame.HIDE_ON_CLOSE):** Cerrar el marco y continuar la ejecución.

Una interfaz gráfica de usuario creada con Swing puede personalizar su aspecto operativo y visual por medio de un tema que controle la ubicación y el comportamiento de botones y otros componentes.

Java 7 presenta un aspecto operativo y visual mejorado denominado Nimbus, pero debe activarse para emplearlo en una clase. Para ello, se invoca el método `setLookAndFeel()` de la clase `UIManager` del paquete principal de Swing. Este método acepta un argumento: el nombre completo de la clase del aspecto operativo y visual.

La siguiente instrucción establece Nimbus como aspecto operativo y visual:

```
UIManager.setLookAndFeel(
   "com.sun.java.swing.plaf.nimbus.NimbusLookAndFeel"
);
```

Para que el marco sea visible debe invocar su método `setVisible()` con `true` como argumento:

```
setVisible(true);
```

De este modo se abre el marco con la anchura y altura definidas. También se puede invocar con `false` para que el marco no se muestre.

El listado 13.1 contiene el código fuente descrito en esta sección. Introduzca estas instrucciones en un nuevo archivo vacío de Java con el nombre `SalutonFrame`.

Listado 13.1. Texto completo de SalutonFrame.java.

```
1: import javax.swing.*;
2:
3: public class SalutonFrame extends JFrame {
4:   public SalutonFrame() {
5:      super("Saluton mondo!");
6:      setLookAndFeel();
```

```
 7:       setSize(350, 100);
 8:       setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
 9:       setVisible(true);
10:    }
11:
12:    private void setLookAndFeel() {
13:       try {
14:          UIManager.setLookAndFeel(
15:             "com.sun.java.swing.plaf.nimbus.NimbusLookAndFeel"
16:          );
17:       } catch (Exception exc) {
18:          // ignorar el error
19:       }
20:    }
21:
22:    public static void main(String[] arguments) {
23:       SalutonFrame sal = new SalutonFrame();
24:    }
25: }
```

Las líneas 22-24 contienen un método `main()`, que convierte esta clase de marco en una aplicación. Al ejecutar la clase, verá el marco mostrado en la figura 13.1.

Figura 13.1. Un marco en una aplicación.

Lo único que muestra `SalutonFrame` es un título, el saludo `"Saluton mondo!"` en esperanto. El marco es una ventana vacía ya que todavía no tiene componentes.

Para añadir componentes a un marco, debe crearlos y añadirlos al contenedor. Cada contenedor tiene un método `add()` que acepta como argumento el componente que se va a mostrar. La clase `SalutonFrame` incluye un método `setLookAndFeel()` que designa Nimbus como aspecto operativo y visual del marco. Este método, de la clase `UIManager`, se invoca en las líneas 14-16.

La invocación de este método se realiza en un bloque `try-catch`, que permite corregir los errores que se produzcan. Las instrucciones `try` y `catch` son nuevas, pero las describiremos en el capítulo 18.

Por el momento, basta con saber que la invocación de `UIManager.setLookAndFeel()` establece el aspecto operativo y visual de una IGU. Si se produce un error, se conserva el aspecto operativo y visual predeterminado en lugar de Nimbus.

Botones

Un sencillo componente que puede añadir a un contenedor es un objeto `JButton`. Como los demás componentes de este capítulo, forma parte del paquete `java.awt.swing`. Un objeto `JButton` es un botón con una etiqueta que describe lo que sucede al pulsarlo. Esta etiqueta puede ser un texto, gráficos o ambos.

La siguiente instrucción crea un objeto `JButton` con el nombre `okButton` y le asigna la etiqueta de texto **OK**:

```
JButton okButton = new JButton("OK");
```

Tras crear un componente como `JButton`, debe añadirlo a un contenedor mediante la invocación de su método `add()`:

```
add(okButton);
```

Al añadir componentes a un contenedor, no se especifica la ubicación en la que mostrarlos. La disposición de los componentes se decide mediante un objeto llamado administrador de diseño. El más sencillo es la clase `FlowLayout`, del paquete `java.awt`.

Para que un contenedor use un administrador de diseño concreto, primero debe crear un objeto de la clase de dicho administrador. Un objeto `FlowLayout` se crea con la siguiente instrucción:

```
FlowLayout flo = new FlowLayout();
```

Seguidamente, se invoca el método `setLayout()` del contenedor para asociar el administrador al mismo. El único argumento debe ser el objeto de administrador de diseño, como en este ejemplo:

```
pane.setLayout(flo);
```

Esta instrucción designa al objeto `flo` como administrador de diseño para el contenedor `pane`.

La siguiente aplicación, la clase `Playback`, muestra un marco con tres botones. Introduzca el texto del listado 13.2 en un nuevo archivo vacío de Java y guárdelo.

Listado 13.2. Texto completo de Playback.java.

```
 1: import javax.swing.*;
 2: import java.awt.*;
 3:
 4: public class Playback extends JFrame {
 5:   public Playback() {
 6:     super("Playback");
 7:     setLookAndFeel();
 8:     setSize(225, 80);
 9:     setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
10:     FlowLayout flo = new FlowLayout();
11:     setLayout(flo);
12:     JButton play = new JButton("Play");
13:     JButton stop = new JButton("Stop");
14:     JButton pause = new JButton("Pause");
15:     add(play);
16:     add(stop);
17:     add(pause);
18:     setVisible(true);
19:   }
20:
```

```
21:    private void setLookAndFeel() {
22:       try {
23:          UIManager.setLookAndFeel(
24:             "com.sun.java.swing.plaf.nimbus.NimbusLookAndFeel"
25:          );
26:       } catch (Exception exc) {
27:          // ignorar el error
28:       }
29:    }
30:
31:    public static void main(String[] arguments) {
32:       Playback pb = new Playback();
33:    }
34: }
```

El programa `Playback` crea un administrador de diseño `FlowLayout` en la línea 10 y establece el marco para usarlo en la línea 11. Al añadir los tres botones al marco en las líneas 15-17, este administrador se encarga de organizarlos.

Al ejecutar la aplicación, el resultado será similar al ilustrado en la figura 13.2. Puede hacer clic en los botones pero no sucederá nada ya que el programa no contiene métodos para recibir entradas del usuario, tema que abordaremos en el capítulo 15.

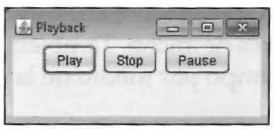


Figura 13.2. Botones en una interfaz.

Puede añadir cualquier componente de Swing a un contenedor mediante esta técnica.

Etiquetas y cuadros de texto

Un componente `JLabel` muestra información que el usuario no puede modificar. Esta información puede ser texto, gráficos o ambos. Estos componentes suelen emplearse para etiquetar a otros componentes de una interfaz, de ahí el nombre de etiqueta, y suelen identificar campos de texto.

Un componente `JTextField` es una zona en la que el usuario puede introducir una sola línea de texto. Puede definir la anchura del cuadro al crear el campo de texto. Las siguientes instrucciones crean un componente `JLabel` y un objeto `JTextField` y los añaden a un contenedor:

```
JLabel pageLabel = new JLabel("Web page address: ", JLabel.RIGHT);
JTextField pageAddress = new JTextField(20);
FlowLayout flo = new FlowLayout();
setLayout(flo);
add(pageLabel);
add(pageAddress);
```

En la figura 13.3 puede ver esta etiqueta y el campo de texto. Ambas instrucciones usan un argumento para configurar el aspecto del componente.

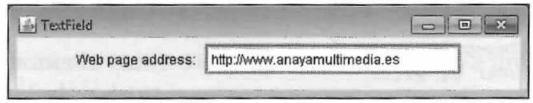


Figura 13.3. Etiquetas y campos de texto.

Nota

Debido a la cantidad de componentes de interfaz presentados en el capítulo, no se muestra el código fuente completo empleado para crear cada imagen. En la página Web del libro encontrará todos los archivos de código.

La etiqueta `pageLabel` se establece en el texto `"Web page address:"` y un argumento `JLabel.RIGHT`. Este valor indica que la etiqueta debe aparecer desplazada a la derecha. `JLabel.LEFT` alinea el texto de la etiqueta a la izquierda y `JLabel.CENTER` lo centra. El argumento usado con `JTextField` indica que el campo de texto debe tener aproximadamente 20 caracteres de ancho. También puede especificar un texto predeterminado que aparezca en el campo por medio de la siguiente instrucción:

```
JTextField country = new JTextField("US", 29);
```

Esta instrucción crea un objeto `JTextField` de 20 caracteres de ancho y con el texto `US` dentro del campo.

Puede recuperar el texto incluido dentro del objeto a través del método `getText()`, que devuelve una cadena:

```
String countryChoice = country.getText();
```

Como habrá imaginado, también puede establecer el texto con el método adecuado:

```
country.setText("Separate Customs Territory of Taiwan, Penghu, Kinmen,
and Matsu");
```

Esto establece el texto en el nombre oficial de China Taipei, el nombre de país más extenso del mundo.

Casillas de verificación

Un componente `JCheckBox` es una casilla junto a una línea de texto que el usuario puede marcar o desactivar. Las siguientes instrucciones crean un objeto `JCheckBox` y lo añaden a un contenedor:

```
JCheckBox jumboSize = new JCheckBox("Jumbo Size");
FlowLayout flo = new FlowLayout();
setLayout(flo);
add(jumboSize);
```

El argumento del constructor `JCheckBox()` indica el texto que se va a mostrar junto a la casilla. Si desea marcarla, use la siguiente instrucción:

```
JCheckBox jumboSize = new JCheckBox("Jumbo Size", true);
```

Puede presentar un `JCheckBox` de forma individual o en grupo. En un grupo de casillas de verificación, solo puede seleccionar una por vez. Para que un objeto `JCheckBox` forme parte de un grupo, debe crear un objeto `ButtonGroup`:

```
JCheckBox frogLegs = new JCheckBox("Frog Leg Grande", true);
JCheckBox fishTacos = new JCheckBox("Fish Taco Platter", false);
JCheckBox emuNuggets = new JCheckBox("Emu Nuggets", false);
FlowLayout flo = new FlowLayout();
ButtonGroup meals = new ButtonGroup();
meals.add(frogLegs);
meals.add(fishTacos);
meals.add(emuNuggets);
setLayout(flo);
add(jumboSize);
add(frogLegs);
add(fishTacos);
add(emuNuggets);
```

Se crean tres casillas de verificación agrupadas bajo el objeto `meals`, de tipo `ButtonGroup`. La casilla Frog Leg Grande aparece inicialmente marcada pero si el usuario marca otra de las casillas, Frog Leg Grande se desactiva automáticamente. En la figura 13.4 puede ver un ejemplo.

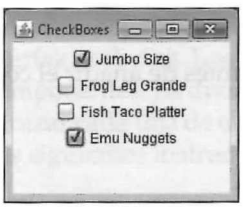


Figura 13.4. Componentes de la casilla de verificación.

Cuadros combinados

Un componente `JComboBox` es una lista desplegable de opciones que también se puede configurar para recibir entradas de texto. Al habilitar ambas opciones, puede seleccionar un elemento con el ratón o emplear el teclado para añadir un texto. El cuadro combinado es similar a un grupo de casillas de verificación, pero solo hay una opción visible a menos que se muestre la lista desplegable.

Para crear un objeto `JComboBox`, debe añadir las siguientes opciones tras crear el objeto:

```
JComboBox profession = new JComboBox();
FlowLayout flo = new FlowLayout();
profession.addItem("Butcher");
profession.addItem("Baker");
profession.addItem("Candlestick maker");
profession.addItem("Fletcher");
profession.addItem("Fighter");
profession.addItem("Technical writer");
setLayout(flo);
add(profession);
```

Este ejemplo crea un componente `JComboBox` que proporciona seis opciones para que el usuario elija. Al seleccionar una, se muestra en el componente. En la figura 13.5 puede ver el ejemplo con la lista desplegable de opciones mostrada.

Figura 13.5. Componentes de cuadro combinado.

Para que un componente `JComboBox` pueda recibir entradas de texto, debe invocar su método `setEditable()` con el argumento `true`:

```
profession.setEditable(true);
```

Debe invocar este método antes de añadir el componente a un contenedor.

Áreas de texto

Un componente `JTextArea` es un campo de texto que permite al usuario introducir más de una línea de texto. Puede especificar la anchura y la altura del componente. Las siguientes instrucciones crean un componente `JTextArea` con una anchura de 40 caracteres y una altura de 8 líneas, y lo añaden a un contenedor:

```
JTextArea comments = new JTextArea(8, 40);
FlowLayout flo = new FlowLayout();
setLayout(flo);
add(comments);
```

En la figura 13.6 se muestra este ejemplo en un marco.

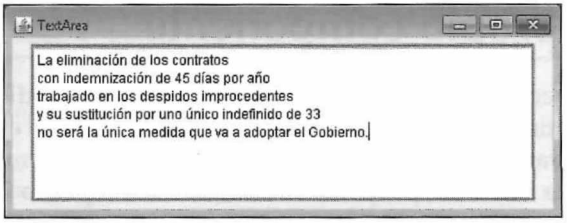


Figura 13.6. Componentes de área de texto.

Puede especificar una cadena en el constructor `JTextArea()` para mostrarla en el área de texto, por medio del carácter de nueva línea (`\n`) para enviar el texto a la nueva línea:

```
JTextArea comments = new JTextArea("I should have been a pair\n"
+ "of ragged claws.", 10, 25);
```

Advertencia

Los componentes de área de texto pueden comportarse de forma inesperada, por lo que pueden aumentar de tamaño cuando el usuario llega a la parte inferior o no incluir barras de desplazamiento en los bordes. Para implementar el tipo de áreas de texto habituales de otras IGU, debe incluir el área dentro de un contenedor denominado panel de desplazamiento, como veremos en el capítulo 16.

Paneles

Los últimos componentes que crearemos en este capítulo son los paneles, que se crean en Swing con la clase `JPanel`. Los objetos `JPanel` son el tipo de contenedor más sencillo que puede usar en una interfaz de Swing. Los objetos `JPanel` dividen el área de pantalla en distintos grupos de componentes, Al dividir la pantalla en secciones, puede emplear distintas reglas para organizar cada una de ellas. Puede crear un objeto `JPanel` y añadirlo a un contenedor con las siguientes instrucciones:

```
JPanel topRow = new JPanel();
FlowLayout flo = new FlowLayout();
setLayout(flo);
add(topRow);
```

Los paneles suelen usarse para organizar los componentes de una interfaz, como veremos en el siguiente capítulo. Para añadir componentes a un panel debe invocar su método `add()`. Para asignar un administrador de diseño directamente a un panel, invoque su método `setLayout()`. También puede emplear paneles si necesita una zona de la interfaz para dibujar algo, como por ejemplo una imagen de un archivo grafico.

Otra aplicación de `JPanel` es para crear sus propios componentes y añadirlos después a otras clases, como veremos en el siguiente ejercicio.

Crear su propio componente

Una de las ventajas de la POO es la posibilidad de reutilizar clases en distintos proyectos. A continuación crearemos un componente de panel especial que podrá reutilizar en sus programas de Java. El componente, `ClockPanel`, muestra la fecha y la hora actuales de forma similar al proyecto `ClockTalk` del capítulo 7.

El primer paso consiste en decidir el componente concreto del que heredar. El componente `ClockPanel` es una subclase de `JPanel`.

La clase `ClockPanel` se define en el listado 13.3. Esta clase representa componentes de panel que incluyen una etiqueta que muestra la fecha y la hora actuales. Introduzca el texto del listado 13.3 en un nuevo archivo vacío de Java y guárdelo.

Listado 13.3. Texto completo de ClockPanel.java.

```
 1: import javax.swing.*;
 2: import java.awt.*;
 3: import java.util.*;
 4:
 5: public class ClockPanel extends JPanel {
 6:   public ClockPanel() {
 7:      super();
 8:      String currentTime = getTime();
 9:      JLabel time = new JLabel("Time: ");
10:      JLabel current = new JLabel(currentTime);
11:      add(time);
12:      add(current);
13:   }
14:
15:   final String getTime() {
16:      String time;
17:      // obtener hora y fechas actuales
18:      Calendar now = Calendar.getInstance();
19:      int hour = now.get(Calendar.HOUR_OF_DAY);
20:      int minute = now.get(Calendar.MINUTE);
21:      int month = now.get(Calendar.MONTH) + 1;
22:      int day = now.get(Calendar.DAY_OF_MONTH);
23:      int year = now.get(Calendar.YEAR);
24:
25:      String monthName = "";
26:      switch (month) {
27:         case (1):
28:            monthName = "January";
29:            break;
30:         case (2):
31:            monthName = "February";
32:            break;
33:         case (3):
34:            monthName = "March";
35:            break;
36:         case (4):
37:            monthName = "April";
38:            break;
```

```
39:          case (5):
40:             monthName = "May";
41:             break;
42:          case (6):
43:             monthName = "June";
44:             break;
45:          case (7):
46:             monthName = "July";
47:             break;
48:          case (8):
49:             monthName = "August";
50:             break;
51:          case (9):
52:             monthName = "September";
53:             break;
54:          case (10):
55:             monthName = "October";
56:             break;
57:          case (11):
58:             monthName = "November";
59:             break;
60:          case (12):
61:             monthName = "December";
62:       }
63:       time = monthName + " " + day + ", " + year + " "
64:          + hour + ":" + minute;
65:       return time;
66:    }
67: }
```

El método `getTime()` de `ClockPanel` contiene la misma técnica para recuperar la fecha y la hora actuales que la aplicación `ClockTalk` del capítulo 7. Este método usa la palabra clave `final` en la declaración de la línea 15:

```
final String getTime() {
   // ...
}
```

Al emplear `final` se evita que el método se reemplace en una subclase. Es necesario para que `ClockPanel` sea un componente de la IGU. El panel se crea en el constructor de las líneas 6-13. Se realizan las siguientes acciones:

- **Línea 8:** Se recupera la fecha y la hora invocando `getTime()` y almacenando la cadena devuelta en la variable `currentTime`.
- **Línea 9:** Se crea la etiqueta `time` con el texto "Time: ".
- **Línea 10:** Se usa `currentTime` como texto del nuevo componente de etiqueta `current`.
- **Línea 11:** La etiqueta `time` se añade al panel mediante la invocación del método `add()` del panel con la etiqueta como argumento.
- **Línea 12:** Se añade la etiqueta `current` al panel de la misma forma.

Para probar el panel, cree la aplicación `ClockFrame`, definida en el listado 13.4.

Listado 13.4. Texto completo de ClockFrame.java.

```
 1: import java.awt.*;
 2: import javax.swing.*;
 3:
 4: public class ClockFrame extends JFrame {
 5:   public ClockFrame() {
 6:      super("Clock");
 7:      setLookAndFeel();
 8:      setSize(225, 125);
 9:      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
10:      FlowLayout flo = new FlowLayout();
11:      setLayout(flo);
12:      ClockPanel time = new ClockPanel();
13:      add(time);
14:      setVisible(true);
15:   }
16:
17:   private void setLookAndFeel() {
18:      try {
19:         UIManager.setLookAndFeel(
20:            "com.sun.java.swing.plaf.nimbus.NimbusLookAndFeel"
21:         );
22:      } catch (Exception exc) {
23:         // ignorar el error
24:      }
25:   }
26:
27:   public static void main(String[] arguments) {
28:      ClockFrame clock = new ClockFrame();
29:   }
30: }
```

Al ejecutar la aplicación obtendrá el resultado mostrado en la figura 13.7.

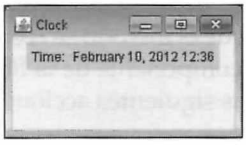


Figura 13.7. Componente de panel de reloj.

Resumen

Los usuarios esperan un entorno visual en el que poder hacer clic en los programas que ejecutan. Esto hace que la creación de software sea un reto, pero Java le permite lograrlo gracias a Swing, que ofrece todas las clases necesarias para crear una interfaz gráfica de usuario funcional, independientemente del tipo de configuración que use para

ejecutar programas de Java. En el siguiente capítulo ampliaremos el tema del diseño de la IGU y trabajaremos con administradores de diseño, clases que permiten especificar la organización de los componentes en un contenedor.

Preguntas y respuestas

P: ¿Cómo se organizan los componentes si no asigno un administrador de diseño a un contenedor?

R: En un contenedor simple como un panel, los componentes se organizan mediante `FlowLayout` de forma predeterminada. Cada componente se añade de la misma forma que las palabras se organizan en un texto, de izquierda a derecha y después a la siguiente línea si no hay más espacio. Marcos, ventanas y applet emplean el estilo `GridLayout`, como veremos en el siguiente capítulo.

P: ¿Por qué muchas clases de interfaz gráfica de usuario comienzan por J, como `JFrame` y `JLabel`?

R: Estas clases forman parte de la estructura Swing, el segundo intento de clases de interfaz gráfica de usuario de la biblioteca de clases de Java. AWT fue el primero, y tenía nombres de clase más sencillos, como `Frame` y `Label`. Las clases AWT pertenecen al paquete `java.awt` y a paquetes relacionados, mientras que Swing pertenece a `javax.swing`, de modo que podrían tener los mismos nombres de clase. El uso de la J evita que las clases se confundan. Las clases Swing también se denominan clases JFC (*Java Foundation Classes*, Clases fundamentales de Java).

Ejercicios

Si todavía no se ha saturado con todos estos conceptos, pruebe sus conocimientos y responda a las siguientes preguntas.

Preguntas

1. ¿Qué componente de usuario se usa para albergar otros componentes?
   A. TupperWare.
   B. `JPanel`.
   C. Choice.
2. ¿Qué es lo primero que debe hacer en un contenedor?
   A. Establecer un administrador de diseño.
   B. Añadir componentes.
   C. Da igual.

3. ¿Qué método determina cómo se organizan los componentes en un contenedor?

   A. `setLayout()`

   B. `setLayoutManager()`

   C. `setVisible()`

Respuestas

1. B. `JPanel`. Puede añadir componentes al panel y después añadir el panel a otro contenedor, como un marco.
2. A. Debe especificar el administrador de diseño antes de poder añadir componentes.
3. A. El método `setLayout()` acepta un argumento: el objeto de administrador de diseño que decide dónde mostrar los componentes.

Actividades

Para ampliar sus conocimientos sobre diseño de interfaces gráficas de usuario, realice las siguientes actividades:

- Modifique la aplicación `SalutonFrame` para que muestre `Saluton Mondo!` en el área principal del marco y no en la barra de título.
- Cree un marco que contenga otro marco y muestre los dos al mismo tiempo.

14
Diseñar una interfaz de usuario

Al comenzar a diseñar interfaces gráficas de usuario (IGU) para sus programas de Java, un obstáculo al que se enfrentará es la forma de mover los objetos. Siempre que un contenedor cambie de tamaño, por ejemplo cuando el usuario cambie un marco, los componentes que contiene pueden reorganizarse para acomodarse a las nuevas dimensiones.

Esta fluidez es una ventaja ya que tiene en cuenta las diferencias de representación de los componentes en cada sistema operativo. Un botón puede tener un aspecto distinto en Windows, Linux o Mac OS. Los componentes se organizan en una interfaz por medio de un conjunto de clases denominadas administradores de diseño. Estas clases definen cómo mostrar los componentes en un contenedor. Cada contenedor puede tener su propio administrador de diseño.

Usar administradores de diseño

En Java, la ubicación de componentes en un contenedor depende del tamaño de otros componentes y de la altura y la anchura del contenedor. La disposición de botones, campos de texto y otros componentes puede verse afectada por estos factores:

- El tamaño del contenedor.
- El tamaño de otros componentes y contenedores.
- El administrador de diseño empleado.

Existen distintos administradores de diseño que puede emplear para modificar la forma de mostrar componentes. El predeterminado es la clase `FlowLayout` del paquete `java.awt`, que utilizamos en el capítulo anterior.

En `FlowLayout`, los componentes se sueltan sobre una zona de la misma forma que las palabras se organizan en un texto: de izquierda a derecha y después en la siguiente línea cuando se agota el espacio.

El siguiente ejemplo se podría usar en un marco para emplear el diseño de flujo al añadir componentes:

```
FlowLayout topLayout = new FlowLayout();
setLayout(topLayout);
```

También puede definir un administrador de diseño para un contenedor concreto, como un objeto `JPanel`. Para ello, use el método `setLayout()` del contenedor.

La aplicación `Crisis` tiene una IGU con cinco botones. Cree un nuevo archivo vacío de Java para la clase `Crisis`. Introduzca el texto del listado 14.1 y guarde el archivo.

Listado 14.1. Texto completo de Crisis.java.

```
 1: import java.awt.*;
 2: import javax.swing.*;
 3:
 4: public class Crisis extends JFrame {
 5:    JButton panicButton;
 6:    JButton dontPanicButton;
 7:    JButton blameButton;
 8:    JButton mediaButton;
 9:    JButton saveButton;
10:
11:    public Crisis() {
12:       super("Crisis");
13:       setLookAndFeel();
14:       setSize(348, 128);
15:       setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
16:       FlowLayout flo = new FlowLayout();
17:       setLayout(flo);
18:       panicButton = new JButton("Panic");
19:       dontPanicButton = new JButton("Don't Panic");
20:       blameButton = new JButton("Blame Others");
21:       mediaButton = new JButton("Notify the Media");
22:       saveButton = new JButton("Save Yourself");
23:       add(panicButton);
24:       add(dontPanicButton);
25:       add(blameButton);
26:       add(mediaButton);
27:       add(saveButton);
28:       setVisible(true);
29:    }
30:
31:    private void setLookAndFeel() {
32:       try {
33:          UIManager.setLookAndFeel(
34:             "com.sun.java.swing.plaf.nimbus.NimbusLookAndFeel"
```

```
35:          );
36:       } catch (Exception exc) {
37:          // ignorar el error
38:       }
39:    }
40:
41:    public static void main(String[] arguments) {
42:       Crisis cr = new Crisis();
43:    }
44: }
```

En la figura 14.1 puede ver la aplicación en funcionamiento.

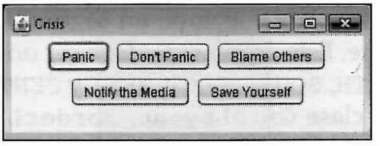


Figura 14.1. Organización de componentes mediante el diseño de flujo.

La clase `FlowLayout` usa las dimensiones de su contenedor como única guía para organizar los componentes. Cambie el tamaño de la ventana de la aplicación para ver cómo se reorganizan automáticamente los componentes. Duplique la anchura y los componentes `JButton` se mostrarán en la misma línea.

GridLayout

La clase `GridLayout` del paquete `java.awt` organiza todos los componentes de un contenedor en una serie concreta de filas y columnas. Todos los componentes reciben la misma cantidad de espacio en la zona de pantalla, de modo que si especifica una cuadrícula con tres columnas y tres filas, el contenedor se divide en nueve zonas del mismo tamaño.

`GridLayout` ubica los componentes al añadirlos a la cuadrícula. Se añaden de izquierda a derecha hasta completar una fila, y después se completa la columna de la izquierda. Las siguientes instrucciones crean un contenedor y lo establecen para emplear un diseño de cuadrícula con dos filas de ancho y tres columnas de alto:

```
GridLayout grid = new GridLayout(2, 3);
setLayout(grid);
```

En la figura 14.2 puede ver la aplicación `Crisis` con un diseño de cuadrícula.

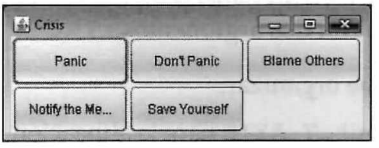


Figura 14.2. Organización de componentes en un diseño de cuadrícula.

Algunas de las etiquetas de la imagen tienen texto reducido. Si el texto es más ancho que la zona disponible en el componente, la etiqueta se recorta con puntos suspensivos (...).

BorderLayout

La clase `BorderLayout`, también de `java.awt`, organiza los componentes en posiciones concretas del contenedor identificadas por cinco direcciones: norte, sur, este, oeste y centro. El administrador `BorderLayout` organiza los componentes en cinco zonas: cuatro indicadas por los puntos cardinales y una zona central. Al añadir un componente en este diseño, el método `add()` incluye un segundo argumento para especificar la ubicación del componente. Este argumento debe ser una de las cinco variables de clase de `BorderLayout`: `NORTH`, `SOUTH`, `EAST`, `WEST` y `CENTER`.

Como sucede con la clase `GridLayout`, `BorderLayout` reserva todo el espacio disponible para los componentes. El situado en el centro recibe todo el espacio que no se necesita para los cuatro del borde, por lo que suele tener mayor tamaño.

Las siguientes instrucciones crean un contenedor con diseño de borde:

```
BorderLayout crisisLayout = new BorderLayout();
setLayout(crisisLayout);
add(panicButton, BorderLayout.NORTH);
add(dontPanicButton, BorderLayout.SOUTH);
add(blameButton, BorderLayout.EAST);
add(mediaButton, BorderLayout.WEST);
add(saveButton, BorderLayout.CENTER);
```

En la figura 14.3 se muestra la aplicación `Crisis` con este diseño.

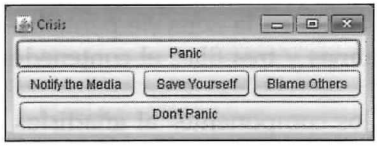


Figura 14.3. Organización de componentes con un diseño de borde.

BoxLayout

Otro administrador de diseño muy útil, `BoxLayout` del paquete `javax.swing`, permite apilar componentes en una misma fila, horizontal o verticalmente.

Para usar este diseño, cree un panel para almacenar los componentes y después un administrador de diseño con dos argumentos:

- El componente que organizar.
- El valor `BoxLayout.Y_AXIS` para la alineación vertical y `BoxLayout.X_AXIS` para la horizontal.

El siguiente código permite apilar los componentes de `Crisis`:

```
JPanel pane = new JPanel();
BoxLayout box = new BoxLayout(pane, BoxLayout.Y_AXIS);
pane.setLayout(box);
pane.add(panicButton);
pane.add(dontPanicButton);
pane.add(blameButton);
pane.add(mediaButton);
pane.add(saveButton);
add(pane);
```

En la figura 14.4 puede ver el resultado.

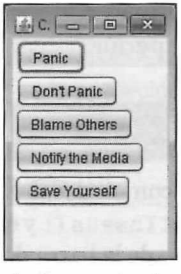


Figura 14.4. Componentes apilados.

Separar componentes con Insets

Al organizar componentes en un contenedor, puede alejarlos de los bordes por medio de `Insets`, un objeto que representa la zona de los bordes de un contenedor.

La clase `Insets`, que forma parte del paquete `java.awt`, tiene un constructor que acepta cuatro argumentos: el espacio libre en la parte superior, inferior, izquierda y derecha del contenedor.

Cada argumento se especifica en píxeles, la misma unidad empleada al definir el tamaño de un marco.

La siguiente instrucción crea un objeto `Insets`:

```
Insets around = new Insets(10, 6, 10, 3);
```

El objeto `around` representa un borde de contenedor 10 píxeles dentro del borde superior, 6 píxeles dentro del izquierdo, 10 píxeles dentro del inferior y 3 píxeles dentro del derecho.

Para emplear el objeto `Insets` en un contenedor, debe reemplazar el método `getInsets()` del contenedor, que carece de argumentos y devuelve un objeto `Insets`, como en el siguiente ejemplo:

```
public Insets getInsets() {
   Insets squeeze = new Insets(50, 15, 10, 15);
   return squeeze;
}
```

En la figura 14.5 puede ver cómo cambia el diseño `FlowLayout` de la interfaz mostrada en la figura 14.1.

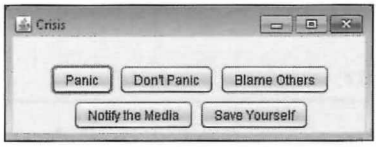


Figura 14.5. Uso de Insets para añadir espacio alrededor de los componentes.

El contenedor mostrado en la figura 14.5 tiene un borde vacío de 15 píxeles con respecto al borde izquierdo, 10 píxeles con respecto al inferior, 15 píxeles con respecto al derecho y 50 con respecto al superior.

Nota

Un contenedor `JFrame` cuenta con un desplazamiento integrado para hacer sitio a la barra de título. Al reemplazar `getInsets()` y establecer valores propios, el contenedor muestra los componentes debajo de la barra de título.

Organizar una aplicación

Los administradores de diseño vistos hasta ahora se aplican a la totalidad del marco, se ha usado el método `setLayout()` del marco y todos los componentes seguían las mismas reglas. Esta configuración puede ser adecuada para algunos programas pero al intentar desarrollar una IGU con Swing, puede que ninguno de los administradores sirva.

Una forma de solucionar el problema consiste en emplear un grupo de objetos `JPanel` como contenedores de para las distintas partes de la IGU. Puede definir distintas reglas de diseño para cada una de estas partes por medio de los métodos `setLayout()` de cada `JPanel`. Una vez que los paneles contengan los componentes necesarios, puede añadirlos directamente al marco.

El siguiente proyecto desarrolla una interfaz completa para el programa que crearemos en el siguiente capítulo. Se trata de un generador de números de lotería que evalúa las probabilidades del usuario de ganar un premio multimillonario a lo largo de su vida. Esta probabilidad se determina ejecutando sorteos aleatorios de seis números repetidamente hasta que el usuario gane. En la figura 14.6 puede ver la IGU de la aplicación.

Cree un nuevo archivo vacío de Java con el nombre `LottoMadness`, introduzca el texto del listado 14.2 y guarde el archivo.

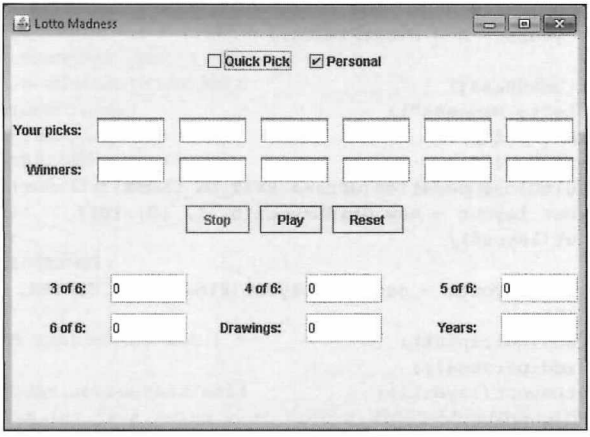


Figura 14.6. IGU de la aplicación LottoMadness.

Listado 14.2. Texto completo de LottoMadness.java.

```
 1: import java.awt.*;
 2: import javax.swing.*;
 3:
 4: public class LottoMadness extends JFrame {
 5:
 6:   // establecer fila 1
 7:   JPanel row1 = new JPanel();
 8:   ButtonGroup option = new ButtonGroup();
 9:   JCheckBox quickpick = new JCheckBox("Quick Pick", false);
10:   JCheckBox personal = new JCheckBox("Personal", true);
11:   // establecer fila 2
12:   JPanel row2 = new JPanel();
13:   JLabel numbersLabel = new JLabel("Your picks: ", JLabel.RIGHT);
14:   JTextField[] numbers = new JTextField[6];
15:   JLabel winnersLabel = new JLabel("Winners: ", JLabel.RIGHT);
16:   JTextField[] winners = new JTextField[6];
17:   // establecer fila 3
18:   JPanel row3 = new JPanel();
19:   JButton stop = new JButton("Stop");
20:   JButton play = new JButton("Play");
21:   JButton reset = new JButton("Reset");
22:   // establecer fila 4
23:   JPanel row4 = new JPanel();
24:   JLabel got3Label = new JLabel("3 of 6: ", JLabel.RIGHT);
25:   JTextField got3 = new JTextField("0");
26:   JLabel got4Label = new JLabel("4 of 6: ", JLabel.RIGHT);
27:   JTextField got4 = new JTextField("0");
28:   JLabel got5Label = new JLabel("5 of 6: ", JLabel.RIGHT);
29:   JTextField got5 = new JTextField("0");
30:   JLabel got6Label = new JLabel("6 of 6: ", JLabel.RIGHT);
31:   JTextField got6 = new JTextField("0", 10);
32:   JLabel drawingsLabel = new JLabel("Drawings: ", JLabel.RIGHT);
33:   JTextField drawings = new JTextField("0");
```

```
34:   JLabel yearsLabel = new JLabel("Years: ", JLabel.RIGHT);
35:   JTextField years = new JTextField();
36:
37:   public LottoMadness() {
38:      super("Lotto Madness");
39:      setLookAndFeel();
40:      setSize(550, 400);
41:      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
42:      GridLayout layout = new GridLayout(5, 1, 10, 10);
43:      setLayout(layout);
44:
45:      FlowLayout layout1 = new FlowLayout(FlowLayout.CENTER,
46:         10, 10);
47:      option.add(quickpick);
48:      option.add(personal);
49:      row1.setLayout(layout1);
50:      row1.add(quickpick);
51:      row1.add(personal);
52:      add(row1);
53:
54:      GridLayout layout2 = new GridLayout(2, 7, 10, 10);
55:      row2.setLayout(layout2);
56:      row2.add(numbersLabel);
57:      for (int i = 0; i < 6; i++) {
58:         numbers[i] = new JTextField();
59:         row2.add(numbers[i]);
60:      }
61:      row2.add(winnersLabel);
62:      for (int i = 0; i < 6; i++) {
63:         winners[i] = new JTextField();
64:         winners[i].setEditable(false);
65:         row2.add(winners[i]);
66:      }
67:      add(row2);
68:
69:      FlowLayout layout3 = new FlowLayout(FlowLayout.CENTER,
70:         10, 10);
71:      row3.setLayout(layout3);
72:      stop.setEnabled(false);
73:      row3.add(stop);
74:      row3.add(play);
75:      row3.add(reset);
76:      add(row3);
77:
78:      GridLayout layout4 = new GridLayout(2, 3, 20, 10);
79:      row4.setLayout(layout4);
80:      row4.add(got3Label);
81:      got3.setEditable(false);
82:      row4.add(got3);
83:      row4.add(got4Label);
84:      got4.setEditable(false);
85:      row4.add(got4);
86:      row4.add(got5Label);
87:      got5.setEditable(false);
88:      row4.add(got5);
89:      row4.add(got6Label);
```

```
 90:       got6.setEditable(false);
 91:       row4.add(got6);
 92:       row4.add(drawingsLabel);
 93:       drawings.setEditable(false);
 94:       row4.add(drawings);
 95:       row4.add(yearsLabel);
 96:       years.setEditable(false);
 97:       row4.add(years);
 98:       add(row4);
 99:
100:       setVisible(true);
101:    }
102:
103:    private void setLookAndFeel() {
104:       try {
105:          UIManager.setLookAndFeel(
106:             "com.sun.java.swing.plaf.nimbus.NimbusLookAndFeel"
107:          );
108:       } catch (Exception exc) {
109:          // ignorar el error
110:       }
111:    }
112:
113:    public static void main(String[] arguments) {
114:       LottoMadness frame = new LottoMadness();
115:    }
116: }
```

Aunque no hayamos añadido instrucciones para que el programa haga algo, puede ejecutar la aplicación para comprobar si la interfaz gráfica se organiza correctamente y recopila la información necesaria. Esta aplicación usa distintos administradores de diseño. Para hacerse una idea de la organización de la interfaz, fíjese en la figura 14.7. La interfaz se divide en cuatro filas horizontales separadas por líneas negras. Cada una de estas filas es un objeto `JPanel`, y el administrador de diseño general de la aplicación organiza las filas en un `GridLayout` de cuatro filas y una columna.

En las filas, se emplean distintos administradores de diseño para determinar cómo mostrar los componentes. Las filas 1 y 3 usan objetos `FlowLayout`. Las líneas 45-46 del programa muestran cómo se crean estos objetos:

```
FlowLayout layout1 = new FlowLayout(FlowLayout.CENTER,
    10, 10);
```

Se emplean tres argumentos con el constructor `FlowLayout()`. El primero, `FlowLayout.CENTER`, indica que los componentes deben centrarse en el contenedor, el `JPanel` horizontal en el que se ubican. Los dos últimos componentes especifican la anchura y la altura que cada componente debe alejarse de otros. Con 10 píxeles se añade una pequeña distancia entre ellos. La fila 2 de la interfaz se organiza en forma de cuadrícula de dos filas de alto y siete columnas de ancho. El constructor `GridLayout()` también especifica que los componentes deben tener una separación de 10 píxeles con respecto al resto. Las líneas 54-55 definen esta cuadrícula:

```
GridLayout layout2 = new GridLayout(2, 7, 10, 10);
row2.setLayout(layout2);
```

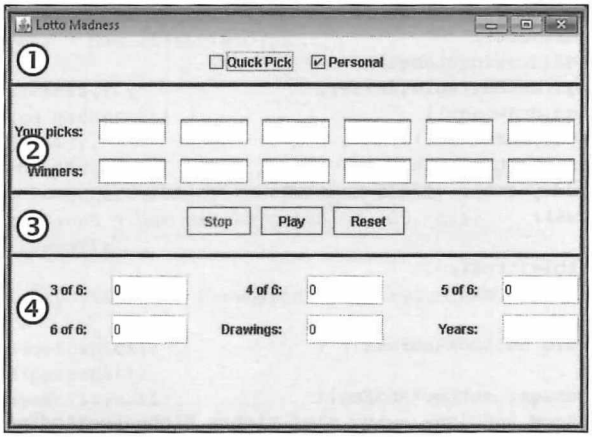


Figura 14.7. División de la aplicación LottoMadness en paneles.

La fila 4 usa `GridLayout` para organizar componentes en una cuadrícula de dos filas de alto por tres columnas de ancho.

La aplicación `LottoMadness` emplea varios de los componentes descritos en este capítulo. Las líneas 7-35 se usan para definir objetos para los componentes que forman la interfaz. Las instrucciones se organizan por fila. Primero se crea un objeto `JPanel` para la fila y después se definen los componentes correspondientes a la fila. Este código crea todos los componentes y contenedores pero no se muestran a menos que se use un método `add()` para añadirlos al marco principal de la aplicación. En las líneas 45-98 se añaden los componentes. Las líneas 45-52 indican el constructor `LottoMadness()`:

```
FlowLayout layout1 = new FlowLayout(FlowLayout.CENTER,
  10, 10);
option.add(quickpick);
option.add(personal);
row1.setLayout(layout1);
row1.add(quickpick);
row1.add(personal);
add(row1);
```

Tras crear un objeto de administrador de diseño, se emplea con el método `setLayout()` del objeto `JPanel` de la fila, `row1` en este caso. Una vez especificado el diseño, se añaden componentes a `JPanel` por medio de su método `add()`. Tras colocar todos los componentes, se añade el objeto `row1` completo al marco mediante la invocación de su método `add()`.

Resumen

Al diseñar la IGU de un programa de Java por primera vez, dudará de que sea una ventaja que los componentes se muevan. Los administradores de diseño le permiten desarrollar una atractiva IGU flexible para solucionar las diferencias de presentación.

En el siguiente capítulo, ahondaremos en el funcionamiento de una interfaz gráfica de usuario. Veremos la interfaz `LottoMadness` en acción.

Preguntas y respuestas

P: ¿Por qué algunos campos de texto de la aplicación `LottoMadness` aparecen en gris y otros en blanco?

R: Se ha utilizado el método `setEditable()` en los campos grises para que no se puedan editar. El comportamiento predeterminado de un campo de texto consiste en permitir a los usuarios cambiar el valor haciendo clic en los bordes e introduciendo los cambios deseados. Sin embargo, algunos campos deben mostrar información y no aceptar entradas del usuario. El método `setEditable()` evita que los usuarios cambien un campo que no deben modificar.

Ejercicios

Para comprobar si sus neuronas están bien organizadas, pruebe sus conocimientos de administración de diseño de Java respondiendo a las siguientes preguntas.

Preguntas

1. ¿Qué contenedor suele usarse para dividir una interfaz en distintos administradores de diseño?

   A `JWindow`

   B. `JPanel`

   C. `Container`

2. ¿Cuál es el administrador de diseño predeterminado de un panel?

   A. `FlowLayout`

   B. `GridLayout`

   C. No existe ninguno predeterminado.

3. ¿De dónde obtiene su nombre la clase `BorderLayout`?

   A. Del borde de cada componente.

   B. De la organización de los componentes con respecto a los bordes de un contenedor.

   C. De un capricho de los programadores de Java.

Respuestas

1. B. `JPanel`, el contenedor más sencillo.
2. A. Los paneles emplean diseño de flujo pero el administrador predeterminado de marcos y ventanas es `GridLayout`.
3. B. Debe especificar la posición de borde de los componentes mediante variables de dirección como `BorderLayout.WEST` y `BorderLayout.EAST` al añadirlos a un contenedor.

Actividades

Si desea mantener el ritmo, realice las siguientes actividades:

- Cree una versión modificada de la aplicación `Crisis` con los objetos `panic` y `dontPanic` organizados bajo un administrador de diseño y los tres botones restantes bajo otro distinto.
- Cree una copia del archivo `LottoMadness.java` y cambie el nombre por `NewMadness.java`. Modifique el programa para que las opciones se muestren en un cuadro combinado y los botones sean casillas de verificación.

15. Responder a entradas del usuario

La interfaz gráfica de usuario (IGU) desarrollada en los dos últimos capítulos puede ejecutarse de forma independiente. Los usuarios pueden pulsar botones, completar campos de texto con texto y cambiar el tamaño de la ventana. Tarde o temprano, querrán más. La IGU que ofrece un programa debe reaccionar a clics del ratón y entradas desde el teclado. Estas reacciones son posibles cuando su programa de Java puede responder a eventos del usuario, actividad denominada control de eventos y que analizaremos en este capítulo.

Hacer que sus programas escuchen

En Java, un evento de usuario es algo que sucede cuando un usuario realiza una acción con el ratón, el teclado u otro dispositivo de entrada.

Antes de poder recibir eventos, debe aprender a hacer que un objeto escuche. La respuesta a eventos del usuario requiere el uso de una o varias interfaces `EventListener`. Las interfaces son una característica de la programación orientada a objetos que permiten a una clase heredar un comportamiento que no podría usar de otra forma. Son una especie de contrato con otras clases que garantiza que la clase tendrá métodos concretos.

Una interfaz `EventListener` contiene métodos que reciben entradas del usuario de un tipo concreto. Para añadir una interfaz `EventListener` se necesitan dos cosas. Por un lado, como las clases que escuchan forman parte del paquete `java.awt.event`, debe habilitarlas con la siguiente instrucción:

```
import java.awt.event.*;
```

Por otra parte, la clase debe emplear la palabra clave `implements` para declarar que admite una o varias interfaces de escucha. La siguiente instrucción crea una clase que usa `ActionListener`, una interfaz para responder a clics de botón y de menú:

```
public class Graph implements ActionListener {
```

Las interfaces `EventListener` permiten que un componente de una IGU genere eventos de usuario. Sin un escuchador, un componente no puede hacer nada para otras partes del programa. Un programa debe incluir una interfaz de escucha por cada tipo de componente al que escuche. Para que el programa responda al clic de un botón o a la pulsación de la tecla **Intro** en un campo de texto debe incluir la interfaz `ActionListener`. Para responder al uso de una lista de opciones o casillas de verificación, necesita la interfaz `ItemListener`. Si necesita más de una interfaz en la misma clase, separe los nombres con comas tras la palabra clave `implements`:

```
public class Graph3D implements ActionListener, MouseListener {
   // ...
}
```

Configurar componentes para ser escuchados

Una vez implementada la interfaz necesaria para un determinado componente, debe configurarlo para que genere eventos de usuario. La interfaz `ActionListener` escucha eventos de acción, como el clic de un botón o la pulsación de la tecla **Intro**. Para que un objeto `JButton` genere un evento, utilice el método `addActionListener()` de esta forma:

```
JButton fireTorpedos = new JButton("Fire torpedos");
fireTorpedos.addActionListener(this);
```

Este código crea el botón `fireTorpedos` e invoca su método `addActionListener()`. La palabra clave `this` se emplea como argumento de `addActionListener()` para indicar que el objeto actual recibe el evento de usuario y lo procesa cuando sea necesario.

Nota

La palabra clave `this` confunde a muchos lectores cuando la ven por primera vez. Hace referencia al objeto en el que aparece. Por ello, si crea una clase `LottoMadness` y usa `this` en una instrucción dentro de la misma, hace referencia al objeto `LottoMadness` que ejecuta el código.

Procesar eventos de usuario

Cuando un evento de usuario se genera desde un componente que tiene un escuchador, se invoca un método automáticamente. El método debe incluirse en la clase especificada al añadir el escuchador al componente.

Cada escuchador tiene distintos métodos que se invocan para recibir sus eventos. La interfaz ActionListener envía eventos a un método llamado `actionPerformed()`. Veamos un breve ejemplo:

```
public void actionPerformed(ActionEvent event) {
   // añadir aquí el método
}
```

Todos los eventos de acción enviados en el programa se incluyen en este método. Si solo un componente del programa puede enviar eventos de acción, puede incluir instrucciones en este método para procesar el evento. Si más de un componente puede enviar estos eventos, debe comprobar el objeto enviado al método.

Un objeto `ActionEvent` se envía al método `actionPerformed()`. Diferentes clases de objetos representan los eventos de usuario que se pueden enviar en un programa. Dichas clases cuentan con métodos que determinan qué componente ha generado el evento. En el método `actionPerformed()`, si el nombre del objeto `ActionEvent` es `event`, puede identificar el componente con la siguiente instrucción:

```
String cmd = event.getActionCommand();
```

El método `getActionCommand()` devuelve una cadena. Si el componente es un botón, la cadena es la etiqueta del mismo. Si es un campo de texto, la cadena es el texto que se introduce en el campo. El método `getSource()` devuelve el objeto que ha provocado el evento. Puede emplear el siguiente método `actionPerformed()` para recibir eventos de tres componentes: un objeto `JButton` con el nombre `start`, un `JTextField` con el nombre `speed` y otro `JTextField` con el nombre `viscosity`:

```
public void actionPerformed(ActionEvent event) {
   Object source = event.getSource();
   if (source == speed) {
      // el campo speed desencadena el evento
   } else if (source == viscosity) {
      // viscosity desencadena el evento
   } else {
      // start desencadena el evento
   }
}
```

Puede invocar el método `getSource()` en todos los eventos de usuario para identificar el objeto concreto que ha generado el evento.

Eventos de casillas de verificación y cuadros combinados

Las casillas de verificación y los cuadros combinados requieren la interfaz `ItemListener`. Invoque el método `addItemListener()` del componente para que genere estos eventos. Las siguientes instrucciones crean la casilla de verificación `superSize` que envía eventos de usuario al marcarla o deseleccionarla:

```
JCheckBox superSize = new JCheckBox("Super Size", true);
superSize.addItemListener(this);
```

Estos eventos se reciben en el método `itemStateChanged()`, que acepta un objeto `ItemEvent` como argumento. Para ver qué objeto ha generado el evento, invoque el método `getItem()`.

Para determinar si una casilla de verificación está seleccionada o no, compare el valor devuelto por `getStateChange()` con las constantes `ItemEvent.SELECTED` e `ItemEvent.DESELECTED`. El siguiente código muestra el objeto `ItemEvent item`:

```
int status = item.getStateChange();
if (status == ItemEvent.SELECTED) {
   // se ha seleccionado item
}
```

Para determinar el valor seleccionado en un objeto `JComboBox`, use `getItem()` y convierta el valor en cadena:

```
Object which = item.getItem();
String answer = which.toString();
```

Eventos de teclado

Cuando un programa debe reaccionar inmediatamente a la pulsación de una tecla, usa eventos de teclado y la interfaz `KeyListener`.

El primer paso para registrar el componente que recibe pulsaciones de teclado es invocar su método `addKeyListener()`. El argumento del método debe ser el objeto que implementa la interfaz `KeyListener`. Si es la clase actual, use `this` como argumento. Un objeto que procese eventos de teclado debe implementar estos métodos:

- **void keyPressed(KeyEvent):** Se invoca al pulsar una tecla.
- **void keyReleased(KeyEvent):** Se invoca al liberar una tecla.
- **void keyTyped(KeyEvent):** Se invoca tras pulsar y liberar una tecla.

Todos deben tener un objeto `KeyEvent` como argumento, que tiene métodos que invocar para saber más sobre el evento. Invoque `getKeyChar()` para saber qué tecla se ha pulsado. La tecla se devuelve como valor `char` y solo se puede emplear con letras, números y signos de puntuación.

Para controlar una tecla del teclado, como **Intro**, **RePág** y **AvPág**, puede invocar `getKeyCode()`. Este método devuelve un valor entero que representa la tecla. Tras ello puede invocar `getKeyText()` con ese entero como argumento para recibir un objeto `String` con el nombre de la tecla (como **Inicio**, **F1**, etc.).

El listado 15.1 contiene una aplicación de Java que dibuja la última tecla pulsada en una etiqueta por medio del método `getKeyChar()`. La aplicación implementa la interfaz `KeyListener`, de modo que la clase contiene los métodos `keyTyped()`, `keyPressed()` y `keyReleased()`. El único que no hace nada es `keyTyped()` en las líneas 22-25. Cree un nuevo archivo de Java con el nombre `KeyViewer`, introduzca el código en el editor de NetBeans y guarde el archivo.

Listado 15.1. Código completo de KeyViewer.java.

```
 1: import javax.swing.*;
 2: import java.awt.event.*;
 3: import java.awt.*;
 4:
 5: public class KeyViewer extends JFrame implements KeyListener {
 6:    JTextField keyText = new JTextField(80);
 7:    JLabel keyLabel = new JLabel("Press any key in the text field.");
 8:
 9:    KeyViewer() {
10:       super("KeyViewer");
11:       setLookAndFeel();
12:       setSize(350, 100);
13:       setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
14:       keyText.addKeyListener(this);
15:       BorderLayout bord = new BorderLayout();
16:       setLayout(bord);
17:       add(keyLabel, BorderLayout.NORTH);
18:       add(keyText, BorderLayout.CENTER);
19:       setVisible(true);
20:    }
21:
22:    public void keyTyped(KeyEvent input) {
23:       char key = input.getKeyChar();
24:       keyLabel.setText("You pressed " + key);
25:    }
26:
27:    public void keyPressed(KeyEvent txt) {
28:       // no hacer nada
29:    }
30:
31:    public void keyReleased(KeyEvent txt) {
32:       // no hacer nada
33:    }
34:
35:    private void setLookAndFeel() {
36:       try {
37:          UIManager.setLookAndFeel(
38:             "com.sun.java.swing.plaf.nimbus.NimbusLookAndFeel"
39:          );
40:       } catch (Exception exc) {
41:          // ignorar el error
42:       }
43:    }
44:
45:    public static void main(String[] arguments) {
46:       KeyViewer frame = new KeyViewer();
47:    }
48: }
```

Al ejecutar la aplicación, obtendrá el resultado mostrado en la figura 15.1.

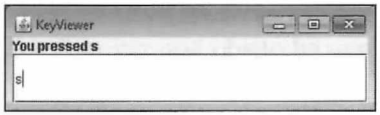


Figura 15.1. Procesamiento de eventos de teclado en un programa.

Habilitar y deshabilitar componentes

Seguramente haya visto componentes de un programa que aparecen sombreados, no con su aspecto normal.

El sombreado indica que el usuario no puede usar el componente porque está deshabilitado. Para deshabilitar y habilitar componentes se emplea su método `setEnabled()`. Se envía un valor Booleano como argumento del método, de modo que `setEnabled(true)` habilita un componente para usarlo y `setEnabled(false)` lo deshabilita.

Las siguientes instrucciones crean botones con las etiquetas **Previous**, **Next** y **Finish**, y desactivan el primero:

```
JButton previousButton = new JButton("Previous");
JButton nextButton = new JButton("Next");
JButton finishButton = new JButton("Finish");
previousButton.setEnabled(false);
```

Este método es una forma eficaz de evitar que un componente envíe un evento de usuario cuando no debe. Por ejemplo, si crea una aplicación de Java que obtiene la dirección de un usuario a través de campos de texto, puede deshabilitar el botón **Save Address** (Guardar dirección) hasta que el usuario haya introducido la calle, la ciudad, la provincia y el código postal.

Completar una aplicación gráfica

Para ver el funcionamiento de las clases de procesamiento de eventos de Swing en un programa de Java, completaremos `LottoMadness`, el simulador de lotería del capítulo anterior. Por ahora, `LottoMadness` es solo una IGU. Puede pulsar botones e introducir texto en los cuadros de texto, pero no hay respuesta. En este ejercicio, crearemos `LottoEvent`, una nueva clase que recibe entradas del usuario, realiza sorteos y controla cuántas veces gana. Al completar la clase, añadiremos nuevas líneas para que `LottoMadness` use `LottoEvent`. Suele ser habitual dividir los proyectos de Swing de esta forma, con la IGU en una clase y los métodos de control de eventos en otra.

El objetivo de esta aplicación es evaluar la probabilidad de ganar un sorteo de lotería de seis números. En la figura 15.2 puede ver un ejemplo del programa en funcionamiento. En lugar de emplear probabilidades para solucionar el problema, el ordenador realiza sorteos de forma rápida sin interrupción hasta que aparece un ganador. Como acertar 6 de 6 es poco probable, el programa también muestra la combinación de tres, cuatro y cinco números ganadores.

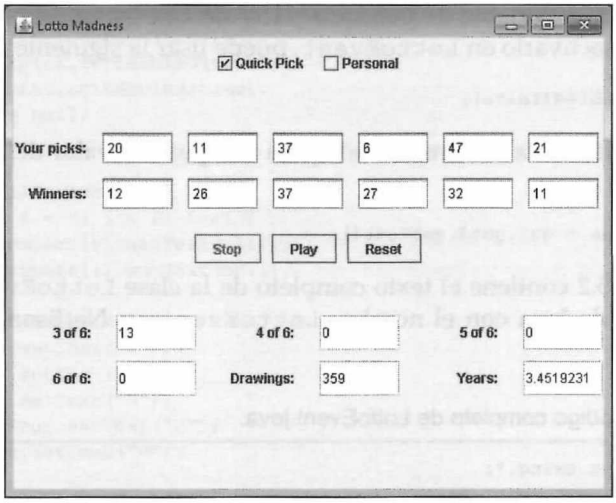


Figura 15.2. Ejecución de la aplicación LottoMadness.

La interfaz incluye 12 campos de texto para los números y dos casillas de verificación: Quick Pick y Personal. Se usan seis campos de texto, que no admiten entradas, para mostrar los números ganadores de cada sorteo. Los otros seis campos de texto permiten al usuario seleccionar sus números. Al marcar la casilla Quick Pick se eligen seis números aleatorios. Al marcar Personal el usuario puede elegir seis números de su elección.

Tres botones controlan la actividad del programa: **Stop**, **Play** y **Reset**. Al pulsar **Play** (Jugar), el programa inicia el subproceso `playing` y realiza los sorteos.

Al pulsar **Stop** (Detener), se detiene el subproceso y al pulsar Reset (Restablecer) se borran los campos para empezar de nuevo. En el capítulo 19 encontrará más información sobre subprocesos.

La clase `LottoEvent` implementa tres interfaces: `ActionListener`, `ItemListener` y `Runnable`, esta última relacionada con subprocesos, como veremos en el capítulo 19. Se necesitan escuchadores para escuchar los eventos de usuario generados por los botones y las casillas de verificación de la aplicación. El programa no tiene que escuchar eventos relacionados con los campos de texto, ya que se emplean estrictamente para almacenar los números elegidos por el usuario. La interfaz de usuario procesa automáticamente este almacenamiento.

La clase debe usar el paquete principal de Swing, `javax.swing`, y el de procesamiento de eventos de Java, `java.awt.event`.

La clase tiene dos variables de instancia:

- `gui`, un objeto `LottoMadness`.
- `playing`, un objeto `Thread` que se emplea para realizar sorteos continuos.

La variable `gui` se usa para la comunicación con el objeto `LottoMadness` que contiene la IGU del programa. Cuando tenga que cambiar la interfaz o recuperar un valor de uno de sus campos de texto, se emplean las variables de instancia del objeto `gui`.

Por ejemplo, la variable de instancia `play` de `LottoMadness` representa el botón `Play`. Para desactivarlo en `LottoEvent`, puede usar la siguiente instrucción:

```
gui.play.setEnabled(false);
```

Puede emplear esta otra instrucción para recuperar el valor del objeto `JTextField got3`:

```
String got3value = gui.got3.getText();
```

El listado 15.2 contiene el texto completo de la clase `LottoEvent`. Cree un nuevo archivo vacío de Java con el nombre `LottoEvent` en NetBeans y añada el código fuente.

Listado 15.2. Código completo de LottoEvent.java.

```
 1: import javax.swing.*;
 2: import java.awt.event.*;
 3:
 4: public class LottoEvent implements ItemListener, ActionListener,
 5:   Runnable {
 6:
 7:   LottoMadness gui;
 8:   Thread playing;
 9:
10:   public LottoEvent(LottoMadness in) {
11:      gui = in;
12:   }
13:
14:   public void actionPerformed(ActionEvent event) {
15:      String command = event.getActionCommand();
16:      if (command.equals("Play")) {
17:         startPlaying();
18:      }
19:      if (command.equals("Stop")) {
20:         stopPlaying();
21:      }
22:      if (command.equals("Reset")) {
23:         clearAllFields();
24:      }
25:   }
26:
27:   void startPlaying() {
28:      playing = new Thread(this);
29:      playing.start();
30:      gui.play.setEnabled(false);
31:      gui.stop.setEnabled(true);
32:      gui.reset.setEnabled(false);
33:      gui.quickpick.setEnabled(false);
34:      gui.personal.setEnabled(false);
35:   }
36:
37:   void stopPlaying() {
38:      gui.stop.setEnabled(false);
```

```
39:       gui.play.setEnabled(true);
40:       gui.reset.setEnabled(true);
41:       gui.quickpick.setEnabled(true);
42:       gui.personal.setEnabled(true);
43:       playing = null;
44:    }
45:
46:    void clearAllFields() {
47:       for (int i = 0; i < 6; i++) {
48:          gui.numbers[i].setText(null);
49:          gui.winners[i].setText(null);
50:       }
51:       gui.got3.setText("0");
52:       gui.got4.setText("0");
53:       gui.got5.setText("0");
54:       gui.got6.setText("0");
55:       gui.drawings.setText("0");
56:       gui.years.setText("0");
57:    }
58:
59:    public void itemStateChanged(ItemEvent event) {
60:       Object item = event.getItem();
61:       if (item == gui.quickpick) {
62:          for (int i = 0; i < 6; i++) {
63:             int pick;
64:             do {
65:                pick = (int) Math.floor(Math.random() * 50 + 1);
66:             } while (numberGone(pick, gui.numbers, i));
67:             gui.numbers[i].setText("" + pick);
68:          }
69:       } else {
70:          for (int i = 0; i < 6; i++) {
71:             gui.numbers[i].setText(null);
72:          }
73:       }
74:    }
75:
76:    void addOneToField(JTextField field) {
77:       int num = Integer.parseInt("0" + field.getText());
78:       num++;
79:       field.setText("" + num);
80:    }
81:
82:    boolean numberGone(int num, JTextField[] pastNums, int count) {
83:       for (int i = 0; i < count; i++) {
84:          if (Integer.parseInt(pastNums[i].getText()) == num) {
85:             return true;
86:          }
87:       }
88:       return false;
89:    }
90:
91:    boolean matchedOne(JTextField win, JTextField[] allPicks) {
92:       for (int i = 0; i < 6; i++) {
93:          String winText = win.getText();
```

```
94:          if ( winText.equals( allPicks[i].getText() ) ) {
95:             return true;
96:           }
97:        }
98:        return false;
99:     }
100:
101:    public void run() {
102:       Thread thisThread = Thread.currentThread();
103:       while (playing == thisThread) {
104:          addOneToField(gui.drawings);
105:          int draw = Integer.parseInt(gui.drawings.getText());
106:          float numYears = (float)draw / 104;
107:          gui.years.setText("" + numYears);
108:
109:          int matches = 0;
110:          for (int i = 0; i < 6; i++) {
111:             int ball;
112:             do {
113:                ball = (int) Math.floor(Math.random() * 50 + 1);
114:             } while (numberGone(ball, gui.winners, i));
115:             gui.winners[i].setText("" + ball);
116:             if (matchedOne(gui.winners[i], gui.numbers)) {
117:                matches++;
118:             }
119:          }
120:          switch (matches) {
121:             case 3:
122:                addOneToField(gui.got3);
123:                break;
124:             case 4:
125:                addOneToField(gui.got4);
126:                break;
127:             case 5:
128:                addOneToField(gui.got5);
129:                break;
130:             case 6:
131:                addOneToField(gui.got6);
132:                gui.stop.setEnabled(false);
133:                gui.play.setEnabled(true);
134:                playing = null;
135:          }
136:          try {
137:             Thread.sleep(100);
138:          } catch (InterruptedException e) {
139:             // no hacer nada
140:          }
141:       }
142:    }
143: }
```

La clase `LottoEvent` tiene un constructor: `LottoEvent(LottoMadness)`. El objeto `LottoMadness` especificado como argumento identifica el objeto que depende de `LottoEvent` para procesar eventos de usuario y realizar los sorteos. En la clase se usan los siguientes métodos:

- `clearAllFields()` vacía todos los campos de texto de la aplicación. Se procesa cuando el usuario pulsa el botón **Reset**.
- `addOneToField()` convierte un campo de texto en un entero, lo incrementa en uno y lo vuelve a convertir en campo de texto. Como se almacenan como cadenas, debe realizar pasos especiales para emplearlos en expresiones.
- `numberGone()` acepta tres argumentos: un número del sorteo, una matriz que contiene varios objetos `JTextField` y un entero `count`. Este método comprueba que cada número del sorteo no se haya seleccionado ya en el mismo sorteo.
- `matchedOne()` acepta dos argumentos: un objeto `JTextField` y una matriz de seis objetos `JTextField`. El método comprueba si uno de los números del usuario coincide con los del sorteo actual.

El método `actionPerformed()` de la aplicación recibe los eventos de acción cuando el usuario pulsa un botón. El método `getActionCommand()` recupera la etiqueta del botón para determinar qué componente se ha pulsado.

Al pulsar el botón **Play** se invoca el método `startPlaying()`. Este método desactiva cuatro componentes. Al pulsar **Stop** se invoca el método `stopPlaying()`, que habilita todos los componentes menos el botón **Stop**.

El método `itemStateChanged()` recibe eventos de usuario desencadenados por la selección de la casillas de verificación Quick Pick o Personal. El método `getItem()` devuelve un objeto que representa la casilla marcada. Si es Quick Pick, se asignan seis números aleatorios del 1 al 50 a los números del usuario. En caso contrario, se borran los campos de texto que contienen los números del usuario.

La clase `LottoEvent` usa números del 1 a 50 para cada bola del sorteo. Se establece en la línea 113, que multiplica el método `Math.random()` por 50, añade `1` al total y lo emplea como argumento del método `Math.floor()`. El resultado final es un entero aleatorio del 1 al 50. Si cambia `50` por otro número aquí y en la línea 65, puede usar `LottoMadness` para sorteos de lotería que generen más valores.

El proyecto `LottoMadness` carece de variables para controlar aspectos como el número de sorteos, los sorteos ganadores y campos de texto de números de lotería. En su lugar, la interfaz almacena los valores y los muestra automáticamente.

Para finalizar el proyecto, vuelva a abrir `LottoMadness.java` en NetBeans. Basta con añadir seis líneas para que funcione con la clase `LottoEvent`.

Primero, añada una nueva variable de instancia para guardar un objeto `LottoEvent`:

```
LottoEvent lotto = new LottoEvent(this);
```

Tras ello, en el constructor `LottoMadness()`, invoque los métodos `addItemListener()` y `addActionListener()` de cada componente de la interfaz de usuario que pueda recibir entradas del usuario:

```
// Añadir escuchadores
quickpick.addItemListener(lotto);
personal.addItemListener(lotto);
```

```
stop.addActionListener(lotto);
play.addActionListener(lotto);
reset.addActionListener(lotto);
```

El listado 15.3 contiene el texto completo de `LottoMadness.java` tras realizar los cambios. Las líneas nuevas se muestran en negrita, el resto no cambia con respecto al anterior.

Listado 15.3. Código completo de LottoMadness.java.

```
 1: import java.awt.*;
 2: import javax.swing.*;
 3:
 4: public class LottoMadness extends JFrame {
 5:   LottoEvent lotto = new LottoEvent(this);
 6:
 7:   // establecer fila 1
 8:   JPanel row1 = new JPanel();
 9:   ButtonGroup option = new ButtonGroup();
10:   JCheckBox quickpick = new JCheckBox("Quick Pick", false);
11:   JCheckBox personal = new JCheckBox("Personal", true);
12:   // establecer fila 2
13:   JPanel row2 = new JPanel();
14:   JLabel numbersLabel = new JLabel("Your picks: ", JLabel.RIGHT);
15:   JTextField[] numbers = new JTextField[6];
16:   JLabel winnersLabel = new JLabel("Winners: ", JLabel.RIGHT);
17:   JTextField[] winners = new JTextField[6];
18:   // establecer fila 3
19:   JPanel row3 = new JPanel();
20:   JButton stop = new JButton("Stop");
21:   JButton play = new JButton("Play");
22:   JButton reset = new JButton("Reset");
23:   // establecer fila 4
24:   JPanel row4 = new JPanel();
25:   JLabel got3Label = new JLabel("3 of 6: ", JLabel.RIGHT);
26:   JTextField got3 = new JTextField("0");
27:   JLabel got4Label = new JLabel("4 of 6: ", JLabel.RIGHT);
28:   JTextField got4 = new JTextField("0");
29:   JLabel got5Label = new JLabel("5 of 6: ", JLabel.RIGHT);
30:   JTextField got5 = new JTextField("0");
31:   JLabel got6Label = new JLabel("6 of 6: ", JLabel.RIGHT);
32:   JTextField got6 = new JTextField("0", 10);
33:   JLabel drawingsLabel = new JLabel("Drawings: ", JLabel.RIGHT);
34:   JTextField drawings = new JTextField("0");
35:   JLabel yearsLabel = new JLabel("Years: ", JLabel.RIGHT);
36:   JTextField years = new JTextField("0");
37:
38:   public LottoMadness() {
39:     super("Lotto Madness");
40:     setLookAndFeel();
41:     setSize(550, 270);
42:     setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
43:     GridLayout layout = new GridLayout(5, 1, 10, 10);
44:     setLayout(layout);
```

```
45:
46:      // Añadir escuchadores
47:      quickpick.addItemListener(lotto);
48:      personal.addItemListener(lotto);
49:      stop.addActionListener(lotto);
50:      play.addActionListener(lotto);
51:      reset.addActionListener(lotto);
52:
53:      FlowLayout layout1 = new FlowLayout(FlowLayout.CENTER,
54:         10, 10);
55:      option.add(quickpick);
56:      option.add(personal);
57:      row1.setLayout(layout1);
58:      row1.add(quickpick);
59:      row1.add(personal);
60:      add(row1);
61:
62:      GridLayout layout2 = new GridLayout(2, 7, 10, 10);
63:      row2.setLayout(layout2);
64:      row2.add(numbersLabel);
65:      for (int i = 0; i < 6; i++) {
66:         numbers[i] = new JTextField();
67:         row2.add(numbers[i]);
68:      }
69:      row2.add(winnersLabel);
70:      for (int i = 0; i < 6; i++) {
71:         winners[i] = new JTextField();
72:         winners[i].setEditable(false);
73:         row2.add(winners[i]);
74:      }
75:      add(row2);
76:
77:      FlowLayout layout3 = new FlowLayout(FlowLayout.CENTER,
78:         10, 10);
79:      row3.setLayout(layout3);
80:      stop.setEnabled(false);
81:      row3.add(stop);
82:      row3.add(play);
83:      row3.add(reset);
84:      add(row3);
85:
86:      GridLayout layout4 = new GridLayout(2, 3, 20, 10);
87:      row4.setLayout(layout4);
88:      row4.add(got3Label);
89:      got3.setEditable(false);
90:      row4.add(got3);
91:      row4.add(got4Label);
92:      got4.setEditable(false);
93:      row4.add(got4);
94:      row4.add(got5Label);
95:      got5.setEditable(false);
96:      row4.add(got5);
97:      row4.add(got6Label);
98:      got6.setEditable(false);
99:      row4.add(got6);
```

```
100:       row4.add(drawingsLabel);
101:       drawings.setEditable(false);
102:       row4.add(drawings);
103:       row4.add(yearsLabel);
104:       years.setEditable(false);
105:       row4.add(years);
106:       add(row4);
107:
108:       setVisible(true);
109:    }
110:
111:    private void setLookAndFeel() {
112:       try {
113:          UIManager.setLookAndFeel(
114:             "com.sun.java.swing.plaf.nimbus.NimbusLookAndFeel"
115:          );
116:       } catch (Exception exc) {
117:          // ignorar el error
118:       }
119:    }
120:
121:    public static void main(String[] arguments) {
122:       LottoMadness frame = new LottoMadness();
123:    }
124: }
```

Tras añadir las líneas en negrita puede ejecutar la aplicación, que mostrará sus probabilidades durante miles de años. Como habrá imaginado, estos sorteos son un ejercicio de futilidad. La probabilidad de ganar un sorteo de seis números en la vida son realmente mínimas, aunque viva tanto como una figura bíblica.

Nota

En el sitio Web del libro encontrará un enlace a una versión de applet del programa `LottoMadness`. Al cierre de esta edición, se habían realizado 410.732.244 sorteos, lo que equivale a 3,9 millones de años de sorteos bisemanales. Hubo 6.364.880 de acertantes de tres números, 337.285 de cuatro, 6.476 de cinco y 51 de seis (aproximadamente uno de cada 8 millones de sorteos). La primera persona en ganar esta lotería virtual fue Bill Teer, el 14 de agosto de 2000, más de cuatro años después de que se publicara el applet. Sus números fueron 3, 7, 1, 15, 34 y 43, y solo necesitó 241.225 sorteos (2.319,47 años) para ganar.

Resumen

Puede crear un programa de aspecto profesional con una modesta cantidad de programación en Swing. Aunque la aplicación `LottoMadness` es más extensa que las de otros capítulos anteriores, la mitad del programa la forman instrucciones para crear la interfaz.

Si dedica más tiempo a ejecutar la aplicación, le corroerá la envidia de los ganadores de estos sorteos de lotería de seis números.

Mi última prueba reveló que podría malgastar 27.000 dólares y los mejores 266 años de mi vida jugando a la lotería, solo para ganar algunos premios de 3 y 4 aciertos. En comparación, la probabilidad de que se gane la vida con sus conocimientos sobre Java es casi una apuesta segura.

Preguntas y respuestas

P: ¿Qué se necesita con el método `paint()` o `repaint()` para indicar que un campo de texto se ha modificado?

R: Tras emplear el método `setText()` de un componente de texto para cambiar su valor, no necesita nada más. Swing se encarga de la actualización necesaria para mostrar el nuevo valor.

P: ¿Por qué se importa una clase y una de sus subclases, como en el listado 15.1, al importar `java.awt.*` y `java.awt.event.*`? ¿La primera de estas instrucciones podría incluir la segunda?

R: Aunque los nombres de los paquetes `java.awt` y `java.awt.event` parezcan estar relacionados, no existe la herencia en los paquetes de Java. Un paquete no puede ser subpaquete de otro.

Al usar un asterisco en una instrucción `import`, todas las clases del paquete estarán disponibles para el programa. El asterisco solo funciona con clases, no paquetes. Lo máximo que puede cargar una instrucción `import` son las clases de un único paquete.

Ejercicios

Tras sufrir con el programa `LottoMadness`, ponga a prueba sus conocimientos con las siguientes preguntas.

Preguntas

1. ¿Por qué los eventos de acción se denominan así?

   A. Se producen en reacción a algo.

   B. Indican que se va a realizar una acción como respuesta.

   C. Es un homenaje al aventurero Action Jackson.

2. ¿Qué significa `this` como argumento de un método `addActionListener()`?
   A. Un escuchador que debe emplearse al producirse un evento.
   B. Un evento que tiene preferencia sobre otros.
   C. Un objeto que procesa los eventos.
3. ¿Qué componente almacena entradas del usuario como enteros?
   A. `JButton`.
   B. `JTextField` o `JTextArea`.
   C. Ninguno de los anteriores.

Respuestas

1. B. Los eventos de acción incluyen el clic de un botón y la selección de un elemento de un menú desplegable.
2. C. La palabra clave `this` hace referencia al objeto actual. Si se usa el nombre del objeto como argumento en lugar de `this`, ese objeto recibe los eventos y se esperará que los procese.
3. B. `JTextField` y `JTextArea` almacenan sus valores como texto, de modo que debe convertirlos antes de poder emplearlos como enteros, números de coma flotante u otros valores que no sean texto.

Actividades

Si el evento principal de este capítulo no le ha proporcionado suficiente acción, pruebe con las siguientes actividades:

- Añada un campo de texto a la aplicación `LottoMadness` que funcione junto a la instrucción `Thread.sleep()` de la clase `LottoEvent` para reducir el ritmo de los sorteos.
- Cambie el proyecto `LottoMadness` para que muestre cinco números del 1 al 90.

16
Crear una interfaz de usuario compleja

La creación de una interfaz gráfica de usuario (IGU) con Swing implica más que aprender a usar los distintos componentes, administradores de diseño y métodos de control de eventos. También debe familiarizarse con todo lo que ofrece Swing.

Sus más de 400 clases diferentes hacen que Swing sea una de las bibliotecas de clases más completas de Java. Muchas de ellas se pueden implementar por medio de las técnicas empleadas en capítulos anteriores; los contenedores y los componentes de Swing comparten superclases, lo que les otorga un comportamiento común.

En este capítulo veremos otros componentes que puede emplear en sus programas de Swing.

Paneles de desplazamiento

Los componentes de una IGU suelen ser mayores que el área de la que disponen. Para pasar de una parte del componente a otra, se usan barras de desplazamiento verticales y horizontales. En Swing puede ofrecer desplazamiento si añade un componente a un panel de desplazamiento, un contenedor representado por la clase `JScrollPane`. Puede crear un panel de desplazamiento por medio de los siguientes constructores:

- **JScrollPane():** Crea un panel de desplazamiento con una barra de desplazamiento horizontal y vertical que aparece cuando se necesita.
- **JScrollPane(int, int):** Crea un panel de desplazamiento con la barra de desplazamiento vertical especificada y barras de desplazamiento horizontales.

- **JScrollPane(Component):** Crea un panel de desplazamiento que contiene el componente de interfaz de usuario especificado.
- **JScrollPane(Component, int, int):** Crea un panel de desplazamiento con el componente especificado, una barra de desplazamiento vertical y otra horizontal.

Los argumentos enteros de estos constructores determinan cómo se emplean las barras de desplazamiento en el panel. Use las siguientes variables de clase como argumentos:

- `JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED` o `JScrollPane.HORIZONTAL_SCROLLBAR_AS_NEEDED`
- `JScrollPane.VERTICAL_SCROLLBAR_NEVER` o `JScrollPane.HORIZONTAL_SCROLLBAR_NEVER`
- `JScrollPane.VERTICAL_SCROLLBAR_ALWAYS` o `JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS`

Si ha creado un panel de desplazamiento sin un componente, puede usar el método `add(Component)` del panel para añadir componentes. Una vez configurado el panel de desplazamiento, debe añadirse a un contenedor en lugar del componente.

Para ver una aplicación que incluye un panel de desplazamiento, introduzca el listado 16.1 en un nuevo archivo vacío de Java con el nombre `MailWriter` y guarde el archivo.

Listado 16.1. Texto completo de MailWriter.java.

```
 1: import javax.swing.*;
 2: import java.awt.*;
 3:
 4: public class MailWriter extends JFrame {
 5:
 6:   public MailWriter() {
 7:     super("Write an E-Mail");
 8:     setLookAndFeel();
 9:     setSize(370, 270);
10:     setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
11:     FlowLayout flow = new FlowLayout(FlowLayout.RIGHT);
12:     setLayout(flow);
13:
14:     JPanel row1 = new JPanel();
15:     JLabel toLabel = new JLabel("To:");
16:     row1.add(toLabel);
17:     JTextField to = new JTextField(24);
18:     row1.add(to);
19:     add(row1);
20:
21:     JPanel row2 = new JPanel();
22:     JLabel subjectLabel = new JLabel("Subject:");
23:     row2.add(subjectLabel);
24:     JTextField subject = new JTextField(24);
25:     row2.add(subject);
26:     add(row2);
```

```
27:
28:       JPanel row3 = new JPanel();
29:       JLabel messageLabel = new JLabel("Message:");
30:       row3.add(messageLabel);
31:       JTextArea message = new JTextArea(4, 22);
32:       message.setLineWrap(true);
33:       message.setWrapStyleWord(true);
34:       JScrollPane scroll = new JScrollPane(message,
35:          JScrollPane.VERTICAL_SCROLLBAR_ALWAYS,
36:          JScrollPane.HORIZONTAL_SCROLLBAR_NEVER);
37:       row3.add(scroll);
38:       add(row3);
39:
40:       JPanel row4 = new JPanel();
41:       JButton send = new JButton("Send");
42:       row4.add(send);
43:       add(row4);
44:
45:       setVisible(true);
46:    }
47:
48:    private void setLookAndFeel() {
49:       try {
50:          UIManager.setLookAndFeel(
51:             "com.sun.java.swing.plaf.nimbus.NimbusLookAndFeel"
52:          );
53:       } catch (Exception exc) {
54:          // ignorar el error
55:       }
56:    }
57:
58:    public static void main(String[] arguments) {
59:       MailWriter mail = new MailWriter();
60:    }
61: }
```

Al ejecutar la aplicación, verá una ventana similar a la mostrada en la figura 16.1.

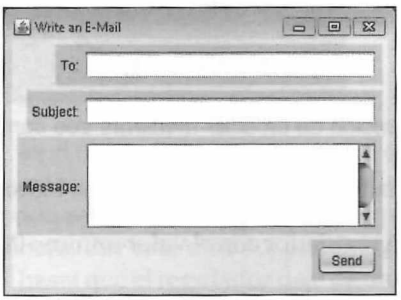


Figura 16.1. Área de texto con desplazamiento en una aplicación.

La aplicación `MailWriter` es una IGU empleada para redactar un correo electrónico. El programa carece de código de control de eventos, de modo que no puede hacer nada con el texto introducido en el formulario.

El texto del correo electrónico se introduce en un área de texto de desplazamiento, que se implementa creando un área de texto y añadiéndola a un panel de desplazamiento con las siguientes instrucciones:

```
JTextArea message = new JTextArea(4, 22);
message.setLineWrap(true);
message.setWrapStyleWord(true);
JScrollPane scroll = new JScrollPane(message,
   JScrollPane.VERTICAL_SCROLLBAR_ALWAYS,
   JScrollPane.HORIZONTAL_SCROLLBAR_NEVER);
row3.add(scroll);
```

Reguladores

La forma más sencilla de obtener entradas numéricas de los usuarios consiste en usar un regulador, un componente que se puede arrastrar horizontal o verticalmente. En Swing, los reguladores se representan con la clase `JSlider`. Permiten elegir un número entre un valor mínimo y otro máximo.

Estos valores se pueden mostrar en una etiqueta que incluye el valor mínimo, el máximo y los valores intermedios. En la figura 16.2 puede ver un ejemplo que crearemos más adelante.

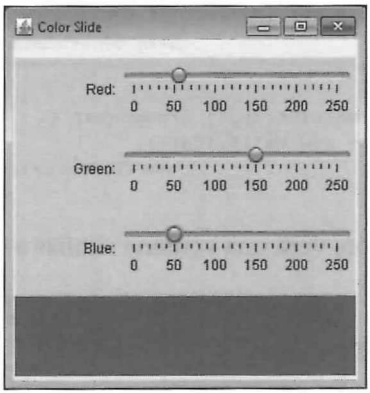


Figura 16.2. Selección de un color mediante tres componentes de regulador.

Puede crear un regulador horizontal con uno de los siguientes constructores:

- **JSlider():** Crea un regulador con el valor mínimo 0, el valor máximo 100 y el valor inicial 50.
- **JSlider(int, int):** Crea un regulador con los valores mínimo y máximo especificados.
- **JSlider(int, int, int):** Crea un regulador con los valores mínimo, máximo e inicial especificados.

Para crear un regulador vertical, use un constructor con un primer argumento adicional: la orientación del regulador. Este argumento debe ser la variable de clase `JSlider.VERTICAL` o `JSlider.HORIZONTAL`.

La siguiente instrucción crea un regulador vertical para un número entre 1 y 1000:

```
JSlider guess = new JSlider(JSlider.VERTICAL, 1, 1000, 500);
```

Este regulador comienza con el selector, la parte del componente para seleccionar el número, en la posición 500.

Para mostrar una etiqueta para el regulador, debe indicar la información que mostrará la etiqueta. Invoque los métodos `setMajorTickSpacing(int)` y `setMinorTickSpacing(int)` del regulador para determinar la frecuencia de las marcas mostradas en la etiqueta. Las marcas mayores se muestran con mayor grosor que las menores. Una vez definida la frecuencia de las marcas, invoque el método `setPaintTicks(boolean)` del regulador con el argumento `true`. También puede mostrar el valor numérico de cada marca si invoca el método `setPaintLabels(boolean)` con `true`.

Escuchadores de cambios

Para controlar la entrada de un regulador, debe contar con una clase que implemente la interfaz `ChangeListener` del paquete `javax.swing.event`. Esta interfaz incluye un solo método:

```
public void stateChanged(ChangeEvent event); {
   // instrucciones para procesar el evento
}
```

Para registrar un objeto como escuchador de cambios, invoque el método `addChangeListener(Object)` del contenedor del regulador. Al mover el regulador, se invoca el método `stateChanged()` del objeto que escucha.

Este método se invoca con un objeto `ChangeEvent` que puede identificar el regulador que ha cambiado de valor. Invoque el método `getSource()` del objeto y convierta el objeto en `JSlider`, como en la siguiente instrucción:

```
JSlider changedSlider = (JSlider) event.getSource();
```

En este ejemplo, `event` es el objeto `ChangeEvent` que es un argumento del método `stateChanged()`. Los eventos de cambio se producen durante el movimiento de un regulador. Comienzan cuando se mueve el regulador por primera vez y no se detienen hasta que el regulador se libera. Por este motivo, puede que no desee hacer nada en el método `stateChanged()` hasta que el regulador deje de moverse. Para comprobar si un regulador se está moviendo, invoque su método `getValueIsAdjusting()`. Devuelve `true` si se produce movimiento o `false` en caso contrario.

Esta técnica se ilustra en el siguiente proyecto, una aplicación de Java que emplea tres reguladores para seleccionar un color. En Java, los colores se crean con la clase `Color` del paquete `java.awt`.

Una forma de crear un objeto `Color` consiste en especificar la cantidad de rojo, verde y azul de un color. Estos valores pueden ser un entero entre 0 y 255, siendo 255 la cantidad máxima de dicho color. La siguiente instrucción crea un objeto `Color` que representa un color caramelo:

```
Color butterscotch = new Color(255, 204, 128);
```

El valor de rojo usado para crear este color es `255`, por lo que contiene la cantidad máxima. También contiene una cantidad elevada de verde y algo de azul.

El listado 16.2 contiene la aplicación `ColorSliders`, con tres reguladores, tres etiquetas y un panel para mostrar el color. Cree un nuevo archivo vacío de Java con el nombre `ColorSliders`, introduzca el código en el editor y guarde el archivo.

Listado 16.2. Texto completo de ColorSliders.java.

```
 1: import javax.swing.*;
 2: import javax.swing.event.*;
 3: import java.awt.*;
 4:
 5: public class ColorSliders extends JFrame implements ChangeListener {
 6:   ColorPanel canvas;
 7:   JSlider red;
 8:   JSlider green;
 9:   JSlider blue;
10:
11:   public ColorSliders() {
12:      super("Color Slide");
13:      setLookAndFeel();
14:      setSize(270, 300);
15:      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
16:      setVisible(true);
17:
18:      canvas = new ColorPanel();
19:      red = new JSlider(0, 255, 255);
20:      green = new JSlider(0, 255, 0);
21:      blue = new JSlider(0, 255, 0);
22:
23:      red.setMajorTickSpacing(50);
24:      red.setMinorTickSpacing(10);
25:      red.setPaintTicks(true);
26:      red.setPaintLabels(true);
27:      red.addChangeListener(this);
28:
29:      green.setMajorTickSpacing(50);
30:      green.setMinorTickSpacing(10);
31:      green.setPaintTicks(true);
32:      green.setPaintLabels(true);
33:      green.addChangeListener(this);
34:
35:      blue.setMajorTickSpacing(50);
36:      blue.setMinorTickSpacing(10);
37:      blue.setPaintTicks(true);
38:      blue.setPaintLabels(true);
39:      blue.addChangeListener(this);
```

```
40:
41:       JLabel redLabel = new JLabel("Red: ");
42:       JLabel greenLabel = new JLabel("Green: ");
43:       JLabel blueLabel = new JLabel("Blue: ");
44:       GridLayout grid = new GridLayout(4, 1);
45:       FlowLayout right = new FlowLayout(FlowLayout.RIGHT);
46:       setLayout(grid);
47:
48:       JPanel redPanel = new JPanel();
49:       redPanel.setLayout(right);
50:       redPanel.add(redLabel);
51:       redPanel.add(red);
52:       add(redPanel);
53:
54:       JPanel greenPanel = new JPanel();
55:       greenPanel.setLayout(right);
56:       greenPanel.add(greenLabel);
57:       greenPanel.add(green);
58:       add(greenPanel);
59:
60:       JPanel bluePanel = new JPanel();
61:       bluePanel.setLayout(right);
62:       bluePanel.add(blueLabel);
63:       bluePanel.add(blue);
64:       add(bluePanel);
65:       add(canvas);
66:
67:       setVisible(true);
68:    }
69:
70:    public void stateChanged(ChangeEvent event) {
71:       JSlider source = (JSlider) event.getSource();
72:       if (source.getValueIsAdjusting() != true) {
73:          Color current = new Color(red.getValue(), green.getValue(),
74:             blue.getValue());
75:          canvas.changeColor(current);
76:          canvas.repaint();
77:       }
78:    }
79:
80:    public Insets getInsets() {
81:       Insets border = new Insets(45, 10, 10, 10);
82:       return border;
83:    }
84:
85:    private void setLookAndFeel() {
86:       try {
87:          UIManager.setLookAndFeel(
88:             "com.sun.java.swing.plaf.nimbus.NimbusLookAndFeel"
89:          );
90:       } catch (Exception exc) {
91:          // ignorar el error
92:       }
93:    }
94:
95:    public static void main(String[] arguments) {
```

```
 96:       ColorSliders cs = new ColorSliders();
 97:    }
 98: }
 99:
100: class ColorPanel extends JPanel {
101:    Color background;
102:
103:    ColorPanel() {
104:       background = Color.red;
105:    }
106:
107:    public void paintComponent(Graphics comp) {
108:       Graphics2D comp2D = (Graphics2D) comp;
109:       comp2D.setColor(background);
110:       comp2D.fillRect(0, 0, getSize().width, getSize().height);
111:    }
112:
113:    void changeColor(Color newBackground) {
114:       background = newBackground;
115:    }
116: }
```

Al ejecutar la aplicación, como vimos en la figura 16.2, un marco contiene tres reguladores que representan la cantidad de rojo, verde y azul en un panel en la parte inferior.

Ajuste los valores de cada regulador para cambiar el color mostrado. En la figura 16.2, la aplicación se emplea para crear el verde North Texas Mean (rojo 50, verde 150 y azul 50). Este tono hace que los alumnos de la Universidad de North Texas se suelten la melena en eventos deportivos y muestren sus feroces garras de águila a los equipos visitantes, que se quedan amarillos (rojo 255, verde 255, naranja 0).

Usar iconos de imágenes y barras de herramientas

Una de las formas más sencillas de mejorar el aspecto visual de una IGU consiste en emplear iconos, pequeñas imágenes para identificar botones y otras partes de una interfaz. Con muchos de los componentes de la biblioteca de clases Swing puede etiquetar un componente con una imagen en lugar de texto por medio de la clase `ImageIcon` del paquete `javax.swing`.

Puede crear un objeto `ImageIcon` a partir de un archivo de su equipo si invoca el método `ImageIcon(String)`. El argumento del método puede ser el nombre del archivo o su ubicación y su nombre, como en estos ejemplos:

```
ImageIcon stopSign = new ImageIcon("stopsign.gif");
ImageIcon saveFile = new ImageIcon("images/savefile.gif");
```

El archivo gráfico empleado para crear el icono debe tener formato GIF, JPEG o PNG. La mayoría son GIF, muy indicado para mostrar pequeños gráficos con un número limitado de colores.

El constructor `ImageIcon` carga inmediatamente la imagen completa desde el archivo. Puede usar iconos de imagen como etiquetas y botones por medio de los métodos de constructor `JLabel(ImageIcon)` y `JButton(ImageIcon)`:

```
ImageIcon siteLogo = new ImageIcon("siteLogo.gif");
JLabel logoLabel = new JLabel(siteLogo);
ImageIcon searchWeb = new ImageIcon("searchGraphic.gif");
JButton search = new JTextField(searchWeb);
```

Varios componentes pueden tener un icono y una etiqueta de texto. La siguiente instrucción crea un botón con ambos elementos:

```
JButton refresh = new JButton("Refresh",
   "images/refreshIcon.gif");
```

Nota

Aunque algunos sistemas operativos emplean el carácter \ para separar carpetas y nombres de archivo, el constructor `ImageIcon` requiere el carácter / como separador.

Los iconos de imagen suelen usarse en barras de herramientas, contenedores que agrupan varios componentes en una fila o columna.

Las barras de herramientas, que se crean con la clase `JToolBar`, se pueden diseñar para que un usuario las cambie de posición en la IGU. El proceso se denomina acoplar y estos componentes también se llaman barras de herramientas acoplables.

Puede crear una barra de herramientas con uno de los siguientes métodos:

- **JToolBar():** Crea una barra de herramientas que alinea los componentes horizontalmente.
- **JToolBar(int):** Crea una barra de herramientas que alinea los componentes en la dirección especificada, que puede ser `SwingConstants.HORIZONTAL` o `SwingConstants.VERTICAL`.

Los componentes se añaden a una barra de herramientas de la misma forma que a otros contenedores: se invoca el método `add(Component)` con el componente que añadir.

Para poder acoplar una barra de herramientas, debe incluirse en un contenedor que use `BorderLayout` como administrador de diseño. Organiza el contenedor en áreas norte, sur, este, oeste y centro. Sin embargo, al emplear una barra de herramientas acoplable, el contenedor solo puede usar dos de estas áreas: el centro y una de dirección.

La barra de herramientas debe añadirse al área de dirección. Las siguientes instrucciones crean una barra de herramientas acoplable vertical con tres botones de icono:

```
BorderLayout border = new BorderLayout();
pane.setLayout(border);
JToolBar bar = new JToolBar(SwingConstants.VERTICAL);
ImageIcon play = new ImageIcon("play.gif");
JButton playButton = new JButton(play);
```

```
ImageIcon stop = new ImageIcon("stop.gif");
JButton stopButton = new JButton(stop);
ImageIcon pause = new ImageIcon("pause.gif");
JButton pauseButton = new JButton(pause);
bar.add(playButton);
bar.add(stopButton);
bar.add(pauseButton);
add(bar, BorderLayout.WEST);
```

El siguiente proyecto del capítulo es `Tool`, una aplicación de Java que incluye iconos de imagen y una barra de herramientas acoplable. Cree un archivo vacío de Java con el nombre `Tool`, introduzca el listado 16.3 y guárdelo.

Listado 16.3. Texto completo de Tool.java.

```
 1: import java.awt.*;
 2: import java.awt.event.*;
 3: import javax.swing.*;
 4:
 5: public class Tool extends JFrame {
 6:   public Tool() {
 7:      super("Tool");
 8:      setLookAndFeel();
 9:      setSize(370, 200);
10:      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
11:
12:      // crear botones de la barra de herramientas
13:      ImageIcon image1 = new ImageIcon("newfile.gif");
14:      JButton button1 = new JButton(image1);
15:      ImageIcon image2 = new ImageIcon("openfile.gif");
16:      JButton button2 = new JButton(image2);
17:      ImageIcon image3 = new ImageIcon("savefile.gif");
18:      JButton button3 = new JButton(image3);
19:
20:      // crear la barra de herramientas
21:      JToolBar bar = new JToolBar();
22:      bar.add(button1);
23:      bar.add(button2);
24:      bar.add(button3);
25:
26:      // crear el área de texto
27:      JTextArea edit = new JTextArea(8, 40);
28:      JScrollPane scroll = new JScrollPane(edit);
29:
30:      // crear marco
31:      BorderLayout border = new BorderLayout();
32:      setLayout(border);
33:      add("North", bar);
34:      add("Center", scroll);
35:      setVisible(true);
36:   }
37:
38:   private void setLookAndFeel() {
39:      try {
40:         UIManager.setLookAndFeel(
```

```
41:             "com.sun.java.swing.plaf.nimbus.NimbusLookAndFeel"
42:          );
43:       } catch (Exception exc) {
44:          // ignorar el error
45:       }
46:    }
47:
48:    public static void main(String[] arguments) {
49:       Tool frame = new Tool();
50:    }
51: }
```

La aplicación `Tool` requiere tres archivos gráficos para crear los iconos de la barra de herramientas: `newfile.gif`, `openfile.gif` y `savefile.gif`. Puede encontrarlos en la página del libro. Guárdelos en la carpeta del proyecto `Java24` (o la que haya designado para sus proyectos de Java en NetBeans). Las figuras 16.3 y 16.4 muestran dos capturas de esta aplicación en funcionamiento. La barra de herramientas se desplaza desde su posición original (véase la figura 16.3) a otro borde de la interfaz (véase la figura 16.4).

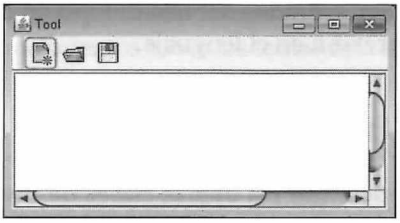


Figura 16.3. Una aplicación con una barra de herramientas.

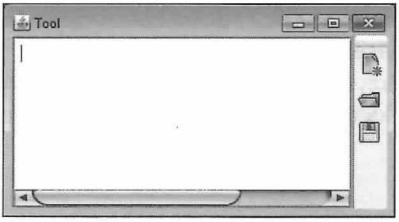


Figura 16.4. Acople de una barra de herramientas en una nueva posición.

Nota

¿Tiene problemas para localizar esta carpeta? Inicie un nuevo proyecto en NetBeans: seleccione Archivo>Proyecto Nuevo, la categoría Java, el tipo de proyecto Java application (Aplicación de Java) y pulse **Siguiente**. El campo de texto Ubicación del proyecto indicará la ubicación de la carpeta en la que se guardarán los iconos.

Compile la aplicación y pruébela moviendo la barra de herramientas. Para ello, puede hacer clic y arrastrarla hasta otro borde del área de texto. Al liberarla, se coloca junto a ese borde y el área de texto se desplaza para hacer sitio.

Oracle ofrece un repositorio de gráficos de icono que puede emplear en sus programas. Los tres iconos utilizados en este ejercicio son de esa colección. Para ver los gráficos, visite `http://java.sun.com/developer/techDocs/hi/repository`.

Nota

También puede sacar una barra de herramientas de una interfaz. Se abrirá una nueva ventana con la barra de herramientas.

Resumen

Este capítulo es el último dedicado a Swing, la parte del lenguaje Java para software de IGU. Aunque Swing es sin duda la parte más amplia de la biblioteca de clases de Java, la mayoría de sus clases se usa de forma similar. Una vez que aprenda a crear un componente, a añadirlo a un contenedor, a aplicar un administrador de diseño a un contenedor y a responder a entradas del usuario, podrá emplear muchas de las nuevas clases de Swing que aparezcan en el lenguaje.

Preguntas y respuestas

P: ¿Dónde puedo encontrar las demás clases de Swing en la biblioteca de clases de Java?

R: En el sitio oficial de Java de Oracle se publica la documentación completa de la biblioteca de clases de Java: `http://download.oracle.com/javase/7/docs/api`. Puede ver las clases incluidas en los paquetes `javax.swing`, `java.awt` y `java.awt.event`, descritos en capítulos anteriores. Todas las clases e interfaces de Swing están documentadas, incluidos sus constructores, variables de clase y de instancia.

Ejercicios

Ejercite sus neuronas con las siguientes preguntas sobre paneles de desplazamiento, iconos de imagen y otras características de Swing.

Preguntas

1. ¿Qué formatos de archivo gráfico admite la clase `ImageIcon`?

   A. GIF.

   B. GIF y JPEG.

   C. GIF, PNG y JPEG.

2. ¿Para qué sirve el método `getValueIsAdjusting()` de un objeto `JSlider`?
   A. Determina si ha cambiado el valor original de un regulador.
   B. Determina si el regulador está cambiando actualmente de valor.
   C. No para mucho, es una verdadera desgracia para su superclase principal.

Respuestas

1. C. La compatibilidad con PNG en `ImageIcon` se añadió en Java 1.3.
2. B. El método `getValueIsAdjusting()` devuelve `true` mientras se mueve el regulador y `false` en caso contrario.

Actividades

Para comprobar sus conocimientos sobre los temas del capítulo, realice las siguientes actividades:

- Cree una IGU que incluya un cuadro combinado en un panel de desplazamiento.
- Añada control de eventos a la aplicación `MailWriter` para que muestre el contenido de los componentes `to`, `subject` y `message` con `System.out.println()` al pulsar un botón **Enviar**.

17

Crear programas Web interactivos

Java se ha convertido en un exitoso lenguaje de propósito general que se ejecuta en diferentes plataformas como teléfonos móviles, servidores Web y aplicaciones de Internet. Cuando apareció en 1990, fue el primer lenguaje de programación que podía ejecutarse en un navegador Web. Los applet son programas de Java diseñados para ejecutarse como parte de una página Web. Al detectar un applet de Java en una página, se descarga al equipo del usuario y se ejecuta.

La programación de applet es diferente a la creación de aplicaciones. Como los applet deben descargarse de una página siempre que se ejecutan, son de menor tamaño que muchas aplicaciones para reducir el tiempo de descarga. Además, como se ejecutan en el equipo del usuario, tienen restricciones de seguridad para evitar que se active código dañino o malintencionado.

Métodos estándar de applet

El primer paso para crear un applet consiste en convertirlo en una subclase de `JApplet`, una clase del paquete `javax.swing` de Swing. Un applet se considera una ventana visual dentro de una página Web, de modo que `JApplet` forma parte de Swing junto a botones, barras de desplazamiento y otros componentes de la interfaz de usuario de un programa. Los applet que diseñe heredan el comportamiento y los atributos necesarios para ejecutarse como parte de una página Web. Antes de crear instrucciones en sus applet, podrán interactuar con un navegador Web, cargarse y descargarse, actualizar sus ventanas en respuesta a cambios en el navegador y realizar otras tareas.

Las aplicaciones inician su ejecución con la primera instrucción del bloque `main()`. Los applet de Java carecen de método `main()`, de forma que no hay un punto de partida definido en el programa. En su lugar, un applet tiene un grupo de métodos estándar que se procesan como respuesta a eventos concretos cuando se ejecuta el applet.

A continuación se muestran los eventos que pueden propiciar el procesamiento de uno de los métodos del applet:

- El programa se carga por primera vez, lo que provoca la invocación de los métodos `init()` y `start()` del applet.
- Sucede algo que requiere que la ventana del applet se actualice, lo que invoca el método `paint()` del applet.
- El navegador detiene el programa, lo que invoca el método `stop()` del applet.
- El programa se reinicia tras detenerse, lo que invoca `start()`.
- El programa se descarga hasta que termina de ejecutarse, lo que invoca `destroy()`.

A continuación se muestra un ejemplo de un sencillo applet:

```
public class Skeleton extends javax.swing.JApplet {
   // añadir aquí el programa
}
```

Los archivos de clase de applet deben ser `public` ya que la clase `JApplet` también es pública (si su applet usa otros archivos de clase propios, no es necesario declararlos como `public`).

La clase del applet hereda todos los métodos que se procesan automáticamente cuando es necesario: `init()`, `paint()`, `start()`, `stop()` y `destroy()`. Sin embargo, estos métodos no hacen nada. Si quiere que suceda algo, debe reemplazarlos por nuevas versiones en su applet.

Pintar una ventana de applet

El método `paint()` se emplea para mostrar texto y gráficos en la ventana del applet. Siempre que es necesario mostrar algo en el applet, el método `paint()` se encarga de ello. También puede forzar la invocación de `paint()` con la instrucción `repaint();`.

El método `paint()` se invoca principalmente cuando algo cambia en el navegador o en el sistema operativo que ejecuta el navegador. Por ejemplo, si el usuario cierra una ventana de otro programa situado delante del navegador, se invoca `paint()` para volver a mostrar el applet. El método `paint()` acepta como argumento un objeto `Graphics` del paquete `java.awt`:

```
public void paint(Graphics screen) {
   Graphics2D screen2D = (Graphics2D) screen;
   // añadir aquí instrucciones de pantalla
}
```

La clase **Graphics** representa un contexto gráfico, un entorno en el que se puede mostrar algo. Como hicimos con las aplicaciones Swing, debe convertirlo a un objeto **Graphics2D** del paquete **java.awt** para usar las funciones graficas de Swing.

Más adelante veremos el método **drawString** () para mostrar texto en las clases **Graphics2D.** Si emplea un objeto **Graphics** o **Graphics2D** en su appiet, debe añadir las siguientes instrucciones **import** antes de la instrucción de clase al inicio del archivo:

```
import java.awt.Graphics;
import java.awt.Graphics2D;
```

También puede disponer de todas las clases de **java.awt** con ayuda del carácter comodín *:

```
import java.awt.*;
```

Inícializar un appiet

El método **init** () se invoca una sola vez al ejecutar un appiet. Es el punto perfecto para definir valores para los objetos y variables necesarios para la correcta ejecución del appiet. Este método también es muy adecuado para definir fuentes, colores y el color de fondo de la ventana de appiet. Veamos un ejemplo:

```
public void init() {
   FlowLayout fio = new FlowLayout();
   setLayout(fio);
   JButton run = new JButton("Run");
   add(run);
}
```

Si va a usar una variable en otros métodos, no debe crearse dentro de un método **init** () ya que solo existe en el ámbito de dicho método. Cree las variables que necesite en una clase como variables de instancia.

Iniciar y detener un appiet

Cuando el appiet comienza su ejecución, se invoca su método **start ()**. Al iniciarse un programa, tras el método **init** () se ejecuta **start** (). Tras ello, el método **start** () solo se vuelve a invocar si el appiet deja de ejecutarse y se reinicia. El método **stop** () se invoca cuando un appiet deja de ejecutarse. Este evento puede producirse cuando el usuario sale de la página Web que contiene el appiet y visita otra diferente. También puede producirse al invocar el método **stop** () directamente en un programa.

Destruir un appiet

El método **destroy** () es el opuesto a **init** ()-Se procesa antes de cerrar un appiet y finalizar su ejecución.

Incluir un applet en una página Web

Los applet se añaden a una página Web por medio de HTML, el lenguaje de marcado empleado para crear páginas Web. HTML es una forma de combinar texto con formato, imágenes, sonido y otros elementos, y mostrarlos en un navegador Web. HTML usa comandos de marcado denominados etiquetas incluidas entre símbolos < y >, como <img> para mostrar imágenes, <p> para añadir párrafos y <hl> y </hl> para indicar que el texto es un título.

El rendimiento de las etiquetas HTML puede modificarse mediante atributos que determinan su funcionamiento. El atributo src de una etiqueta img proporciona el nombre del archivo de imagen que debe mostrarse, como en este ejemplo:

```
<img src="graduation.jpg">
```

Este marcado hace que una página Web muestre la imagen almacenada en el archivo graduation.jpg. Una forma de añadir applet a una página Web consiste en emplear la etiqueta applet y varios atributos. El siguiente marcado HTML ejecuta un applet en una página:

```
<applet code="StripYahtzee.class" codebase="javadir" height="300"
width="400">
<p>Sorry, no dice ... this requires a Java-enabled browser.</p>
</applet>
```

El atributo code identifica el nombre del archivo de clase del applet. Si se usa más de un archivo de clase con el applet, code debe hacer referencia a la clase que sea subclase de **JApplet.**

El applet codebase contiene la ruta a la carpeta o subcarpeta del applet y sus archivos. Si se omite el atributo codebase, todos los archivos asociados al applet deben residir en la misma carpeta que la página Web que lo contiene. En el ejemplo anterior, codebase indica que el applet StripYahtzee se encuentra en la subcarpeta javadir.

Los atributos height y width definen el tamaño en píxeles de la ventana del applet en la página Web. Debe tener el tamaño suficiente para acomodar los elementos que muestre en el applet.

Entre las etiquetas <applet> y </applet> puede incluir marcado HTML que mostrar a los usuarios Web cuyos navegadores no admitan Java o lo tengan desactivado.

En el ejemplo anterior, el párrafo **Sorry, no dice...this requires a Java-enabled browser** se muestra en lugar del applet en navegadores sin Java. Aquí puede incluir instrucciones para descargar un navegador compatible con Java desde Oracle **(http://www.java.com/es/).** También puede incluir hipervínculos y otros elementos HTML.

Otro atributo muy útil, align, indica cómo mostrar el applet con respecto al material restante de la página, como texto y gráficos. El valor align="left" alinea el applet a la izquierda de los elementos contiguos de la página y align="right" a la derecha.

Crear un applet

El primer proyecto de este capítulo es un applet que muestra la cadena **Saluton mondo!**, el tradicional saludo en esperanto cada vez más conocido. Puede comprobar la estructura de los applet si recrea la aplicación **Saluton** del capítulo 23 como programa que ejecutar en una página Web. Cree un nuevo archivo vacío de Java con el nombre **SalutonApplet**, introduzca el texto del listado 17.1 y guárdelo.

Listado 17.1. Texto completo de SalutonApplet.java.

```
 1: import java.awt.*;
 2:
 3: public class SalutonApplet extends javax.swing.JApplet {
 4:    String greeting;
 5:
 6:    public void init() {
 7:       greeting = "Saluton mondo!";
 8:    }
 9:
10:    public void paint(Graphics screen) {
11:       Graphics2D screen2D = (Graphics2D) screen;
12:       screen2D.drawString(greeting, 25, 50);
13:    }
14: }
```

El programa **SalutonApplet** almacena la cadena **Saluton mondo!** en el método **init** () de las líneas 6-8 y lo muestra en la ventana del applet en la línea 12. No es necesario que el applet use los métodos **start** (), **stop** () ni **destroy** (), por lo que no se incluyen en el programa. Ejecutaremos el applet después de examinar su código.

Dibujar en una ventana de applet

Para mostrar texto en la ventana de un applet se emplea el método **drawString** () de la clase **Graphics2D**, que dibuja texto en un componente de interfaz de usuario. **drawString** () es similar al método **System.out.println** () que ya hemos usado para mostrar texto en aplicaciones.

Los tres siguientes argumentos se envían a **drawString** ():

- El texto que se va a mostrar, que puede ser distintas cadenas y variables combinadas con el operador +.
- La posición x de un sistema de coordenadas (x,y) en la que mostrar la cadena.
- La posición **y** de un sistema de coordenadas (x,y) en la que mostrar la cadena.

El sistema de coordenadas (x,y) de un applet se emplea con distintos métodos. Comienza con el punto (0,0) de la esquina superior izquierda de la ventana del applet. Los valores x alimentan al desplazar a la derecha y los valores **y** al hacerlo hacia abajo.

Probar el programa SalutonApplet

Los applet de Java no se pueden ejecutar como las aplicaciones, ya que carecen de un método **main ()**. Para ejecutarlos, debe añadir marcado a una página Web que contenga el applet. Para crear una página Web de ejemplo para **SalutonApplet,** siga estos pasos en NetBeans:

1. Seleccione Archivo>Archivo Nuevo para abrir el cuadro de diálogo Archivo Nuevo.
2. Seleccione Otro en el panel Categorías y Archivo HTML en Tipos de Archivos. Pulse Siguiente. Se abrirá el cuadro de diálogo Nuevo Archivo HTML.
3. Asigne el nombre **SalutonApplet** al archivo y pulse Terminar.

NetBeans abre el editor de código fuente con marcado HTML predeterminado. Bórrelo, introduzca el texto del listado 17.2 y guarde el archivo.

Listado 17.2. Texto completo de SalutonApplet.html.

```
 1: <html>
 2: <head>
 3: <title>Saluton Mondo!</title>
 4: </head>
 5: Cbody bgcolor="#000000" text=,,#FF00FF">
 6: <p>This is a Java applet.</p>
 7: <applet
 8:   code="SalutonApplet.class"
 9:   codebase="..\\..\\build\\classes"
10:    height="150"
11:    width="300"
12: >
13: <p>You need a Java-enabled browser to see this.</p>
14: </applet>
15: </body>
16: </htral>
```

La etiqueta **<applet>** se define en las líneas 7-14, pero la línea 13 se ignora en los navegadores con Java.

Tras guardar el archivo, puede verlo en un navegador Web. En el panel Proyecto a la izquierda del editor de código, haga clic con el botón derecho del ratón sobre el archivo **SalutonApplet.html y** seleccione Ver. La página Web se abre en el navegador predeterminado de su equipo (véase la figura 17.1). Al ejecutar el applet en un navegador, se le pedirá permiso para ejecutar el programa. Muchos navegadores Web deben configurarse para permitir programas de Java antes de poder ejecutar un applet.

En los navegadores actuales, los applet de Java se ejecutan por medio de Java Plug-in, un intérprete de Oracle que admite la versión más actualizada del lenguaje.

Un complemento (*plug-in*) es un programa que funciona junto a un navegador Web para ampliar su funcionalidad. Los complementos procesan un tipo de datos que el navegador normalmente no procesa. Apple ofrece un complemento para reproducir películas

QuickTime, Macromedia distribuye otro para ejecutar archivos de animación Flash y otros muchos tipos de contenido especial se admiten de esta forma. El complemento se puede descargar desde el sitio de Java (`http://www.java.com/es/`).

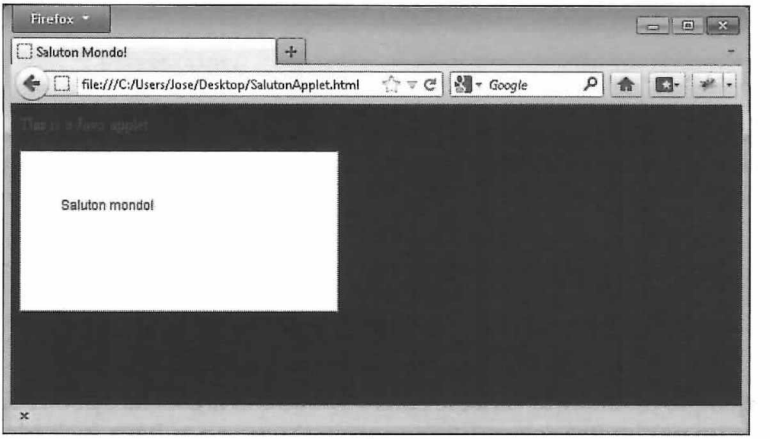


Figura 17.1. El applet `SalutonApplet` en un navegador.

Truco

NetBeans puede probar applet sin necesidad de crear una página Web. Abra el archivo de clase del applet en el editor de código y seleccione Ejecutar>Ejecutar archivo. El applet se carga en la herramienta `appletviewer`, que forma parte del JDK.

Enviar parámetros desde una página Web

La funcionalidad de un applet se puede personalizar mediante parámetros, ajustes almacenados en marcado HTML similares a los argumentos de línea de comandos de una aplicación. Los parámetros se almacenan como parte de la página Web que contiene el applet. Se crean con la etiqueta HTML `param` y sus dos atributos: `name` y `value`. Puede incluir más de una etiqueta `param` en un applet, peo debe hacerlo entre las etiquetas `<applet>` y `</applet>`. La siguiente etiqueta `applet` contiene varios parámetros:

```
<applet code="ScrollingHeadline" height="50" width="400">
   <param name="headline1" value="Dewey defeats Truman">
   <param name="headline2" value="Stix nix hix pix">
   <param name="headline3" value="Man bites dog">
</applet>
```

Este marcado podría usarse con un applet que muestre titulares de noticias en una página Web. Como las noticias se actualizan constantemente, la única forma de diseñar un programa como éste es emplear parámetros.

El atributo `name` asigna un nombre al parámetro y `value` le asigna un valor.

Recibir parámetros en el applet

Para acceder a los parámetros de un applet debe invocar su método **getParameter (String)**, heredado de **JApplet,** con su nombre como argumento, como en esta instrucción:

```
String displayl = getParameter("headlinel");
```

El método **getParameter** () devuelve valores como cadenas, por lo que debe convertirlas a otros tipos. Si desea usar un parámetro como entero, puede emplear estas instrucciones:

```
int speed;
String speedParam = getParameter("speed");
if (speedParam != nuil) {
   speed = Integer.parselnt(speedParam);
}
```

Este ejemplo establece la variable entera **speed** con la cadena **speedParam.** Al intentar recuperar un parámetro con **getParameter** () que no se haya incluido en una página Web con la etiqueta **param,** se envía como **nuil,** el valor de una cadena vacía.

Advertencia

El método **Integer.parselnt (String)** requiere el uso de excepciones, una técnica que veremos en el siguiente capítulo.

Procesar parámetros en un applet

El ejercicio de este capítulo muestra el peso de una persona en distintas unidades. El applet acepta dos parámetros: un peso en libras y el nombre de la persona pesada. El peso se usa para determinar el peso de la persona en onzas, kilos y toneladas métricas.

Cree un nuevo archivo vacío de Java con el nombre **WeightScale,** introduzca el texto del listado 17,3 y guárdelo.

Listado 17.3. Texto completo de WeightScale.java.

```
1: import java.awt.*;
2:
3: public class WeightScale extends javax.swing.JApplet {
4:    float lbs =   OF;
5:    float ozs;
6:    float kgs;
7:    float metricTons;
8:    String ñame = "somebody";
9:
```

```
10:    public void init() {
11:       String lbsValue = getParameter("weight");
12:       if (lbsValue != null) {
13:          lbs = Float.valueOf(lbsValue);
14:       }
15:       String personValue = getParameter("person");
16:       if (personValue != null) {
17:          name = personValue;
18:       }
19:       ozs = (float) (lbs * 16);
20:       kgs = (float) (lbs / 2.204623);
21:       metricTons = (float) (lbs / 2204.623);
22:    }
23:
24:    public void paint(Graphics screen) {
25:       Graphics2D screen2D = (Graphics2D) screen;
26:       screen2D.drawString("Studying the weight of " + name, 5, 30);
27:       screen2D.drawString("In pounds: " + lbs, 55, 50);
28:       screen2D.drawString("In ounces: " + ozs, 55, 70);
29:       screen2D.drawString("In kilograms: " + kgs, 55, 90);
30:       screen2D.drawString("In metric tons: " + metricTons, 55, 110);
31:    }
32: }
```

El método `init()` es donde se cargan los dos parámetros del applet. Como provienen de la página Web como cadenas, es necesario convertir el parámetro `weight` a número de coma flotante para poder emplearlo en expresiones matemáticas. La clase `Float` incluye `valueOf(String)`, que devuelve el valor de una cadena como `Float` y que se convierte automáticamente a tipo de variable `float` en la línea 13.

Las líneas 19-21 convierten la variable `lbs` a distintas unidades de medida. Cada instrucción muestra `(float)` por delante de la ecuación de conversión. Se usa para convertir el resultado de la ecuación en un número de coma flotante. El método `paint()` del applet emplea `drawString()` para mostrar una línea de texto. El método `paint()` tiene tres argumentos: el texto que mostrar y sus posiciones `x` e `y`.

Antes de poder probar el applet `WeightScale` debe crear una página Web para incluirlo. En NetBeans genere un nuevo archivo HTML con el nombre `WeightScale`. Introduzca el listado 17.4 y abra la nueva página en un navegador Web: haga clic con el botón derecho del ratón sobre `WeightScale.html` en el panel **Proyecto** y seleccione **Vista**.

Listado 17.4. Texto completo de WeightScale.html.

```
1: <applet code="WeightScale.class" codebase="..\\..\\build\\classes"
2:   height="170" width="210">
3:   <param name="person" value="Konishiki">
4:   <param name="weight" value="605">
5: </applet>
```

En el ejemplo se usa Konishiki, un campeón de sumo norteamericano que pesaba 605 libras, el mayor de todos los luchadores en bikini. Puede sustituirlo por quien desee. En la figura 17.2 puede ver un ejemplo del resultado.

Figura 17.2. El appiet WeightScale en un navegador.

Para que el appiet muestre un nombre diferente junto con un valor diferente para el parámetro **weight,** debe cambiar el archivo **WeightScale.html.** El appiet seguirá funcionando correctamente.

Emplear [a etiqueta Object

La versión más reciente de HTML, HTML5, ha cambiado la etiqueta **<applet>** por una etiqueta **<object>** para cargar los appiet de Java, programas Flash y otros contenidos interactivos. Esta etiqueta tiene atributos **height** y **width** como **<applet>.** También cuenta con un atributo **type** que debe ser **"application/x-java-applet",** el tipo MIME designado para los appiet de Java (los tipos MIME categorizan formatos de archivo para usar en Internet). Veamos el inicio del formato de un objeto:

```
cobject type="application/x-java-applet" height="300" width="400">
</object>
```

code y **codebase** no se designan como atributos, sino que se incluyen los parámetros **code** y **codebase** entre las etiquetas **<object>** y **</objectx**

El siguiente marcado HTML5 muestra un appiet:

```
cobject type="application/x-java-applet" height="300" width="400">
   <param name="coden value="StripYahtzee" />
   <param name="codebase" value="javadir" />
   <p>Sorry, no dice ... this requires a Java-enabled browser.</p>
</object>
```

Resumen

Muchos capítulos del übro se centran en aplicaciones, principalmente porque muchos programadores no diseñan demasiados appiet para la Web. Los appiet están limitados por restricciones de seguridad predeterminadas que permiten su ejecución segura en los

equipos de los usuarios. No pueden guardar archivos en el ordenador, leerlos, acceder a carpetas de archivos ni crear ventanas emergentes que no se identifiquen como applet de Java, entre otras medidas.

Estas restricciones se pueden evitar si el applet se firma con una firma digital y se le pide al usuario que lo apruebe. Una alternativa al desarrollo de programas de Java como applet consiste en emplear Java Web Start, una tecnología para iniciar aplicaciones de Java desde un servidor Web.

Preguntas y respuestas

P: ¿Hay algún motivo por el que el atributo `codebase` deba usarse en una etiqueta `applet`?

R: Si todos los programas de Java se agruparan en su propia subcarpeta con `codebase`, esta estructura mejoraría la organización de un sitio Web, pero no hay una razón de peso. Es una cuestión de preferencias personales.

P: ¿Por qué los applet no tienen un método `main()`?

R: Los applet no emplean `main()` porque su ciclo vital es más complicado que el de las aplicaciones. Una aplicación se inicia, se ejecuta hasta finalizar su labor y finaliza. Un applet se puede iniciar y detener varias veces en un navegador al mostrar la página en la que se incluye.

Si un usuario usa el botón **Atrás** para salir de la página y después regresa, se vuelve a invocar el método `start()` del applet. Si una ventana emergente que oculta el applet se cierra, se invoca el método `paint()` del applet.

La clase `JApplet` se ha diseñado para que estas complejas interacciones funcionen en un navegador.

Ejercicios

Las siguientes preguntas ponen a prueba sus conocimientos sobre applet.

Preguntas

1. ¿Qué tipo de argumento se emplea con el método `paint()`?

   A. Un objeto `Graphics`.

   B. Un objeto `Graphics2D`.

   C. Ninguno.

2. ¿Qué método se procesa justo antes de que termine de ejecutarse un applet?

   A. `decline()`

   B. `destroy()`

   C. `defenestrate()`

3. ¿Por qué no se pueden crear todas las variables necesarias para un applet dentro del método `init()`?

   A. El ámbito de las variables se limitaría a ese método.

   B. La legislación federal lo prohíbe,

   C. Se pueden crear sin ningún problema.

Respuestas

1. A. El objeto `Graphics` controla el comportamiento y los atributos necesarios para mostrar elementos en pantalla en la ventana del applet. Podría crear un objeto `Graphics2D` dentro del método pero no se envía como argumento.
2. B. Se puede usar el método `destroy()` para liberar recursos empleados por el applet.
3. A. Las variables usadas en más de un método de una clase deben crearse después de la instrucción de la clase pero antes de los métodos.

Actividades

Aplique sus conocimientos sobre programación de applet con las siguientes actividades:

- Cree un applet en el que el texto mostrado se mueva al actualizar la ventana del applet.
- Instale Java Plug-in en su navegador preferido y pruebe los applet de `www.javaonthebrain.com`.

18
Procesar errores en un programa

Errores, problemas, equivocaciones y erratas impiden el funcionamiento correcto de un programa y son una parte natural del proceso de desarrollo de software. Natural es probablemente la palabra más amable utilizada para describirlos. En mis programas, cuando no puedo detectar la causa de un error, utilizo palabras que harían sonrojar a cualquier rapero. Algunos errores son detectados por el compilador e impiden que se pueda crear una clase. Otros los captura el intérprete como respuesta a un problema. Java divide los errores en dos categorías:

- **Excepciones:** Eventos que indican una circunstancia inusual en la ejecución de un programa.
- **Errores:** Fallos que indican que el intérprete tiene problemas que pueden no estar relacionados con el programa.

Los errores son algo de lo que un programa no suele recuperarse, por lo que no los describiremos en este capítulo. Seguramente haya encontrado un error `OutOfMemoryError` en un programa; no se puede hacer nada al respecto. El programa termina con el error.

Las excepciones se pueden controlar de forma que el programa siga ejecutándose con normalidad.

Excepciones

Aunque las acabemos de presentar, seguramente ya las conozca de los últimos 17 capítulos. Estos fallos surgen al crear un programa de Java que se compila correctamente pero que detecta un problema al ejecutarse.

Por ejemplo, un error de programación habitual es hacer referencia a un elemento de una matriz que no existe:

```
String[] greek = { "Alpha", "Beta", "Gamma" };
System.out.println(greek[3]);
```

La matriz `String` `greek` tiene tres elementos. Como el primero se numera como `0` y no como `1`, el primer elemento es `greek[0]`, el segundo es `greek[1]` y el tercero es `greek[2]`. Por ello, la excepción que intenta mostrar `greek[3]` es errónea. Las instrucciones anteriores se compilan correctamente, pero al ejecutar el programa, el intérprete de Java se detiene con el siguiente mensaje:

```
Exception in thread "main" java.lang.ArrayIndexOutBoundsException: 3
   at SampleProgram.main(SampleProgram.java:4)
```

Este mensaje indica que la aplicación ha generado una excepción, que el intérprete indica mostrando un mensaje de error y deteniendo el programa.

El mensaje de error se refiere a la clase `ArrayIndexOutOfBoundsException` del paquete `java.lang`. Esta clase es una excepción, un objeto que representa una circunstancia excepcional que se produce en un programa de Java.

Cuando una clase de Java detecta una excepción, alerta del error a los usuarios de la clase. En este ejemplo, el usuario de la clase es el intérprete de Java.

Nota

En este proceso, los objetos generan excepciones para indicar de su presencia a otros. Las excepciones son capturadas por otros objetos o por el intérprete de Java.

Todas las excepciones son subclases de `Exception`, del paquete `java.lang`. `ArrayIndexOutOfBoundsException` hace lo que imagina: informa del acceso a un elemento de matriz fuera de los límites de la matriz.

Existen cientos de excepciones en Java. Muchas, como la de excepción, indican un problema que se puede corregir con un cambio de programación. Se pueden comparar a errores de compilación; una vez corregido el problema, ya no tiene que preocuparse de la excepción.

Otras excepciones deben corregirse siempre que ejecute un programa por medio de cinco nuevas palabras clave: `try`, `catch`, `finally`, `throw` y `throws`.

Capturar excepciones en un bloque try-catch

Hasta el momento, hemos solucionado las excepciones corrigiendo el problema que las genera. En ocasiones no podrá hacerlo de esta forma y tendrá que recurrir a una clase de Java.

Como introducción a su relevancia, añada la aplicación del listado 18.1 a un nuevo archivo vacío de Java con el nombre `Calculator` y guárdelo.

Listado 18.1. Texto completo de Calculator.java.

```
1: public class Calculator {
2:   public static void main(String[] arguments) {
3:      float sum = 0;
4:      for (int i = 0; i < arguments.length; i++) {
5:         sum = sum + Float.parseFloat(arguments[i]);
6:      }
7:      System.out.println("Those numbers add up to " + sum);
8:   }
9: }
```

La aplicación `Calculator` acepta uno o varios números como argumentos de línea de comandos, los suma y muestra el total.

Como los argumentos de línea de comandos se representan mediante cadenas, el programa debe convertirlos a números de coma flotante antes de sumarlos. El método `Float.parseFloat()` de la línea 5 se encarga de esto y suma el número convertido a la variable `sum`.

Antes de ejecutar la aplicación con los argumentos de línea de comandos, debe establecerlos en NetBeans por medio de Ejecutar>Establecer la configuración del proyecto>Personalizar: `8 6 7 5 3 0 9`. Seleccione Ejecutar>Ejecutar Proyecto Principal para ejecutar la aplicación y ver el resultado de la figura 18.1.

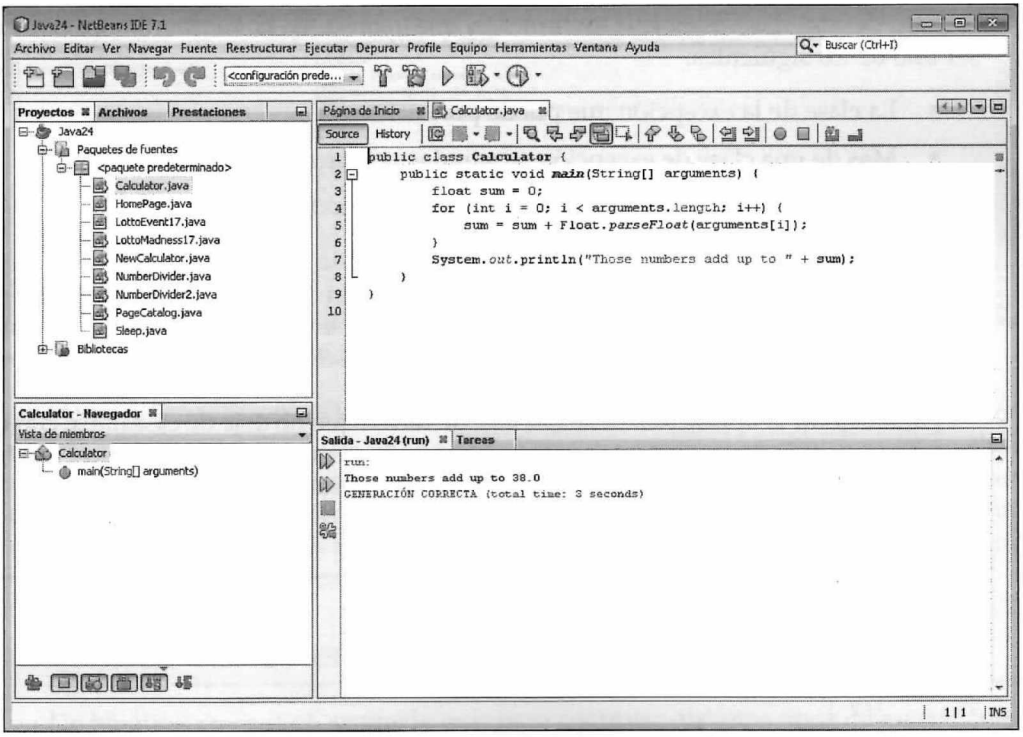


Figura 18.1. Resultado de la aplicación Calculator.

Ejecute varias veces el programa con distintos números como argumentos. Se ejecutará correctamente, lo que le hará preguntarse qué tiene que ver con las excepciones.

Para ver su importancia, cambie los argumentos de línea de comandos por `1 3 5x`. El tercer argumento contiene una errata: no debería incluir la `x`. La aplicación `Calculator` no sabe que se trata de un error, por lo que intenta sumar `5x` a los demás números, lo que provoca la siguiente excepción:

```
Exception in thread "main" java.lang.NumberFormatException: For input
string: "5x" at sun.misc.FloatingDecimal.readJavaFormatString
(FloatingDecimal.java:1224)
   at java.lang.Float.parseFloat(Float.java:422)
   at Calculator.main(Calculator.java:5)
```

Este mensaje es válido para el programador pero no demasiado para el usuario. Los programas de Java pueden encargarse de sus propias excepciones mediante un bloque `try-catch`, que adopta el siguiente formato:

```
try {
   // instrucciones que pueden generar la excepción
} catch (Exception e) {
   // qué hacer si se produce la excepción
}
```

Debe usarse un bloque `try-catch` en todas las excepciones que deba procesar un método de una clase. El objeto `Exception` que aparece en la instrucción `catch` debe ser uno de los siguientes:

- La clase de la excepción que puede producirse.
- Más de una clase de excepción, separadas por caracteres `|`.
- Una superclase de distintas excepciones que pueden producirse.

La sección `try` del bloque `try-catch` contiene la instrucción (o instrucciones) que pueden generar una excepción. En la aplicación `Calculator`, la invocación del método `Float.parseFloat(String)` de la línea 5 del listado 18.1 genera `NumberFormatException` siempre que se emplea con un carácter de cadena que no se puede convertir a valor de coma flotante.

Para mejorar la aplicación `Calculator` de forma que no deje de ejecutarse con este tipo de error, puede usar un bloque `try-catch`. Cree un nuevo archivo vacío de Java con el nombre `NewCalculator` e introduzca el texto del listado 18.2.

Listado 18.2. Texto completo de NewCalculator.java.

```
1: public class NewCalculator {
2: public static void main(String[] arguments) {
3:    float sum = 0;
4:    for (int i = 0; i < arguments.length; i++) {
5:       try {
6:          sum = sum + Float.parseFloat(arguments[i]);
7:       } catch (NumberFormatException e) {
```

```
 8:          System.out.println(arguments[i] + " is not a number.");
 9:          }
10:       }
11:       System.out.println("Those numbers add up to " + sum);
12:    }
13: }
```

Tras guardar la aplicación, ejecútela con el argumento de línea de comandos `1 3 5x` para ver el resultado mostrado en la figura 18.2.

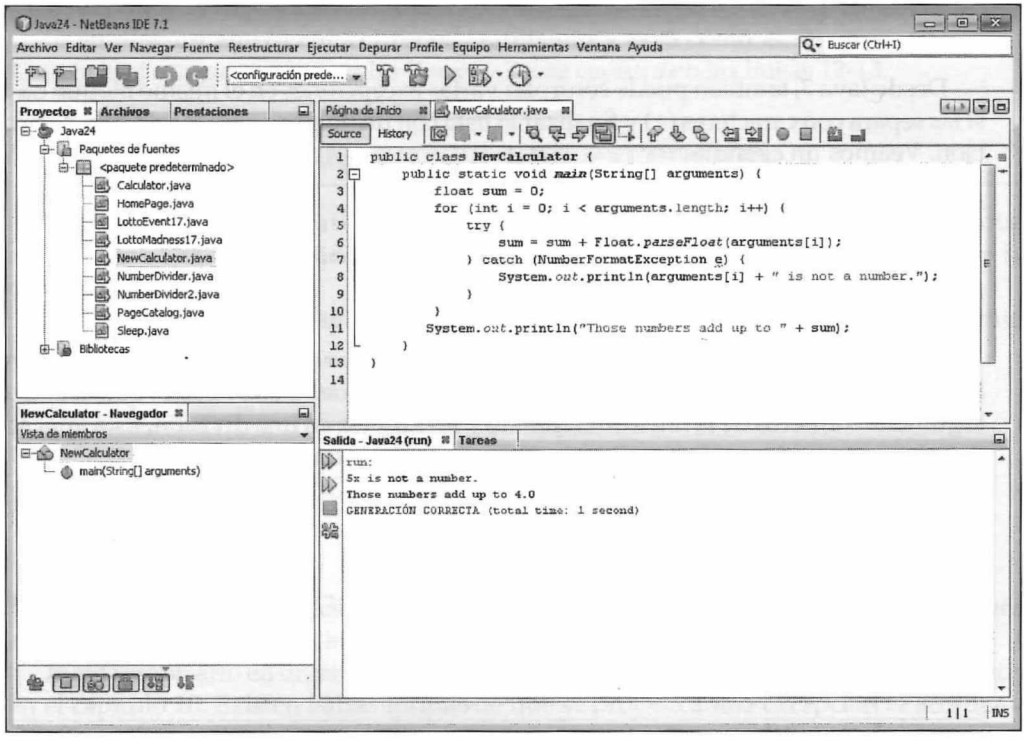


Figura 18.2. Resultado de la aplicación NewCalculator.

El bloque `try-catch` de las líneas 5-9 controla los errores `NumberFormatException` generados por `Float.parseFloat()`. Estas excepciones se capturan en la clase `NewCalculator`, que muestra un mensaje de error siempre que el argumento no es un número. Como la excepción se controla en la clase, el intérprete de Java no muestra un error. Por lo general, puede solucionar problemas relacionados con entradas de usuario y otros datos inesperados con ayuda de bloques `try-catch`.

Capturar varias excepciones diferentes

Un bloque `try-catch` se puede usar para controlar distintos tipos de excepciones, aunque se generen en instrucciones diferentes.

Una forma de controlar varias clases de excepciones es dedicar un bloque `catch` a cada una, como en este código:

```
String textValue = "35";
int value;
try {
   value = Integer.parseInt(textValue);
catch (NumberFormatException exc) {
   // código para procesar la excepción
} catch (Arithmetic Exception exc) {
   // código para procesar la excepción
}
```

Desde Java 7, también puede controlar varias excepciones en el mismo bloque `catch` si las separa con caracteres (|) y finalizar la lista con un nombre para la variable de excepción. Veamos un ejemplo:

```
try {
   value = Integer.parseInt(textValue);
catch (NumberFormatException | Arithmetic Exception exc) {
   // código para procesar excepciones
}
```

Si se captura `NumberFormatException` o `ArithmeticException`, se asigna a la variable `exc`. El listado 18.3 contiene la aplicación `NumberDivider`, que acepta dos argumentos enteros de la línea de comando y los emplea en una división. Esta aplicación debe poder solucionar dos posibles problemas en la entrada del usuario:

- Argumentos no numéricos.
- División por cero.

Cree un nuevo archivo de Java con el nombre `NumberDivider` e introduzca el texto del listado 18.3 en el editor de código.

Listado 18.3. Texto completo de NumberDivider.java.

```
 1: public class NumberDivider {
 2:   public static void main(String[] arguments) {
 3:      if (arguments.length == 2) {
 4:         int result = 0;
 5:         try {
 6:            result = Integer.parseInt(arguments[0]) /
 7:               Integer.parseInt(arguments[1]);
 8:            System.out.println(arguments[0] + " divided by " +
 9:               arguments[1] + " equals " + result);
10:         } catch (NumberFormatException e) {
11:            System.out.println("Both arguments must be numbers.");
12:         } catch (ArithmeticException e) {
13:            System.out.println("You cannot divide by zero.");
14:         }
15:      }
16:   }
17: }
```

Mediante argumentos de línea de comandos para especificar dos argumentos puede ejecutar la aplicación con enteros, números de coma flotante y argumentos no numéricos.

La instrucción `if` de la línea 3 comprueba que se envíen dos argumentos a la aplicación. En caso contrario, el programa sale sin mostrar nada.

La aplicación `NumberDivider` realiza divisiones enteras, de modo que el resultado es un entero. En una división entera, 5 dividido por 2 equivale a 2, no a 2.5. Si usa un argumento de coma flotante o no numérico, se genera `NumberFormatException` en las líneas 6-7 y se captura en las líneas 10-11.

Si emplea un entero como primer argumento y cero como el segundo, se genera `ArithmeticExpression` en las líneas 6-7 y se captura en las líneas 12-13.

Controlar algo que no sea una excepción

Cuando se enfrenta a varias excepciones con `try` y `catch`, en ocasiones querrá que el programa haga algo al final del bloque, independientemente de que se produzca la excepción o no.

Para ello, puede usar un bloque `try-catch-finally`, que tiene el siguiente formato:

```
try {
   // instrucciones que pueden generar la excepción
} catch (Exception e) {
   // qué hacer si se produce la excepción
} finally {
   // instrucciones que ejecutar siempre
}
```

La instrucción o instrucciones de la sección `finally` del bloque se ejecutan después de todos los elementos del bloque, aunque se produzca una excepción.

Puede emplearlo en un programa que lea datos de un archivo en disco, como veremos en el capítulo 20. Existen varias formas de que se produzca una excepción al acceder a datos: puede que el archivo no exista, que se produzca un error de disco, etc. Si las instrucciones para leer el disco se encuentran en una sección `try` y los errores se procesan en una sección `catch`, puede cerrar el archivo en la sección `finally`. De este modo se asegura que el archivo se cierra independientemente de que se produzca una excepción.

Generar excepciones

Al invocar un método de otra clase, dicha clase puede controlar el uso del método mediante la generación de excepciones. Al usar las clases de la biblioteca de clases de Java, el compilador suele mostrar el siguiente mensaje:

```
NetReader.java:14: unreported exception java.net.MalformedURLException;
must be caught or declared to be thrown
```

Siempre que vea un error que indica que una excepción debe capturarse o declararse para ser generada, indica que el método que intenta utilizar genera una excepción.

Cualquier clase que invoque estos métodos, como una aplicación que cree, debe realizar lo siguiente:

- Procesar la excepción con un bloque `try-catch`.
- Generar la excepción.
- Procesar la excepción con un bloque `try-catch` y después volver a generarla.

Hasta el momento, hemos visto cómo controlar excepciones. Si desea generar una excepción tras controlarla, puede emplear una instrucción `throw` seguida del objeto de excepción que va a generar.

Las siguientes instrucciones procesan un error `NumberFormatException` en un bloque `catch` y después generan la excepción:

```
try {
   principal = Float.parseFloat(loanText) * 1.1F;
} catch (NumberFormatException e) {
   System.out.println(arguments[i] + " is not a number.");
   throw e;
}
```

El siguiente código procesa todas las excepciones que pueden generarse en el bloque `try` y las genera:

```
try {
   principal = Float.parseFloat(loanText) * 1.1F;
} catch (Exception e) {
   System.out.println("Error " + e.getMessage());
   throw e;
}
```

`Exception` es la principal de todas las subclases de excepción. Una instrucción `catch` captura la clase y las subclases posteriores en la jerarquía de clases.

Al generar una excepción con `throw`, por lo general significa que no ha hecho nada de lo que debería para controlar la excepción.

Veamos un ejemplo práctico. Imagine un hipotético programa llamado `CreditCardChecker`, una aplicación que verifica compras con tarjeta de crédito. Esta aplicación usa la clase `CheckDatabase`, que realiza las siguientes tareas:

1. Establece una conexión con el ordenador del titular de la tarjeta.
2. Pregunta al ordenador si el número de la tarjeta es válido.
3. Pregunta al ordenador si el cliente tiene crédito suficiente para realizar la compra.

Mientras la clase `CheckDatabase` realiza su trabajo, ¿qué sucede si el ordenador del titular de la tarjeta no responde? Es el tipo de error para el que se ha diseñado el bloque `try-catch` y se emplea en `CheckDatabase` para controlar errores de conexión.

Si la clase `CheckDatabase` procesa este error por su cuenta, la aplicación `CreditCardChecker` desconoce que la excepción se ha producido. No es una buena idea, ya que la aplicación debe saber cuándo no se puede establecer una conexión para poder informar de este hecho al usuario de la aplicación.

Una forma de notificar algo a la aplicación `CreditCardChecker` es que `CheckDatabase` capture la excepción en un bloque `catch` y después la vuelva a generar con una instrucción `throw`. La excepción se genera en `CheckDatabase`, que después debe procesarla como cualquier otra excepción.

El control de excepciones es una forma de comunicación de las clases entre ellas en caso de error o de otras circunstancias inusuales.

Al usar `throw` en un bloque `catch` que captura una clase principal, como `Exception`, al generar la excepción se genera dicha clase. Se pierde parte del detalle del tipo de error producido, ya que una subclase como `NumberFormatException` ofrece más información sobre el problema que la clase `Exception`.

Java 7 le ofrece una nueva forma de conservar estos detalles: la palabra clave `final` en una instrucción `catch`:

```
try {
   principal = Float.parseFloat(loanText) * 1.1F;
} catch (final Exception e) {
   System.out.println("Error " + e.getMessage());
   throw e;
}
```

La palabra clave `final` en `catch` hace que `throw` se comporte de forma diferente. Se genera la clase concreta que se ha capturado.

Ignorar excepciones

La última técnica que veremos en este capítulo consiste en ignorar una excepción. Un método de una clase puede ignorar excepciones si emplea una cláusula `throws` como parte de la definición del método.

El siguiente método genera `MalformedURLException`, un error que se produce al trabajar con direcciones Web en un programa de Java:

```
public loadURL(String address) throws MalformedURLException {
   URL page = new URL(address);
   loadWebPage(page);
}
```

La segunda instrucción crea un objeto `URL`, que representa una dirección en la Web. El constructor de la clase `URL` genera `MalformedURLException` para indicar que se usa una dirección Web no válida, de modo que no se construyen objetos. La siguiente instrucción hace que se genere una de estas excepciones:

```
URL source = new URL("http:www.java24hours.com");
```

La cadena `http:www.java24hours.com` no es una URL válida. Faltan los caracteres `//`.

Como el método `loadURL()` se ha declarado para generar errores `MalformedURLException`, no tiene que procesarlos en su interior. La responsabilidad de capturar la excepción recae en los métodos de invoquen el método `loadURL()`.

Generar y capturar excepciones

En el siguiente proyecto crearemos una clase que emplea excepciones para indicar a otra clase que se ha producido un error. Las clases de este proyecto son `HomePage`, que representa una página personal en la Web, y `PageCatalog`, una aplicación que cataloga dichas páginas.

Introduzca el texto del listado 18.4 en un nuevo archivo vacío de Java con el nombre `HomePage`.

Listado 18.4. Texto completo de HomePage.java.

```
 1: import java.net.*;
 2:
 3: public class HomePage {
 4:    String owner;
 5:    URL address;
 6:    String category = "none";
 7:
 8:    public HomePage(String inOwner, String inAddress)
 9:       throws MalformedURLException {
10:
11:       owner = inOwner;
12:       address = new URL(inAddress);
13:    }
14:
15:    public HomePage(String inOwner, String inAddress, String inCategory)
16:       throws MalformedURLException {
17:
18:       this(inOwner, inAddress);
19:       category = inCategory;
20:    }
21: }
```

Puede usar la clase `HomePage` en otros programas. Representa páginas Web personales en la red. Tiene tres variables de instancia: `address`, un objeto `URL` que representa la dirección de la página; `owner`, el propietario de la página y `category`, un breve comentario que describe la página.

Como otras clases que crean objetos `URL`, `HomePage` debe procesar errores `MalformedURLException` en un bloque `try-catch` o declarar que ignora estos errores. La clase opta por la segunda opción, como se muestra en las líneas 8-9 y 15-16. Al emplear `throws` en los dos constructores, `HomePage` elimina la necesidad de controlar

errores `MalformedURLException`. Para crear una aplicación que use la clase `HomePage`, vuelva a NetBeans y cree un archivo vacío de Java con el nombre `PageCatalog` que contenga el texto del listado 18.5.

Listado 18.5. Texto completo de PageCatalog.java.

```
 1: import java.net.*;
 2:
 3: public class PageCatalog {
 4:   public static void main(String[] arguments) {
 5:     HomePage[] catalog = new HomePage[5];
 6:     try {
 7:       catalog[0] = new HomePage("Mark Evanier",
 8:         "http://www.newsfromme.com", "comic books");
 9:       catalog[1] = new HomePage("Todd Smith",
10:         "http://www.sharkbitten.com", "music");
11:       catalog[2] = new HomePage("Rogers Cadenhead",
12:         "http://workbench.cadenhead.org", "programming");
13:       catalog[3] = new HomePage("Juan Cole",
14:         "http://www.juancole.com", "politics");
15:       catalog[4] = new HomePage("Rafe Colburn",
16:         "www.rc3.org");
17:       for (int i = 0; i < catalog.length; i++) {
18:         System.out.println(catalog[i].owner + ": " +
19:           catalog[i].address + " - " +
20:           catalog[i].category);
21:       }
22:     } catch (MalformedURLException e) {
23:       System.out.println("Error: " + e.getMessage());
24:     }
25:   }
26: }
```

Al ejecutar la aplicación compilada, se muestra el siguiente resultado:

```
Error: no protocol: www.rc3.org
```

La aplicación `PageCatalog` crea una matriz de objetos `HomePage` y después muestra su contenido. Los objetos `HomePage` se crean con hasta tres argumentos:

- El nombre del propietario de la página.
- La dirección de la página (como `String`, no `URL`).
- La categoría de la página.

El tercer argumento es opcional y no se usa en las líneas 15-16. Los constructores de la clase `HomePage` generan errores `MalformedURLException` cuando reciben una cadena que no pueden convertir en un objeto `URL` válido. Estas excepciones se controlan en la aplicación `PageCatalog` con un bloque `try-catch`.

Para corregir los problemas que causa el error "sin protocolo", cambie la línea 16 para que la cadena comience por `http://`, como las demás direcciones Web de las líneas 7-14. Al ejecutar el programa, verá el resultado mostrado en la figura 18.3.

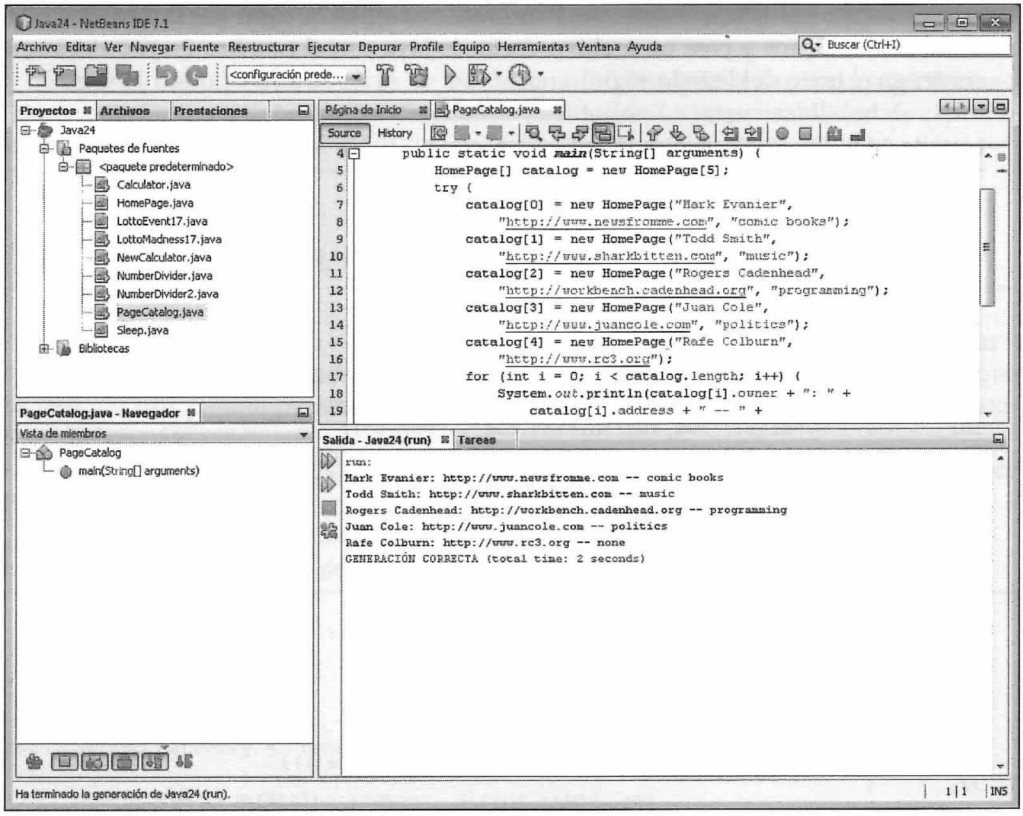


Figura 18.3. Resultado de la aplicación PageCatalog.

Resumen

Una vez vistas las técnicas de Java para controlar excepciones, el tema de los errores será menos árido que al comienzo del capítulo. Estas técnicas le ofrecen numerosas posibilidades:

- Capturar una excepción y procesarla.
- Ignorar una excepción y dejar que otra clase o el intérprete de Java se encargue de ella.
- Capturar varias excepciones en el mismo bloque `try-catch`.
- Generar su propia excepción.

La gestión de excepciones en sus programas de Java hace que sean más fiables, versátiles y fáciles de usar, ya que no muestran crípticos mensajes de error a los usuarios del software.

Preguntas y respuestas

P: ¿Puedo crear mis propias excepciones?

R: Puede crear sus propias excepciones fácilmente si las convierte en una subclase de una excepción existente, como `Exception`, la superclase de todas las excepciones. En una subclase de `Exception`, solo hay dos métodos que reemplazar: `Exception()` sin argumentos y `Exception()` con un cadena como argumento. En éste, la cadena debe ser un mensaje que describa el error.

P: ¿Por qué no se ha descrito la forma de generar y capturar errores en el capítulo?

R: Java divide los problemas en errores y excepciones, por ser de gravedad diferente. Las excepciones son menos graves, de modo que deben controlarse en sus programas mediante `try-catch` o `throws` en la declaración del método. Los errores, por su parte, son más graves y no se pueden controlar en un programa. Dos ejemplos de errores son los desbordamientos de pila y la falta de memoria. Pueden provocar el fallo del intérprete de Java y no se pueden corregir en el programa mientras el intérprete lo ejecuta.

Ejercicios

Aunque este capítulo está literalmente plagado de errores, responda a las siguientes preguntas sin cometer fallos.

Preguntas

1. ¿Cuántas excepciones puede controlar una única instrucción `catch`?
   A. Solo una.
   B. Distintas excepciones.
   C. Esta respuesta debe dejarse en blanco.
2. ¿Cuándo se ejecutan las instrucciones de una sección `finally`?
   A. Cuando un bloque `try-catch` termina con una excepción.
   B. Cuando un bloque `try-catch` termina sin una excepción.
   C. En ambos casos.

Respuestas

1. B. Un objeto `Exception` en una instrucción `catch` puede controlar todas las excepciones de su propia clase y sus superclases.

2. C. La instrucción (o instrucciones) de una sección `finally` siempre se ejecuta después del bloque `try-catch`, independientemente de que se produzca una excepción o no.

Actividades

Para comprobar si es un programador excepcional de Java, intente no cometer errores en las siguientes actividades:

- Modifique la aplicación `NumberDivider` para que genere las excepciones que capture y ejecute el programa para ver los resultados.
- La clase `LottoEvent` creada en el capítulo 15 incluye un bloque `try-catch`. Úselo como guía para crear una clase `Sleep`, que procesa `InterruptedException` para que otras clases como `LottoEvent` no tengan que hacerlo.

19
Crear un programa con subprocesos

Un término informático que suele emplearse para describir el ritmo héctico de la vida diaria es la multitarea, que significa hacer más de una cosa al mismo tiempo, como navegar por la red mientras se habla por teléfono. Un ordenador multitarea es que el puede ejecutar más de un programa a la vez.

Una sofisticada característica del lenguaje Java es su capacidad para crear programas multitarea, gracias a una clase de objetos denominados subprocesos.

Subprocesos

En un programa de Java, cada una de las tareas simultáneas que controla un ordenador se denomina subproceso y el proceso general, subprocesamiento. El subprocesamiento es muy útil en las animaciones y en otros programas.

Los subprocesos permiten organizar un programa para que realice más de una tarea a la vez. Cada tarea se incluye en un subproceso propio lo que se consigue implementando cada tarea como clase independiente.

Los subprocesos se representan con la clase `Thread` y la interfaz `Runnable`, que forman parte del paquete de clases `java.lang`. Al pertenecer a este paquete, no es necesario usar una instrucción `import` para emplearlas en sus programas.

Una de las aplicaciones más básicas de la clase `Thread` consiste en ralentizar la velocidad con la que un programa realiza una acción.

Ralentizar un programa

La clase `Thread` cuenta con un método `sleep()` que puede invocar en cualquier programa que deba dejar de ejecutarse durante un tiempo. Esta técnica se suele usar en programas con animaciones para evitar que las imágenes se muestren más rápidamente de lo que el intérprete de Java puede procesar.

Para emplear el método `sleep()`, invoque `Thread.sleep()` con el número de milisegundos que debe detenerse:

```
Thread.sleep(5000);
```

Esta instrucción hace que el intérprete de Java se detenga durante cinco segundos antes de continuar. Si por algún motivo no puede detenerse tanto tiempo, el método `sleep()` genera `InterruptedException`. Como se puede generar esta excepción, debe procesarla de alguna forma en el método `sleep()`. Una solución consiste en incluir la instrucción `Thread.sleep()` en un bloque `try-catch`:

```
try {
   Thread.sleep(5000);
} catch (InterruptedException e) {
   // activar antes
}
```

Cuando un programa de Java tenga que procesar más de una acción a la vez, debe organizarlo en subprocesos. Su programa puede tener todos los subprocesos que necesite y se pueden ejecutar de forma simultánea sin que unos afecten a otros.

Crear un subproceso

Una clase de Java que se pueda ejecutar como subproceso se denomina ejecutable. Aunque puede usar subprocesos para detener la ejecución de un programa durante unos segundos, los programadores suelen emplearlos para lo contrario: para acelerar un programa. Si incluye tareas de elevado consumo de tiempo en sus propios subprocesos, el resto del programa se ejecutará más rápidamente. Suele usarse para evitar que una tarea ralentice la capacidad de respuesta de la interfaz gráfica de usuario de un programa.

Por ejemplo, si tiene una aplicación que carga datos del mercado bursátil desde el disco y genera estadísticas, la tarea que más tiempo consume es la carga de datos desde el disco. Si la aplicación no emplea subprocesos, su interfaz puede responder lentamente al cargarse los datos, lo que puede resultar frustrante para el usuario.

Hay dos formas de incluir una tarea en su propio subproceso:

- Incluirla en una clase que implemente la interfaz `Runnable`.
- Incluirla en una subclase de `Thread`.

Para admitir la interfaz `Runnable`, se usa la palabra clave `implements` al crear la clase:

```
public class LoadStocks implements Runnable {
   // cuerpo de la clase
}
```

Cuando una clase implementa una interfaz, indica que la clase contiene un comportamiento adicional. Las clases que implementan la interfaz `Runnable` deben incluir el método `run()`, que tiene la siguiente estructura:

```
public void run() {
   // cuerpo del método
}
```

El método `run()` debe encargarse de la tarea para la que se ha creado el subproceso. En el ejemplo de análisis bursátil, `run()` podría incluir instrucciones para cargar datos desde el disco y generar estadísticas basadas en dichos datos.

Al ejecutar una aplicación con subprocesos, las instrucciones de su método `run()` no se ejecutan inmediatamente. En Java, los subprocesos se pueden iniciar y detener, y un subproceso no comienza a ejecutarse hasta que realice dos acciones:

- Crear un objeto de la clase de subproceso invocando el constructor `Thread`.
- Iniciar el subproceso invocando su método `start()`.

El constructor `Thread` acepta un solo argumento, el objeto que contiene el método `run()`. Por lo general, puede usar la palabra clave `this` como argumento para indicar que la clase actual contiene el método `run()`.

El listado 19.1 contiene una aplicación de Java que muestra una secuencia de números primos en un area de texto. Cree un nuevo archivo vacío de Java con el nombre `PrimeFinder`, introduzca el código y guárdelo.

Listado 19.1. Texto completo de PrimeFinder.java.

```
 1: import java.awt.*;
 2: import javax.swing.*;
 3: import java.awt.event.*;
 4:
 5: class PrimeFinder extends JFrame implements Runnable, ActionListener {
 6:    Thread go;
 7:    JLabel howManyLabel;
 8:    JTextField howMany;
 9:    JButton display;
10:    JTextArea primes;
11:
12:    PrimeFinder() {
13:       super("Find Prime Numbers");
14:       setLookAndFeel();
15:       setSize(400, 300);
16:       setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
17:       BorderLayout bord = new BorderLayout();
18:       setLayout(bord);
19:
20:       howManyLabel = new JLabel("Quantity: ");
```

```
21:      howMany = new JTextField("400", 10);
22:      display = new JButton("Display primes");
23:      primes = new JTextArea(8, 40);
24:
25:      display.addActionListener(this);
26:      JPanel topPanel = new JPanel();
27:      topPanel.add(howManyLabel);
28:      topPanel.add(howMany);
29:      topPanel.add(display);
30:      add(topPanel, BorderLayout.NORTH);
31:
32:      primes.setLineWrap(true);
33:      JScrollPane textPane = new JScrollPane(primes);
34:      add(textPane, BorderLayout.CENTER);
35:
36:      setVisible(true);
37:   }
38:
39:   public void actionPerformed(ActionEvent event) {
40:      display.setEnabled(false);
41:      if (go == null) {
42:         go = new Thread(this);
43:         go.start();
44:      }
45:   }
46:
47:   public void run() {
48:      int quantity = Integer.parseInt(howMany.getText());
49:      int numPrimes = 0;
50:      // candidate: el número que puede ser primo
51:      int candidate = 2;
52:      primes.append("First " + quantity + " primes:");
53:      while (numPrimes < quantity) {
54:         if (isPrime(candidate)) {
55:            primes.append(candidate + " ");
56:            numPrimes++;
57:         }
58:         candidate++;
59:      }
60:   }
61:
62:   public static boolean isPrime(int checkNumber) {
63:      double root = Math.sqrt(checkNumber);
64:      for (int i = 2; i <= root; i++) {
65:         if (checkNumber % i == 0) {
66:            return false;
67:         }
68:      }
69:      return true;
70:   }
71:
72:   private void setLookAndFeel() {
73:      try {
74:         UIManager.setLookAndFeel(
75:            "com.sun.java.swing.plaf.nimbus.NimbusLookAndFeel"
```

```
76:          );
77:       } catch (Exception exc) {
78:          // ignorar el error
79:       }
80:    }
81:    public static void main(String[] arguments) {
82:       PrimeFinder fp = new PrimeFinder();
83:    }
84: }
```

La aplicación `PrimeFinder` muestra un campo de texto, un botón **Display Primes** y un área de texto (véase la figura 19.1).

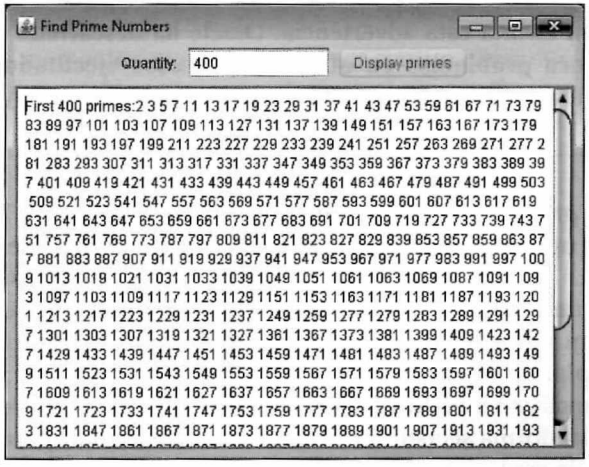


Figura 19.1. Ejecución de la aplicación PrimeFinder.

Muchas instrucciones de la aplicación se emplean para crear la IGU o para mostrar una secuencia de números primos. Las siguientes se usan para implementar subprocesos en el programa:

- **Línea 5:** La interfaz `Runnable` se aplica a la clase `PrimeFinder`.
- **Línea 6:** Se crea una variable de objeto `Thread` con el nombre `go` pero sin un valor.
- **Líneas 41-44:** Si la variable de objeto `go` tiene el valor `null`, lo que indica que todavía no se ha cerrado el subproceso, se crea un nuevo objeto `Thread` y se almacena en la variable. El subproceso se inicia con la invocación de su método `start()`, que provoca la invocación del método `run()` de la clase `PrimeFinder`.
- **Líneas 47-60:** El método `run()` busca una secuencia de números primos empezando por 2 y muestra los números en el componente de área de texto `primes` invocando su método `append()`. La cantidad de números primos de la secuencia se determina por medio del valor del campo de texto `howMany`.

Trabajar con subprocesos

Puede iniciar un subproceso si invoca su método `start()`, lo que seguramente le haga pensar que también existe un método `stop()` para detenerlo.

Aunque Java incluye un método `stop()` en la clase `Thread`, está obsoleto. En Java, un elemento obsoleto es una clase, interfaz, método o variable que se ha sustituido por otro que funciona mejor.

Nota

Debe tener en cuenta esta advertencia. Oracle ha descartado el método `stop()` ya que provoca problemas en otros subprocesos ejecutados en el intérprete de Java. También se han descartado los métodos `resume()` y `suspend()` de la clase.

El siguiente proyecto muestra cómo detener un subproceso. El programa que crearemos itera por una lista de títulos de sitios Web y las direcciones empleadas para visitarlos.

El título de cada página y la dirección Web se muestran en un ciclo continuo. Los usuarios pueden visitar el sitio mostrado actualmente si hacen clic en un botón de la ventana del applet.

Este programa funciona durante un periodo de tiempo y muestra información sobre cada sitio Web de forma secuencial. Debido a este factor temporal, los subprocesos son la mejor forma de controlar el programa.

En lugar de introducir el programa primero en el editor de NetBeans y analizarlo después, añadiremos el texto completo del applet `LinkRotator` al final del capítulo y antes describiremos sus secciones.

La declaración class

Lo primero que necesita hacer en el applet es usar `import` para las clases de los paquetes `java.awt`, `java.net`, `java.applet`, `java.awt.event` y `javax.swing`. Tras ello, ya puede iniciar el applet con la siguiente instrucción:

```
public class LinkRotator extends JApplet
   implements Runnable, ActionListener {
```

Esta instrucción crea la clase `LinkRotator` como subclase de `JApplet`. También indica que la clase admite dos interfaces: `Runnable` y `ActionListener`. Al implementar la clase `Runnable`, puede emplear un método `run()` en este applet para iniciar la ejecución de un subproceso. La interfaz `ActionListener` permite al applet responder a eventos del ratón.

Configurar variables

Lo primero que debe hacer en `LinkRotator` es crear las variables y objetos de la clase. Cree una matriz de seis elementos de objetos `String` con el nombre `pageTitle` y otra de seis elementos de objetos `URL` con el nombre `pageLink`:

```
String[] pageTitle = new String[6];
URL[] pageLink = new URL[6];
```

La matriz `pageTitle` contiene los títulos de los seis sitios Web mostrados. La clase de objetos `URL` contiene el valor de la dirección de un sitio Web. `URL` tiene el comportamiento y los atributos necesarios para controlar una dirección Web y usarla para cargar la página en un navegador.

Los tres últimos elementos que se van a crear son un objeto `Color` con el nombre `butterscotch`, una variable entera con el nombre `current` y un objeto `Thread` con el nombre `runner`:

```
Color butterscotch = new Color(255, 204, 158);
int current = 0;
Thread runner;
```

Los objetos `Color` representan colores que puede emplear con la fuente, componentes de la interfaz de usuario y otros aspectos visuales de Swing. Encontrará más información al respecto en el capítulo 23.

La variable `current` controla qué sitio se muestra para que pueda iterar por los distintos sitios. El objeto `Thread runner` representa el subproceso en el que se ejecuta el programa. Puede invocar métodos del objeto `runner` al iniciar, detener o pausar el funcionamiento del applet.

Iniciar con init()

El método `init()` de un applet se procesa automáticamente al comenzar a ejecutar el applet por primera vez. Este método se emplea para asignar valores a las matrices `pageTitle` y `pageLink`. También permite crear un botón para el applet. El método está formado por las siguientes instrucciones:

```
public void init() {
   pageTitle = new String[] {
      "Sun's Java site",
      "Cafe au Lait",
      "JavaWorld",
      "Java in 24 Hours",
      "Sams Publishing",
      "Workbench"
      };
   pageLink[0] = getURL("http://java.sun.com");
   pageLink[1] = getURL("http://www.ibiblio.org/javafaq");
   pageLink[2] = getURL("http://www.javaworld.com");
```

```
    pageLink[3] = getURL("http://www.java24hours.com");
    pageLink[4] = getURL("http://www.samspublishing.com");
    pageLink[5] = getURL("http://workbench.cadenhead.org");
    Button goButton = new Button("Go");
    goButton.addActionListener(this);
    FlowLayout flow = new FlowLayout();
    setLayout(flow);
    add(goButton);
}
```

El título de cada página se almacena en los seis elementos de la matriz `pageTitle`, que se inicializa con seis cadenas. Los elementos de la matriz `pageLink` reciben un valor devuelto por el método `getURL()`, todavía por crear.

Las últimas siete instrucciones del método `init()` crean y ubican el botón **Go** en la ventana del applet.

Capturar errores al definir la URL

Al definir un objeto `URL`, debe asegurarse de que el texto usado en la dirección tiene el formato correcto. `http://workbench.cadenhead.org` y `http://www.samspublishing.com` son correctos pero `http:www.javaworld.com` no al faltar los símbolos `/`.

El método `getURL(String)` acepta una dirección Web como argumento y devuelve el objeto `URL` que representa dicha dirección. Si la cadena no es una dirección válida, el método devuelve `null`:

```
URL getURL(String urlText) {
    URL pageURL = null;
    try {
        pageURL = new URL(getDocumentBase(), urlText);
    } catch (MalformedURLException m) {
        // no hacer nada
    }
    return pageURL;
}
```

El bloque `try-catch` procesa los errores `MalformedURLException` que puedan producirse al crear objetos `URL`. Como no debe suceder nada si se genera esta excepción, el bloque `catch` solo contiene un comentario.

Controlar actualizaciones de pantalla en el método paint()

El método `paint()` de un applet se ejecuta cuando es necesario actualizar la ventana del applet. También puede invocarlo manualmente.

La invocación de `repaint()` fuerza la invocación del método `paint()`. Esta instrucción indica a la IGU que ha sucedido algo.

El applet `LinkRotator` tiene un breve método `paint()`:

```
public void paint(Graphics screen) {
   Graphics2D screen2D = (Graphics2D) screen;
   screen2D.setColor(butterscotch);
   screen2D.fillRect(0, 0, getSize().width, getSize().height);
   screen2D.setColor(Color.black);
   screen2D.drawString(pageTitle[current], 5, 60);
   screen2D.drawString("" + pageLink[current], 5, 80);
}
```

La primera instrucción crea un objeto `screen2D` que representa el area de dibujo de la ventana del applet. Todos los dibujos se realizan invocando los métodos de este objeto.

El método `setColor()` de `Graphics2D` selecciona el color empleado para posteriores dibujos. El color se establece en `butterscotch` antes de trazar un rectángulo que ocupa toda la ventana del applet.

Tras ello, el color se establece en `black` y se muestran líneas de texto en las posiciones (5,60) y (5,80) de la pantalla. La primera línea es un elemento de la matriz `pageTitle`. La segunda muestra la dirección del objeto `URL`, que se almacena en la matriz `pageLink`. La variable `current` determina el elemento de las matrices que se van a mostrar.

Iniciar el subproceso

En este applet, el subproceso `runner` comienza al invocar el método `start()` del applet y se detiene al invocar `stop()`.

El método `start()` se invoca después del método `init()` y siempre que se reinicie el programa. Veamos este método:

```
public void start() {
   if (runner == null) {
      runner = new Thread(this);
      runner.start();
   }
}
```

Este método inicia el subproceso `runner` si no se encuentra ya iniciado.

La instrucción `runner = new Thread(this)` crea un nuevo objeto `Thread` con un argumento: la palabra clave `this`. Hace referencia al propio applet y lo designa como la clase que se ejecuta junto al subproceso.

La invocación de `runner.start()` provoca la ejecución del subproceso. Al iniciarse se invoca su método `run()`. Como el subproceso `runner` es el propio applet, se invoca el método `run()` del applet.

Ejecutar el subproceso

En el método `run()` se concentra la actividad del subproceso. En el applet `LinkRotator`, el método `run()` se representa de esta forma:

```
public void run() {
   Thread thisThread = Thread.currentThread();
   while (runner == thisThread) {
      current++;
      if (current > 5) {
         current = 0;
      }
      repaint();
      try {
         Thread.sleep(10000);
      } catch (InterruptedException e) {
         // no hacer nada
      }
   }
}
```

Lo primero que sucede en `run()` es la creación del objeto `thisThread`. Un método de la clase `Thread`, `currentThread()`, define el valor del objeto `thisThread`. El método `currentThread()` controla el subproceso actualmente en ejecución.

Todas las instrucciones de este método forman parte de un bucle `while` que compara el objeto `runner` con el objeto `thisThread`. Ambos son subprocesos, y mientras hagan referencia al mismo objeto, el bucle `while` sigue ejecutándose. No hay instrucciones dentro de este bucle que hagan que los objetos `runner` y `thisThread` tengan valores diferentes, de modo que continúa indefinidamente hasta que un elemento externo cambia uno de los objetos `Thread`.

El método `run()` invoca `repaint()`. Tras ello, se aumenta en uno el valor de la variable `current` y si supera 5, se vuelve a establecer en 0. La variable `current` se usa en el método `paint()` para determinar de qué sitio Web mostrar la información. Al cambiar `current` se muestra un sitio distinto con `paint()`.

Este método incluye otro bloque `try-catch` para controlar errores. La instrucción `Thread.sleep(10000)` detiene el subproceso durante 10 segundos, tiempo suficiente para que los usuarios lean el nombre y la dirección del sitio Web. La instrucción `catch` se encarga de los errores `InterruptedException` que puedan producirse mientras se procesa la instrucción `Thread.sleep()`. Estos errores se producen si algo interrumpe el subproceso mientras está detenido.

Detener el subproceso

El método `stop()` se invoca siempre que el applet se detiene por salir de su página, de modo que es un lugar perfecto para detener un subproceso en ejecución. El método `stop()` del applet `LinkRotator` contiene las siguientes instrucciones:

```
public void stop() {
   if (runner != null) {
      runner = null;
   }
}
```

La instrucción `if` comprueba si el objeto `runner` es igual a `null`. En caso afirmativo, no existe un subproceso activo que detener. En caso contrario, la instrucción establece `runner` en `null`. De este modo, el objeto `runner` tiene un valor distinto al objeto `thisThread`. Cuando esto sucede, deja de ejecutarse el bucle `while` dentro del método `run()`.

Procesar clics del ratón

El último factor que tener en cuenta en el applet `LinkRotator` es el control de eventos. Cuando un usuario pulsa el botón **Go**, el navegador Web debe abrir el sitio Web mostrado. Lo hace con el método `actionPerformed()`, que se invoca siempre que se pulse el botón. Veamos el método `actionPerformed()` del applet `LinkRotator`:

```
public void actionPerformed(ActionEvent event) {
   if (runner != null) {
      runner = null;
   }
   AppletContext browser = getAppletContext();
   if (pageLink[current] != null) {
      browser.showDocument(pageLink[current]);
   }
}
```

Lo primero que sucede en este método es que el subproceso `runner` se detiene. La siguiente instrucción crea un nuevo objeto `AppletContext` con el nombre `browser`. Un objeto `AppletContext` representa el entorno en el que se presenta el applet, es decir, la página en la que se encuentra y el navegador Web que la ha cargado.

El método `showDocument(URL)` carga la dirección Web especificada en el navegador. Si `pageLink[current]` es una dirección válida, `showDocument()` solicita al navegador que cargue la página.

Mostrar enlaces circulares

Ya puede crear el programa y probarlo. Genere un nuevo archivo vacío de Java con el nombre `LinkRotator` e introduzca el texto del listado.

Listado 19.2. Texto completo de LinkRotator.java.

```
1: import java.applet.*;
2: import java.awt.*;
3: import java.awt.event.*;
```

```
 4: import javax.swing.*;
 5: import java.net.*;
 6:
 7: public class LinkRotator extends JApplet
 8:   implements Runnable, ActionListener {
 9:
10:   String[] pageTitle = new String[6];
11:   URL[] pageLink = new URL[6];
12:   Color butterscotch = new Color(255, 204, 158);
13:   int current = 0;
14:   Thread runner;
15:
16:   public void init() {
17:      pageTitle = new String[] {
18:         "Sun's Java site",
19:         "Cafe au Lait",
20:         "JavaWorld",
21:         "Java in 24 Hours",
22:         "Sams Publishing",
23:         "Workbench"
24:      };
25:      pageLink[0] = getURL("http://java.sun.com");
26:      pageLink[1] = getURL("http://www.ibiblio.org/javafaq");
27:      pageLink[2] = getURL("http://www.javaworld.com");
28:      pageLink[3] = getURL("http://www.java24hours.com");
29:      pageLink[4] = getURL("http://www.samspublishing.com");
30:      pageLink[5] = getURL("http:// workbench.cadenhead.org");
31:      Button goButton = new Button("Go");
32:      goButton.addActionListener(this);
33:      FlowLayout flow = new FlowLayout();
34:      setLayout(flow);
35:      add(goButton);
36:   }
37:
38:   URL getURL(String urlText) {
39:      URL pageURL = null;
40:      try {
41:         pageURL = new URL(getDocumentBase(), urlText);
42:      } catch (MalformedURLException m) { }
43:      return pageURL;
44:   }
45:
46:   public void paint(Graphics screen) {
47:      Graphics2D screen2D = (Graphics2D) screen;
48:      screen2D.setColor(butterscotch);
49:      screen2D.fillRect(0, 0, getSize().width, getSize().height);
50:      screen2D.setColor(Color.black);
51:      screen2D.drawString(pageTitle[current], 5, 60);
52:      screen2D.drawString("" + pageLink[current], 5, 80);
53:   }
54:
55:   public void start() {
56:      if (runner == null) {
57:         runner = new Thread(this);
58:         runner.start();
59:      }
```

```
60:    }
61:
62:    public void run() {
63:       Thread thisThread = Thread.currentThread();
64:       while (runner == thisThread) {
65:          current++;
66:          if (current > 5) {
67:             current = 0;
68:          }
69:          repaint();
70:          try {
71:             Thread.sleep(10000);
72:          } catch (InterruptedException e) {
73:             // no hacer nada
74:          }
75:       }
76:    }
77:
78:    public void stop() {
79:       if (runner != null) {
80:          runner = null;
81:       }
82:    }
83:
84:    public void actionPerformed(ActionEvent event) {
85:       if (runner != null) {
86:          runner = null;
87:       }
88:       AppletContext browser = getAppletContext();
89:       if (pageLink[current] != null) {
90:          browser.showDocument(pageLink[current]);
91:       }
92:    }
93: }
```

Tras guardar el programa, debe crear una página Web a la que añadir el applet; no funcionará correctamente si emplea Ejecutar>Ejecutar archivo para probarlo en NetBeans ya que los enlaces no se pueden abrir de esta forma.

Cree una nueva página Web. Seleccione Archivo>Archivo Nuevo y haga clic en Otro para localizar la opción Archivo HTML en el panel Tipos de Archivos del cuadro de diálogo Escoja el tipo de archivo. Asigne el nombre `LinkRotator` a la página, que NetBeans guarde como `NetBeans.html`, e introduzca el listado 19.3 como marcado de la página.

Listado 19.3. Texto completo de LinkRotator.html.

```
1: <applet
2:    code="LinkRotator.class"
3:    codebase="..\\build\\classes"
4:    width="300"
5:    height="100"
6: >
7: </applet>
```

Cuando termine, haga clic con el botón derecho del ratón sobre `LinkRotator.html` en el panel **Proyecto** y seleccione **Vista**. La página se abre en un navegador Web y el applet muestra todos los enlaces en rotación. Pulse el botón **Go** para visitar un sitio. En la figura 19.2 puede ver el applet.

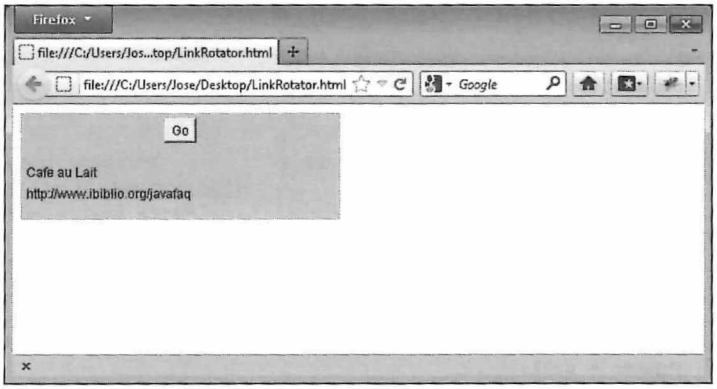


Figura 19.2. Enlaces giratorios en la ventana de un applet.

Resumen

Los subprocesos son un potente concepto implementado con un número reducido de clases e interfaces en Java. Al admitir el subprocesamiento en sus programas, su capacidad de respuesta es mayor y se pueden acelerar las tareas que realizan.

Aunque no haya aprendido nada más en este capítulo, ahora tiene un nuevo término para describir su frenético ritmo de vida.

Preguntas y respuestas

P: ¿Hay algún motivo para no hacer nada con una instrucción `catch`, como sucede en el applet `LinkRotator`?

R: Depende del tipo de error o de excepción capturado. En `LinkRotator`, sabe el origen de la excepción en las dos instrucciones `catch`, por lo que sabe que no hacer nada siempre es correcto. En el método `getURL()`, `MalformedURLExceptio` solo se produce si la URL enviada al método no es válida.

Ejercicios

Responda a las siguientes preguntas sobre el subprocesamiento múltiple en Java.

Preguntas

1. ¿Qué interfaz debe implementarse para que un programa use subprocesos?
   A. `Runnable`
   B. `Thread`
   C. `JApplet`
2. Si una interfaz contiene tres métodos distintos, ¿cuántos deben incluirse en una clase que implemente dicha interfaz?
   A. Ninguno.
   B. Todos.
   C. Sé la respuesta pero me la callo.

Respuestas

1. A. Debe usarse `Runnable` con la instrucción `implements`. `Thread` se emplea en un programa con subprocesos múltiples pero no se necesita en la instrucción de clase que inicia un programa.
2. B. Una interfaz garantiza que la clase incluye todos los métodos de la interfaz.

Actividades

Si este extenso ejercicio no ha acabado con sus fuerzas, amplíe sus conocimientos con las siguientes actividades:

- Si se siente cómodo con HTML, cree su propia página Web que incluya el applet `LinkRotator` y seis de sus sitios Web preferidos. Use el applet junto a los demás gráficos y texto de la página.
- Añada un botón a la aplicación `PrimeFinder` que permita detener el subproceso mientras la secuencia de números primos se está calculando.

20
Leer y escribir archivos

Existen diversas formas de representar datos en un ordenador. Ya hemos visto una: la creación de objetos. Un objeto incluye datos en forma de variables y referencias a objetos. También incluye métodos que usan los datos para realizar tareas.

Para trabajar con otros tipos de datos, como archivos de un disco duro o documentos de un servidor Web, puede usar las clases del paquete `java.io`. La parte `io` de este nombre corresponde a *input/output* (entrada/salida) y a las clases usadas para acceder a un origen de datos, como un disco duro, un CD-ROM o la memoria del ordenador.

Puede añadir datos a un programa y enviarlos por medio de un sistema de comunicación denominado flujo, objetos que transfieren información de un punto a otro.

Flujos

Para guardar datos de forma permanente en un programa de Java o para recuperarlos después, debe usar al menos un flujo. Un flujo es un objeto que toma información de un origen y la envía a otro punto. Los flujos conectan diversos orígenes, como programas informáticos, discos duros, servidores de Internet, la memoria del ordenador y los DVD-ROM. Una vez que aprenda a trabajar con un tipo de datos mediante flujos, podrá trabajar con otros de la misma forma. En este capítulo usaremos flujos para leer y escribir datos almacenados en archivos de su equipo. Hay dos tipos de flujos:

- Flujos de entrada, para leer datos de un origen.
- Flujos de salida, para escribir datos en un origen.

Todos los flujos de entrada y salida están formados por bytes, enteros con valores comprendidos entre 0 y 255. Puede usar este formato para representar datos, como programas ejecutables, documentos de procesadores de texto o archivos de música MP3, una pequeña muestra de lo que puede representar con bytes. Un flujo de bytes se usa para leer y escribir este tipo de datos.

Nota

Los archivos de clase de Java almacenados como bytes se denominan código de bytes. El intérprete de Java ejecuta código de bytes, que no se reproduce necesariamente en el lenguaje Java. Puede ejecutar un código de bytes compilado por otros lenguajes como NetRexx y Jython. El intérprete de Java también se denomina intérprete de código de bytes.

Una forma más especializada de trabajar con datos son los caracteres, como letras, números o signos de puntuación. Puede usar un flujo de caracteres para leer y escribir un origen de texto. Independientemente de que trabaje con un flujo de bytes, de caracteres u otro tipo de información, el proceso es el mismo:

- Crear un objeto de flujo asociado a los datos.
- Invocar métodos del flujo para añadir o recuperar información del flujo.
- Cerrar el flujo mediante la invocación del método `close()` del objeto.

Archivos

En Java, los archivos se representan con la clase `File`, también del paquete `java.io`. Los archivos se pueden leer de discos duros, CD-ROM y otros dispositivos de almacenamiento. Un objeto `File` puede representar archivos que ya existan o que desee crear. Para crear un objeto `File`, use el nombre del archivo como constructor:

```
File bookName = new File("address.dat");
```

Se crea un objeto para el archivo `address.dat` en la carpeta actual. También puede incluir una ruta en el nombre del archivo:

```
File bookName = new File("data\\address.dat");
```

Nota

Este ejemplo funciona en Windows, que usa el carácter de barra invertida (`\\`) como separador en rutas y nombres de archivo. En Linux y otros sistemas basados en Unix se usa el carácter de barra inclinada (`/`). Para crear un programa de Java que haga

referencia a archivos y que funcione en todos los sistemas, use la variable de clase `File.pathSeparator` en lugar de las barras, como en esta instrucción:

```
File bookName = new
File("data" +
File.pathSeparator
   + "address.dat");
```

Una vez creado el objeto `File`, puede invocar diversos métodos:

- **exists():** `true` si el archivo existe, `false` en caso contrario.
- **getName():** El nombre del archivo como cadena.
- **length():** El tamaño del archivo, como valor `long`.
- **createNewFile():** Crea un archivo del mismo nombre en caso de que no exista ya.
- **delete():** Elimina el archivo si existe.
- **renameTo(Archivo):** Cambia el nombre del archivo, usando como argumento el nombre del objeto `File` especificado.

También puede usar un objeto `File` para representar una carpeta de su sistema en lugar de un archivo. Especifique el nombre de la carpeta en el constructor de `File`, ya sea absoluta (como `C:\\MisDocumentos\\`) o relativa (como `java\\database`).

Una vez conseguido el objeto que representa la carpeta, puede invocar su método `listFiles()` para ver su contenido. Este método devuelve una matriz de objetos `File` que representan todos los archivos y subcarpetas que contiene.

Leer datos de un flujo

El primer proyecto del capítulo consiste en leer datos de un archivo mediante un flujo de entrada. Para ello usaremos la clase `FileInputStream`, que representa flujos de entrada que se leen como bytes desde un archivo. Puede crear un flujo de entrada si especifica un nombre de archivo o un objeto `File` como argumento del constructor `FileInputStream()`. El archivo debe existir antes de crear el flujo de entrada. En caso contrario, se genera una `IOException` al intentar crear el flujo. Muchos de los métodos asociados a la lectura y escritura de archivos generan esta excepción, por lo que suele ser conveniente incluir todas las instrucciones en su propio bloque `try-catch`:

```
try {
   File cookie = new File("cookie.web");
   FileInputStream stream = new FileInputStream(cookie);
   System.out.println("Length of file: " + cookie.length());
} catch (IOException e) {
   System.out.println("Could not read file.");
}
```

Los flujos de entrada leen datos en bytes. Puede leer un solo byte si invoca el método `read()` del flujo sin argumentos. Si no hay más bytes disponibles en el flujo por haber alcanzado el final del archivo, se devuelve el valor de byte `-1`.

Al leer un flujo de entrada, comienza con el primer byte del flujo, por ejemplo el primero de un archivo. Puede ignorar bytes de un flujo si invoca su método `skip()` con un argumento `int` que represente el número de bytes que se van a ignorar. La siguiente instrucción ignora los siguientes 1.024 bytes del flujo `scanData`:

```
scanData.skip(1024);
```

Si desea leer más de un byte por vez, siga estos pasos:

- Cree una matriz de bytes con el tamaño exacto del número de bytes que desea leer.
- Invoque el método `read()` del flujo con la matriz como argumento. La matriz se completa con bytes leídos desde el flujo.

Puede crear una aplicación que lea datos ID3 de un archivo de audio MP3. Como MP3 es un conocido formato de archivo de música, suelen añadirse 128 bytes al final de un archivo ID3 para guardar información sobre la canción, como el título o el álbum.

La aplicación `ID3Reader` lee un archivo MP3 mediante un flujo de entrada e ignora todo menos los últimos 128. Los bytes restantes se examinan para comprobar si contienen datos ID3. En caso afirmativo, los tres primeros bytes son los números 84, 65 y 71.

Nota

En el conjunto de caracteres ASCII, incluido en el estándar Unicode admitido por Java, estos tres números representan las letras mayúsculas T, A y G, respectivamente.

Cree un nuevo archivo vacío de Java con el nombre `ID3Reader` y añada el código del listado 20.1.

Listado 20.1. Texto completo de ID3Reader.java.

```
 1: import java.io.*;
 2:
 3: public class ID3Reader {
 4:   public static void main(String[] arguments) {
 5:     try {
 6:       File song = new File(arguments[0]);
 7:       FileInputStream file = new FileInputStream(song);
 8:       int size = (int) song.length();
 9:       file.skip(size - 128);
10:       byte[] last128 = new byte[128];
11:       file.read(last128);
12:       String id3 = new String(last128);
```

```
13:          String tag = id3.substring(0, 3);
14:          if (tag.equals("TAG")) {
15:             System.out.println("Title: " + id3.substring(3, 32));
16:             System.out.println("Artist: " + id3.substring(33, 62));
17:             System.out.println("Album: " + id3.substring(63, 91));
18:             System.out.println("Year: " + id3.substring(93, 97));
19:          } else {
20:             System.out.println(arguments[0] + " does not contain"
21:                + " ID3 info.");
22:          }
23:          file.close();
24:       } catch (Exception e) {
25:          System.out.println("Error - " + e.toString());
26:       }
27:    }
28: }
```

Antes de ejecutar la clase como aplicación, debe especificar un archivo MP3 como argumento de línea de comandos. El programa se puede ejecutar con cualquier MP3, como por ejemplo `Come On and Gettit.mp3`, el injustamente olvidado clásico soul de 1973 de Marion Black.

La figura 20.1 muestra el resultado de la aplicación `ID3Reader` con un archivo MP3.

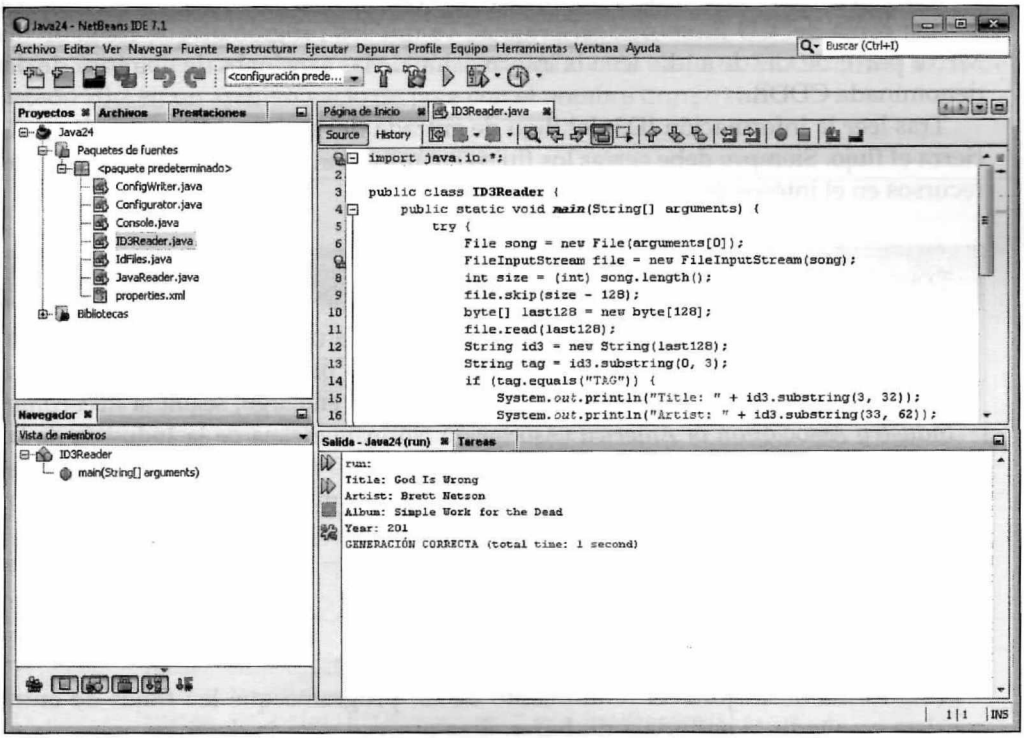


Figura 20.1. Ejecución de la aplicación ID3Reader.

Truco

Si no tiene `Come On and Gettit.mp3` en su equipo (craso error, en mi modesta opinión), puede buscar canciones MP3 con la licencia Creative Commons desde Yahoo! Search (`http://es.yahoo.com/`). Creative Commons es un conjunto de licencias de copyright que estipulan cómo distribuir, editar o publicar libros o canciones. En el sitio Web Rock Proper (`www.rockproper.com`) encontrará una colección de álbumes en MP3 con licencia para ser compartidos bajo Creative Commons.

La aplicación lee los últimos 128 bytes del MP3 en las líneas 10-11 del listado 20.1 y los almacena en la matriz `byte`. Esta matriz se usa en la línea 12 para crear un objeto `String` con los caracteres representados por dichos bytes.

Si los tres primeros caracteres de la cadena son `TAG`, el archivo MP3 examinado contiene información ID3 en un formato compatible con la aplicación.

En las líneas 15-18, se invoca el método `substring()` de la cadena para mostrar partes de la misma. Los caracteres que se van a mostrar tienen formato ID3, que sitúa la información sobre el artista, canción, título y año en las mismas posiciones de los últimos 128 bytes de un archivo MP3. Algunos archivos MP3 no contienen información ID3 o la incluyen en un formato diferente.

El archivo `Come On and Gettit.mp3` contiene información ID3 legible si lo ha creado de una copia del CD *Eccentric Soul* original, ya que los programas que crean archivos MP3 a partir de CD de audio leen la información de las canciones de una base de datos denominada CDDB.

Tras leer la información ID3 del flujo de entrada del archivo MP3, en la línea 23 se cierra el flujo. Siempre debe cerrar los flujos cuando termine de usarlos para conservar recursos en el intérprete de Java.

Nota

Puede que se vea tentado a buscar una copia de `Come On and Gettit.mp3` en un servicio como BitTorrent, uno de los más populares para compartir archivos. En este caso, comprendo perfectamente la tentación. Sin embargo, según la *Recording Industry Association of America* (Asociación Norteamericana de la Industria de la Grabación), todo el que descargue archivos MP3 de CD que no posea, arderá en los infiernos. Puede adquirir *Eccentric Soul* a través de Amazon, eBay, Apple iTunes y otros servicios similares.

Flujos de entrada en búfer

Una forma de mejorar el rendimiento de un programa que lea flujos de entrada consiste en añadir la entrada a un búfer. Este proceso guarda datos en memoria para usarlos posteriormente, cuando el programa los necesite. Cuando un programa de Java

necesita datos de un flujo de entrada en búfer, busca primero en el búfer, una acción más rápida que leer desde un origen como un archivo. Para usar un flujo de entrada en búfer, puede crearlo como objeto `FileInputStream` y después usar este objeto para crear un flujo de entrada. Invoque el constructor `BufferedInputStream(InputStream)` con el flujo de entrada como único argumento. Los datos se guardan en búfer al leerlo del flujo de entrada.

Para leer un flujo en búfer, invoque su método `read()` sin argumentos. Se devuelve un entero comprendido entre 0 y 255 que representa el siguiente byte de datos del flujo. Si no hay más bytes disponibles, se devuelve `-1`.

Como ejemplo de flujo en búfer, el siguiente programa añade una función a Java que muchos programadores de otros lenguajes echan en falta: la entrada de consola.

La entrada de consola es la capacidad de leer caracteres desde la consola (también denominada línea de comandos) mientras se ejecuta una aplicación.

La clase `System`, que contiene la variable `out` en las instrucciones `System.out.print()` y `System.out.println()`, tiene la variable de clase `in` que representa el objeto `InputStream`. Este objeto recibe la entrada del teclado y la ofrece como flujo.

Puede usar este flujo de entrada como cualquier otro.

La siguiente instrucción crea un flujo de entrada en búfer asociado al flujo de entrada `System.in`:

```
BufferedInputStream bin = new BufferedInputStream(System.in);
```

El siguiente proyecto, la clase `Console`, contiene un método de clase que puede usar para recibir la entrada de consola desde una aplicación de Java. Introduzca el texto del listado 20.2 en un archivo vacío de Java con el nombre `Console`.

Listado 20.2. Texto completo de Console.java.

```
 1: import java.io.*;
 2:
 3: public class Console {
 4:   public static String readLine() {
 5:     StringBuffer response = new StringBuffer();
 6:      try {
 7:        BufferedInputStream bin = new
 8:           BufferedInputStream(System.in);
 9:        int in = 0;
10:        char inChar;
11:        do {
12:           in = bin.read();
13:           inChar = (char) in;
14:           if (in != -1) {
15:              response.append(inChar);
16:           }
17:        } while ((in != -1) & (inChar != '\n'));
18:        bin.close();
19:        return response.toString();
20:     } catch (IOException e) {
21:        System.out.println("Exception: " + e.getMessage());
22:        return null;
23:     }
```

```
24:    }
25:
26:    public static void main(String[] arguments) {
27:       System.out.print("You are standing at the end of the road ");
28:       System.out.print("before a small brick building. Around you ");
29:       System.out.print("is a forest. A small stream flows out of ");
30:       System.out.println("the building and down a gully.\n");
31:       System.out.print("> ");
32:       String input = Console.readLine();
33:       System.out.println("That's not a verb I recognize.");
34:    }
35: }
```

La clase `Console` incluye un método `main()` que ilustra cómo se puede usar. Al ejecutar la aplicación, se obtiene el resultado mostrado en la figura 20.2.

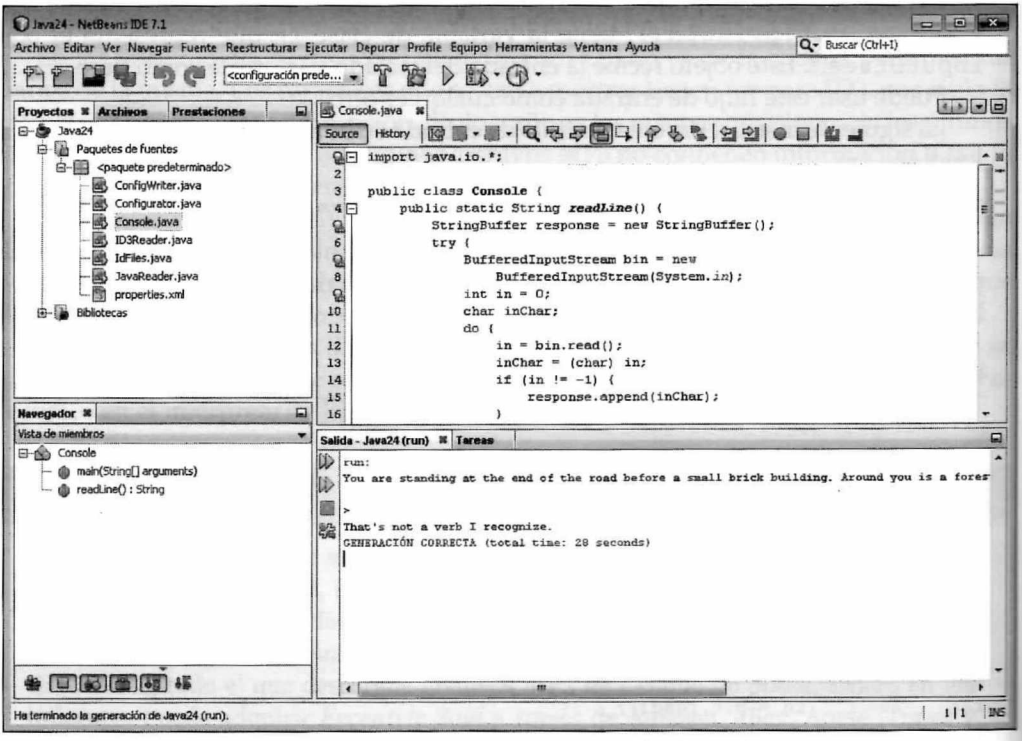


Figura 20.2. Ejecución de la aplicación Console.

La clase `Console` contiene un método de clase, `readLine()`, que recibe caracteres de la consola. Al pulsar **Intro**, `readLine()` devuelve un objeto `String` que contiene todos los caracteres recibidos.

Si guarda la clase `Console` en una carpeta indicada en su variable de entorno `CLASSPATH` (en Windows), puede invocar `Console.readLine()` desde cualquier programa de Java.

Nota

La clase `Console` también es el juego de aventura de texto menos satisfactorio del mundo. No puede entrar en el edificio, vadear el flujo ni huir. Si prefiere una versión más completa de este juego, llamado Adventure, pruebe con el archivo de Interactive Fiction (`www.wurb.com/if/game/1`).

Escribir datos en un flujo

En el paquete `java.io`, las clases para trabajar con flujos se combinan en grupos. Las clases `FileInputStream` y `FileOutputStreams` para trabajar con flujos de bytes, `FileReader` y `FileWriter` para trabajar con flujos de caracteres y otros muchos grupos para trabajar con otros datos de flujos. Para empezar a escribir datos, primero se crea un objeto `File` asociado a un flujo de salida. No es necesario que este archivo exista en su sistema. Puede crear `FileOutputStream` de dos formas. Si desea añadir bytes a un archivo existente, invoque el constructor `FileOutputStream()` con dos argumentos: un objeto `File` que representa el archivo y el valor Booleano `true`. Los bytes escritos en el flujo se adjuntan al final del archivo.

Si desea escribir en un nuevo archivo, invoque el constructor `FileOutputStream()` con un objeto `File` como único argumento.

Una vez obtenido el flujo de salida, puede invocar distintos métodos `write()` para escribir bytes en su interior:

- Invoque `write()` con un byte como único argumento para escribir ese byte en el flujo.
- Invoque `write()` con una matriz de bytes como único argumento para escribir todos los bytes de la matriz en el flujo.
- Especifique tres argumentos en el método `write(byte[], int, int)`: una matriz de bytes, un entero que represente el primer elemento de la matriz que se va a escribir en el flujo y el número de bytes a escribir.

La siguiente instrucción crea una matriz de bytes con 10 bytes y escribe los últimos 5 en el flujo de salida:

```
File dat = new File("data.dat");
FileOutputStream datStream = new FileOutputStream(dat);
byte[] data = new byte[] { 5, 12, 4, 13, 3, 15, 2, 17, 1, 18 };
datStream.write(data, 5, 5);
```

Al escribir bytes en un flujo, puede convertir el texto en una matriz de bytes invocando el método `getBytes()` en el objeto `String`, como en este ejemplo:

```
String name = "Puddin N. Tane";
byte[] nameBytes = name.getBytes();
```

Una vez escritos los bytes en un flujo, debe cerrarlo mediante la invocación de su método `close()`. El siguiente proyecto es una sencilla aplicación, `ConfigWriter`, que guarda varias líneas de texto en un archivo escribiendo bytes en un flujo de salida. Cree un archivo vacío de Java e introduzca el texto del listado 20.3 en el editor de código.

Listado 20.3. Texto completo de ConfigWriter.java.

```
 1: import java.io.*;
 2:
 3: class ConfigWriter {
 4:    String newline = System.getProperty("line.separator");
 5:
 6:    ConfigWriter() {
 7:       try {
 8:          File file = new File("program.properties");
 9:          FileOutputStream fileStream = new FileOutputStream(file);
10:          write(fileStream, "username=max");
11:          write(fileStream, "score=12550");
12:          write(fileStream, "level=5");
13:       } catch (IOException ioe) {
14:          System.out.println("Could not write file");
15:       }
16:    }
17:
18:    void write(FileOutputStream stream, String output)
19:       throws IOException {
20:
21:       output = output + newline;
22:       byte[] data = output.getBytes();
23:       stream.write(data, 0, data.length);
24:    }
25:
26:    public static void main(String[] arguments) {
27:       ConfigWriter cw = new ConfigWriter();
28:    }
29: }
```

Al ejecutar la aplicación, se crea el archivo `program.properties` con las tres siguientes líneas de texto:

```
username=max
score=12550
level=5
```

Leer y escribir propiedades de configuración

Los programas de Java son más versátiles cuando se pueden configurar mediante argumentos de línea de comandos, como hemos hecho en aplicaciones de otros capítulos. El paquete `java.util` incluye la clase `Properties`, que permite cargar parámetros de configuración desde un archivo de texto.

El archivo se puede leer como otros orígenes de archivos en Java:

- Crear un objeto `File` que represente el archivo.
- Crear un objeto `FileInputStream` desde ese archivo `File`.
- Invocar `load()` para recuperar las propiedades de ese flujo de entrada.

Un archivo de propiedades tiene un conjunto de nombres de propiedad seguidos de un signo igual (=) y sus valores. Veamos un ejemplo:

```
username=lepton
lastCommand=open database
windowSize=32
```

Cada propiedad tiene su propia línea, por lo que hay que configurar las propiedades `username`, `lastCommand` y `windowSize` con los valores `"lepton"`, `"open database"` y `"32"`, respectivamente (el mismo formato se usó en la clase `ConfigWriter`).

El siguiente código carga el archivo de propiedades `config.dat`:

```
File configFile = new File("config.dat");
FileInputStream inStream = new FileInputStream(configFile);
Properties config = new Properties();
config.load(inStream);
```

Los parámetros de configuración, denominados propiedades, se almacenan como cadenas en el objeto `Properties`. Cada propiedad se identifica mediante una clase, similar al parámetro de un applet. El método `getProperty()` recupera una propiedad a través de su clave, como en esta instrucción:

```
String username = config.getProperty("username");
```

Al almacenar las propiedades como cadenas, debe convertirlas para poder usarlas como valor numérico:

```
String windowProp = config.getProperty("windowSize");
int windowSize = 24;
try {
   windowSize = Integer.parseInt(windowProp);
} catch (NumberFormatException exception) {
   // no hacer nada
}
```

Las propiedades se pueden almacenar mediante la invocación del método `setProperty()` con dos argumentos: la clave y el valor:

```
config.setProperty("username", "max");
```

Puede mostrar todas las propiedades si invoca el método `list(PrintStream)` del objeto `Properties`. `PrintStream` es la clase de la variable `out` de la clase `System`, que ya hemos usado en el libro para mostrar resultados en instrucciones `System.out.println()`. El siguiente código invoca `list()` para mostrar todas las propiedades:

```
config.list(System.out);
```

Una vez modificadas las propiedades, puede almacenarlas en el archivo:

- Cree un objeto `File` que represente el archivo.
- Cree un objeto `FileOutputStream` a partir de ese objeto `File`.
- Invoque `store(OutputStream, String)` para guardar las propiedades en el flujo de salida indicado, con una descripción del archivo de propiedades como cadena.

En el siguiente proyecto ampliaremos la aplicación `ConfigWriter`, que escribe parámetros de programa en un archivo. La aplicación `Configurator` lee dichos parámetros en un archivo de propiedades de Java, añade la propiedad `runtime` con la fecha y hora actuales, y guarda el archivo. Cree un nuevo archivo vacío de Java para guardar la clase `Configurator` e introduzca el texto del listado 20.4.

Listado 20.4. Texto completo de Configurator.java.

```
 1: import java.io.*;
 2: import java.util.*;
 3:
 4: class Configurator {
 5:
 6:    Configurator() {
 7:       try {
 8:          // cargar el archivo de propiedades
 9:          File configFile = new File("program.properties");
10:          FileInputStream inStream = new FileInputStream(configFile);
11:          Properties config = new Properties();
12:          config.load(inStream);
13:          // crear una nueva propiedad
14:          Date current = new Date();
15:          config.setProperty("runtime", current.toString());
16:          // guardar el archivo de propiedades
17:          FileOutputStream outStream = new FileOutputStream(configFile);
18:          config.store(outStream, "Properties settings");
19:          inStream.close();
20:          config.list(System.out);
21:       } catch (IOException ioe) {
22:          System.out.println("IO error " + ioe.getMessage());
23:       }
24:    }
25:
26:    public static void main(String[] arguments) {
27:       Configurator con = new Configurator();
28:    }
29: }
```

El resultado de la aplicación `Configurator` se muestra en la figura 20.3. El archivo `program.properties` contiene el siguiente texto:

```
#Properties settings
#Tue May 12 22:51:26 EDT 2009
runtime=Tue May 12 22\:51\:26 EDT 2009
```

```
score=12550
level=5
username=max
```

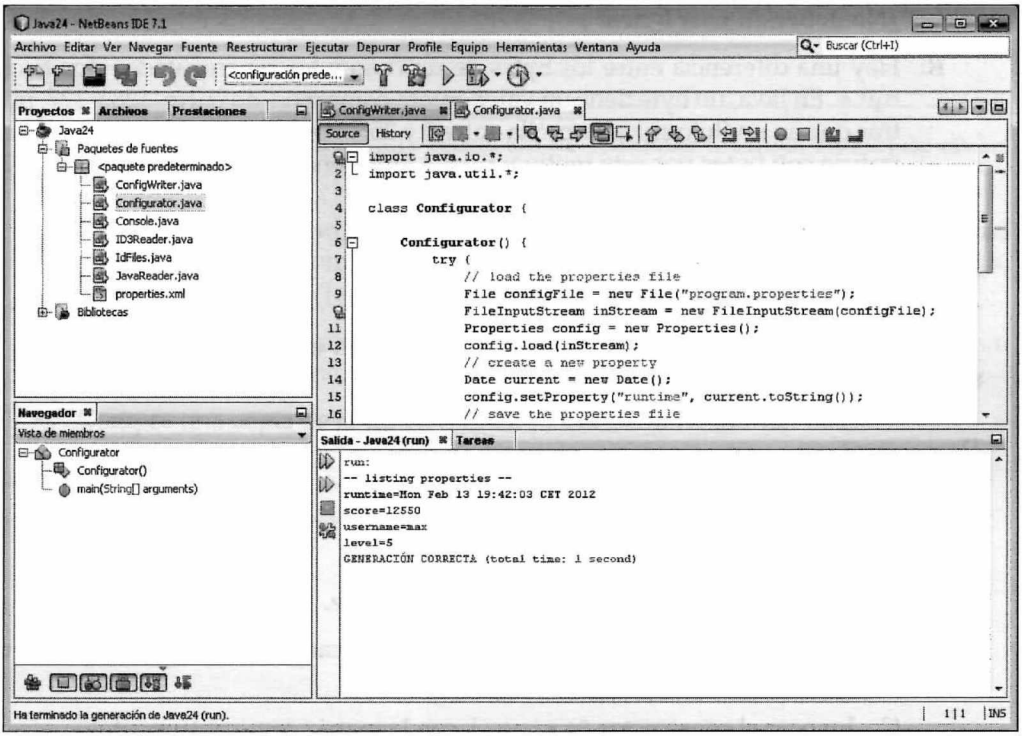


Figura 20.3. Ejecución de la aplicación Configurator.

El carácter de barra invertida (\), que difiere del resultado de la aplicación, garantiza que el archivo de propiedades se guarde correctamente.

Resumen

En este capítulo hemos trabajado con flujos de entrada y salida para escribir bytes, la forma más sencilla de representar datos en un flujo. El paquete `java.io` cuenta con otras muchas clases para trabajar con flujos. También existe un paquete `java.net` que le permite leer y escribir flujos sobre una conexión de Internet.

Los flujos de bytes se pueden adaptar a distintos usos, lo que le permite convertir bytes a otros tipos de datos como enteros, caracteres y cadenas.

El primer proyecto del capítulo, la aplicación `ID3Reader`, lee bytes de un flujo y los convierte en cadena ya que es más fácil leer los datos ID3 en este formato de una canción como *Come On and Gettit*, de Marian Black, del álbum *Eccentric Soul*. Por cierto, ¿he mencionado que debería comprar la canción?

Preguntas y respuestas

P: ¿Por qué algunos de los métodos del capítulo usan enteros como argumentos? ¿No deberían usar bytes?

R: Hay una diferencia entre los bytes de un flujo y los representados por la clase `byte`. En Java, un byte tiene un valor entero comprendido entre -128 y 127, mientras que en un flujo su valor oscila entre 0 y 255. Por lo general usará `int` cuando trabaje con bytes por este motivo: puede almacenar los valores entre 128 y 255, mientras que `byte` no.

Ejercicios

Para comprobar los conocimientos adquiridos en este capítulo, responda a las siguientes preguntas sobre flujos en Java.

Preguntas

1. De las siguientes técnicas, ¿cuál se usa para convertir una matriz de bytes en una cadena?

   A. Invocar el método `toString()` de la matriz.

   B. Convertir los bytes en caracteres y asignar cada uno a un elemento de una matriz `String`.

   C. Invocar el constructor `String()` con la matriz como argumento.

2. ¿Qué tipo de flujo se usa para leer de un archivo en un programa de Java?

   A. Un flujo de entrada.

   B. Un flujo de salida.

   C. Ninguno.

3. ¿Qué método de la clase `File` se puede usar para determinar el tamaño de un archivo?

   A. `getSize()`

   B. `read()`

   C. `length()`

Respuestas

1. C. Puede procesar individualmente cada byte, como sugiere la respuesta B, pero es más sencillo crear cadenas a partir de otros tipos de datos.

2. A. Un flujo de entrada se crea a partir de un objeto `File` o proporcionando un nombre de archivo al método constructor del flujo de entrada.
3. C. Este método devuelve un `long` que representa el número de bytes del flujo

Actividades

Para experimentar la refrescante sensación de remontar un flujo, sumérjase en las siguientes actividades.

- Cree una aplicación que lea las etiquetas ID3 de todos los archivos MP3 de una carpeta y cambie el nombre de los archivos utilizando la información sobre artistas, canciones y álbumes (siempre que exista).
- Cree un programa que lea un archivo de código fuente de Java y lo escriba sin cambios bajo un nuevo nombre.
- Compre una copia de la canción *Come on and Gettit* de Marian Black.

21
Leer y escribir datos XML

El aumento presencial de Java en los años 90 coincidió con otro cambio radical en el desarrollo del software: la aparición de XML (*Extensible Markup Language*, Lenguaje de marcado extensible). XML, un formato para organizar y almacenar datos para poder leerlos en cualquier programa, se ha convertido en algo omnipresente.

Gracias a XML, se pueden leer y escribir datos independientemente del software usado para crearlos. Es un importante cambio con respecto al pasado, cuando todos los programas parecían tener su propio formato propietario.

Los datos XML se pueden leer con un analizador, un programa que reconoce el formato y que puede extraer las partes necesarias de los datos.

En este capítulo aprenderemos a leer y a escribir datos XML con el XOM (*XML Object Model*, Modelo de objetos XML), una biblioteca de clases de Java que facilita el trabajo con datos XML en programas de Java.

Crear un archivo XML

Antes de explorar el XOM, debe conocer ciertos aspectos sobre XML y cómo almacena los datos. Los datos XML aparecen por todas partes: se pueden almacenar como archivos, transferir sobre una red o Internet y guardar en la memoria de un programa.

Diversas clases de Java pueden leer y escribir XML, incluida la clase `Properties` del paquete `java.util` que vimos en el capítulo anterior.

Un objeto `Properties` se puede almacenar como XML en lugar de usar el formato nombre=valor que vimos antes.

Una vez completado el objeto con propiedades de configuración, su método `storeToXML()` lo guarda en un archivo XML.

Este método acepta dos argumentos:

- `FileOutputStream`, un flujo sobre el que debe guardarse el archivo.
- Un comentario, que puede ser la cadena vacía `""` si los datos no requieren comentarios.

El primer proyecto del capítulo es una sencilla aplicación, `PropertyFileCreator`, que almacena propiedades de configuración en formato XML. Abra NetBeans, introduzca el texto del listado 21.1 en un nuevo archivo vacío de Java con el nombre `PropertyFileCreator` y guarde el archivo.

Listado 21.1. Texto completo de PropertyFileCreator.java.

```
 1: import java.io.*;
 2: import java.util.*;
 3:
 4: public class PropertyFileCreator {
 5:    public PropertyFileCreator() {
 6:       Properties prop = new Properties();
 7:       prop.setProperty("username", "rcade");
 8:       prop.setProperty("browser", "Mozilla Firefox");
 9:       prop.setProperty("showEmail", "no");
10:       try {
11:          File propFile = new File("properties.xml");
12:          FileOutputStream propStream = new FileOutputStream(propFile);
13:          Date now = new Date();
14:          prop.storeToXML(propStream, "Created on " + now);
15:       } catch (IOException exception) {
16:          System.out.println("Error: " + exception.getMessage());
17:       }
18:    }
19:
20:    public static void main(String[] arguments) {
21:       PropertyFileCreator pfc = new PropertyFileCreator();
22:    }
23: }
```

Al ejecutar la aplicación, se crea un archivo de propiedades con tres parámetros: `username` **rcade**, `browser` **Mozilla Firefox** y `showEmail` **no**.

Si las propiedades se hubieran guardado en otro formato, tendría este aspecto:

```
#Created on Wed Jun 15 20:56:33 EDT 2011
# Thu Wed Jun 15 20:56:33 EDT 2011
showEmail=no
browser=Mozilla Firefox
username=rcade
```

Al ejecutar la aplicación, genera el archivo XML `properties.xml`, presentado en el listado 21.2.

Listado 21.2. Texto completo de properties.xml.

```
1: <?xml version="1.0" encoding="UTF-8"?>
2: <!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
3: <properties>
4: <comment>Created on Wed Jun 15 20:56:33 EDT 2011</comment>
5: <entry key="showEmail">no</entry>
6: <entry key="browser">Mozilla Firefox</entry>
7: <entry key="username">rcade</entry>
8: </properties>
```

XML organiza los datos de forma que se documentan automáticamente, lo que permite comprenderlos con un simple vistazo. Si se fija en el listado 21.2, verá cómo almacena las propiedades de configuración. Las etiquetas `?xml` y `!DOCTYPE` pueden parecer complejas, pero el resto del archivo es muy simple.

En un archivo XML, los datos se incluyen entre etiquetas similares a las de HTML, el lenguaje de marcado usado en la Web. Las etiquetas de inicio empiezan con un carácter < seguido del nombre de un elemento y un carácter >, como en `<properties>`, en la línea 3 del listado 21.2. Las etiquetas de cierre comienzan por < seguidas del mismo nombre de elementos y los caractéres />, como en `</properties>` en la línea 8.

Todo lo comprendido entre ambas etiquetas se considera el valor del elemento.

Los datos XML deben tener un único elemento raíz que incluya todos sus datos. En el listado 21.2, la raíz es el elemento `properties` definido en las líneas 3-8.

Un elemento puede contener texto, un elemento secundario o varios. El elemento `properties` contiene cuatro secundarios: un elemento `comment` y tres elementos `entry`.

Veamos el elemento `comment`:

```
<comment>Created on Wed Jun 15 20:56:33 EDT 2011</comment>
```

El valor de este elemento es el texto que contiene: `Created on Wed Jun 15 20:56:33 EDT 2011`. Un elemento XML también puede tener uno o varios atributos, definidos dentro de su etiqueta de inicio como pares de nombre=valor. Los atributos se separan mediante espacios y ofrecen información adicional sobre el elemento.

Cada elemento `entry` tiene un atributo y un valor:

```
<entry key="showEmail">no</entry>
```

Este elemento tiene el valor `no` y un atributo `key` con el valor `showEmail`.

Un tipo de elemento XML no incluido en el listado 21.2 es el que se define con una única etiqueta. Estos elementos empiezan con el carácter <, seguido del nombre del elemento y los caracteres />.

Por ejemplo, este elemento podría ser un secundario de `properties`:

```
<inactive />
```

Aunque XML se ha descrito como formato y se ha comparado a HTML, en realidad no es un lenguaje. Describe cómo crear formatos de datos específicos para las tareas que desee realizar con un programa informático. Los formatos XML se denominan dialectos.

El dialecto XML creado por la clase `Properties` de Java es un ejemplo. Oracle ha desarrollado este formato para la representación de parámetros de configuración de software. Los datos que cumplen las reglas del formato XML se describen como bien formados. El software que lee o escribe XML debe aceptar datos bien formados.

Los datos también pueden someterse a un estándar más meticuloso denominado validez. Un archivo XML válido contiene los elementos adecuados en los puntos adecuados, y requiere una forma de definir los elementos válidos.

Leer un archivo XML

Como hemos visto en capítulos anteriores, existe una gran cantidad de código de Java ya creado para simplificar su trabajo. En la biblioteca de clases de Java, puede adoptar clases de Swing para programación de interfaces de usuario, las clases de `java.io` para acceder a archivos, las de `java.awt.event` para aceptar entradas del usuario y otras muchas para reducir al máximo las labores de programación. Un aspecto esencial para la programación con Java es saber dónde buscar clases y paquetes de Java que poder usar en sus propios proyectos. La reutilización de una biblioteca de clases bien desarrollada es mucho más sencilla que crear sus propias clases desde cero.

El equipo de Java en Oracle no es el único que crea clases de Java, como comprobará a lo largo del capítulo cuando use XOM, una biblioteca de clases desarrollada por el programador y escritor Elliotte Rusty Harold. Harold, un experto en Java y XML, estaba descontento con el funcionamiento de las bibliotecas XML existentes (seguramente le recuerde a la sensación de James Gosling con otro lenguaje). Harold creó su propia biblioteca de clases para representar datos XML como un árbol con cada elemento en un nodo. Puede descargar la biblioteca en `www.xom.nu`.

Descomprima el archivo en una carpeta de su sistema. Tras ello, debe añadir la biblioteca a su proyecto actual de NetBeans:

1. Seleccione Archivo>Propiedades del proyecto para abrir el cuadro de diálogo Propiedades del proyecto.
2. Haga clic en Bibliotecas en el panel Categorías y pulse el botón **Añadir biblioteca**. Se abrirá el cuadro de diálogo Añadir biblioteca.
3. Pulse el botón **Crear** para abrir el cuadro de diálogo Crear una biblioteca nueva.
4. Introduzca **XOM** en el campo Nombre biblioteca y pulse **Aceptar**. Se abrirá el cuadro de diálogo Personalizar biblioteca.
5. Haga clic en **Agregar archivo JAR/Carpeta**. Se abrirá el cuadro de diálogo Examinar archivos JAR/Carpeta.
6. Busque la carpeta en la que haya guardado XOM y seleccione los archivos `xom-1.2.1` y `xom-samples` (puede que el número de versión sea diferente). Pulse **Agregar archivo JAR/Carpeta**.

El dialecto XML creado por la clase `Properties` de Java es un ejemplo. Oracle ha desarrollado este formato para la representación de parámetros de configuración de software. Los datos que cumplen las reglas del formato XML se describen como bien formados. El software que lee o escribe XML debe aceptar datos bien formados.

Los datos también pueden someterse a un estándar más meticuloso denominado validez. Un archivo XML válido contiene los elementos adecuados en los puntos adecuados, y requiere una forma de definir los elementos válidos.

Leer un archivo XML

Como hemos visto en capítulos anteriores, existe una gran cantidad de código de Java ya creado para simplificar su trabajo. En la biblioteca de clases de Java, puede adoptar clases de Swing para programación de interfaces de usuario, las clases de `java.io` para acceder a archivos, las de `java.awt.event` para aceptar entradas del usuario y otras muchas para reducir al máximo las labores de programación. Un aspecto esencial para la programación con Java es saber dónde buscar clases y paquetes de Java que poder usar en sus propios proyectos. La reutilización de una biblioteca de clases bien desarrollada es mucho más sencilla que crear sus propias clases desde cero.

El equipo de Java en Oracle no es el único que crea clases de Java, como comprobará a lo largo del capítulo cuando use XOM, una biblioteca de clases desarrollada por el programador y escritor Elliotte Rusty Harold. Harold, un experto en Java y XML, estaba descontento con el funcionamiento de las bibliotecas XML existentes (seguramente le recuerde a la sensación de James Gosling con otro lenguaje). Harold creó su propia biblioteca de clases para representar datos XML como un árbol con cada elemento en un nodo. Puede descargar la biblioteca en `www.xom.nu`.

Descomprima el archivo en una carpeta de su sistema. Tras ello, debe añadir la biblioteca a su proyecto actual de NetBeans:

1. Seleccione Archivo>Propiedades del proyecto para abrir el cuadro de diálogo Propiedades del proyecto.
2. Haga clic en Bibliotecas en el panel Categorías y pulse el botón **Añadir biblioteca**. Se abrirá el cuadro de diálogo Añadir biblioteca.
3. Pulse el botón **Crear** para abrir el cuadro de diálogo Crear una biblioteca nueva.
4. Introduzca **XOM** en el campo Nombre biblioteca y pulse **Aceptar**. Se abrirá el cuadro de diálogo Personalizar biblioteca.
5. Haga clic en **Agregar archivo JAR/Carpeta**. Se abrirá el cuadro de diálogo Examinar archivos JAR/Carpeta.
6. Busque la carpeta en la que haya guardado XOM y seleccione los archivos `xom-1.2.1` y `xom-samples` (puede que el número de versión sea diferente). Pulse **Agregar archivo JAR/Carpeta**.

7. En el cuadro de diálogo Personalizar biblioteca, pulse **Aceptar**.
8. En el cuadro de diálogo Añadir biblioteca, seleccione XOM y pulse **Añadir biblioteca**.
9. Pulse **Aceptar** en el cuadro de diálogo Propiedades del proyecto.

La biblioteca XOM ya está disponible para su proyecto. Cuenta con clases para leer y escribir datos XML, y para guardarlos en archivos y otros destinos.

Advertencia

XOM está disponible de forma gratuita bajo una licencia de código abierto (LGPL). Puede distribuir la biblioteca XOM sin modificaciones con sus aplicaciones de Java. También puede modificar las clases de la biblioteca, pero dichos cambios deben ofrecerse bajo LGPL. Puede consultar los detalles completos de la licencia en `www.xom.nu/license.xhtml`.

El siguiente programa que crearemos es la aplicación `WeatherStation`, que lee información meteorológica ofrecida en un dialecto XML desde el sitio Weather Underground (`www.wunderground.com`).

Las clases principales de la biblioteca XOM se encuentran en el paquete `nu.xom`, que puede incluir en sus programas con una instrucción `import`:

```
import nu.xom.*;
```

La clase `Builder` puede cargar y analizar datos XML en cualquier dialecto, siempre que estén bien formados. El siguiente código crea un generador y carga el archivo `forecast`:

```
File file = new File("forecast.xml");
Builder builder = new Builder();
Document doc = builder.build(propFile);
```

XOM también puede cargar datos XML sobre la Web. En lugar de invocar `build(Archivo)`, invoque el método con la dirección Web de los datos:

```
Builder builder = new Builder();
Document doc = builder.build("http://tinyurl.com/rd4r72");
```

Nota

La dirección Web `http://tinyurl.com/rd4r72` es una URL abreviada que redirige a la dirección real del sitio Weather Underground, mucho más difícil de escribir correctamente. La dirección completa sería `http://wunderground.com/auto/wui/geo/ForecastXML/index.xml?query=Wasilla,AK`. Contiene información meteorológica de la localidad de Wasilla, en Alaska.

Cuando el generador carga los datos XML, los ofrece como objeto `Document`, que contiene el documento XML completo. Puede recuperar el elemento raíz del documento por medio de su método `getRootElement()`:

```
Element root = doc.getRootElement();
```

La clase `Element` representa un elemento. Cada uno tiene varios métodos que puede usar para examinar sus contenidos:

- El método `getFirstChildElement()` obtiene el primer elemento secundario que coincida con un nombre especificado.
- El método `get(int)` lee un elemento que coincida con un índice especificado, numero en orden desde `0`.
- El método `getChildElements()` obtiene todos sus elementos secundarios.
- El método `getValue()` lee su texto.
- El método `getAttribute()` recupera uno de sus atributos.

Las siguientes instrucciones recuperan el elemento `comment` y su valor:

```
Element highF = high.FirstChildElement("fahrenheit");
String highTemp = highF.getValue();
```

Este enfoque no funciona cuando varios elementos comparten el mismo nombre, como sucede con el elemento `forecastday`. En ese caso, puede recuperar todos los elementos y ejecutar un bucle `for` en los mismos:

```
Elements days = root.getChildElements("simpleday");
for (int current = 0; current < days.size(); current++) {
   Element day = days.get(current);
}
```

Este programa no usa atributos pero se puede acceder al atributo de un elemento por medio del método `getAttribute()`, que acepta el nombre del atributo como argumento:

```
Attribute key = day.getAttribute("key");
```

Una vez obtenido el atributo, su método `getValue()` muestra el valor:

```
String keyValue = key.getValue();
```

Cree un nuevo archivo vacío de Java con el nombre `WeatherStation` e introduzca el texto del listado 21.3.

Listado 21.3. Texto completo de WeatherStation.java.

```
1: import java.io.*;
2: import nu.xom.*;
3:
4: public class WeatherStation {
```

```
 5:   int[] highTemp = new int[6];
 6:   int[] lowTemp = new int[6];
 7:   String[] conditions = new String[6];
 8:
 9:   public WeatherStation() throws ParsingException, IOException {
10:      // obtener el documento XML
11:      Builder builder = new Builder();
12:      Document doc = builder.build("http://tinyurl.com/rd4r72");
13:      // obtener el elemento raíz, <forecast>
14:      Element root = doc.getRootElement();
15:      // obtener el elemento <simpleforecast>
16:      Element simple = root.getFirstChildElement("simpleforecast");
17:      // obtener los elementos  <forecastday>
18:      Elements days = simple.getChildElements("forecastday");
19:      for (int current = 0; current < days.size(); current++) {
20:         // obtener el elemento <forecastday> actual
21:         Element day = days.get(current);
22:         // obtener el elemento <high> actual
23:         Element high = day.getFirstChildElement("high");
24:         Element highF = high.getFirstChildElement("fahrenheit");
25:         // obtener el elemento <low> actual
26:         Element low = day.getFirstChildElement("low");
27:         Element lowF = low.getFirstChildElement("fahrenheit");
28:         // obtener el elemento <icon> actual
29:         Element icon = day.getFirstChildElement("icon");
30:         // almacenar los valores en variables de objeto
31:         lowTemp[current] = -1;
32:         highTemp[current] = -1;
33:         try {
34:            lowTemp[current] = Integer.parseInt(lowF.getValue());
35:            highTemp[current] = Integer.parseInt(highF.getValue());
36:         } catch (NumberFormatException nfe) {
37:            // no hacer nada
38:         }
39:         conditions[current] = icon.getValue();
40:      }
41:   }
42:
43:   public void display() {
44:      for (int i = 0; i < conditions.length; i++) {
45:         System.out.println("Period " + i);
46:         System.out.println("\tConditions: " + conditions[i]);
47:         System.out.println("\tHigh: " + highTemp[i]);
48:         System.out.println("\tLow: " + lowTemp[i]);
49:      }
50:   }
51:
52:   public static void main(String[] arguments) {
53:      try {
54:         WeatherStation station = new WeatherStation();
55:         station.display();
56:      } catch (Exception exception) {
57:         System.out.println("Error: " + exception.getMessage());
58:      }
59:   }
60: }
```

La aplicación `WeatherStation`, que no requiere argumentos de línea de comandos, genera siete predicciones meteorológicas para la localidad de Wasilla en Alaska:

```
Period 0
   Conditions: rain
   High: 56
   Low: 40
Period 1
   Conditions: partlycloudy
   High: 59
   Low: 40
Period 2
   Conditions: partlycloudy
   High: 61
   Low: 41
Period 3
   Conditions: chancerain
   High: 67
   Low: 43
Period 4
   Conditions: chancerain
   High: 67
   Low: 47
Period 5
   Conditions: chancerain
   High: 65
   Low: 49
```

Nota

La biblioteca de clases de Java incluye JAXP (*Java API for XML Processing*, API de Java para procesamiento XML), un conjunto de clases que realizan las mismas funciones que XOM. JAXP puede representar datos XML como objeto o como flujo de eventos, y ofrece al programador mayor control sobre la forma de analizar los datos. XOM es más fácil de aprender y requiere XML bien formado y válido en todo momento. Encontrará más información sobre JAXP en `http://jaxp.java.net`.

Leer información de suscripciones RSS

Existen cientos de dialectos XML que representan datos de forma independiente a plataformas y software. Uno de los más conocidos es RSS, un formato para intercambiar titulares y enlaces de sitios de noticias *online*, blogs y otras fuentes de información.

RSS ofrece el contenido Web en formato XML, perfecto para leerlo en archivos de software accesibles desde la Web denominados *feeds* (información de suscripciones). Los lectores RSS, llamados agregadores de noticias, son utilizados por millones de usuarios para estar al tanto de sus sitios Web preferidos. También existen aplicaciones Web que recopilan y comparten elementos RSS.

La clase `Builder` del paquete `nu.xom` puede cargar XML sobre Internet desde cualquier URL:

```
String rssUrl = "http://feeds.drudge.com/retort";
Builder builder = new Builder();
Document doc = builder.build(rssUrl);
```

En el ejercicio de este capítulo se emplea esta técnica para leer un archivo RSS y presentar los últimos 15 elementos.

Abra su editor e introduzca el texto del listado 21.4. Guarde el resultado como `Aggregator.java`.

Listado 21.4. Texto completo de Aggregator.java.

```
 1: import java.io.*;
 2: import nu.xom.*;
 3:
 4: public class Aggregator {
 5:    public String[] title = new String[15];
 6:    public String[] link = new String[15];
 7:    public int count = 0;
 8:
 9:    public Aggregator(String rssUrl) {
10:    try {
11:       // recuperar el elemento XML
12:       Builder builder = new Builder();
13:       Document doc = builder.build(rssUrl);
14:       // recuperar el elemento raíz del documento
15:       Element root = doc.getRootElement();
16:       // recuperar el elemento channel de la raíz
17:       Element channel = root.getFirstChildElement("channel");
18:       // obtener los elementos item de channel
19:       if (channel != null) {
20:          Elements items = channel.getChildElements("item");
21:          for (int current = 0; current < items.size(); current++) {
22:             if (count > 15) {
23:                break;
24:             }
25:             // obtener el elemento actual
26:             Element item = items.get(current);
27:             Element titleElement = item.getFirstChildElement("title");
28:             Element linkElement = item.getFirstChildElement("link");
29:             title[current] = titleElement.getValue();
30:             link[current] = linkElement.getValue();
31:             count++;
32:          }
33:       }
34:    } catch (ParsingException exception) {
35:       System.out.println("XML error: " + exception.getMessage());
36:       exception.printStackTrace();
37:    } catch (IOException ioException) {
38:       System.out.println("IO error: " + ioException.getMessage());
39:       ioException.printStackTrace();
40:    }
```

```
41: }
42:
43: public void listItems() {
44:    for (int i = 0; i < 15; i++) {
45:       if (title[i] != null) {
46:          System.out.println("\n" + title[i]);
47:          System.out.println(link[i]);
48:          i++;
49:       }
50:    }
51: }
52:
53:    public static void main(String[] arguments) {
54:       if (arguments.length > 0) {
55:          Aggregator aggie = new Aggregator(arguments[0]);
56:          aggie.listItems();
57:       } else {
58:          System.out.println("Usage: java Aggregator rssUrl");
59:       }
60:    }
61: }
```

Antes de ejecutar la aplicación, defina un argumento de línea de comandos para el *feed* que desee leer, que puede ser cualquier *feed* RSS. Si no conoce ninguno, use `http://feeds.drudge.com/retort`, que contiene titulares de Drudge Retort, un sitio de noticias *online* que publico. En la figura 21.1 puede ver un ejemplo.

Nota

Encontrará más información sobre el dialecto XML RSS en el sitio Web de RSS Advisory Board (`www.rssboard.org`). Soy el director de este servicio, que ofrece asistencia sobre el formato y un directorio de software que puede usar para leer *feed* RSS.

Resumen

Java libera al software de la dependencia de un determinado sistema operativo. El programa que cree con el lenguaje en Windows genera archivos que se pueden ejecutar en un servidor Linux o en un equipo con Mac OS X.

XML logra una liberación similar para los datos generados por software. Si los datos XML cumplen las sencillas reglas necesarias para que estén bien formados, puede leerlos con cualquier software que analice XML. No necesita el programa original para contar con una forma de acceso.

La biblioteca XOM facilita la lectura y escritura de datos XML.

Al usar Java y XML, puede declarar su independencia ante dos de los principales obstáculos a los que se enfrentan los programadores informáticos desde hace décadas: datos obsoletos y sistemas operativos obsoletos.

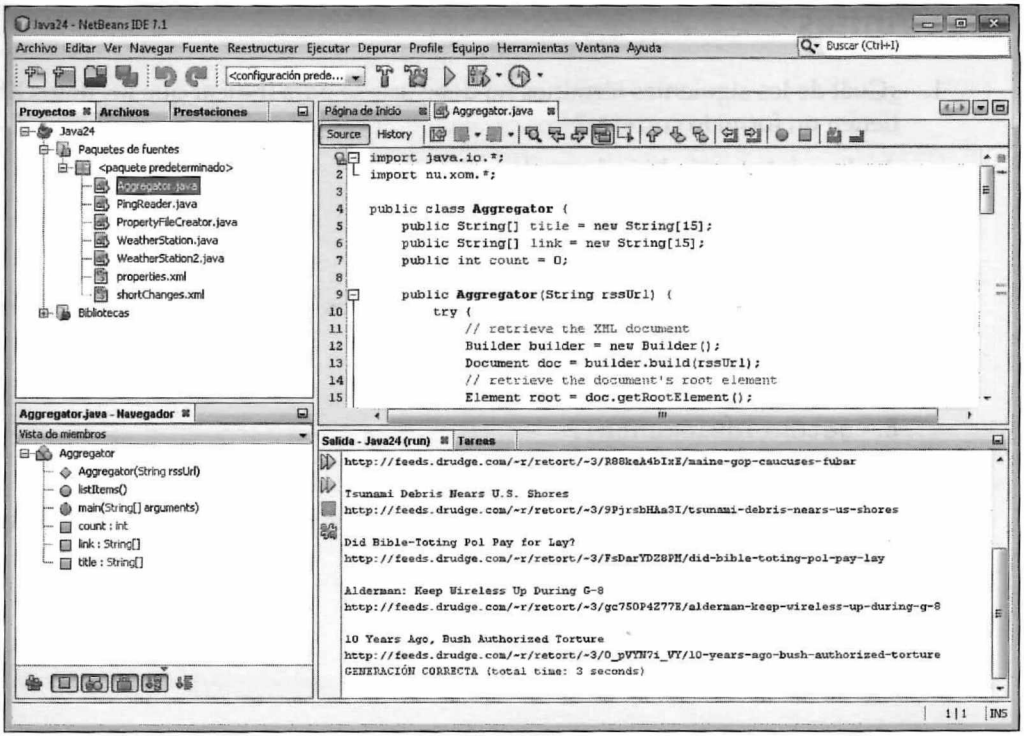


Figura 21.1. Ejecución de la aplicación Aggregator.

Preguntas y respuestas

P: ¿Para qué sirve la instrucción DOCTYPE del archivo XML generado por la aplicación `PropertyFileCreator`?

R: Es una referencia a una definición de tipo de document (DTD), un archivo que define las reglas que los datos XML deben cumplir para considerarse válidos en su dialecto. Si carga la página Web a la que se hace referencia en esa instrucción, `http://java.sun.com/dtd/properties.dtd`, encontrará referencias a todos los elementos y atributos incluidos en el archivo XML generado por la clase `Properties` de la biblioteca de clases de Java. Aunque Sun ofrezca esta DTD, la documentación oficial de Java indica que no debe usarse al evaluar datos de configuración de propiedades. Supuestamente los analizadores la ignorarán.

Ejercicios

Para comprobar sus conocimientos sobre XML en Java, responda a las siguientes preguntas.

Preguntas

1. ¿Cuál de los siguientes términos no debe usarse para indicar que los datos XML tienen un formato correcto?

   A. Los datos están bien formados.

   B. Los datos son válidos.

   C. Los datos son espectaculares.

2. ¿Qué método lee todos los elementos incluidos dentro de un elemento principal?

   A. `get()`

   B. `getChildElements()`

   C. `getFirstChildElement()`

3. ¿Qué método de la clase `Elements` puede usarse para determinar el número de elementos que contiene?

   A. `count()`

   B. `length()`

   C. `size()`

Respuestas

1. C. Los datos bien formados tiene etiquetas de inicio y cierre correctamente estructuradas, un solo elemento raíz con todos los elementos secundarios y una declaración `?XML` en la parte superior. Los datos válidos siguen las reglas de un determinado dialecto XML.
2. B. El método `getChildElements()` devuelve un objeto `Elements` que contiene todos los elementos.
3. C. Como los vectores, `Elements` usa un método `size()` que ofrece el número de elementos que contiene.

Actividades

Para ampliar sus conocimientos sobre XML, analice las siguientes actividades:

- Revise la aplicación `WeatherStation` para mostrar un elemento adicional de los datos meteorológicos de Weather Underground.
- Cree un programa que muestre los datos contenidos en `shortChanges.xml`, un documento XML de información sobre blogs disponible en `www.weblogs.com/shortChanges.xml`.

22
Crear servicios Web con JAX-WS

Ahora que Internet es omnipresente y controla millones de equipos de escritorio, servidores Web, consolas de videojuegos y otros dispositivos, el deseo de conectarlo todo ha propiciado el nacimiento de los servicios Web, software que se comunica con otro software a través de HTTP, el protocolo de la Web. Una de las novedades más atractivas de Java es JAX-WS (*Java API for XML Web Services*, API de Java para servicios Web XML). Se trata de un conjunto de paquetes y clases que crean clientes que realizan solicitudes de servicios Web y servicios que reciben dichas solicitudes.

JAX-WS admite servicios Web implementados mediante SOAP (*Simple Object Access Protocol*, Protocolo simple de acceso a objetos) y REST (*Representational State Transfer*, Transferencia de estado de representación). JAX-WS simplifica considerablemente la inclusión de estos protocolos. Como programador, creará objetos e invocará métodos par usar servicios Web; el resto se realiza entre bastidores.

Definir una interfaz de punto final de servicios

El primer paso para crear un servicio Web JAX-WS consiste e crear una interfaz de punto final de servicios, una interfaz de Java que define los métodos que los clientes pueden invocar al usar el servicio Web. El servicio Web `SquareRootServer` que desarrollaremos en este capítulo realiza dos tareas:

- Calcular la raíz cuadrada de un número.
- Mostrar la fecha y hora actuales.

Una interfaz es un conjunto de métodos que ofrece nombres, argumentos y tipos devueltos pero que no contiene código para implementar los métodos. La interfaz actúa de contrato entre objetos: si un objeto implementa una interfaz, otros objetos saben que pueden invocar todos los métodos de la misma en dicho objeto.

En el capítulo 15 implementamos la interfaz `ActionListener` en las clases de Java que necesitaban recibir eventos de acción al pulsar un botón.

En este proyecto, controlaremos el otro lado del contrato. La interfaz `SquareRootServer` define dos métodos que debe incluir una clase que implemente el servicio Web: `squareRoot(double)` y `getTime()`. Las siguientes instrucciones definen la interfaz:

```
public interface SquareRootServer {
   double getSquareRoot(double input);
   String getTime();
}
```

Las definiciones de métodos de una interfaz acaban en punto y coma, y no en los caracteres { y } de una instrucción de bloque. Las interfaces no definen el comportamiento de los métodos; de eso se encargan las clases que implementan la interfaz.

Como estos métodos se pueden invocar como servicio Web JAX-WS, debe añadirse un modificador adicional delante de cada uno, la anotación `@WebMethod`:

```
public interface SquareRootServer {
   @WebMethod double getSquareRoot(double input);
   @WebMethod String getTime();
}
```

Usar anotaciones para simplificar el código de Java

Las anotaciones son una especie de comentarios inteligentes comprensibles para el intérprete de Java, el compilador y las herramientas de programación. Permiten definir información sobre un programa que no forma parte del propio programa pero que puede desencadenar acciones al compilar o ejecutar el programa.

Las anotaciones comienzan por un signo `@` seguido del nombre de la anotación. Una de las más habituales es `@Override`, que indica un método que reemplaza el método de una superclase.

Veamos un ejemplo:

```
@Overrides public void paintComponent(Graphics comp) {
   // añadir aquí definición de método
}
```

Si comete un error y no reemplaza un método, lo que sucedería si usa el tipo o número incorrecto de parámetros, el compilador puede capturar el error.

La anotación `@WebMethod` indica que un método se puede invocar como servicio Web. La interfaz `SquareRootServer` también usa una anotación `@WebService` que indica que define una interfaz de punto final de servicio.

Las anotaciones pueden aceptar parámetros que ofrecen una mayor personalización. `SquareRootServer` incluye una última anotación:

```
@SOAPBinding(style = Style.RPC)
```

Esta anotación define el contrato entre el servicio Web y los programas cliente que lo invocan. Encontrará más información al respecto en un apartado posterior.

Empecemos con el código del servicio Web. Cree un nuevo archivo vacío de Java con el nombre de clase `SquareRootServer` y el nombre de paquete `com.java24hours.ws`. Introduzca el código del listado 22.1.

Listado 22.1. Texto completo de SquareRootServer.java.

```
 1: package com.java24hours.ws;
 2:
 3: import javax.jws.*;
 4: import javax.jws.soap.*;
 5: import javax.jws.soap.SOAPBinding.*;
 6:
 7: @WebService
 8:
 9: @SOAPBinding(style = Style.RPC)
10:
11: public interface SquareRootServer {
12:    // obtener la raíz cuadrada de un número
13:    @WebMethod double getSquareRoot(double input);
14:
15:    // obtener la fecha y hora actuales como cadena
16:    @WebMethod String getTime();
17:
18: }
```

Esta clase se incluye en el paquete `com.java24hours.ws`, una decisión de diseño que facilita la implementación del servicio Web en otro software para el acceso por Internet. Una vez definida la interfaz, ya puede crear el código para implementar sus dos métodos: `getSquareRoot()` y `getTime()`.

Crear un Bean de implementación de servicio

La clase de Java que implementa la interfaz de punto final de servicio se denomina Bean de implementación de servicio. El aprendizaje de nuevos términos técnicos es un aspecto habitual de JAX-WS.

La clase `SquareRootServerImpl` implementa la interfaz `SquareRootServer`, como indica la declaración:

```
public class SquareRootServerImpl implements SquareRootServer {
```

Estos significa que la clase debe contener todos los métodos de la interfaz, cada uno con los parámetros adecuados.

Los métodos `getSquareRoot(double)` y `getTime()` se implementan mediante técnicas que ya conoce. La única novedad es la siguiente anotación, que aparece antes de la instrucción de clase:

```
@WebService(endpointInterface = "com.java24hours.ws.SquareRootServer")
```

Esta anotación indica que la clase es un Bean de implementación de servicio de la interfaz `com.java24hours.ws.SquareRootServer`. Debe usar el nombre completo de la clase, incluido el nombre del paquete.

Recuerde que las anotaciones no acaban en punto y coma, como las instrucciones.

Para comenzar el código de la clase, cree un nuevo archivo vacío de Java con el nombre `SquareRootServerImpl` en el paquete `com.java24hours.ws` y añada el código del listado 22.2.

Listado 22.2. Texto completo de SquareRootServerImpl.java.

```
 1: package com.java24hours.ws;
 2:
 3: import java.util.*;
 4: import javax.jws.*;
 5:
 6: @WebService(endpointInterface = "com.java24hours.ws.SquareRootServer")
 7:
 8: public class SquareRootServerImpl implements SquareRootServer {
 9:
10:   public double getSquareRoot(double input) {
11:      return Math.sqrt(input);
12:   }
13:
14:   public String getTime() {
15:      Date now = new Date();
16:      return now.toString();
17:   }
18: }
```

Con las dos clases creadas ya puede iniciar el servicio Web para poder invocarlo desde otro software.

Nota

El nombre de Bean de implementación de servicio hace referencia a JavaBeans, clases especiales de Java diseñadas para funcionar como componentes reutilizables de software en Java Enterprise Edition. Sin embargo, la referencia es un tanto errónea en lo que respecta a JAX-WS. Cualquier objeto de Java puede ser un Bean de implementación de servicio siempre que siga las reglas para métodos de servicios Web y se haya creado con las anotaciones correctas.

Publicar el servicio Web

Los servicios Web JAX-WS se pueden implementar en servidores de aplicaciones de Java como BEAWebLogic, GlassFish, JBoss y Jetty. Si ha creado el servicio Web `SquareRootServer` en un entorno de desarrollo que admita dichos servidores, ya puede iniciarlo. También puede crear una aplicación de Java que cargue el servicio Web y lo ofrezca por Internet. La aplicación `SquareRootServerPublisher` se encarga de esta tarea, que solo requiere dos pasos:

- Cargar la clase que implementa el servicio Web.
- Publicar ese objeto en una dirección accesible a Internet.

La clase `EndPoint` del paquete `javax.xml.ws` cuenta con el método `publish(String, Object)` para implementar servicios Web.

El primer argumento de este método es la dirección Web desde la que se accede al servicio, que en este proyecto es `http://127.0.0.1:5335/service`. Esta dirección Web comienza con el nombre de host, `127.0.0.1`, denominado host local (`localhost`) por ser el ordenador local que usa para crear y ejecutar sus programas de Java. La segunda parte de la dirección es el número de puerto del host local en el que el servicio Web espera conexiones. El puerto `5335` se ha elegido de forma arbitraria ya que es improbable que otros programas lo usen en su equipo. La parte final de la dirección, `/service`, es la ruta. Todos los servicios Web deben tener una ruta exclusiva. Si ejecuta otros servicios Web en su equipo, no pueden tener la misma ruta que `SquareRootServer`.

Para implementar el servicio Web, cree un archivo vacío de Java con el nombre `SquareRootServerPublisher` en el paquete `com.java24hours.ws`, e introduzca el texto del listado 22.3.

Listado 22.3. Texto completo de SquareRootServerPublisher.java.

```
 1: package com.java24hours.ws;
 2:
 3: import javax.xml.ws.*;
 4:
 5: public class SquareRootServerPublisher {
 6:   public static void main(String[] arguments) {
 7:     SquareRootServerImpl srsi = new SquareRootServerImpl();
 8:     Endpoint.publish(
 9:       "http://127.0.0.1:5335/service",
10:       srsi
11:     );
12:   }
13: }
```

Al ejecutar la aplicación, espera conexiones en el puerto 5335 de su equipo. Puede invocar los métodos del servicio Web desde cualquier programa compatible con servicios Web SOAP o REST, se haya escrito en Java o en otro lenguaje. Mientras el servicio Web esté en Internet, cualquier software conectado a Internet podrá invocar sus métodos.

Usar archivos de lenguaje de definición de servicios Web

Antes de probar el servicio Web, puede probar la disponibilidad de la aplicación `SquareRootServerPublisher` con cualquier navegador. Abra un navegador y diríjase a la dirección `http://127.0.0.1:5335/service?wsdl`. El navegador mostrará el archivo XML del listado 22.4, proporcionado por la aplicación que acaba de crear.

Este archivo es un contrato de servicio escrito en WSDL (*Web Service Description Language*, Lenguaje de descripción de servicios Web), un dialecto XML para describir cómo funciona un servicio Web a los servidores y clientes que vayan a usarlo.

No es necesario entender WSDL para crear servicios JAX-WS y clientes para acceder a dichos servicios. No obstante, merece la pena examinar los contenidos para hacerse una idea de cómo funcionan los servicios Web basados en SOAP y REST.

Listado 22.4. Contrato de lenguaje de descripción de servicios Web.

```
 1: <?xml version="1.0" encoding="UTF-8"?>
 2: <!- Published by JAX-WS RI at http://jax-ws.dev.java.net. RI's version
 3: is JAX-WS RI 2.2.2 in JDK 7. ->
 4: <!- Generated by JAX-WS RI at http://jax-ws.dev.java.net. RI's version
 5: is JAX-WS RI 2.2.2 in JDK 7. ->
 6: <definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
 7: xmlns:tns="http://ws.java24hours.com/"
 8: xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 9: xmlns="http://schemas.xmlsoap.org/wsdl/"
10: targetNamespace="http://ws.java24hours.com/"
11: name="SquareRootServerImplService">
12: <types></types>
13: <message name="getSquareRoot">
14: <part name="arg0" type="xsd:double"></part>
15: </message>
16: <message name="getSquareRootResponse">
17: <part name="return" type="xsd:double"></part>
18: </message>
19: <message name="getTime"></message>
20: <message name="getTimeResponse">
21: <part name="return" type="xsd:string"></part>
22: </message>
23: <portType name="SquareRootServer">
24: <operation name="getSquareRoot" parameterOrder="arg0">
25: <input message="tns:getSquareRoot"></input>
26: <output message="tns:getSquareRootResponse"></output>
27: </operation>
28: <operation name="getTime" parameterOrder="">
29: <input message="tns:getTime"></input>
30: <output message="tns:getTimeResponse"></output>
31: </operation>
32: </portType>
33: <binding name="SquareRootServerImplPortBinding"
34: type="tns:SquareRootServer">
```

```
35: <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
36: style="rpc"></soap:binding>
37: <operation name="getSquareRoot">
38: <soap:operation soapAction=""></soap:operation>
39: <input>
40: <soap:body use="literal"
41: namespace="http://ws.java24hours.com/"></soap:body>
42: </input>
43: <output>
44: <soap:body use="literal"
45: namespace="http://ws.java24hours.com/"></soap:body>
46: </output>
47: </operation>
48: <operation name="getTime">
49: <soap:operation soapAction=""></soap:operation>
50: <input>
51: <soap:body use="literal"
52: namespace="http://ws.java24hours.com/"></soap:body>
53: </input>
54: <output>
55: <soap:body use="literal"
56: namespace="http://ws.java24hours.com/"></soap:body>
57: </output>
58: </operation>
59: </binding>
60: <service name="SquareRootServerImplService">
61: <port name="SquareRootServerImplPort"
62: binding="tns:SquareRootServerImplPortBinding">
63: <soap:address location="http://127.0.0.1:5335/service"></soap:address>
64: </port>
65: </service>
66: </definitions>
```

Un archivo WSDL se denomina contrato de servicio ya que estipula cómo acceder al servicio, los mensajes que se pueden intercambiar y los tipos de datos de la información transferida.

Las líneas 13-22 del contrato WSDL definen los métodos del servicio Web, los parámetros de dichos métodos y los datos devueltos en la respuesta. Fíjese en las líneas e intente determinar dónde se indica que el método `getSquareRoot()` acepta un parámetro `double` y devuelve un valor `double`.

Los tipos de datos a los que se hace referencia en el contrato no son de Java. Son tipos de datos generales para usarlos en cualquier lenguaje de programación compatible con SOAP (no existen servicios Web especiales para Java).

Nota

Como un contrato WSDL define un servicio Web con gran detalle, puede usarlo para automatizar la programación de servicios Web. El JDK incluye una herramienta de línea de comandos, `wsimport`, que acepta un archivo WSDL como entrada y crea clases de Java para acceder al servicio Web.

Crear un cliente de servicio Web

En este apartado, crearemos `SquareRootClient`, una aplicación de Java que puede invocar los métodos del servicio Web que acabamos de crear. Evidentemente, el servicio debe estar en ejecución para que el cliente pueda conectarse.

Como la tecnología de servicios Web como la biblioteca JAX-WS admite estándares como SOAP, REST, HTTP y XML, no es necesario usar un programa de Java para conectarse al servicio Web `square root`. Perl, Python, Ruby y otros lenguajes cuentan con bibliotecas que admiten servicios Web. La biblioteca JAX-WS ofrece la clase `Service` del paquete `javax.xml.ws`, una factoría que crea objetos que pueden invocar un servicio Web. El método de clase `Service.create(URL, QName)` crea la factoría. Los argumentos son un objeto `URL` de `java.net` y `QName` de `javax.xml.namespace`.

La URL debe ser la dirección del contrato WSDL del servicio Web:

```
URL url = new URL("http://127.0.0.1:5335/service?wsdl");
```

`QName` es un nombre cualificado, un identificador XML asociado al proveedor del servicio Web. Un nombre cualificado está formado por un URI de espacio de nombres y un identificador local. Los URI de espacio de nombres son similares a las URL pero no actúan necesariamente como direcciones Web. Como el nombre de paquete del servicio Web `square root` es `com.java24hours.ws`, que por convención en Java se asocia al nombre de host de Internet `ws.java24hours.com`, el URI de espacio de nombres de este servicio Web es `http://ws.java24hours.com`. El identificador local del servicio Web es el nombre del Bean de implementación de servicio con la palabra `Service` añadida. Veamos la instrucción que crea el nombre cualificado:

```
QName qname = new QName(
   "http://ws.java24hours.com/",
   "SquareRootServerImplService"
);
```

Con la URL y el nombre cualificado puede crear la factoría cliente del servicio Web:

```
Service service = Service.create(url, qname);
```

La factoría tiene un método `getPort(Class)` que crea un objeto de la clase especificada. Para identificar una clase de Java que usar como argumento del método, use una variable de clase de la clase `class`. ¿Confuso? Tendrá más sentido si lo lee una instrucción de Java:

```
SquareRootServer srs = service.getPort(SquareRootServer.class);
```

Advertencia

Como hemos indicado, un URI no tiene que ser una dirección Web funcional. Aunque `http://ws.java24hours.com` lo parezca, se usa como identificador exclusivo. Soy el propietario del nombre de dominio `java24hours.com` y puedo controlar cómo se

usan sus subdominios. Por ello, al designar `http://ws.java24hours.com` como URI, puedo estar seguro que ningún otro proveedor de servicios Web adoptara dicha identificación.

La invocación de `getPort()` con `SquareRootServer.class` como argumento hace que la factoría cree un objeto `SquareRootServer`, que se almacena en la variable `srs`. Invoque los métodos del objeto `SquareRootServer` como en cualquier objeto de Java:

```
System.out.println(srs.getTime());
System.out.println(srs.getSquareRoot(625D));
```

La biblioteca JAX-WS empaqueta estos métodos como mensajes SOAP, los envía por Internet al servicio Web y transmite las invocaciones. Cuando el servicio Web responde a las invocaciones, empaqueta las respuestas como mensajes SOAP, las devuelve por Internet y se vuelven a convertir en tipos de datos de Java.

Para combinar todas las piezas, cree un nuevo archivo vacío de Java con el nombre `SquareRootClient` e introduzca el texto del listado 22.5.

Listado 22.5. Texto completo de SquareRootClient.java.

```
 1: package com.java24hours.ws;
 2:
 3: import java.net.*;
 4: import javax.xml.namespace.*;
 5: import javax.xml.ws.*;
 6:
 7: class SquareRootClient {
 8:   public static void main(String[] arguments) throws Exception {
 9:     URL url = new URL("http://127.0.0.1:5335/service?wsdl");
10:     QName qname = new QName(
11:       "http://ws.java24hours.com/",
12:       "SquareRootServerImplService"
13:     );
14:     Service service = Service.create(url, qname);
15:     SquareRootServer srs = service.getPort(SquareRootServer.class);
16:
17:     System.out.println(srs.getTime());
18:     System.out.println(srs.getSquareRoot(625D));
19:   }
20: }
```

Al ejecutar la aplicación, verá el resultado de la figura 22.1 si la aplicación `SquareRootPublisher` está en ejecución.

Resumen

El conjunto JAX-WS de paquetes y clases es el sucesor de JAX-RPC (*Java API for XML-based RPC*, API de Java para RPC basado en XML), una tecnología para realizar llamadas de procedimiento remoto entre objetos Java por Internet.

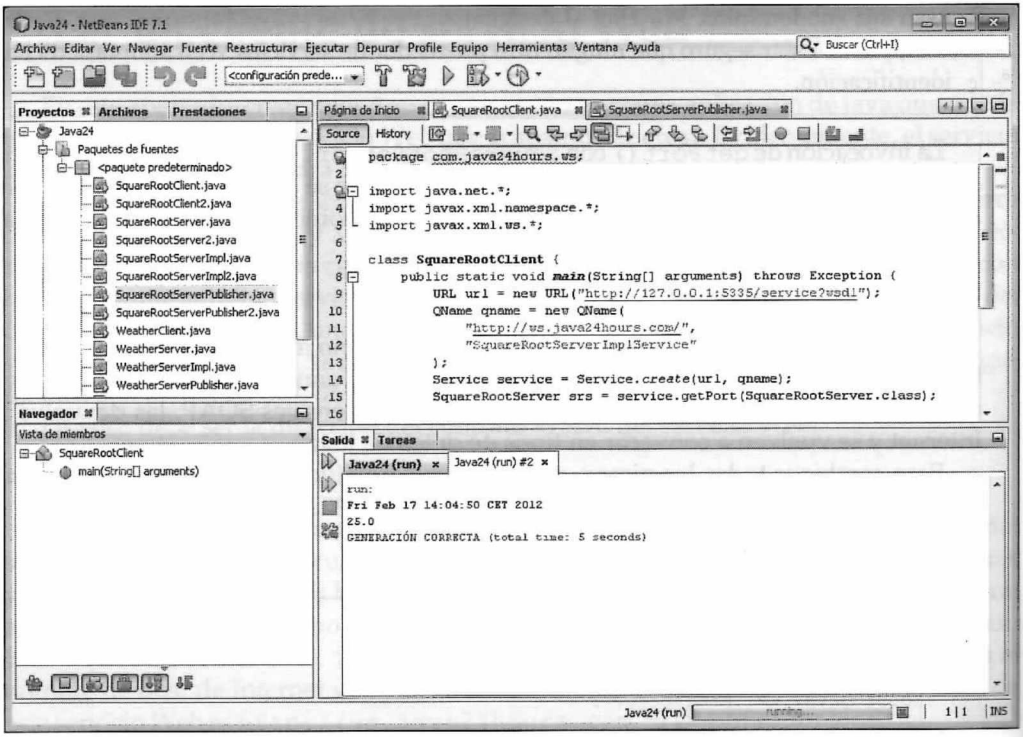


Figura 22.1. Invocación de un servicio Web y la respuesta mostrada.

La posibilidad de invocar otro software, independientemente de su ubicación y del lenguaje de programación usado para crearlo, es una de las piezas básicas de la tendencia de desarrollo de software llamada Web 2.0. La Web 2.0 permite al software aprovechar la omnipresente conectividad de Internet que como humanos llevamos disfrutando desde mediados de los años 90. En este capítulo hemos analizado los pasos necesarios para crear y usar un servicio Web con JAX-WS. Puede crear una interfaz para el servicio (una interfaz de punto final de servicio), implementarlo (un Bean de implementación de servicios), publicarlo en Internet y crear un cliente para acceder al mismo.

Muchas herramientas de programación, como NetBeans y el JDK, permiten crear código automáticamente para simplificar el diseño y el acceso a servicios Web.

Preguntas y respuestas

P: ¿Cómo encaja XML-RPC en los servicios Web SOAP?

R: XML-RPC es un protocolo para invocar métodos y recibir datos en formato XML sobre HTTP, el protocolo de la Web. SOAP es todo eso y, de hecho, ambos protocolos comparten un origen común. XML-RPC se creó como borrador del proto-

colo que acabaría convirtiéndose en SOAP. Como XML-RPC fue anterior y es más sencillo de implementar, su desarrollo continuó y sigue siendo muy popular. La biblioteca Java XML-RPC de Apache (`http://ws.apache.org/xmlrpc`), admite la creación de servicios Web y clientes que usan XML-RPC. SOAP es un protocolo de servicios Web más sofisticado y admite una mayor variedad de interacciones entre clientes y servicios.

P: **No tengo muy claro por qué un paquete como `com.java24hours.ws` se asocia al host de Internet `ws.java24hours.com`. ¿Cómo funciona?**

R: Los nombres de paquetes de Java los crean los programadores que desarrollan el paquete. Oracle comienza los nombres de paquetes de la biblioteca de clases de Java por `java` o `javax`, como en `java.util` y `javax.swing`. Cuando otros programadores crean el paquete, usan una convención que evite que dos entidades seleccionen el mismo nombre de paquete. La convención consiste en elegir un nombre de paquete que se base en alguna propiedad de la entidad, como un nombre de dominio. Como propietario del dominio `cadenhead.org`, he creado clases de Java con nombres de paquete que empiezan por `org.cadenhead`, como `org.cadenhead.web`. La *Apache Software Foundation*, propietaria de `apache.org`, llama a su paquete XML-RPC `org.apache.xmlrpc`.

Ejercicios

Compruebe sus conocimientos sobre servicios Web respondiendo a las siguientes preguntas.

Preguntas

1. ¿Qué es un Bean de implementación de servicio?
   A. Una interfaz que identifica los métodos que hay que alcanzar a través de un servicio Web.
   B. Una clase que implementa un servicio Web.
   C. Un contrato de servicio entre un servicio Web y clientes que invocan el servicio.
2. Cuando aparece un texto como `@WebMethod` o `@Override` en la declaración de un método, ¿qué nombre recibe?
   A. Anotación.
   B. Aserción.
   C. Agravante.

3. ¿Qué significan las siglas WSDL?
   A. Lenguaje de implementación de servicios Web.
   B. Lenguaje de descripción de servicios Web.
   C. Lucy in the Sky with Diamonds.

Respuestas

1. B. La respuesta A se refiere a una interfaz de punto final de servicios.
2. A. Aunque la respuesta C también podría ser válida, en función de cuántos problemas haya tenido en este apartado del capítulo.
3. B. Suele denominarse, equivocadamente, Lenguaje de definición de servicios Web.

Actividades

Para ampliar sus conocimientos sobre el tema de este capítulo, realice las siguientes actividades:

- Añada un método al servicio Web `square root` que multiplique un número por 10, y modifique la aplicación `SquareRootClient` para invocar un método.
- Cree un nuevo servicio Web que use la clase `WeatherStation` del capítulo 21 y ofrezca la temperatura mínima y máxima actual, y las condiciones meteorológicas a través de un servicio Web.

23
Crear gráficos Java2D

En este capítulo vamos a aprender a convertir contenedores de Swing, los sencillos paneles y marcos grises para componentes de interfaz gráfica de usuario (IGU), en un lienzo artístico en el que dibujar fuentes, colores, formas y gráficos.

Usar la clase Font

En Java, los colores y las fuentes se representan por las clases `Color` y `Font` del paquete `java.awt`. Con estas clases puede representar el texto con distintas fuentes y tamaños, y cambiar el color de texto y gráficos. Las fuentes se crean con el constructor `Font(String, int, int)`, que acepta tres argumentos:

- El tipo de fuente como nombre genérico (Dialog, DialogInput, Monospaced, SanSerif o Serif) o un nombre de fuente (Arial Black, Helvetica o Courier New).
- El estilo como una de las tres siguientes variables de clase: `Font.BOLD`, `Font.ITALIC` o `Font.PLAIN`.
- El tamaño de la fuente en puntos.

La siguiente instrucción crea un objeto `Font` Serif de 12 puntos en cursiva:

```
Font current = new Font("Serif", Font.ITALIC, 12);
```

Si usa fuentes diferentes a las genéricas, deben instalarse en el equipo desde el que se ejecute el programa. Puede combinar estilos de fuente si los añade unos a otros:

```
Font headline = new Font("Courier New", Font.BOLD + Font.ITALIC, 72);
```

Una vez definida la fuente, puede invocar el método `setFont(Fuente)` del componente `Graphics2D` para designarla como la actual. Las posteriores operaciones de dibujo usarán dicha fuente hasta que defina otra distinta. Las instrucciones del siguiente ejemplo crean un objeto de fuente Comic Sans y lo designan como fuente actual antes de trazar texto:

```
public void paintComponent(Graphics comp) {
   Graphics2D comp2D = (Graphics2D) comp;
   Font font = new Font("Comic Sans", Font.BOLD, 15);
   comp2D.setFont(font);
   comp2D.drawString("Potrzebie!", 5, 50);
}
```

Java admite el suavizado para dibujar fuentes y gráficos con un aspecto menos cuadrado. Para habilitarlo, debe establecer un consejo de renderización en Swing. El objeto `Graphics2D` tiene el método `setRenderingHint(int, int)`, que acepta dos argumentos:

- La clave del consejo.
- El valor que asociar a la clave.

Estos valores son variables de clase de la clase `RenderingHints` de `java.awt`. Para activar el suavizado, invoque `setRenderingHint()` con dos argumentos:

```
comp2D.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
   RenderingHints.VALUE_ANTIALIAS_ON);
```

El objeto `comp2D` de este ejemplo es el objeto `Graphics2D` que representa el entorno de dibujo de un contenedor.

Usar la clase Color

En Java, los colores se representan por medio de la clase `Color`, que incluye las siguientes constantes como variables de clase: `black`, `blue`, `cyan`, `darkGray`, `gray`, `green`, `lightGray`, `magenta`, `orange`, `pink`, `red`, `white` y `yellow`.

En un contenedor, puede establecer el color de fondo de un componente con estas constantes si invoca el método `setBackground(Color)` de esta forma:

```
setBackground(Color.orange);
```

El color actual, como la fuente actual, debe establecerse antes de dibujar, por medio del método `setColor(Color)`. El siguiente código incluye una instrucción para establecer el color actual en naranja y dibujar el texto con ese color:

```
public void paintComponent(Graphics comp) {
   Graphics2D comp2D = (Graphics2D) comp;
   comp2D.setColor(Color.orange);
   comp2D.drawString("Go, Buccaneers!", 5, 50);
}
```

Al contrario de lo que sucede con el método `setBackground()`, que puede invocar directamente en un contenedor, debe invocar el método `setColor()` en un objeto `Graphics2D`.

Crear colores personalizados

En Java puede crear colores personalizados si especifica su valor sRGB (*Standard Red Green Blue*, Rojo Verde Azul estándar). sRGB define un color mediante la cantidad de rojo, verde y azul que incluye. Cada valor oscila entre 0 (no tiene nada de ese color) y 255 (la cantidad máxima).

El constructor `Color(int, int, int)` acepta argumentos que representan los valores rojo, verde y azul. El siguiente código dibuja un panel que muestra texto naranja claro (230 de rojo, 220 de verde, 0 de azul) sobre un fondo rojo oscuro (235 de rojo, 50 de verde, 50 de azul):

```
import java.awt.*;
import javax.swing.*;

public class GoBucs extends JPanel {
   Color lightOrange = new Color(230, 220, 0);
   Color darkRed = new Color(235, 50, 50);

   public void paintComponent(Graphics comp) {
      Graphics2D comp2D = (Graphics2D) comp;
      comp2D.setColor(darkRed);
      comp2D.fillRect(0, 0, 200, 100);
      comp2D.setColor(lightOrange);
      comp2D.drawString("Go, Buccaneers!", 5, 50);
   }
}
```

Este ejemplo invoca el método `fillRect()` de `Graphics2D` para dibujar un rectángulo relleno con el color actual.

Nota

Los valores sRGB permiten crear 16.5 millones de combinaciones posibles, aunque la mayoría de monitores informáticos ofrece una versión aproximada de los mismos. Si necesita más información al respecto, consulte un manual especializado sobre el tema.

Dibujar líneas y formas

El dibujo de formas como líneas y rectángulos es tan sencillo en un programa de Java como mostrar texto. Basta con un objeto `Graphics2D` para definir la superficie de dibujo y objetos para representar lo que se va a dibujar.

El objeto `Graphics2D` tiene métodos para dibujar texto con un comando como el siguiente:

```
comp2D.drawString("Draw, pardner!", 15, 40);
```

Este código traza el texto `"Draw, pardner!"` en las coordenadas (15,40). Los métodos de dibujo usan el mismo sistema de coordenadas (x,y) que el texto. La coordenada (0,0) se encuentra en la esquina superior izquierda del contenedor, los valores X aumentan hacia la derecha y los valores Y hacia abajo. Puede determinar el valor (x,y) máximo que usar en un applet con las siguientes instrucciones:

```
int maxXValue = getSize().width;
int maxYValue = getSize().height;
```

A excepción de las líneas, las formas pueden tener relleno o no. Una forma con relleno se traza con el color actual ocupando totalmente el espacio de la forma. Las formas sin relleno trazan un borde con el color actual.

Dibujar líneas

El dibujo 2D de un objeto es creado para representar la forma trazada.

Los objetos que definen formas pertenecen al paquete `java.awt.geom`.

La clase `Line2D.Float` crea una línea que conecta un punto (x,y) inicial con otro final. La siguiente instrucción crea una línea desde el punto (40,200) al punto (70,130):

```
Line2D.Float line = new Line2D.Float(40F, 200F, 70F, 130F);
```

Los argumentos incluyen la letra `F` para indicar que son valores de coma flotante. Si se omite, Java los procesaría como enteros.

> **Nota**
>
> `Line2D.Float` incluye un punto en el centro del nombre de la clase, algo que no hemos visto anteriormente. Se debe a que `Float` es una clase interna de la clase `Line2D`, como detallamos en el capítulo 11.

Todas las formas, menos las líneas, se trazan invocando un método de la clase `Graphics2D`: `draw()` para contornos y `fill()` para formas con relleno.

La siguiente instrucción traza el objeto de línea creado en el ejemplo anterior:

```
comp2D.draw(line);
```

Dibujar rectángulos

Los rectángulos pueden tener relleno o no, y esquinas cuadradas o redondeadas. Se crean mediante el constructor `Rectangle2D.Float(int, int, int, int)` con estos argumentos:

- La coordenada x de la parte superior izquierda del rectángulo.
- La coordenada y de la parte superior izquierda.
- La anchura del rectángulo.
- La altura.

La siguiente instrucción dibuja un rectángulo sin relleno con cuatro esquinas cuadradas:

```
Rectangle2D.Float box = new
   Rectangle2D.Float(245F, 65F, 20F, 10F);
```

Esta instrucción crea un rectángulo con su esquina superior izquierda en la coordenada (245,65), una anchura de 20 píxeles y una altura de 10. Para trazarlo como contorno, puede usar lo siguiente:

```
comp2D.draw(box);
```

Si desea rellenar el rectángulo, use el método `fill()`:

```
comp.fill(box);
```

Puede crear rectángulos con esquinas redondeadas en lugar de cuadradas si usa la clase `RoundRectangle2D.Float`. El constructor de esta clase comienza con los mismos cuatro argumentos que `Rectangle2D.Float` pero añade otros dos:

- Una serie de píxeles en la dirección x alejándose de la esquina del rectángulo.
- Una serie de píxeles en la dirección y alejándose de la esquina.

Estas distancias se usan para determinar dónde iniciar el redondeo de la esquina del rectángulo. La siguiente instrucción crea un rectángulo redondeado:

```
RoundRectangle2D.Float ro = new RoundRectangle.Float(
   10F, 10F,
   100F, 80F,
   15F, 15F);
```

En este rectángulo, la esquina superior izquierda se encuentra en la coordenada (10,10). El tercer y cuarto argumentos especifican la anchura y altura del rectángulo. En este caso, tiene 100 píxeles de ancho y 80 de alto.

Los dos últimos argumentos de `drawRoundRect()` especifican que las cuatro esquinas deben empezar a redondearse a 15 píxeles con respecto a la esquina, en la posición (10,10).

Dibujar elipses y círculos

Puede crear elipses y círculos con la misma clase, `Ellipse2D.Float`, que acepta cuatro argumentos:

- La coordenada x de la elipse.
- La coordenada y de la elipse.
- Su anchura.
- Su altura.

Las coordenadas (x,y) no indican un punto en el centro de la elipse o el círculo, como habría esperado. En su lugar, describen un rectángulo invisible en el que se encaja la elipse. La coordenada (x,y) es la esquina superior izquierda de este rectángulo. Si tiene la misma anchura y altura, es un círculo.

La siguiente instrucción crea un círculo dentro del rectángulo en la coordenada (245,45), con 5 píxeles de alto y de ancho:

```
Ellipse2D.Float cir = new Ellipse2D.Float(
   245F, 45F, 5F, 5F);
```

Dibujar arcos

Otra forma circular que puede trazar en Java es un arco, una elipse o círculo parcial. Los arcos se crean con la clase `Arc2D.Float`, que tiene un método constructor con muchos de los mismos argumentos. Puede dibujar el arco si especifica una elipse, la parte de la elipse que debe ser visible (en grados) y el punto del arco que debe empezar en la elipse. Para crear un arco, especifique los siguientes argumentos enteros en el constructor:

- La coordenada x del rectángulo invisible para la elipse.
- La coordenada y del rectángulo.
- La anchura del rectángulo.
- La altura del rectángulo.
- El punto de la elipse en el que debe comenzar el arco (en grados de 0 a 359).
- El tamaño del arco (también en grados).
- El tipo de arco.

El punto inicial y el tamaño del arco oscilan entre 0 y 359 grados en sentido contrario a las agujas del reloj, comenzando por 0 grados en la posición de las 3 (véase la figura 23.1). El tipo de arco se especifica mediante variables de clase: `PIE` para porciones de gráficos circulares, `CLOSED` si los extremos se conectan mediante una línea recta y `OPEN` si los extremos no se conectan.

La siguiente instrucción traza un arco abierto en (100,50), es decir, 120 grados de longitud, que comienza en la marca de los 30 grados y tiene una anchura de 65 y una altura de 75:

```
Arc2D.Float smile = new Arc2D.Float(100F, 50F, 65F, 75F,
   30F, 120F, Arc2D.Float.OPEN);
```

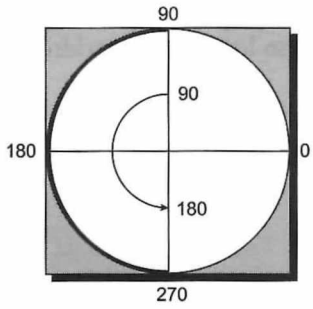


Figura 23.1. Definición de arcos en grados.

Crear un gráfico circular

En este capítulo crearemos un `PiePanel`, un componente de IGU que muestra un gráfico circular. Este componente es una subclase de `JPanel`, un sencillo contenedor de Swing muy útil para dibujar elementos. Una forma de empezar a crear una clase consiste en definir cómo se crean sus objetos. Los programas que usan la clase `PiePanel` deben:

- Crear un objeto `PiePanel` por medio del método de constructor `PiePanel(int)`. El entero especificado como argumento es el número de porciones del gráfico circular.
- Invocar el método `addSlice(Color, float)` del objeto para asignar a la porción un color y un valor.

El valor de cada porción de `PiePanel` es la cantidad representada por dicha porción. Por ejemplo, la tabla 23.1 muestra datos sobre el estado de la devolución de préstamos universitarios en Estados Unidos durante los primeros 38 años del programa.

Tabla 23.1. Devolución de préstamos universitarios.

Cantidad devuelta por los alumnos	$101 mil millones
Cantidad prestada a alumnos activos	$68 mil millones
Cantidad prestada a alumnos que abonan los pagos	$91 mil millones
Cantidad prestada a alumnos que no abonan los pagos	$25 mil millones

Podría usar `PiePanel` para representar estos datos en un gráfico circular con las siguientes instrucciones:

```
PiePanel loans = new PiePanel(4);
loans.addSlice(Color.green, 101F);
loans.addSlice(Color.yellow, 68F);
loans.addSlice(Color.blue, 91F);
loans.addSlice(Color.red, 25F);
```

La figura 23.2 muestra el resultado en un marco de aplicación que contiene un componente: un `PiePanel` creado con los datos anteriores.

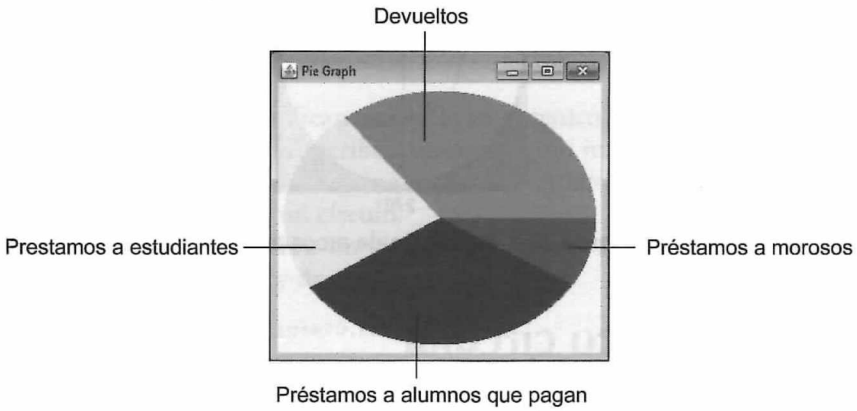


Figura 23.2. Datos de préstamos universitarios en un gráfico circular.

Al crear un objeto `PiePanel`, el número de porciones se especifica en el constructor. Debe conocer otros tres aspectos para poder dibujar cada porción:

- El color de la porción, representado por un objeto `Color`.
- El valor representado por cada porción.
- El valor total representado por todas las porciones.

Se usa una nueva clase de ayuda, `PieSlice`, para representar las porciones del gráfico circular:

```
import java.awt.*;

class PieSlice {
   Color color = Color.lightGray;
   float size = 0;

   PieSlice(Color pColor, float pSize) {
      color = pColor;
      size = pSize;
   }
}
```

Cada porción se crea invocando `PieSlice(Color, float)`. El valor combinado de todas las porciones se almacena como variable de instancia privada de la clase `PiePanel`, `totalSize`. También hay variables de instancia para el color de fondo del panel (`background`) y un contador usado para controlar las porciones (`current`):

```
private int current = 0;
private float totalSize = 0;
private Color background;
```

Una vez obtenida la clase `PieSlice` con la que trabajar, puede crear una matriz de objetos `PieSlice` con otra variable de instancia:

```
private PieSlice[] slice;
```

Al crear un objeto `PiePanel`, ninguna de las porciones tienen un color o un tamaño asignado. Lo único que debe hacer en el constructor es definir el tamaño de la matriz `slice` y guardar el color de fondo del panel:

```
public PiePanel(int sliceCount) {
   slice = new PieSlice[sliceCount];
   background = getBackground();
}
```

Use el método `addSlice(Color, float)` para añadir una porción del gráfico al panel:

```
public void addSlice(Color sColor, float sSize) {
   if (current <= slice.length) {
      slice[current] = new PieSlice(sColor, sSize);
      totalSize += sSize;
      current++;
   }
}
```

La variable de instancia `current` se usa para incluir cada porción en su propio elemento de la matriz `slice`. La variable `length` de una matriz contiene el número de elementos que la matriz debe contener; mientras que `current` no sea mayor que `slice.length`, puede seguir añadiendo porciones al panel. La clase `PiePanel` procesa todas las operaciones gráficas en su método `paintComponent()`, como es de esperar. Lo más complicado de esta tarea es dibujar los arcos que representan cada porción del gráfico, lo que se hace con las siguientes instrucciones:

```
float start = 0;
for (int i = 0; i < slice.length; i++) {
   float extent = slice[i].size * 360F / totalSize;
   comp2D.setColor(slice[i].color);
   Arc2D.Float drawSlice = new Arc2D.Float(
      xInset, yInset, width, height, start, extent,
      Arc2D.Float.PIE);
   start += extent;
   comp2D.fill(drawSlice);
}
```

La variable `start` controla dónde comenzar a dibujar el arco y `extent` el tamaño del arco. Si conoce el tamaño total de todas las porciones del gráfico y el tamaño de una en concreto, puede determinar `extent` si multiplica el tamaño del arco por 360 y lo divide por el total de todas las porciones.

Todos los arcos se dibujan en un bucle `for`: tras calcular `extent` en cada arco, se crea el arco y se añade `extent` a `start`. De este modo, cada porción comienza junto a la anterior. La invocación del método `fill()` de `Graphics2D` dibuja el arco.

Para combinar todos los elementos, cree un nuevo archivo vacío de Java con el nombr
`PiePanel` e introduzca el texto del listado 23.1.

Listado 23.1. Texto completo de PiePanel.java.

```
 1: import java.awt.*;
 2: import javax.swing.*;
 3: import java.awt.geom.*;
 4:
 5: public class PiePanel extends JPanel {
 6:   private PieSlice[] slice;
 7:   private int current = 0;
 8:   private float totalSize = 0;
 9:   private Color background;
10:
11:   public PiePanel(int sliceCount) {
12:      slice = new PieSlice[sliceCount];
13:      background = getBackground();
14:   }
15:
16:   public void addSlice(Color sColor, float sSize) {
17:      if (current <= slice.length) {
18:         slice[current] = new PieSlice(sColor, sSize);
19:         totalSize += sSize;
20:         current++;
21:      }
22:   }
23:
24:   public void paintComponent(Graphics comp) {
25:      super.paintComponent(comp);
26:      Graphics2D comp2D = (Graphics2D) comp;
27:      int width = getSize().width - 10;
28:      int height = getSize().height - 15;
29:      int xInset = 5;
30:      int yInset = 5;
31:      if (width < 5) {
32:         xInset = width;
33:      }
34:      if (height < 5) {
35:         yInset = height;
36:      }
37:      comp2D.setColor(background);
38:      comp2D.fillRect(0, 0, getSize().width, getSize().height);
39:      comp2D.setColor(Color.lightGray);
40:      Ellipse2D.Float pie = new Ellipse2D.Float(
41:         xInset, yInset, width, height);
42:      comp2D.fill(pie);
43:      float start = 0;
44:      for (int i = 0; i < slice.length; i++) {
45:         float extent = slice[i].size * 360F / totalSize;
46:         comp2D.setColor(slice[i].color);
47:         Arc2D.Float drawSlice = new Arc2D.Float(
48:            xInset, yInset, width, height, start, extent,
49:            Arc2D.Float.PIE);
50:         start += extent;
```

```
51:          comp2D.fill(drawSlice);
52:       }
53:    }
54: }
55:
56: class PieSlice {
57:    Color color = Color.lightGray;
58:    float size = 0;
59:
60:    PieSlice(Color pColor, float pSize) {
61:       color = pColor;
62:       size = pSize;
63:    }
64: }
```

El listado 23.1 define una clase `PiePanel` en las líneas 1-54 y una clase de ayuda `PieSlice` en las líneas 56-64. La clase `PiePanel` se puede usar como componente en cualquier IGU de un programa de Java. Para probar `PiePanel`, debe crear una clase que la utilice.

El listado 23.2 contiene una aplicación que usa estos paneles, `PieFrame`. Cree un nuevo archivo vacío de Java e introduzca el código del listado.

Listado 23.2. Texto completo de PieFrame.java

```
 1: import javax.swing.*;
 2: import javax.swing.event.*;
 3: import java.awt.*;
 4:
 5: public class PieFrame extends JFrame {
 6:    Color uneasyBeingGreen = new Color(0xCC, 0xCC, 0x99);
 7:    Color zuzusPetals = new Color(0xCC, 0x66, 0xFF);
 8:    Color zootSuit = new Color(0x66, 0x66, 0x99);
 9:    Color sweetHomeAvocado = new Color(0x66, 0x99, 0x66);
10:    Color shrinkingViolet = new Color(0x66, 0x66, 0x99);
11:    Color miamiNice = new Color(0x33, 0xFF, 0xFF);
12:    Color inBetweenGreen = new Color(0x00, 0x99, 0x66);
13:    Color norwegianBlue = new Color(0x33, 0xCC, 0xCC);
14:    Color purpleRain = new Color(0x66, 0x33, 0x99);
15:    Color freckle = new Color(0x99, 0x66, 0x33);
16:
17:    public PieFrame() {
18:       super("Pie Graph");
19:       setLookAndFeel();
20:       setSize(320, 290);
21:       setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
22:       setVisible(true);
23:
24:       PiePanel pie = new PiePanel(10);
25:       pie.addSlice(uneasyBeingGreen, 1337);
26:       pie.addSlice(zuzusPetals, 1189);
27:       pie.addSlice(zootSuit, 311);
28:       pie.addSlice(sweetHomeAvocado, 246);
29:       pie.addSlice(shrinkingViolet, 203);
30:       pie.addSlice(miamiNice, 187);
```

```
31:       pie.addSlice(inBetweenGreen, 166);
32:       pie.addSlice(norwegianBlue, 159);
33:       pie.addSlice(purpleRain, 139);
34:       pie.addSlice(freckle, 127);
35:       add(pie);
36:    }
37:
38:    private void setLookAndFeel() {
39:       try {
40:          UIManager.setLookAndFeel(
41:             "com.sun.java.swing.plaf.nimbus.NimbusLookAndFeel"
42:          );
43:       } catch (Exception exc) {
44:          // ignorar el error
45:       }
46:    }
47:
48:    public static void main(String[] arguments) {
49:       PieFrame pf = new PieFrame();
50:    }
51: }
```

La clase `PieFrame` es una sencilla interfaz gráfica de usuario que contiene un componente, un objeto `PiePanel` creado en la línea 24. El método `addSlice()` del objeto se invoca 10 veces en las líneas 25-35 para añadir porciones al gráfico circular.

Al ejecutar la aplicación, `PieFrame` muestra un gráfico circular con la población de los 10 países más poblados del mundo (en millones de habitantes), de acuerdo a un informe de 2011 de la base de datos del censo internacional estadounidense. En orden son: China (1.337 millones), India (1.189 millones), Estados Unidos (311 millones), Indonesia (246 millones), Brasil (203 millones), Pakistán (187 millones), Nigeria (166 millones), Bangladesh (159 millones), Rusia (139 millones) y Japón (127 millones).

Como Java solo usa algunos de los colores definidos en la clase `Color`, se crean 10 nuevos y se les asignan nombres descriptivos. Los colores se expresan como valores hexadecimales, precedidos de `0x`, pero también se pueden especificar como valores decimales en cada constructor `Color()`. En la figura 23.3 puede ver la aplicación en ejecución.

Nota

Puede consultar el censo mundial actual en `www.cadenhead.org/census`.

Resumen

Por medio de fuentes, colores y gráficos puede atraer la atención a los elementos de sus programas y hacer que resulten más atractivos para sus usuarios.

Dibujar con las formas disponibles en Java puede parecer más complicado de lo que realmente es. Sin embargo, los gráficos trazados con polígonos tienen dos ventajas con respecto a los cargados desde archivos de imagen:

- **Velocidad:** Incluso un pequeño gráfico, como un icono, tarda más en cargarse y mostrarse que una serie de polígonos.
- **Escala:** Puede cambiar el tamaño de una imagen que use polígonos si modifica los valores usados para crearla. Por ejemplo, puede añadir una función a la clase `Sign` que multiplique por dos todos los puntos (x,y) de las formas antes de crearlas, lo que genera una imagen del doble de tamaño. Las imágenes con polígonos se escalan más rápidamente que los archivos de imagen y generan mejores resultados.

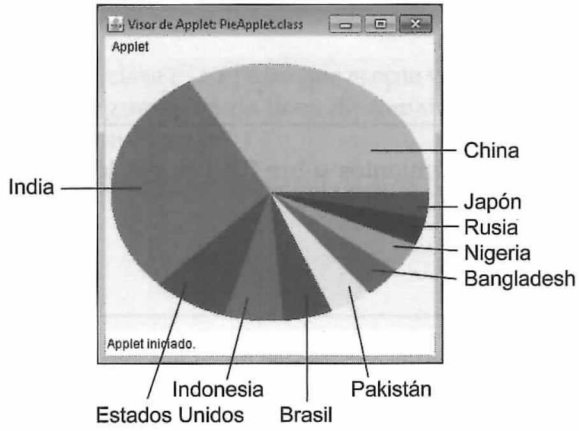


Figura 23.3. Datos de población en un gráfico circular.

Preguntas y respuestas

P: ¿Cómo puedo trazar arcos en el sentido de las agujas del reloj?

R: Puede especificar el tamaño del arco como número negativo. El arco comienza en el mismo punto, pero se desplaza en la dirección contraria en un trazado elíptico. Por ejemplo, la siguiente instrucción traza un archivo abierto en la posición (35,20), 90 grados de largo, que comienza en la marca de los 0 grados, en sentido de las agujas del reloj, con una altura de 20 y una anchura de 15:

```
Arc2D.Float smile = new Arc2D.Float(35F, 20F, 15F, 20F,
   0F, -90F, Arc2D.Float.OPEN);
```

P: Las elipses y los círculos no tienen esquinas. ¿Qué son las coordenadas (x,y) especificadas en el constructor `Ellipses.Float`?

R: Estas coordenadas representan los valores x e y más pequeños del óvalo o círculo. Si trazara un rectángulo invisible a su alrededor, la esquina superior izquierda del rectángulo sería las coordenadas x e y usadas como argumentos del método.

P: ¿Cómo puedo usar XRender con Java?

R: Java 7 permite dibujar gráficos Java2D con el motor de renderización XRender en entornos basados en X11, generalmente en Linux. Esta funcionalidad está desactivada de forma predeterminada y debe activarse mediante la opción de línea de comandos `Dsun.java2d.xrender=true`. XRender permite a los programas de Java usar las prestaciones de modernas unidades de procesamiento de gráficos (GPU). En NetBeans puede establecer estas opciones si selecciona Ejecutar>Establecer la configuración del proyecto>Personalizar. Use el campo Opciones de la máquina virtual para establecer esta opción y pulse **Aceptar**.

Ejercicios

Compruebe sus conocimientos sobre fuentes y colores por medio de las siguientes preguntas.

Preguntas

1. ¿Cuál de las siguientes constantes no se usa para seleccionar un color?
   A. `Color.cyan`
   B. `Color.teal`
   C. `Color.magenta`
2. Al cambiar el color de un objeto y redibujarlo en un contenedor, ¿qué debe hacer para que sea visible?
   A. Usar el método `drawColor()`.
   B. Usar la instrucción `repaint()`.
   C. Nada.
3. ¿Qué significan las siglas RGB?
   A. Roy G. Biv.
   B. Rojo, verde y azul
   C. Lucy in the Sky with Diamonds.

Respuestas

1. B. El color principal de los Jacksonville Jaguars, `teal`, no se representa en `Color`.

2. B. Al invocar `repaint()` es necesario invocar manualmente el método `paintComponent()`.
3. B. Si C fuera la respuesta correcta, podría usar colores que solo fueran visibles años después durante un *flashback*.

Actividades

Para explorar el espectro completo de posibilidades al usar fuentes y colores en sus programas, realice las siguientes actividades:

- Cree una versión de la clase `PieFrame` que acepte valores de color y de porciones de un gráfico como argumentos de línea de comandos en lugar de incluirlos en el código fuente de la aplicación.
- Cree una aplicación que dibuje una señal de stop con colores, formas y fuentes.

24
Crear aplicaciones de Android

Java es un lenguaje de programación de propósito general que se puede ejecutar en distintas plataformas. Una de ellas se ha convertido en los últimos cuatro años en la gran impulsora del desarrollo con Java. El sistema operativo Android, que nació para teléfonos móviles y se ha instaurado en otros muchos dispositivos, ejecuta exclusivamente programas escritos en Java. Estos programas, denominados *apps* (aplicaciones), se crean sobre una plataforma móvil de código abierto que ofrece total libertad a los programadores. Cualquiera puede crear, implementar y comercializar aplicaciones Android.

En este capítulo analizaremos Android, sus características y por qué miles de programadores usan esta plataforma de desarrollo. También crearemos una aplicación y la ejecutaremos en un teléfono Android (si lo tiene) y en un emulador (si no lo tiene).

Introducción a Android

Google lanzó Android en 2007, dos años después de adquirir la tecnología, como parte de un esfuerzo por establecer una nueva plataforma móvil no propietaria y abierta, lo contrario a la tecnología en la que se basan BlackBerry de RIM y el iPhone de Apple. Algunas de las principales empresas de telefonía móvil y tecnología, como Google, Intel, Motorola, Nvidia o Samsung, formaron la *Open Handset Alliance* (Alianza de dispositivos abiertos) para promover la nueva plataforma en beneficio propio. Google publicó el *Android Software Development Kit* (Kit de desarrollo de software de Android o SDK), un conjunto gratuito de herramientas para desarrollar aplicaciones de Android. El primer teléfono en ejecutar Android, el modelo T-Mobile G1, apareció en junio de 2008.

Este esfuerzo tuvo unos inicios lentos, pero desde principios de 2010 ha explotado y se ha convertido en el verdadero rival de iPhone y otras plataformas móviles. Los principales operadores telefónicos ofrecen dispositivos Android. También existe un nuevo mercado para tablet y lectores de libros electrónicos.

Antes de la aparición de Android, el desarrollo de aplicaciones móviles requería costosas herramientas de programación y programas de desarrollo. El mercado de la telefonía controlaba quién podía crear aplicaciones y si éstas se podían vender a los usuarios. Android acaba con esas restricciones.

La naturaleza no propietaria y de código abierto de Android implica que cualquiera puede desarrollar, publicar y vender aplicaciones. El único coste es la cuota para enviar aplicaciones al mercado de Google. Todo lo demás es gratuito.

Para descargar el SDK de Android y aprender a crear programas para la plataforma, visite el sitio Android Developer (`http://developer.android.com`). Lo consultará continuamente mientras crea sus propias aplicaciones ya que documenta todas las clases de la biblioteca de clases de Android y ofrece una completa referencia *online*.

La creación de aplicaciones de Android es muy sencilla si usa un entorno de desarrollo integrado (IDE) compatible con el SDK de Android. El más conocido es Eclipse, también gratuito y de código abierto. El complemento Android para Eclipse permite que el SDK se integre perfectamente con el IDE.

Puede usar Eclipse para crear aplicaciones de Android, probarlas en un emulador que actúa como teléfono Android e incluso implementarlas. Desde su aparición, el lenguaje Java se ha utilizado para crear programas que ejecutar en ordenadores de escritorio, servidores Web o navegadores Web.

Android usa Java en todas partes. Sus programas se pueden implementar en millones de teléfonos y otros dispositivos móviles. Esto materializa el objetivo de diseño original de Java cuando James Gosling inventó el lenguaje mientras trabajaba en Sun Microsystems, a mediados de los 90. Sun quería un lenguaje que pudiera ejecutarse en cualquier dispositivo, como teléfonos, tarjetas inteligentes, etc. Los programadores de Java abandonaron esos sueños cuando el lenguaje se convirtió en un medio para ejecutar programas Web interactivos para después pasar a ser un lenguaje de propósito general.

15 años después, la plataforma Android aloja miles de millones de programas de Java por todo el mundo, de acuerdo a un informe del sector.

Android tiene el potencial de ser el área de programación de Java más duradera, y lucrativa, de los próximos años. Puede que también sea la más divertida.

Crear una aplicación de Android

Las aplicaciones de Android son programas convencionales de Java que usan una estructura de aplicación, un conjunto básico de clases y archivos que todas las aplicaciones comparten. La estructura se rige por una serie de reglas que determinan la estructura de las aplicaciones para que se ejecuten correctamente en dispositivos Android. Para empezar a crear aplicaciones, debe instalar y configurar el SDK de Android, el IDE Eclipse y el complemento Android para Eclipse.

Si es su primera toma de contacto con Android, en el apéndice C se indica cómo obtener y configurar estas herramientas. Adelante. Mientras espero, me pondré al día con algunos amigos de Facebook. ¿Ya? Perfecto.

El primer proyecto es la aplicación SalutonMondo, un sencillo programa que muestra una línea de texto en la pantalla de un dispositivo Android.

1. Ejecute el IDE Eclipse, con un aspecto y funcionamiento similares a NetBeans.
2. Seleccione Archivo>Nuevo>Android Project (Proyecto Android). Se abrirá el asistente New Android Project (Nuevo proyecto Android).
3. En el campo Project Name (Nombre del proyecto), introduzca **SalutonMondo**.
4. Marque la opción Create new project in workspace (Crear nuevo proyecto en espacio de trabajo).
5. La casilla de verificación Use default location (Usar ubicación predeterminada) determina dónde almacenar el proyecto. Si usa la ubicación predeterminada, deje marcada la casilla. En caso contrario, anule su selección, pulse **Browse** (Examinar) y seleccione una carpeta para almacenar el proyecto. Pulse **Siguiente**.
6. Todos los proyectos de Android requieren un destino de generación. Este destino representa la versión más antigua de Android que puede ejecutar su aplicación. Como cada versión de Android ofrece nuevas funciones, su elección determinará las opciones que puede usar. En una sencilla aplicación como ésta, un destino antiguo es válido. Seleccione Android 2.2. Pulse **Siguiente**.
7. En el campo Application name (Nombre de la aplicación), asigne a la aplicación el nombre **Saluton Mondo!**; es el que se mostrará en los dispositivos Android.
8. El campo Package name (Nombre del paquete) debe indicar el nombre del paquete de Java al que pertenecen las clases de esta aplicación. Introduzca **org.cadenhead.android**.
9. La casilla de verificación Create Activity (Crear actividad) indica si la nueva aplicación se creará con una clase `Activity`. Una actividad es una tarea que la aplicación puede realizar. Mantenga marcada la casilla e introduzca **SalutonActivity** en el campo contiguo.
10. Pulse **Finalizar**. Se crea la nueva aplicación y el elemento SalutonMondo aparece en el panel Explorador de paquetes. En la figura 24.1 puede ver los tres pasos del proceso de creación de un nuevo archivo de Android.

Explorar un nuevo proyecto de Android

Un nuevo proyecto de Android consta de unos 20 archivos y carpetas que siempre se organizan de la misma forma en la aplicación. Puede haber más archivos en función de las prestaciones de la aplicación, pero estos archivos y carpetas iniciales siempre deben estar presentes.

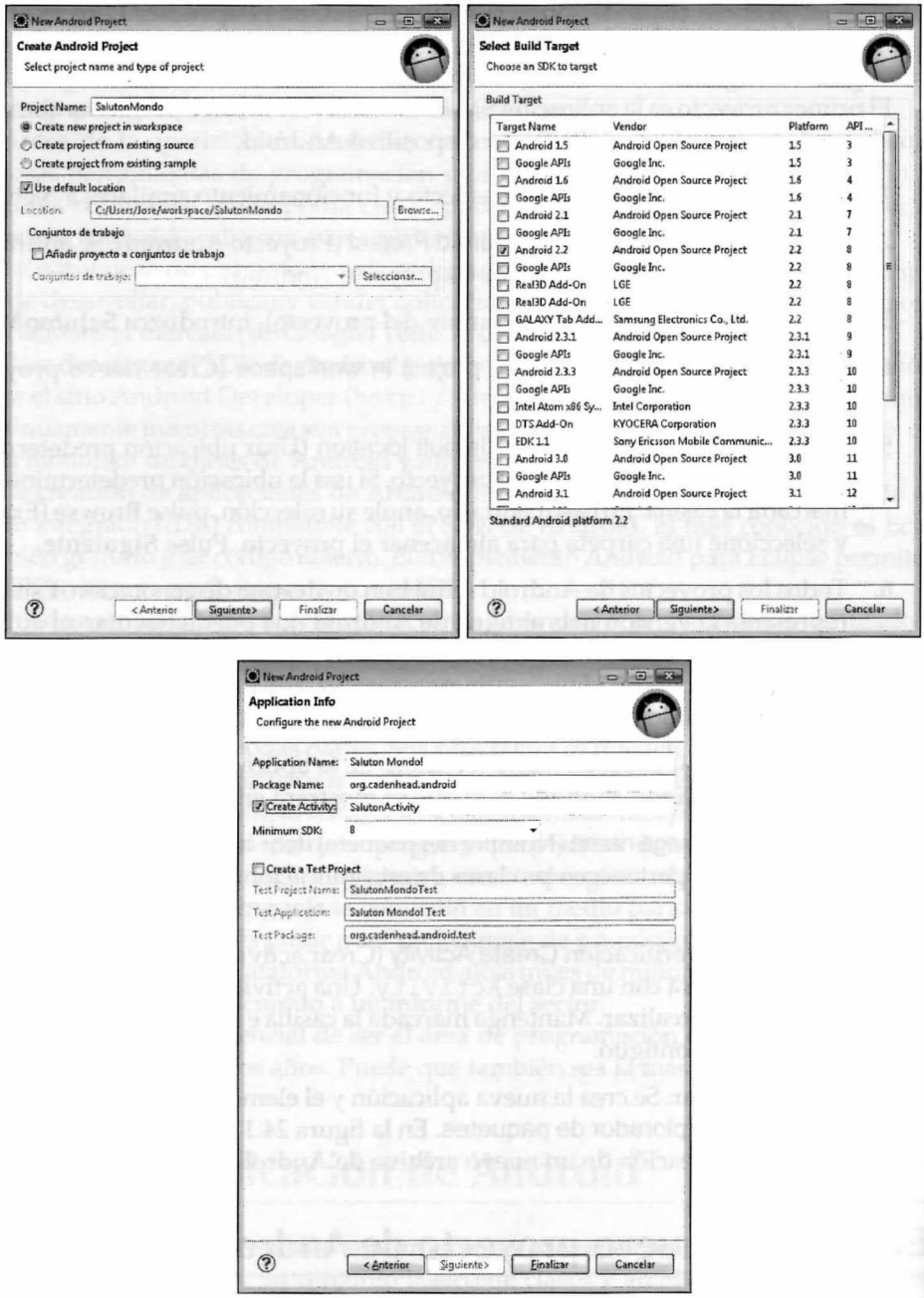


Figura 24.1. Creación de un nuevo proyecto de Android en Eclipse.

En la figura 24.2 puede ver el Explorador de paquetes de Eclipse tras crear un nuevo proyecto.

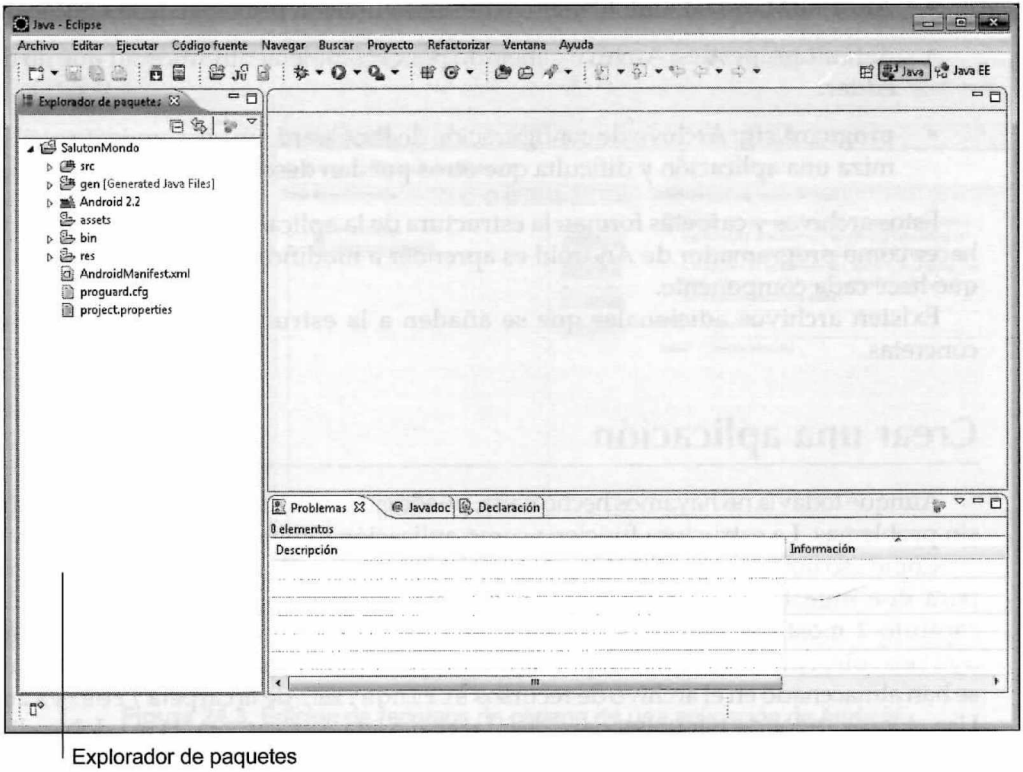


Figura 24.2. Las partes de un proyecto de Android.

Puede usar la carpeta para examinar la estructura de archivos y carpetas del proyecto. La nueva aplicación `SalutonMondo` comienza con los siguientes componentes:

- **Carpeta /src:** La carpeta raíz del código fuente Java de la aplicación.
- **/src/org.cadenhead.android/SalutonActivity.java:** Clase de la actividad que se inicia de forma predeterminada al ejecutar la aplicación.
- **Carpeta /gen:** Carpeta del código fuente de Java generado que no se edita manualmente.
- **/gen/org.cadenhead.android/R.java:** Código fuente de administración de recursos, generado automáticamente, para la aplicación. No debe editarlo nunca.
- **/assets:** Carpeta para recursos de archivo que no se compilan en la aplicación.
- **/res:** Carpeta de recursos de la aplicación como cadenas, números, archivos de diseño, gráficos y animaciones. Existen subcarpetas para tipos de recursos concreto: `layout`, `values`, `drawable-hdpi`, `drawable-ldpi` y `drawable-mdpi`. Estas carpetas contienen cinco archivos de recursos: tres versiones de `ic_launcher.png`, `main.xml` y `strings.xml`.

- **AndroidManifest.xml:** El archivo de configuración principal de la aplicación.
- **default.properties:** Archivo generado por el complemento Android que no debe editar.
- **proguard.cfg:** Archivo de configuración de ProGuard, una herramienta que optimiza una aplicación y dificulta que otros puedan descompilar el código fuente.

Estos archivos y carpetas forman la estructura de la aplicación, Lo primero que debe hacer como programador de Android es aprender a modificar la estructura para saber qué hace cada componente.

Existen archivos adicionales que se añaden a la estructura para realizar tareas concretas.

Crear una aplicación

Aunque todavía no hayamos hecho nada, puede ejecutar el nuevo proyecto de Android sin problemas. La estructura funciona como apücación totalmente operativa.

Como eso no tiene atractivo alguno, puede personalizar la aplicación SalutonMondo para que muestre el tradicional saludo de programación **Saluton Mondo!.** En el capítulo 2 mostramos este texto como cadena mediante la invocación del método System.out.println().Las aplicaciones de Android muestran cadenas que primero se han almacenado en el archivo de recursos strings.xml de la carpeta /res/values. Use el Explorador de paquetes para desplazarse hasta esta carpeta. Haga doble clic en strings.xml. Se abrirá un editor de recursos (véase la figura 24.3).

Las cadenas y otros recursos cuentan con un nombre y un valor, al igual que las variables de Java. El panel de recursos muestra dos recursos de cadena: helio y app_name. Los nombres de recursos se rigen por tres reglas:

- Deben estar en minúscula.
- No deben incluir espacios.
- Solo pueden usar el guión bajo (_) como signo de puntuación.

Haga clic en una cadena del panel de recursos. Se muestran los campos de texto Nare (Nombre) y Valué (Valor) junto con consejos para editar cadenas (véase la figura 24.3

El recurso de cadena app_name lo seleccionamos al ejecutar el asistente New Androc Project. El nombre debe coincidir con el asignado antes pero puede modificarlo cu desee si cambia esta cadena.

La cadena helio contiene el texto que se va a mostrar en la pantalla principal única) de la aplicación al ejecutarla. Haga clic en el nombre de esta cadena para edi En el campo **Valué,** introduzca **Saluton Mondo!.**

Los recursos se almacenan en archivos XML. El editor de recursos es un sen editor XML. También puede editar directamente el código XML. Haga clic en la strings.xml de la parte inferior del editor para abrir el archivo y editarlo direct (puede ver esta etiqueta en la figura 24.3).

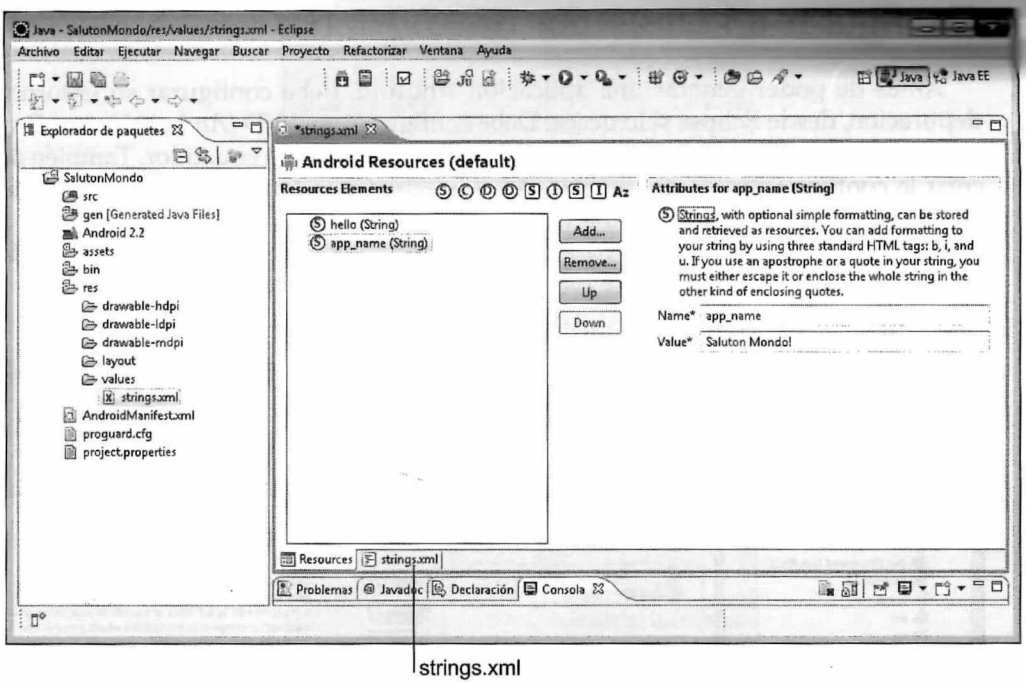


Figura 24.3. Edición de recursos de cadena de una aplicación de Android.

`strings.xml` tiene este aspecto:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <string name="hello">Saluton Mondo!</string>
   <string name="app_name">Saluton Mondo!</string>
</resources>
```

Este editor permite cambiar cualquier elemento del archivo XML, incluso las etiquetas de marcado. El elemento `string` contiene un atributo `name` que identifica el nombre del recurso. El valor se incluye entre etiquetas como datos de caracteres.

Para volver al editor de recursos, haga clic en la ficha Resources (Recursos). Haga clic en el botón **Guardar** de la barra de herramientas de Eclipse para guardar el cambio en el archivo `strings.xml`. Con este cambio, ya casi podemos ejecutar la aplicación.

Advertencia

Aunque pueda editar directamente el código XML, no lo haga. No es necesario al crear recursos para una aplicación de Android. La excepción es cuando el editor de Eclipse no admite algo que desea definir en un recurso. No es el caso de las cadenas, por lo que es recomendable limitarse al editor de recursos. Cometerá más errores si edita directamente los archivos XML.

Configurar un emulador de Android

Antes de poder generar una aplicación Android, debe configurar su entorno de depuración, desde Eclipse si lo desea. Debe configurar un AVD (*Android Virtual Device*, Dispositivo virtual de Android) que ejecute la aplicación como emulador. También debe crear la configuración de depuración del proyecto. Cuando termine, podrá generar la aplicación y ejecutarla en el emulador.

Para configurar un dispositivo virtual de Android, haga clic en el icono verde de la barra de herramientas de Eclipse (véase la figura 24.4).

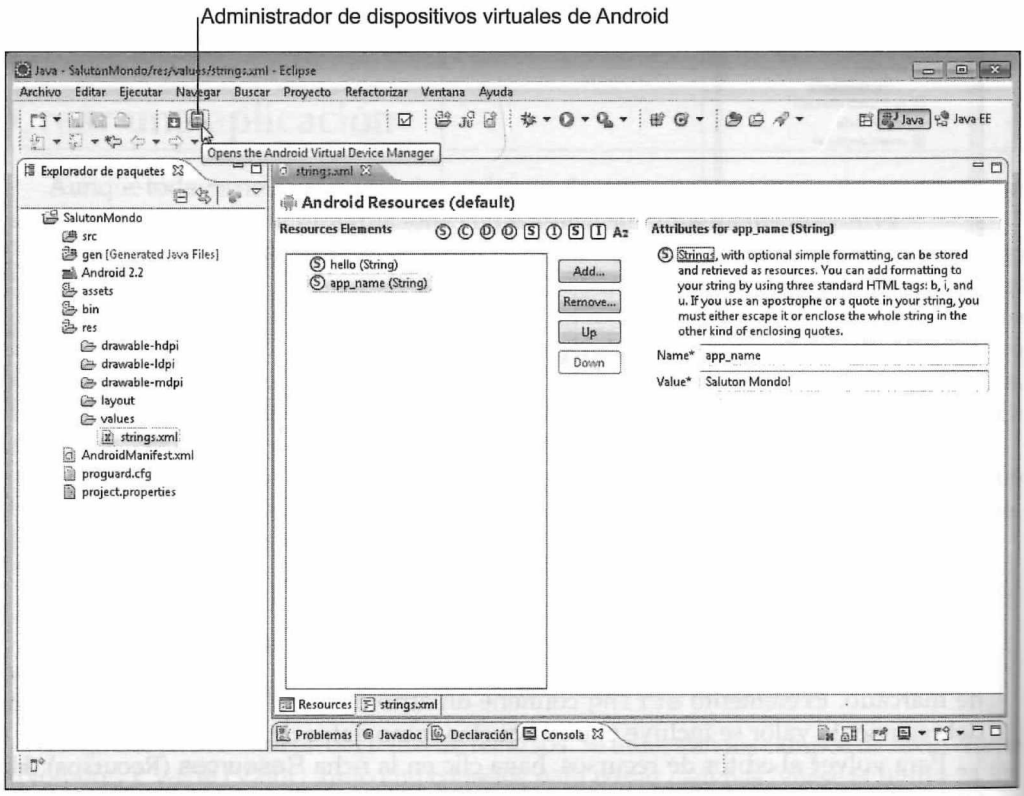


Figura 24.4. Configuración de un dispositivo virtual de Android.

Se abre la ventana Android Virtual Device Manager (Administrador de dispositivos virtuales de Android), una de las herramientas del SDK de Android. Los emuladores creados se enumeran a la derecha. En la figura 24.5 puede ver el administrador.

Para añadir un nuevo emulador, pulse **New** (Nuevo) y siga estos pasos:

1. En el campo Name (Nombre) introduzca **SimpleAVD**.
2. En el campo Target (Destino) seleccione una versión de Android, como Android 2.2 - API Level 8.

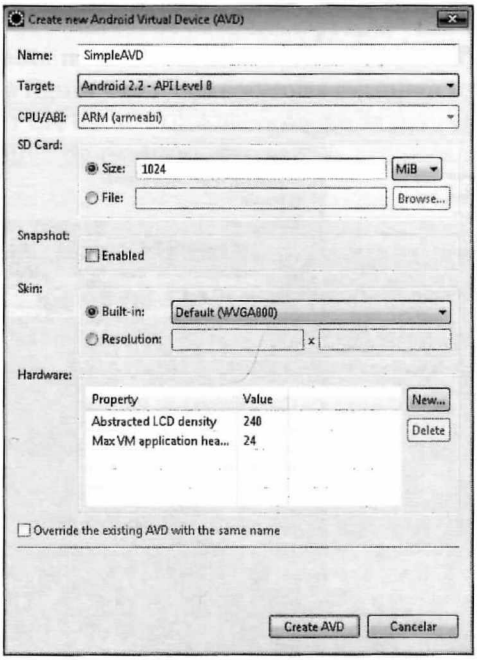


Figura 24.5. Creación de un nuevo emulador de Android.

3. En el campo Size (Tamaño), seleccione un tamaño para la tarjeta SD falsa. Introduzca **1024** y seleccione MiB en el menú desplegable, para una tarjeta SD de 1024MB. Debe tener este espacio disponible en su equipo, de modo que puede elegir otro valor si lo desea. El tamaño mínimo es de 9MB.
4. Pulse **Create AVD** (Crear DVA). Se creará el nuevo emulador, proceso que puede tardar un minuto.

Puede crear todos los emuladores que necesite y personalizarlos para distintas versiones de Android y diferentes tipos de pantallas. Cierre el administrador para volver a la interfaz principal de Eclipse.

Crear una configuración de depuración

El último paso necesario antes de poder iniciar la aplicación SalutonMondo consiste en crear una configuración de depuración en Eclipse. Siga los pasos:

1. Seleccione Ejecutar>Configuraciones de depuración para abrir la ventana Configuraciones de depuración.
2. En el panel izquierdo, haga doble clic en Android Application (Aplicación Android) (véase la figura 24.6). Se añadirá la entrada Configuración_nueva. El panel de la derecha muestra opciones de configuración para la nueva opción.

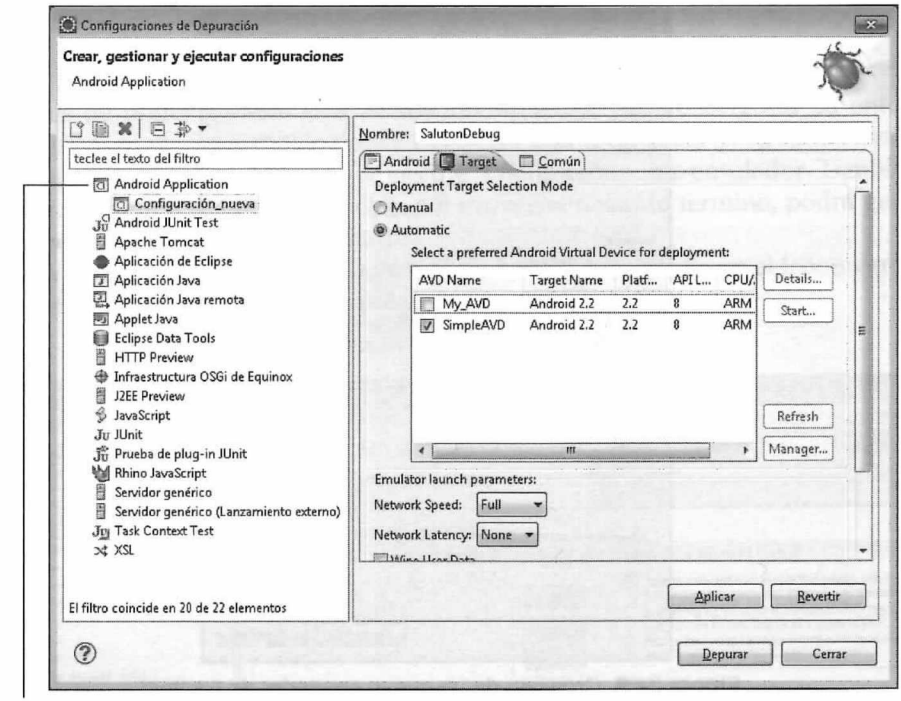


Figura 24.6. Creación de una configuración de depuración de Android.

3. En el panel derecho, en el campo Name, introduzca **SalutonDebug**.
4. Pulse el botón **Browse** (Examinar) para abrir el cuadro de diálogo Project Selection (Seleccionar proyecto):
5. Seleccione el proyecto SalutonMondo y pulse **Aceptar**.
6. Haga clic en la ficha Target (Destino)
7. En Deployment Target Selection Mode (Modo de selección de destino de implementación), seleccione Automatic (Automático). Podrá seleccionar un DVA de destino en una tabla.
8. En la tabla, marque la casilla de verificación del emulador SimpleAVD.
9. Pulse **Aplicar** para guardar los cambios y después pulse **Cerrar**.

Ejecutar la aplicación

Una vez creado el emulador de Android y una configuración de depuración, ya puede ejecutar su primera aplicación. Haga clic en `SalutonMondo`, el elemento superior del Explorador de paquetes, y pulse el icono con forma de insecto de la barra de herramientas

de Eclipse. El emulador de Android se abre en su propia ventana, lo que puede tardar varios minutos. El emulador muestra **Saluton Mondo!** como texto y en la barra de título de la aplicación (véase la figura 24.7). Los controles permiten usarlo como un teléfono, pero con el ratón el lugar del dedo. Pulse el botón **Atrás** para cerrar la aplicación y ver la emulación del dispositivo de Android.

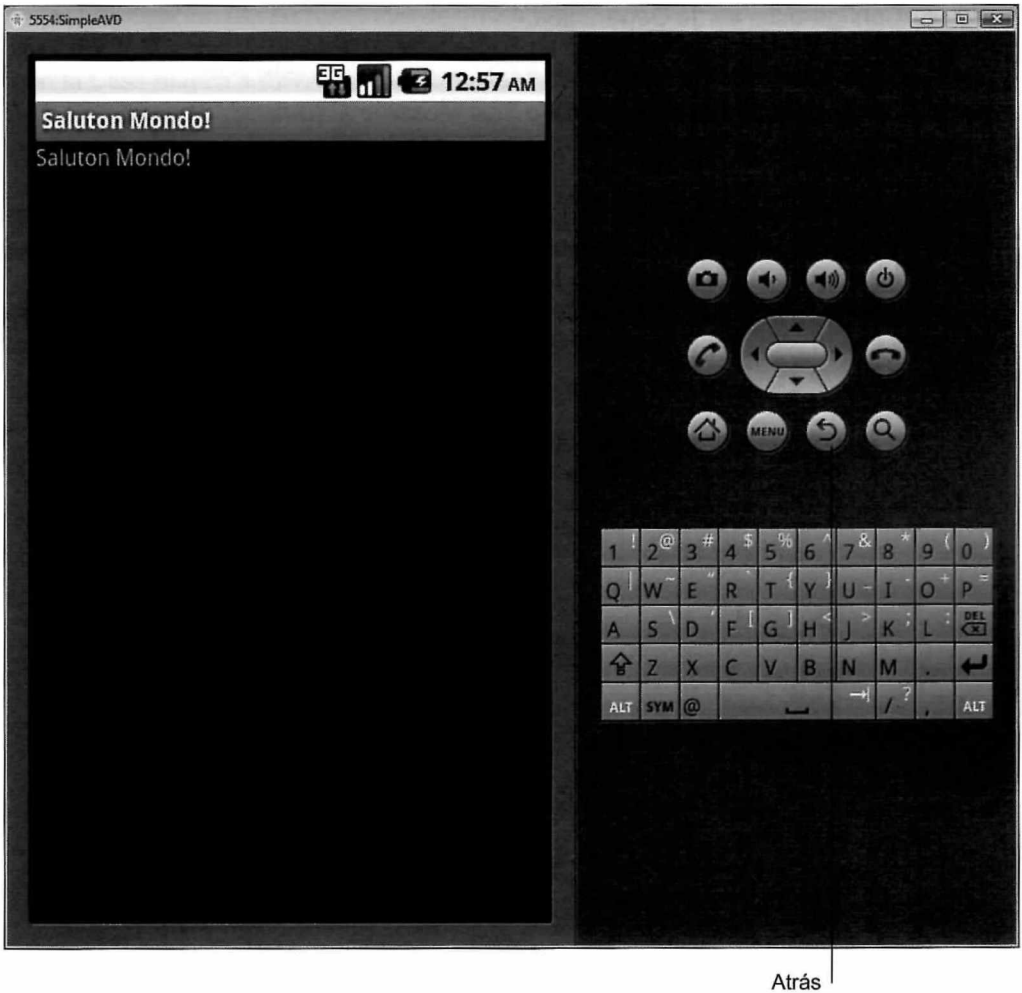


Figura 24.7. Ejecución de una aplicación en el emulador de Android.

Un emulador puede hacer muchas de las cosas que hace un dispositivo real, como conectarse a Internet si el ordenador dispone de una conexión activa. También puede recibir llamadas falsas y mensajes SMS.

Como no es un dispositivo totalmente funcional, las aplicaciones que desarrolle deben probarse en teléfonos y tablet Android reales.

Si puede conectar un teléfono Android (u otro dispositivo) a su ordenador mediante un cable USB, podrá ejecutar la aplicación si establece el teléfono en modo de depuración. Las aplicaciones desarrolladas con el SDK de Android solo se pueden implementar en un teléfono de esta forma. En el teléfono, debe seleccionar una opción de depuración USB. Tras ello, en Eclipse, siga los pasos descritos a continuación:

1. Seleccione Ejecutar>Configuraciones de Depuración para abrir la ventana Configuraciones de Depuración.
2. Haga clic en la ficha Target del panel derecho para activarla.
3. Cambie Deployment Target Selection Mode de Automatic a Manual.
4. Pulse **Aplicar** y **Cerrar**.

Conecte su teléfono Android con el cable USB. El icono de Android aparecerá en la barra superior de la pantalla. Si arrastra esta barra hacia abajo verá un mensaje que indica que la depuración USB está conectada.

En Eclipse, haga clic en el icono con forma de insecto de la barra de herramientas. Se abrirá el cuadro de diálogo Android Device Chooser (Selector de dispositivos Android) (véase la figura 24.8).

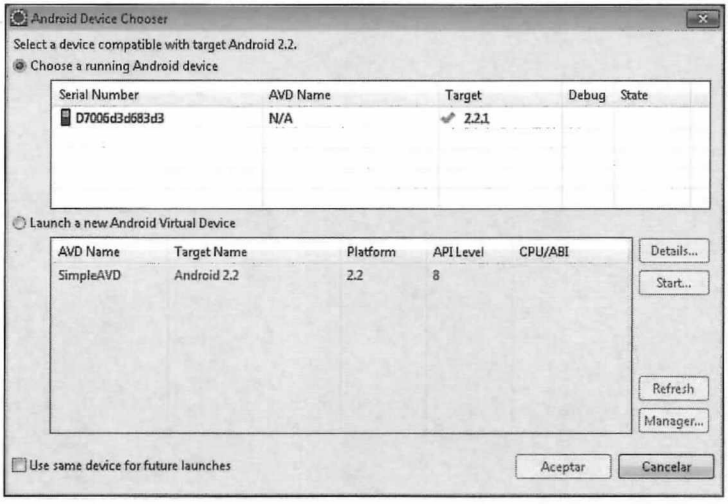


Figura 24.8. Implementación de una aplicación en un teléfono Android.

Si se ha detectado un teléfono Android, aparecerá en la parte superior, bajo la opción Choose a running Android device (Seleccionar dispositivo en ejecución).

Seleccione esta opción, haga clic en el dispositivo y pulse **Aceptar**. La aplicación se ejecutará en el teléfono como sucedió en el emulador.

Al igual que el primer programa de Java que creamos en el capítulo 2, la primera aplicación creada en Android es excepcionalmente simple. El siguiente proyecto es más ambicioso.

Diseñar una aplicación real

Las aplicaciones Android pueden aprovechar al máximo todas las funciones del dispositivo, como la mensajería SMS, los servicios basados en la ubicación o la entrada táctil. En el último proyecto de programación del libro crearemos la aplicación Take Me To Your Leader.

Esta aplicación usa las funciones de un teléfono Android para realizar llamadas, visitar un sitio Web y cargar una ubicación en Google Maps. La aplicación le pone en contacto con la Casa Blanca a través del teléfono, la Web y un mapa (si el presidente de Estados Unidos no es su líder, la aplicación se puede personalizar).

Para empezar debe crear un nuevo proyecto en Eclipse:

1. Seleccione Archivo>Nuevo>Android Project para abrir el asistente New Android Project (véase la figura 24.9)

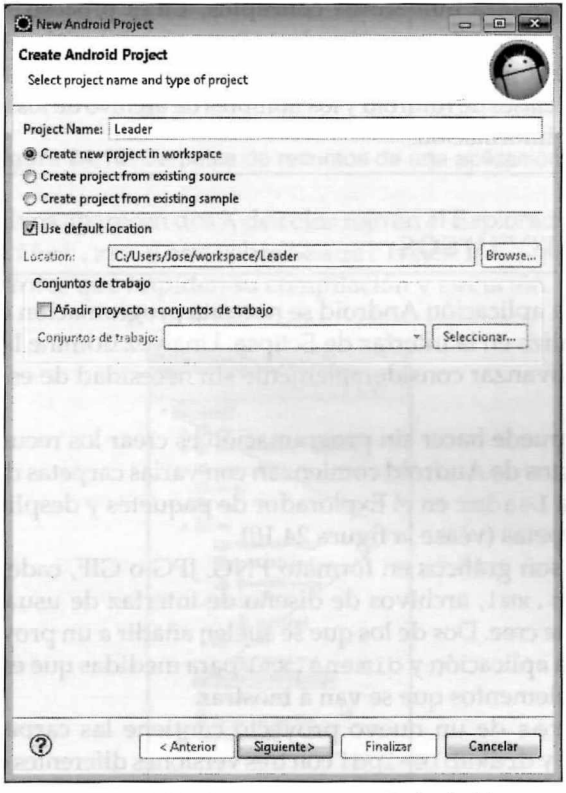


Figura 24.9. Nuevo proyecto de Android.

2. En el campo Project Name, introduzca **Leader**.
3. Seleccione la opción Create new project in workspace. Pulse **Siguiente**.

4. Seleccione la opción Build Target Android 2.2. Pulse **Siguiente**.
5. En el campo Application Name, introduzca **Take Me To Your Leader**.
6. En el campo Package Name, introduzca **org.cadenhead.android**.
7. Seleccione Create Activity e introduzca LeaderActivity en el campo contiguo.
8. Pulse **Finalizar**.

El proyecto aparece en el Explorador de paquetes, como sucedía con SalutonMondo. Para evitar confusiones, cierre SalutonMondo antes de continuar. Haga clic con el botón derecho del ratón sobre SalutonMondo en el Explorador de paquetes y seleccione Cerrar proyecto en el menú contextual.

Nota

Este proyecto analiza numerosos conceptos. En el proceso, puede tener un navegador abierto por la sección de referencia del sitio Android Developer (`http://developer.android.com/reference`). Puede buscar clases de Java en la biblioteca de clases de Android y los nombres de archivo de los archivos del proyecto si necesita más información.

Organizar recursos

Para crear una aplicación Android se necesita programación de Java pero gran parte del trabajo se realiza en la interfaz de Eclipse. Una vez domine las opciones del SDK de Android, podrá avanzar considerablemente sin necesidad de escribir una sola línea de código de Java.

Algo que no puede hacer sin programación es crear los recursos para la aplicación. Todos los proyectos de Android comienzan con varias carpetas de recursos. Para verlas, amplíe la carpeta `Leader` en el Explorador de paquetes y despliegue la carpeta `/res` y todas sus subcarpetas (véase la figura 24.10).

Los recursos son gráficos en formato PNG, JPG o GIF, cadenas almacenadas en el archivo `strings.xml`, archivos de diseño de interfaz de usuario en formato XML y otros archivos que cree. Dos de los que se suelen añadir a un proyecto son `colors.xml` para colores de la aplicación y `dimens.xml` para medidas que establecen el tamaño del texto y de otros elementos que se van a mostrar.

La carpeta `/res` de un nuevo proyecto contiene las carpetas `drawable-hdpi`, `drawable-mdpi` y `drawable-lpdi` con tres versiones diferentes de `ic_launcher.png`, el icono de la aplicación, el pequeño gráfico usado para iniciarla.

Las tres versiones de `ic_launcher.png` son el mismo gráfico con distinto tamaño para pantallas de resolución alta, media y baja. No usaremos estos iconos, por lo que puede borrarlos. Seleccione uno de los archivos `ic_launcher.png` en el Explorador de paquetes y pulse **Supr**. Se le pedirá que confirme la operación.

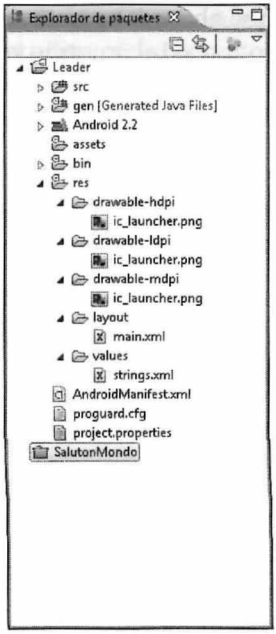


Figura 24.10. Carpetas de recursos de una aplicación.

Al borrar los archivos aparecen dos X de color rojo en el Explorador de paquetes: una sobre `AndroidManifest.xml` y otra sobre `Leader` (véase la figura 24.11). Indican que la aplicación tiene errores que impiden su compilación y ejecución.

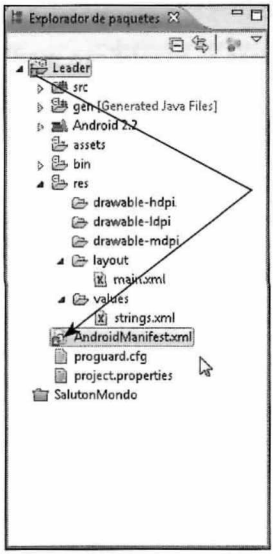


Figura 24.11. Detección y corrección de errores en la aplicación.

Los errores aparecen porque ahora falta un icono en la aplicación. Añadiremos un nuevo archivo gráfico (`appicon.png`) al proyecto y lo designaremos como su icono en el archivo `AndroidManifest.xml`, el archivo de configuración principal de la aplicación. En el sitio Web del libro encontrará `appicon.png` y otros cuatro archivos gráficos para la aplicación: `browser.png`, `maps.png`, `phone.png` y `whitehouse.png`. Descargue los cinco y guárdelos en una carpeta temporal de su equipo. La compatibilidad de Android con varias resoluciones es muy útil pero en este caso no la necesitamos. En lugar de usar las carpetas `drawable` existentes, crearemos una nueva:

1. En el Explorador de paquetes, haga clic en la carpeta `/res` para seleccionarla.
2. Seleccione Archivo>Nuevo>Carpeta para abrir el cuadro de diálogo Nueva Carpeta.
3. Introduzca **drawable** en el campo Nombre de la carpeta.
4. Pulse **Finalizar**.

Se creará una nueva carpeta `drawable` dentro de `/res`. Todos los gráficos necesarios para la aplicación se pueden almacenar aquí independientemente de su resolución.

Puede añadir archivos a recursos si los arrastra y suelta. Abra la carpeta temporal con los cinco archivos, selecciónelos y arrástrelos hasta la carpeta `drawable` en el Explorador de paquetes.

Ahora que el proyecto tiene un nuevo icono, puede establecerlo como icono de la aplicación y deshacerse de los errores indicados en el Explorador de paquetes. Para ello, editaremos `AndroidManifest.xml`.

Advertencia

En una aplicación, los recursos se identifican mediante un ID creado con su nombre de archivo sin la extensión. `appicon.png` tiene el ID `appicon`, `browser.png` el ID `browser`, etc. No puede haber dos recursos con el mismo ID (a excepción del mismo gráfico almacenado a distintas resoluciones en las tres carpetas `drawable-*dpi`, ya que cuentan como un mismo recurso).

Si dos recursos tienen el mismo nombre sin la extensión, como `appicon.png` y `appicon.gif`, Eclipse detecta el error y la aplicación no se compila. Los recursos también deben tener nombres que solo incluyan letras minúsculas, números, guiones bajos y puntos. Los archivos de este proyecto cumplen estas reglas.

Configurar el archivo de manifiesto de la aplicación

La principal herramienta de configuración de una aplicación Android es el archivo `AndroidManifest.xml` de la carpeta principal de la aplicación. Todos los archivos XML utilizados por una aplicación se pueden editar manualmente o por medio del editor de

Eclipse, más sencillo y menos proclive a errores. A menos que tenga gran experiencia en la edición de XML, debe usar el editor. Para seleccionar el icono correcto para la aplicación, siga estos pasos:

1. Haga doble clic en `AndroidManifest.xml` en el Explorador de paquetes. El archivo se abrirá para edición en la ventana principal de Eclipse con el editor integrado.
2. En la parte inferior del editor hay varias fichas. Haga clic en la ficha Application (Aplicación) para ver los parámetros relacionados con la aplicación (véase la figura 24.12).

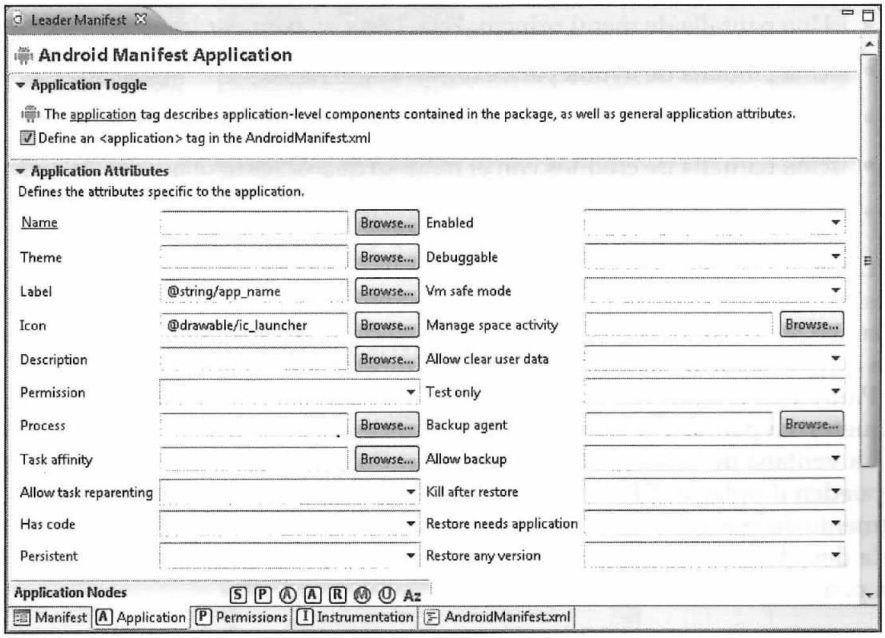


Figura 24.12. Edición del archivo AndroidManifest.xml de la aplicación.

3. El campo Icon (Icono) identifica el icono de la aplicación, que ahora muestra el valor incorrecto `@drawable/ic_launcher`. Haga clic en el botón **Browse** situado junto al campo. Se abrirá el cuadro de diálogo Resource Chooser (Selector de recursos) con los cinco recursos `drawable` de la aplicación.
4. Seleccione appicon y pulse **Aceptar**. El campo Icon muestra ahora el valor correcto.
5 Guarde el archivo: pulse el botón **Guardar** de la barra de herramientas o seleccione Archivo>Guardar.

Las X rojas desaparecen del Explorador de paquetes, lo que indica que la aplicación tiene ahora el icono correcto.

Diseñar una interfaz de usuario

La interfaz gráfica de usuario de una aplicación consta de diseños, contenedores que contienen elementos como campos de texto, botones, gráficos y otros elementos personalizados que diseñe. Cada pantalla mostrada al usuario puede tener uno o varios diseños. Existen diseños para apilar componentes de forma vertical u horizontal, para organizarlos en una tabla, etc.

Una aplicación puede tener una única pantalla o varias. Un juego se podría organizar de esta forma:

- Una pantalla de inicio que muestre el juego mientras se carga.
- Una pantalla de menú principal con botones para ver las demás pantallas.
- Una pantalla de ayuda para explicar cómo jugar.
- Una pantalla de puntuación con los marcadores más altos.
- Una pantalla de créditos con el nombre de los desarrolladores del juego.
- Una pantalla de juego para jugar las partidas.

La aplicación Leader solo tiene una pantalla, con cuatro botones para contactar con el presidente de los Estados Unidos u otro distinto. Las pantallas de la aplicación se almacenan en la carpeta `/res/layout`. Un proyecto nuevo tiene un archivo `main.xml` en esta carpeta ya designado como pantalla a mostrar al cargar la aplicación.

Para editar el diseño de esta pantalla, haga doble clic en `main.xml` en el Explorador de paquetes. La pantalla se abre en la ventana principal de Eclipse (véase la figura 24.13).

La ventana de edición contiene un panel Palette (Paleta) con distintas carpetas que se pueden desplegar. El subpanel Form Widgets (Componentes de formulario), probablemente desplegado, muestra sencillos componentes que pude arrastrar a la pantalla de la derecha. Siga los pasos descritos a continuación para añadir tres botones gráficos a la pantalla:

1. Borre el componente textview que muestra el texto **Hello World**. Selecciónelo y pulse **Supr**.
2. Haga doble clic en la carpeta Images & Media (Imágenes y medios) del panel Palette para desplegarla.
3. Arrastre un componente **ImageButton** desde la paleta a la pantalla. Aparecerá un cuadro azul en pantalla y un mensaje de error por debajo. El error se genera porque el botón carece de imagen.
4. Arrastre otros dos componentes **ImageButton** a la pantalla. Se apilarán verticalmente.
5. El panel Esquema enumera los componentes de la pantalla. Seleccione el elemento imageButton1. Las propiedades del botón se abren en el panel Propiedades (véase la figura 24.14).

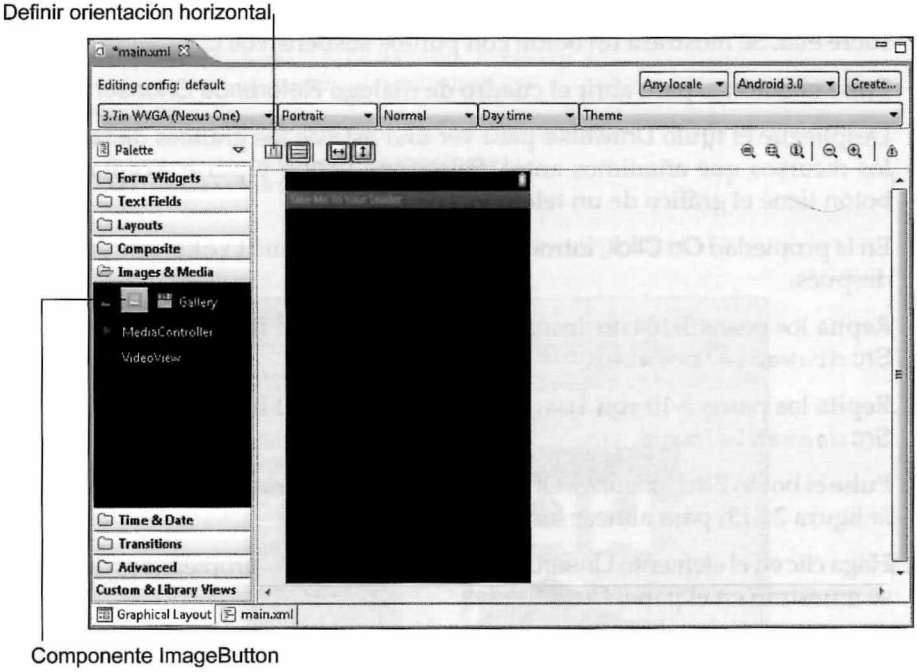


Figura 24.13. Edición del archivo main.xml de la aplicación.

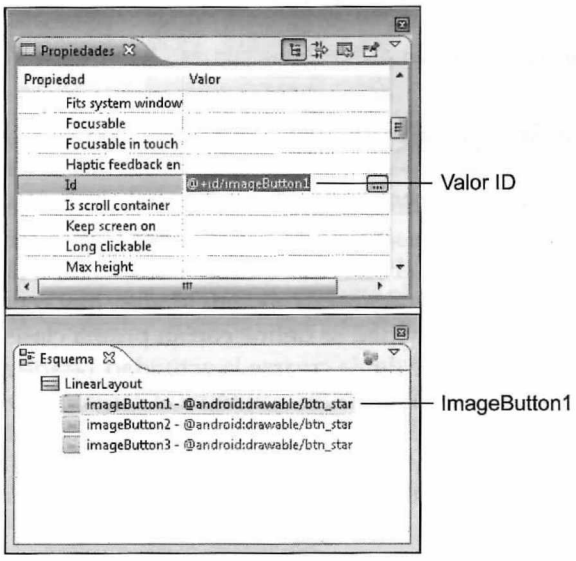


Figura 24.14. Personalización de las propiedades de un componente.

6. Desplácese hacia abajo en el panel Propiedades hasta que vea la propiedad Id. Su valor actual es `@+id/imageButton1`. Cámbielo por **@+id/phonebutton**.

7. Desplácese hasta la propiedad Src, que tiene el valor `drawable/icon`. Haga clic sobre ella. Se mostrará un botón con puntos suspensivos (...).
8. Pulse el botón ... para abrir el cuadro de diálogo Reference Chooser.
9. Despliegue el título Drawable para ver una lista de los gráficos de la aplicación, los recursos que añadimos antes. Seleccione phone y pulse **Aceptar**. Ahora el botón tiene el gráfico de un teléfono.
10. En la propiedad On Click, introduzca el valor `processClicks`, que explicaremos después.
11. Repita los pasos 5-10 con `imageButton2`. Asigne el ID `@+id/webbutton` y el Src `drawable/browser`.
12. Repita los pasos 5-10 con `imageButton3`. Asigne el ID `@+id/mapbutton` y el Src `drawable/maps`.
13. Pulse el botón Set Horizontal Orientation (Establecer orientación horizontal) (véase la figura 24.13) para alinear los botones.
14. Haga clic en el elemento LinearLayout de Esquema. Las propiedades de la pantalla se muestran en el panel Propiedades.
15. Haga clic en el valor de Background y pulse el botón .... Se abrirá la ventana Reference Chooser.
16. Despliegue Drawable, seleccione whitehouse y pulse **Aceptar**. El fondo de la pantalla será un gráfico de la Casa Blanca.
17. Pulse el botón **Guardar**. En la figura 24.15 puede ver la pantalla completa.

Crear código de Java

Ya hemos completado gran parte del trabajo de la aplicación, pero sin una sola línea de código de Java. El desarrollo de aplicaciones es más sencillo cuando se usa el mayor número de funciones del SDK de Android sin recurrir a la programación.

Las aplicaciones se organizan en actividades, que representan lo que la aplicación puede hacer. Cada actividad se define con su propia clase en Java. Al crear esta aplicación, especificamos que debía crearse la actividad `LeaderActivity`. Una clase con el mismo nombre se ejecuta automáticamente al cargar la aplicación.

El código fuente de `LeaderActivity.java` se encuentra en el Explorador de paquetes, en la carpeta `/src/org.cadenhead.android`. Haga doble clic en el archivo para editarlo. Al empezar, la clase incluye el código del listado 24.1.

Listado 24.1. Texto inicial de LeaderActivity.java.

```
1: package org.cadenhead.android;
2:
3: import android.app.Activity;
```

```
 4: import android.os.Bundle;
 5:
 6: public class LeaderActivity extends Activity {
 7:    /** Se invoca al crear la actividad. */
 8:    @Override
 9:    public void onCreate(Bundle savedInstanceState) {
10:       super.onCreate(savedInstanceState);
11:       setContentView(R.layout.main);
12:    }
13: }
```

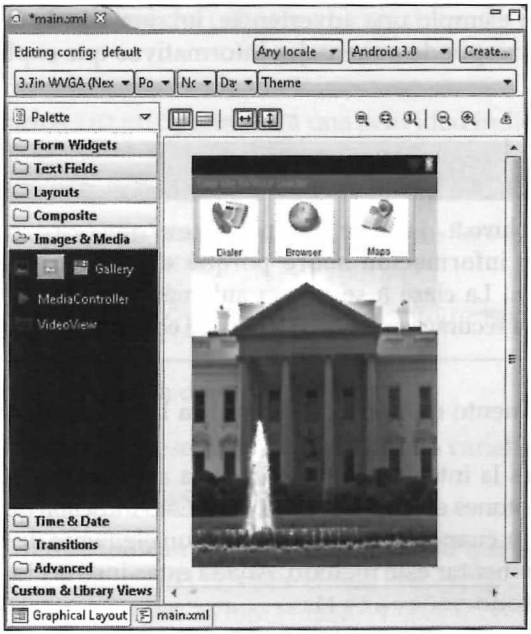


Figura 24.15. Vista previa de la interfaz gráfica de usuario de una aplicación.

Como sucede con todas las actividades, la clase `LeaderActivity` es una subclase de `Activity`, del paquete `android.app`, que contiene el comportamiento necesario para mostrar una pantalla, recopilar entradas del usuario, guardar preferencias del usuario, etc..

El método `onCreate()` definido en las líneas 9-12 se invoca al cargar la clase. Lo primero que hace es usar `super()` para invocar el mismo método en su superclase. Tras ello, invoca `setContentView()`, un método que selecciona la pantalla que se va a mostrar. El argumento de este método es una variable de instancia, `R.layout.main`, que hace referencia al archivo `main.xml` en `/res/layout`. Como recordará, el ID de un recurso es su nombre de archivo sin la extensión.

Lo primero que debe hacer en la clase `LeaderActivity` es asignar una variable de clase. Añada la siguiente instrucción por debajo de la definición de clase:

```
public static final String TAG = "Leader";
```

Esta variable actúa de identificador de la clase, que puede usar para registrar los eventos que se producen al ejecutarse. Las clases de Android pueden registrar sus actividades para que sepa qué sucede en la aplicación. Veamos una de las instrucciones de registro que añadirá después:

```
Log.i(TAG, "Making call");
```

Esta instrucción muestra un mensaje de registro etiquetado con el nombre `Leader`.

La clase `Log` del paquete `android.util` muestra mensajes del registro. Esta clase tiene cinco métodos diferentes para registrar mensajes y cada uno indica el tipo de mensaje, como por ejemplo una advertencia, un mensaje de depuración o un error. El método `i()` se corresponde a mensajes informativos que explican lo que sucede en la aplicación.

Nota

Puede abrir el archivo `R.java` de la carpeta `/res/gen/ org.cadenhead.android` si necesita más información sobre porqué el recurso principal se denomina `R.layout.main`. La clase `R` se genera automáticamente en el SDK para permitir hacer referencia a recursos por sus ID. No debe editar esta clase.

El primer argumento de `Log.i()` identifica la aplicación y el segundo contiene el mensaje.

Al diseñar antes la interfaz de usuario de la aplicación, establecimos la propiedad `On Click` de los botones en `processClicks`. Esto indicaba que se invocaría el método `processClicks()` cuando el usuario pulsara un elemento de la pantalla. Ha llegado el momento de implementar este método. Añada estas instrucciones a `LeaderActivity` por debajo del método `onCreate()`:

```
public void processClicks(View display) {
   Intent action;
   int id = display.getId();
}
```

Este método se invoca con un argumento, un objeto `View` del paquete `android.view`. `View` es una representación visual en una aplicación. En este caso es la pantalla que contiene los botones **Dialer**, **Browser** y **Maps**.

El método `getId()` del objeto `View` devuelve el ID del botón pulsado: `phonebutton`, `webbutton` o `mapbutton`. Este ID se almacena en la variable `id` para poder usarlo en una instrucción `switch` para realizar una acción en función del clic:

```
switch (id) {
   case (R.id.phonebutton):
      // ...
      break;
   case (R.id.webbutton):
      // ...
```

```
        break;
    case (R.id.mapbutton):
        // ...
        break;
    default:
        break;
}
```

Este código realiza una de tres acciones, usando el entero de cada ID como condición en `switch`.

La primera instrucción del método `processClicks()` crea una variable para almacenar un objeto `Intent`, una clase del paquete `android.content`:

```
Intent action;
```

En Android, los objetos `Intent` permiten a una actividad indicar a otra lo que debe hacer. También permiten a la aplicación comunicarse con el dispositivo Android. Veamos los tres objetos `Intent` usados en este método:

```
action = new Intent(Intent.ACTION_DIAL, Uri.parse("tel:202-456-1111"));
action = new Intent(Intent.ACTION_VIEW,
    Uri.parse("http://whitehouse.gov"));
action = new Intent(Intent.ACTION_VIEW, Uri.parse("geo:0,0?q=White House,
    Washington, DC"));
```

El constructor `Intent()` acepta dos argumentos:

- La acción que realizar, representada por una de sus variables de clase.
- Los datos asociados a la acción.

Estos tres objetos `Intent` indican al dispositivo Android que realice una llamada al teléfono de la Casa Blanca, (202) 456-1111, visite el sitio Web `http://whitehouse.gov` y abra Google Maps con la dirección "White House, Washington, DCQ".

Una vez creado el objeto `Intent`, la siguiente instrucción consigue que haga algo:

```
startActivity(action);
```

El listado 24.2 muestra el texto completo de la clase `LeaderActivity`. Añada las instrucciones `import` en las líneas 3-8 y el método `processClicks()` a lo que ya hemos introducido. Asegúrese de que su código coincide con el listado.

Listado 24.2. Texto completo de LeaderActivity.java.

```
1: package org.cadenhead.android;
2:
3: import android.app.Activity;
4: import android.content.Intent;
5: import android.net.Uri;
6: import android.os.Bundle;
7: import android.util.Log;
8: import android.view.View;
```

```
 9:
10: public class LeaderActivity extends Activity {
11:   public static final String TAG = "Leader";
12:
13:   /** Se invoca al crear la actividad. */
14:   @Override
15:   public void onCreate(Bundle savedInstanceState) {
16:      super.onCreate(savedInstanceState);
17:      setContentView(R.layout.main);
18:   }
19:
20:   public void processClicks(View display) {
21:      Intent action;
22:      int id = display.getId();
23:      switch (id) {
24:         case (R.id.phonebutton):
25:            Log.i(TAG, "Making call");
26:            action = new Intent(Intent.ACTION_DIAL,
27:               Uri.parse("tel:202-456-1111"));
28:            startActivity(action);
29:            break;
30:         case (R.id.webbutton):
31:            Log.i(TAG, "Loading browser");
32:            action = new Intent(Intent.ACTION_VIEW,
33:               Uri.parse("http://whitehouse.gov"));
34:            startActivity(action);
35:            break;
36:         case (R.id.mapbutton):
37:            Log.i(TAG, "Loading map");
38:            action = new Intent(Intent.ACTION_VIEW,
39:               Uri.parse("geo:0,0?q=White House, Washington, DC"));
40:            startActivity(action);
41:            break;
42:         default:
43:            break;
44:      }
45:   }
46: }
```

Cuando termine, guarde el archivo. Se compilará correctamente (Eclipse lo hará de forma automática); en caso contrario, aparecerán las X de color rojo en el Explorador de paquetes para indicar los archivos del proyecto con errores. Si no hay errores, ya está listo para ejecutar la aplicación, pero primero debe crear una nueva configuración para el proyecto:

1. Haga clic en la flecha situada junto al botón **Depurar** de la barra de herramientas de Eclipse y seleccione Configuraciones de Depuración para abrir la ventana Configuraciones de Depuración.
2. Haga doble clic en Android Application. Se añadirá la entrada Configuración_ nueva (1).
3. Introduzca **LeaderDebug** en el campo Name.

4. Pulse el botón **Browse** (Examinar), seleccione el proyecto Leader y pulse **Aceptar**.
5. Haga clic en la ficha Target para activarla.
6. En Deployment Target Selection Mode marque la opción Automatic y seleccione el dispositivo virtual SimpleAVD.
7. Cambie Deployment Target Selection Mode a Manual y pulse **Aplicar** y **Cerrar**.

Se creará la nueva configuración de depuración LeaderDebug.

Para ejecutar la aplicación, haga clic en la flecha situada junto al botón **Depurar** y seleccione LeaderDebug (si se muestra). En caso contrario, seleccione Configuraciones de depuración, seleccione LeaderDebug y pulse **Depurar**. Se abrirá el cuadro de diálogo Android Device Chooser (Selector de dispositivos de Android). Marque la opción Launch a new Android Virtual Device (Iniciar un nuevo dispositivo virtual de Android), seleccione SimpleAVD y pulse **Aceptar**.

El emulador se abre tras unos minutos y ejecuta automáticamente la aplicación.

El emulador no reproduce todo lo que puede hacer un dispositivo de Android. Los botones **Dialer** y **Browser** de la aplicación Leader deberían funcionar correctamente pero puede que tenga problemas con **Maps**.

La aplicación también se puede ejecutar en un teléfono Android, siempre que funcione con el SDK de Android y se haya configurado en modo de depuración.

Haga clic en la flecha situada junto a **Depurar** y seleccione LeaderDebug, que esta vez sí aparecerá. Marque la opción Choose a running Android device (Seleccionar un dispositivo de Android en ejecución), seleccione su teléfono en la lista y pulse **Aceptar**.

En la figura 24.16 puede ver la aplicación ejecutada en un teléfono. Si lo cambia de modo vertical a modo horizontal, la aplicación también cambiará.

Figura 24.16. ¡Take me to your leader!

La aplicación Leader también se añade a las aplicaciones del teléfono con su propio icono **Take Me to Your Leader**. Se conservará en el teléfono incluso después de desconectar el cable USB.

¡Enhorabuena! Ahora existe una nueva aplicación de Android en el mundo.

Nota

Como habrá imaginado, la programación de Android es mucho más amplia de lo que hemos visto en un solo capítulo, incluso aunque sea obra de un autor de demostrada experiencia, enorme talento y humildad sin igual como un servidor. Si desea ampliar sus conocimientos, consulte un manual especializado sobre programación con Android.

Resumen

El objetivo de este libro es ayudarle a familiarizarse con los conceptos de programación y a confiar en su capacidad para crear sus propias aplicaciones, independientemente de que las ejecute en un ordenador de escritorio, una página Web, un servidor Web o un teléfono. El enfoque de Java es un tanto difícil de dominar (puede tachar la parte "un tanto" de la frase ya que no hace honor a la verdad).

Al aumentar su experiencia con Java, lo hará sobre aspectos de gran relevancia, ya que conceptos como la programación orientada a objetos, los equipos virtuales y los entornos seguros están presentes en el desarrollo de software más puntero.

En los apéndices encontrará información adicional de gran utilidad.

Al finalizar este capítulo, puede explorar Java en distintos puntos. Los programadores analizan el lenguaje en los blogs de `http://weblogs.java.net`. Numerosas ofertas de trabajo sobre Java aparecen en sitios Web como `http://www.careerbuilder.com`.

Si ha completado todos los capítulos, enhorabuena. Use sus nuevos conocimientos de programación para conseguir un buen trabajo y reimpulsar la economía mundial.

Preguntas y respuestas

P: ¿Por qué se usa Eclipse para crear aplicaciones de Android en lugar de NetBeans?

R: Puede usar NetBeans para desarrollar aplicaciones pero es un IDE más complicado y menos compatible con la programación de Android. Google ha diseñado Eclipse como el IDE más indicado para Android. La documentación oficial y los cursos prácticos del sitio Android Developer (`http://developer.android.com`) usan Eclipse.

Muchos libros de programación para Android también usan Eclipse. Aunque existe cierta curva de aprendizaje para cambiar de NetBeans a Eclipse al adentrarse en Android, una vez dominados los aspectos básicos de crear, depurar e implementar una aplicación, encontrará que Eclipse es más fácil de usar debido a su mayor aceptación entre programadores y escritores técnicos.

P: ¿Cómo dificulta ProGuard que el código fuente de una aplicación se pueda descompilar?

R: Los archivos de clase de Java pueden ser sometidos a ingeniería inversa, el proceso de averiguar el código fuente usado para crear un código ejecutable. Como los diseñadores de aplicaciones de Android puede que no quieran que otros programadores copien su código fuente en sus aplicaciones, ProGuard está disponible en todos los proyectos de Android que cree.

ProGuard optimiza una aplicación eliminando código sin usar de sus archivos de clase al compilarlos. También cambia los nombres de clases, campos y métodos por otros sin sentido. De esa forma, aunque alguien descompile el código de Java, resulta mucho más complicado determinar el código fuente.

La función de ofuscación solo se aplica al generar una aplicación en modo de publicación. Si se hace antes, la depuración sería mucho más compleja.

Ejercicios

Si desea probar sus conocimientos sobre desarrollo con Android, responda a las siguientes preguntas.

Preguntas

1. ¿Cuál de las siguientes empresas no forma parte de *Open Handset Initiative*, el grupo creador de Android?
   A. Google.
   B. Apple.
   C. Motorola.
2. ¿Qué herramienta dificulta a los desarrolladores averiguar el código fuente de un programa de Java?
   A. Un descompilador.
   B. Un recompilador.
   C. Un ofuscador.
3. ¿Cuál de las siguientes tareas no puede realizar un emulador de Android?
   A. Recibir un mensaje SMS.
   B. Conectarse a Internet.
   C. Realizar una llamada telefónica.

Respuestas

1. B. Apple, ya que Android se creó, en parte, como alternativa de código abierto y no propietaria al iPhone de Apple.
2. C. Menudo trabalenguas, ¿verdad? Diga rápidamente ofuscador cinco veces.
3. C. Los emuladores no pueden hacer nada de lo que hace un dispositivo real, de modo que solo forman parte del proceso de pruebas de una aplicación.

Actividades

Para ampliar sus conocimientos sobre Android, realice las siguientes actividades:

- Cambie el texto de la aplicación `SalutonMondo` por `Hello, Android` y ejecútela en el emulador y en un dispositivo Android (si tiene).
- Cree una nueva versión de Take Me To Your Leader para otro líder mundial distinto y personalice el teléfono, la Web y los destinos de los mapas.

Apéndices

Apéndice A
Usar el entorno de desarrollo integrado NetBeans

Aunque se puedan crear programas de Java con el JDK y un editor de texto, el proceso es menos masoquista si se recurre a un entorno de desarrollo integrado (IDE).

En los 23 primeros capítulos del libro usamos NetBeans, un IDE gratuito de Oracle para programadores de Java. NetBeans es un programa que facilita reconocer, escribir, compilar y probar software de Java. Incluye un administrador de proyectos y de archivos, un diseñador de interfaces graficas de usuario y muchas otras herramientas. Una de sus principales funciones es el editor de código, que detecta errores sintácticos de Java mientras se escribe.

La versión 7.0 de NetBeans se ha convertido en la preferida por los programadores profesionales de Java, ya que ofrece una funcionalidad y un rendimiento que valen 10 veces su precio. También es uno de los IDE más sencillos para los no versados en Java.

En este apéndice veremos cómo instalar el software y aprenderemos a usarlo con los proyectos del libro.

Instalar NetBeans

Desde sus inicios, el IDE NetBeans ha evolucionado hasta convertirse en una de las principales herramientas de programación para los desarrolladores de Java. James Gosling, el creador de Java, le otorgó su voto de confianza definitivo en su prólogo del libro *NetBeans Field Guide*: "Uso NetBeans en todos mis proyectos de desarrollo de Java". Yo también me he convertido.

Aunque se puedan crear programas de Java con el JDK y un editor de texto, el proceso es menos masoquista si se recurre a un entorno de desarrollo integrado (IDE).

En los 23 primeros capítulos del libro usamos NetBeans, un IDE gratuito de Oracle para programadores de Java. NetBeans es un programa que facilita reconocer, escribir, compilar y probar software de Java. Incluye un administrador de proyectos y de archivos, un diseñador de interfaces graficas de usuario y muchas otras herramientas. Una de sus principales funciones es el editor de código, que detecta errores sintácticos de Java mientras se escribe.

La versión 7.0 de NetBeans se ha convertido en la preferida por los programadores profesionales de Java, ya que ofrece una funcionalidad y un rendimiento que valen 10 veces su precio. También es uno de los IDE más sencillos para los no versados en Java.

En este apéndice veremos cómo instalar el software y aprenderemos a usarlo con los proyectos del libro.

Instalar NetBeans

Desde sus inicios, el IDE NetBeans ha evolucionado hasta convertirse en una de las principales herramientas de programación para los desarrolladores de Java. James Gosling, el creador de Java, le otorgó su voto de confianza definitivo en su prólogo del libro *NetBeans Field Guide*: "Uso NetBeans en todos mis proyectos de desarrollo de Java". Yo también me he convertido.

NetBeans admite todos los aspectos de la programación Java de las tres ediciones del lenguaje: JSE (*Java Standard Edition*, Java Edición estándar), JEE (*Java Enterprise Edition*, Java Edición empresarial) y JME (*Java Mobile Edition*, Java Edición móvil). También admite el desarrollo de aplicaciones Web, servicios Web y JavaBeans.

Puede descargar el software, disponible para Windows, MacOS y Linux, desde `www.netbeans.org`. NetBeans se incluye en la descarga del JDK (*Java Development Kit*, Kit de desarrollo de Java), una opción que elegir si no lo tiene en su equipo.

Crear un nuevo proyecto

El JDK y NetBeans se descargan como asistentes de instalación para configurar el software en su sistema. Puede instarlo en la carpeta y en el grupo de menús que desee pero conviene conservar las opciones predeterminadas, a menos que tenga un buen motivo para no hacerlo.

Al ejecutar NetBeans por primera vez tras instalarlo, verá una página de inicio que muestra enlaces a noticias y tutoriales (véase la figura A.1). Puede leerlos desde el IDE por medio del navegador integrado de NetBeans.

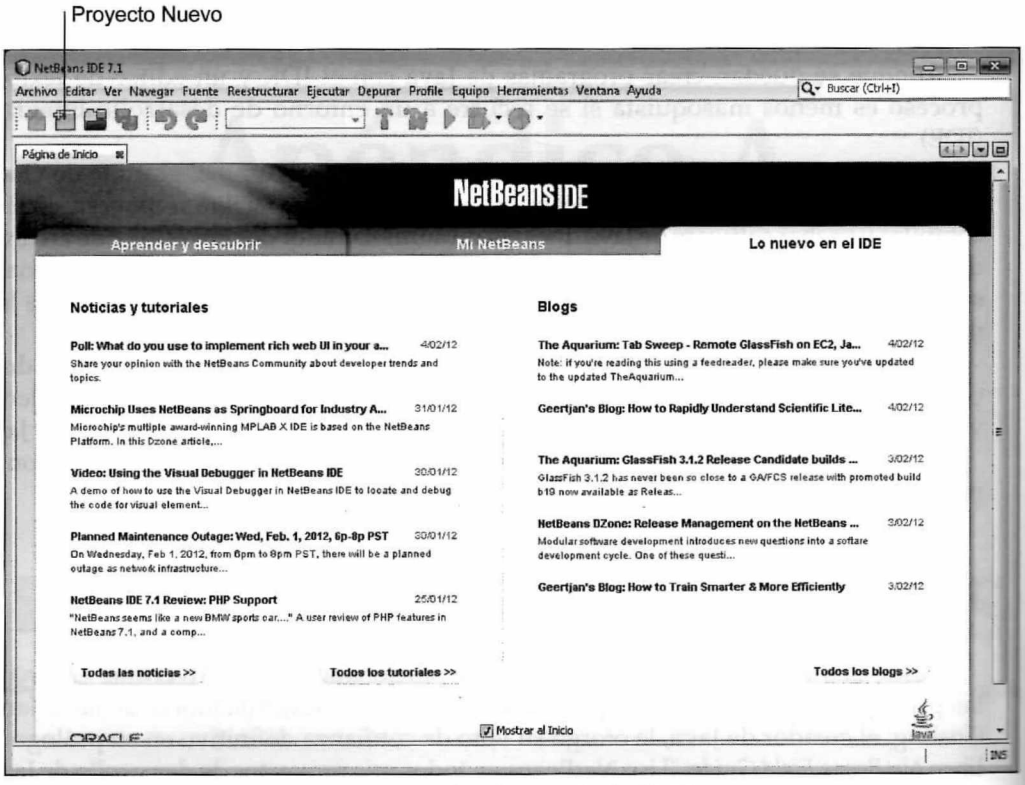


Figura A.1. Interfaz de usuario de NetBeans.

Un proyecto de NetBeans está formado por una serie de clases de Java relacionadas, archivos usados por dichas clases y bibliotecas de clases. Cada proyecto tiene su propia carpeta, que puede examinar y modificar fuera de NetBeans con editores de texto y otras herramientas de programación. Para iniciar un nuevo proyecto, pulse el botón **Proyecto Nuevo** indicado en la figura A.1 o seleccione Archivo>Proyecto Nuevo. Se abrirá el asistente Proyecto Nuevo (véase la figura A.2).

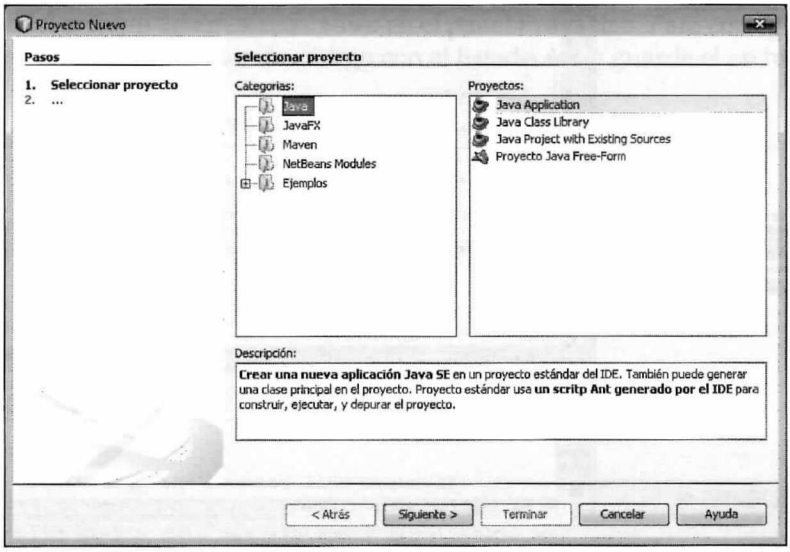


Figura A.2. El asistente Proyecto Nuevo.

NetBeans puede crear distintos tipos de proyectos de Java pero en este libro puede centrarse en uno de ellos: aplicaciones de Java. Para su primer proyecto (y la mayoría de los proyectos del libro), seleccione el tipo Java Application (Aplicación de Java) y pulse **Siguiente**. El asistente le pedirá que elija un nombre y una ubicación para el proyecto.

El campo Ubicación del proyecto identifica la carpeta raíz de los proyectos de programación creados con NetBeans. En Windows, es la subcarpeta `NetBeansProjects` de `Mis documentos`. Todos los proyectos que cree se almacenan en esta carpeta, cada uno en su propia subcarpeta. En el campo Nombre proyecto, introduzca **Java24**. El cuadro de texto Crear clase principal cambia como respuesta a la entrada y le recomienda `java24.Java24` como nombre de la clase principal de Java del proyecto. Cámbielo por **Spartacus** y pulse **Terminar**. Acepte las demás opciones predeterminadas. NetBeans crea el proyecto y su primera clase.

Crear una nueva clase de Java

Cuando NetBeans crea un nuevo proyecto, configura todos los archivos y carpetas necesarias y crea la clase principal. En la figura A.3 puede ver la primera clase del proyecto, `Spartacus.java`, en el editor de código.

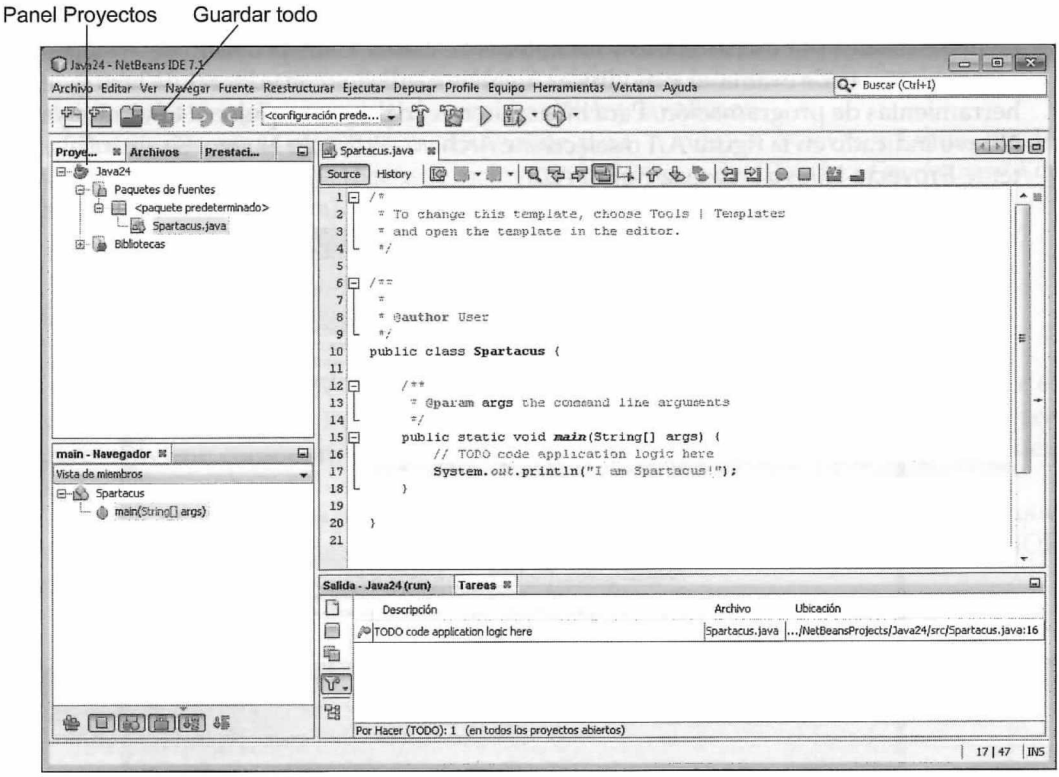


Figura A.3. El editor de código de NetBeans.

`Spartacus.java` es una clase básica de Java formada solamente por un método `main()`. Las líneas grises son comentarios que explican el funcionamiento de la clase. Estos comentarios se ignoran al ejecutar la clase.

Para que una clase nueva haga algo, añada la siguiente línea de código por debajo del comentario `// TODO code application logic here`:

```
System.out.println("I am Spartacus!");
```

El método `System.out.println()` muestra una cadena de texto, en este caso la frase `I am Spartacus!`.

Asegúrese de escribirlo tal y como se muestra. Al escribir, el editor de código intuye lo que hace y muestra información de ayuda relacionada con la clase `System`, la variable de instancia `out` y el método `println()`. Esto le será de gran utilidad después, pero por ahora puede ignorarlo.

Tras añadir correctamente la línea y finalizarla con punto y coma, pulse el botón **Guardar todo** de la barra de herramientas para guardar la clase.

Las clases de Java deben compilarse en código de bytes ejecutable antes de poder ejecutarlas. NetBeans trata de compilar las clases automáticamente, aunque también puede hacerlo de forma manual:

- Seleccione Ejecutar>Compilar File (Compilar archivo).
- Haga clic con el botón derecho del ratón sobre `Spartacus.java` en el panel Proyectos para abrir un menú contextual y seleccione Compilar File.

Si NetBeans no le permite seleccionar estas opciones, significa que ya ha compilado la clase automáticamente.

Si la clase no se compila correctamente, aparece un signo de exclamación rojo junto al nombre del archivo `Spartacus.java` en el panel Proyectos. Para corregirlo, compare lo que ha escrito en el editor de código con el listado A.1 y guarde el archivo.

Listado A.1. La clase de Java Spartacus.java.

```
 1: /*
 2: * To change this template, choose Tools | Templates
 3: * and open the template in the editor.
 4: */
 5:
 6: /**
 7: *
 8: * @author User
 9: */
10: public class Spartacus {
11:
12:   /**
13:   * @param args the command line arguments
14:   */
15:   public static void main(String[] args) {
16:      // TODO code application logic here
17:      System.out.println("I am Spartacus!");
18:
19:   }
20:
21: }
```

La clase se define en las líneas 10-21. Las líneas 1-9 son comentarios que NetBeans incluye en todas las clases nuevas.

Ejecutar la aplicación

Una vez creada la clase de Java `Spartacus.java` y tras compilarla correctamente, puede ejecutarla desde NetBeans de dos formas:

- Seleccione Ejecutar>Ejecutar archivo en el menú.
- Haga clic con el botón derecho del ratón sobre `Spartacus.java` en el panel Proyectos y seleccione Ejecutar archivo.

Al ejecutar una clase de Java, el compilador invoca su método `main()`. La cadena `I am Spartacus!` se muestra en el panel Salida (véase la figura A.4).

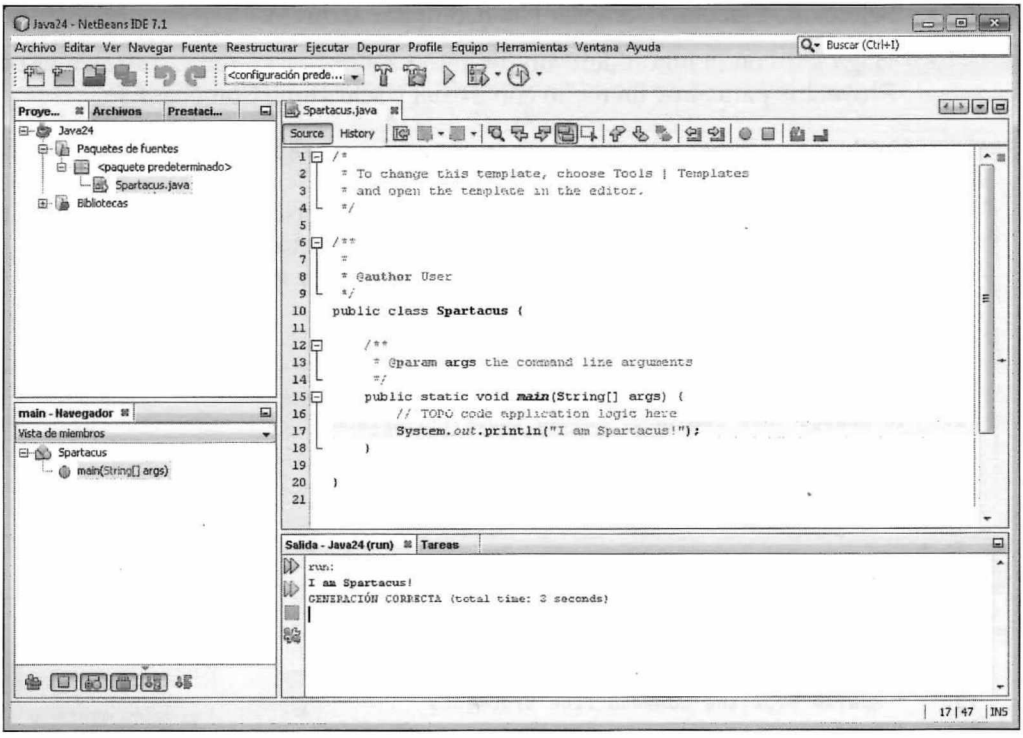


Figura A.4. Resultado de la aplicación Spartacus.

Una clase de Java debe tener un método `main()` para poder ejecutarla. Si intenta ejecutarla sin este método, NetBeans generará un error.

Corregir errores

Una vez creada, compilada y ejecutada la aplicación `Spartacus`, es hora de estropear algo para que vea cómo responde NetBeans a los errores.

Como cualquier programador, tendrá mucha práctica en cometer errores, pero preste atención de todas formas.

Vuelva a `Spartacus.java` en el editor de código fuente y borre el punto y coma al final de la línea que invoca `System.out.println()` (línea 17 del listado A.1). Incluso antes de guardar el archivo, NetBeans detecta el error y muestra un icono de advertencia rojo a la izquierda de la línea (véase la figura A.5).

Desplace el ratón sobre el icono de advertencia para abrir un cuadro de diálogo en el que se describe el error detectado por NetBeans.

El editor de código de NetBeans puede identificar la mayoría de errores de programación y erratas que detecte mientras escribe un programa de Java. Impide que el archivo se compile hasta que los errores se corrijan.

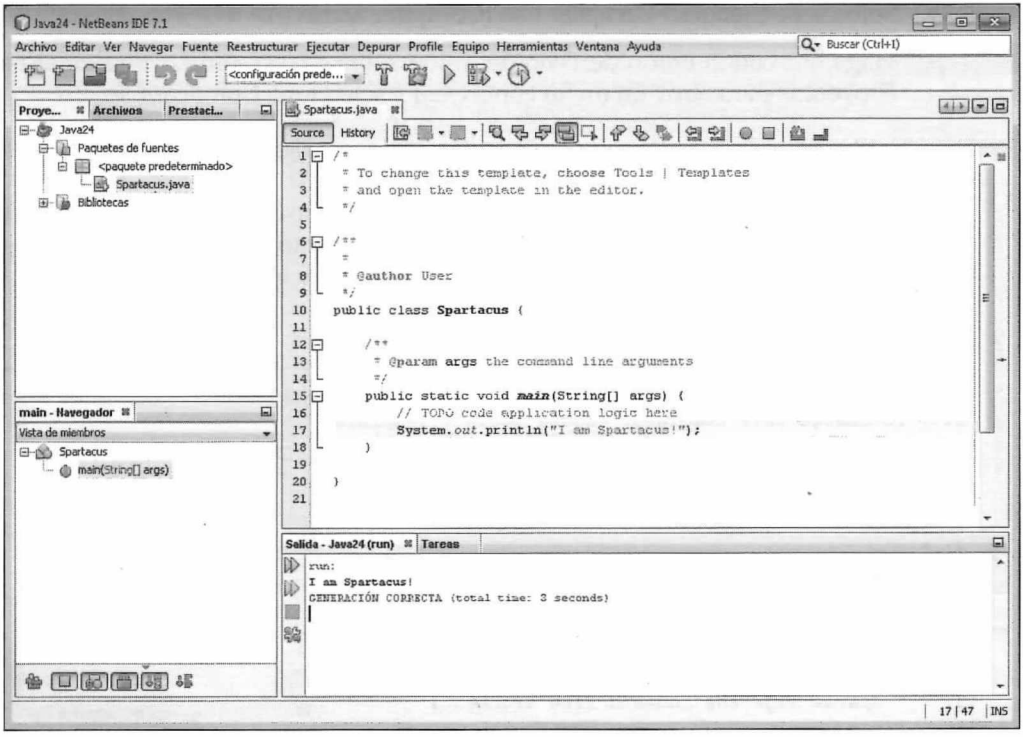


Figura A.4. Resultado de la aplicación Spartacus.

Una clase de Java debe tener un método `main()` para poder ejecutarla. Si intenta ejecutarla sin este método, NetBeans generará un error.

Corregir errores

Una vez creada, compilada y ejecutada la aplicación `Spartacus`, es hora de estropear algo para que vea cómo responde NetBeans a los errores.

Como cualquier programador, tendrá mucha práctica en cometer errores, pero preste atención de todas formas.

Vuelva a `Spartacus.java` en el editor de código fuente y borre el punto y coma al final de la línea que invoca `System.out.println()` (línea 17 del listado A.1). Incluso antes de guardar el archivo, NetBeans detecta el error y muestra un icono de advertencia rojo a la izquierda de la línea (véase la figura A.5).

Desplace el ratón sobre el icono de advertencia para abrir un cuadro de diálogo en el que se describe el error detectado por NetBeans.

El editor de código de NetBeans puede identificar la mayoría de errores de programación y erratas que detecte mientras escribe un programa de Java. Impide que el archivo se compile hasta que los errores se corrijan.

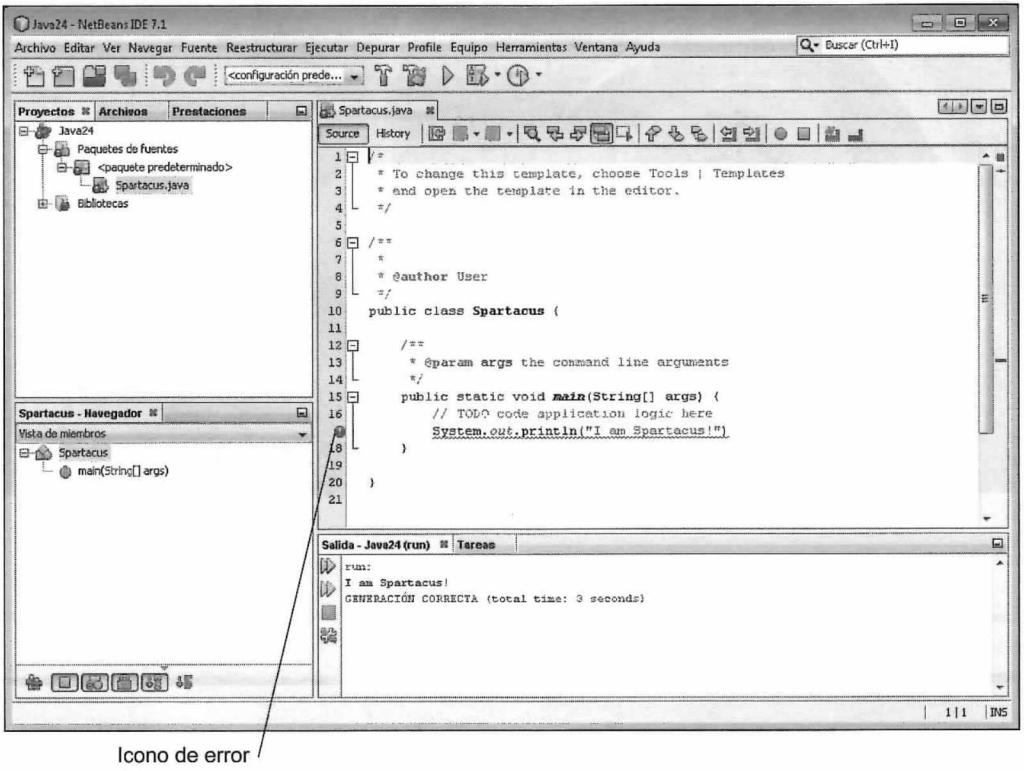


Figura A.5. Errores en el editor de código.

Vuelva a añadir el punto y coma al final de la línea. El icono de error desaparece y ya puede guardar y volver a ejecutar la clase.

Estas funciones básicas son todo lo que necesita para crear y compilar los programas de Java del libro.

NetBeans cuenta con muchas más prestaciones que las que hemos indicado aquí, pero debería centrarse en aprender Java antes de adentrarse en el IDE. Utilice NetBeans como si fuera un sencillo administrador de proyectos y editor de texto. Cree clases, identifique errores e intente compilar y ejecutar correctamente todos sus programas.

Si desea aprender más sobre NetBeans, Oracle ofrece recursos de formación y documentación en `www.netbeans.org/kb`.

Apéndice B
Recursos para Java

Tras finalizar este manual, seguramente se pregunte dónde dirigirse para mejorar sus conocimientos de programación de Java. En este apéndice se indican sitios Web, grupos de Internet y otos recursos que puede usar para ampliar sus conocimientos.

Sitio de Java oficial de Oracle

La división de software de Java de Oracle mantiene tres sitios Web muy interesantes para programadores y usuarios del lenguaje.

Oracle Technology Network for Java Developers (Red tecnológica de Oracle para desarrolladores de Java), publicada en `http://www.oracle.com/technetwork/java`, es la primera visita si busca información relacionada con Java. Encontrará nuevas versiones del JDK y otros recursos de programación para descargar, además de documentación sobre la biblioteca de clases de Java. También cuenta con una base de datos de errores, un directorio de grupos de usuario y foros.

Java.net (`www.java.net`) es una gran comunidad de programadores de Java. Puede abrir su propio blog sobre el lenguaje, crear un nuevo proyecto de código abierto y alojarlo gratuitamente en el sitio, y colaborar con otros programadores.

Java.com (`http://www.java.com/es/`) destaca los beneficios del lenguaje para programadores y no programadores. Puede descargar el entorno de tiempo de ejecución desde el sitio, que permite a los usuarios ejecutar programas creados con Java. También incluye una galería con ejemplos de uso de Java.

Documentación de clases de Java

Puede que la parte más completa del sitio Java de Oracle sea la documentación de todas las clases, variables y métodos de la biblioteca de clases de Java. Existen miles de páginas *online* sin coste alguno que puede consultar para saber cómo usar las clases en sus programas. Para consultar la documentación de las clases de Java 7, visite `http://download.oracle.com/javase/7/docs/api`.

Otros sitios Web de Java

Como gran parte del fenómeno Java se inspiró gracias a su uso en páginas Web, existen numerosos sitios Web sobre Java y la programación con Java.

Café au Lait

Elliotte Rusty Harold, autor de excelentes libros sobre programación con Java, ofrece Café au Lait, un consolidado blog que incluye noticias sobre Java, publicación de productos y otros sitios de interés para programadores. El sitio es un magnífico recurso para todos los interesados en Java. Se puede visitar en `www.cafeaulait.org`. Harold también ofrece una lista de las preguntas más habituales relacionadas con Java. Últimamente no ofrece demasiadas actualizaciones debido a una remodelación en curso del sitio pero puede que en breve lo vuelva a lanzar.

Workbench

También publico un blog, Workbench, relacionado con Java, tecnología de Internet, libros de informática y temas similares. Puede visitarlo en `http://workbench.cadenhead.org`.

Java 7 Developer Blog

Los programadores de Java Ben Evans y Martijn Verburg analizan el progreso de Java 7 en su blog Java 7 Developer (`www.java7developer.com`). Ofrecen ejemplos de código que ilustran las novedades de la versión actual del lenguaje, consejos para usarlas de forma eficaz y analizan las funciones que se esperan en Java 8.

Otros blogs sobre Java

Existen cientos de blogs sobre programación con Java, ya sea como tema principal o como parte de otros temas más diversos. En el motor de búsqueda IceRocket encontrará una lista etiquetada de los últimos blogs sobre Java: `www.icerocket.com/tag/java`.

InformIT

El sitio de referencia InformIT, `www.informit.com`, es un completo recurso, que dedica sus secciones a distintos temas relacionados con el desarrollo de software e Internet. La sección sobre Java incluye artículos prácticos y una referencia para principiantes.

Stack Overflow

La comunidad *online* Stack Overflow es el lugar donde los programadores realizan preguntas y valoran las respuestas proporcionadas por otros usuarios. El sitio está etiquetado, de modo que puede limitar sus búsquedas al lenguaje o tema que le interese. Visite `http://stackoverflow.com/questions/tagged/java`.

Java Review Service

Java Review Service examina programas, componentes y herramientas publicadas en la Web, que clasifica como Top 1%, Top 5% o Top 25%. También categoriza los recursos por tema, con una descripción de cada uno y enlaces para descargar el código fuente, si existe. Puede visitarlo en `www.jars.com`.

JavaWorld Magazine

Una revista en activo desde la aparición del lenguaje. Suele publicar artículos y tutoriales, junto con noticias sobre desarrollo con Java. Incluye también podcasts de vídeo y audio. Puede visitar la revista en `www.javaworld.com`.

Directorio Java de Developer.com

Como Java es un lenguaje orientado a objetos, puede usar recursos creados por otros programadores en sus programas. Antes de iniciar cualquier proyecto de Java, busque recursos en la Web que pueda reutilizar en su programa.

Un buen punto de partida es el directorio de Java de Developer.com. Este sitio cataloga los programas de Java, los recursos de programación e información similar (`www.developer.com/java`).

Twitter

Si necesita algo más interactivo para buscar consejos para programadores de Java, pruebe con Twitter, el conocido servicio usado por millones de usuarios para enviar mensajes breves a sus amigos y a otros seguidores.

El hashtag #java identifica los mensajes relacionados con Java, aunque algunos podrían referirse a la isla o al café, ya que los hashtag son informal y los crean los usuarios. Para buscar en Twitter los últimos mensajes sobre Java, introduzca http://twitter.com/#!/search-home en un navegador Web y busque por #java.

Oportunidades laborales

Si es uno de los usuarios que aprende Java como parte de un plan para convertirse en un líder del sector, algunos de los recursos indicados en este apéndice incluyen una sección con ofertas de trabajo. Consulte las ofertas relacionadas con Java que puedan estar disponibles.

El motor de búsqueda de ofertas de trabajo indeed incluye una sección dedicada a puestos de trabajo de Java. Visite www.indeed.com/q-Java-jobs.html para ver las últimas ofertas para programadores versados en el lenguaje. También puede probar suerte en www.dice.com.

Aunque no sea un recurso específico de búsqueda de empleados de Java, el sitio Web CareerBuilder (www.careerbuilder.com) le permite buscar ofertas de trabajo en distintas bases de datos, incluidos periódicos y otras fuentes. Puede buscar en más de 100.000 ofertas mediante palabras clave como Java, Internet o encantador de serpientes.

Apéndice C Configurar un entorno de desarrollo de Android

Aunque las aplicaciones de Android se crean con Java, requieren algo más que las herramientas de programación estándar. Necesitan el JDK (*Java Development Kit*, Kit de desarrollo de Java), el SDK de Android (*Software Development Kit*, Kit de desarrollo de software), un entorno de desarrollo integrado compatible con la programación Android y controladores para dispositivos Android.

Eclipse es el IDE más utilizado y con mayor compatibilidad para Android.

En este apéndice, configuraremos estas herramientas y las utilizaremos para ejecutar una aplicación de Android. Todas ellas son gratuitas y se pueden descargar en Internet.

Primeros pasos

Puede realizar la programación de Android en los siguientes sistemas operativos:

- Windows XP o posterior.
- Mac OS X 10.5.8 o posterior (x86).
- Linux.

Necesitará 600MB de espacio en disco para instalar el SDK de Android y otros 1.2GB para Eclipse. Ya debería tener instalado el JDK, ya que lo hemos utilizado en los distintos capítulos del libro junto con NetBeans para ejecutar programas de Java. Android requiere JDK 5.0 o posterior. Si necesita el JDK, puede descargarlo de `http://oracle.com/technetwork/java/javase`.

Instalar Eclipse

Aunque otros IDE como NetBeans admiten desarrollo de Android, Eclipse se ha convertido en el más utilizado para crear aplicaciones para Android. Los desarrolladores de Android han seleccionado Eclipse como su entorno preferido y lo utilizan en la documentación oficial y los tutoriales.

Eclipse, como NetBeans, ofrece una interfaz gráfica de usuario para crear programas Java. Puede usarla para crear cualquier tipo de programa de Java (y también admite otros lenguajes de programación). Android requiere Eclipse 3.5 o posterior. Para descargar Eclipse, visite `http://eclipse.org/downloads`. Existen diversas versiones del IDE. Seleccione *Eclipse IDE for Java EE Developers* (IDE Eclipse para desarrolladores de Java EE). Java EE es la edición empresarial de Java, que incluye dos elementos que usará en sus proyectos de Android: el complemento JDT (*Java Development Tools*, Herramientas de desarrollo de Java) y WTP (*Web Tools Platform*, Plataforma de herramientas Web).

Eclipse se empaqueta como archivo ZIP. No cuenta con un programa de instalación que le guíe para configurarlo en su equipo. El archivo ZIP contiene una carpeta `eclipse` de nivel superior con todos los archivos necesarios para ejecutar Eclipse. Descomprima la carpeta en una ubicación donde guarde sus programas. En mi sistema de Windows uso la carpeta `Program Files (x86)`. Tras ello, abra la carpeta `eclipse` recién creada y busque la aplicación ejecutable Eclipse. Cree un acceso directo a la misma y añádala al escritorio o a la barra de tareas. Antes de iniciar Eclipse debe instalar el SDK de Android.

Instalar el SDK de Android

El SDK de Android es un conjunto gratuito de herramientas para crear, depurar y ejecutar aplicaciones de Android. Se usa en Eclipse cuando se trabaja con aplicaciones Android. Puede descargar el SDK del sitio Web oficinal de Android: `http://developer.android.com/sdk`. Está disponible para Windows, Mac OS y Linux. La versión para Windows se ofrece como asistente de instalación que le guía por el proceso de configuración. Las demás, al cierre de esta edición, son un archivo ZIP (Mac OS) o TGZ (Linux).

En cualquier caso, incluya Android en la que carpeta que use para almacenar programas, en teoría la misma carpeta principal de la carpeta de Eclipse. En mi caso, uso `Program Files (x86)`. El SDK incluye un administrador SDK y AVD que se usa para actualizar y ampliar el SDK una vez instalado. Este administrador, que se ejecuta desde Eclipse, permite actualizar el SDK con las novedades publicadas de Android. Una vez instalado el SDK, ya puede ejecutar Eclipse.

Instalar el complemento Android para Eclipse

El IDE Eclipse admite diversos lenguajes de programación y tecnología pero no todos se admiten desde el principio. El IDE se mejora mediante complementos que ofrecen la funcionalidad necesaria.

Instalar Eclipse

Aunque otros IDE como NetBeans admiten desarrollo de Android, Eclipse se ha convertido en el más utilizado para crear aplicaciones para Android. Los desarrolladores de Android han seleccionado Eclipse como su entorno preferido y lo utilizan en la documentación oficial y los tutoriales.

Eclipse, como NetBeans, ofrece una interfaz gráfica de usuario para crear programas Java. Puede usarla para crear cualquier tipo de programa de Java (y también admite otros lenguajes de programación). Android requiere Eclipse 3.5 o posterior. Para descargar Eclipse, visite `http://eclipse.org/downloads`. Existen diversas versiones del IDE. Seleccione *Eclipse IDE for Java EE Developers* (IDE Eclipse para desarrolladores de Java EE). Java EE es la edición empresarial de Java, que incluye dos elementos que usará en sus proyectos de Android: el complemento JDT (*Java Development Tools*, Herramientas de desarrollo de Java) y WTP (*Web Tools Platform*, Plataforma de herramientas Web).

Eclipse se empaqueta como archivo ZIP. No cuenta con un programa de instalación que le guíe para configurarlo en su equipo. El archivo ZIP contiene una carpeta `eclipse` de nivel superior con todos los archivos necesarios para ejecutar Eclipse. Descomprima la carpeta en una ubicación donde guarde sus programas. En mi sistema de Windows uso la carpeta `Program Files (x86)`. Tras ello, abra la carpeta `eclipse` recién creada y busque la aplicación ejecutable Eclipse. Cree un acceso directo a la misma y añádala al escritorio o a la barra de tareas. Antes de iniciar Eclipse debe instalar el SDK de Android.

Instalar el SDK de Android

El SDK de Android es un conjunto gratuito de herramientas para crear, depurar y ejecutar aplicaciones de Android. Se usa en Eclipse cuando se trabaja con aplicaciones Android. Puede descargar el SDK del sitio Web oficinal de Android: `http://developer.android.com/sdk`. Está disponible para Windows, Mac OS y Linux. La versión para Windows se ofrece como asistente de instalación que le guía por el proceso de configuración. Las demás, al cierre de esta edición, son un archivo ZIP (Mac OS) o TGZ (Linux).

En cualquier caso, incluya Android en la que carpeta que use para almacenar programas, en teoría la misma carpeta principal de la carpeta de Eclipse. En mi caso, uso `Program Files (x86)`. El SDK incluye un administrador SDK y AVD que se usa para actualizar y ampliar el SDK una vez instalado. Este administrador, que se ejecuta desde Eclipse, permite actualizar el SDK con las novedades publicadas de Android. Una vez instalado el SDK, ya puede ejecutar Eclipse.

Instalar el complemento Android para Eclipse

El IDE Eclipse admite diversos lenguajes de programación y tecnología pero no todos se admiten desde el principio. El IDE se mejora mediante complementos que ofrecen la funcionalidad necesaria.

Eclipse necesita un complemento para integrar el IDE con el SDK de Android. El complemento añade comandos de menú a la interfaz del IDE relacionados con Android y permite crear y administrar aplicaciones de Android.

Siga los pasos descritos a continuación:

1. Inicie Eclipse desde el acceso directo creado antes o desde la carpeta en la que se instaló. El programa se abre con diversas ventanas, una barra de menús y una barra de herramientas en la parte superior.
2. Seleccione Ayuda>Install New Software (Instalar nuevo software). Se abrirá el asistente Instalar, que le permite buscar e instalar complementos para Eclipse. Los complementos se descargan desde repositorios de software pero Eclipse debe conocer primero la ubicación del repositorio antes de poder buscar los complementos.
3. Pulse el botón **Añadir** para abrir el cuadro de diálogo Add Repository (Añadir repositorio).
4. Deje en blanco el campo Nombre. En el campo Ubicación, introduzca la dirección Web **http://dl-ssl.google.com/android/eclipse/** y pulse **Aceptar**. En la ventana Instalar aparecerá la entrada Developer Tools (Herramientas de desarrollador) (véase la figura C.1).

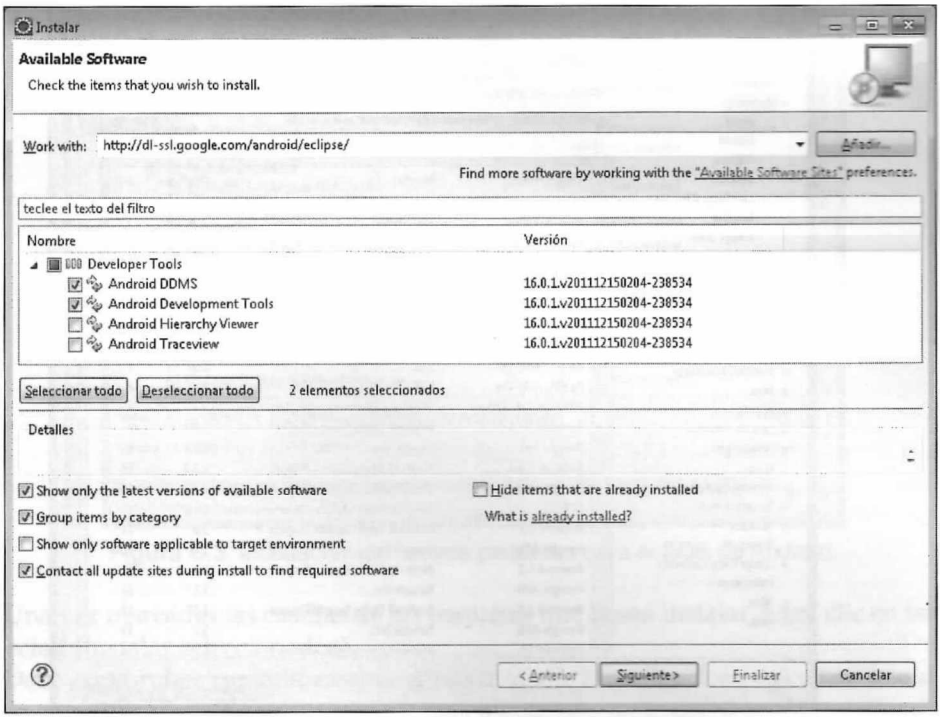


Figura C.1. Nuevos complementos para Eclipse.

5. Haga clic en la flecha de esta entrada para desplegarla. Verá diversas opciones de herramientas relacionadas con Android que puede añadir a Eclipse, como se aprecia en la imagen.
6. Marque las casillas de verificación Android DDMS y Android Development Tools (también puede añadir las demás pero no las analizaremos).
7. Pulse **Siguiente** para confirmar el acuerdo de licencia y los elementos que instalar. Al llegar al final del asistente, pulse **Finalizar**.

Tras instalar el complemento, cierre y reinicie Eclipse. Debe comprobar sus preferencias en Eclipse para asegurarse de que el IDE puede localizar el SDK de Android. Para ello, siga los pasos descritos a continuación:

1. Seleccione Ventana>Preferencias. Se abrirá el cuadro de diálogo Preferencias con una lista de categorías en la parte izquierda.
2. Haga clic en Android para ver las preferencias generales de Android.
3. Compruebe que el campo SDK Location (Ubicación del SDK) contiene la carpeta en la que ha instalado el SDK de Android. En caso contrario, pulse **Examinar** para desplazarse hasta esa carpeta y seleccionarla.
4. Una vez localizado el SDK, verá una tabla con una lista de destinos SDK.

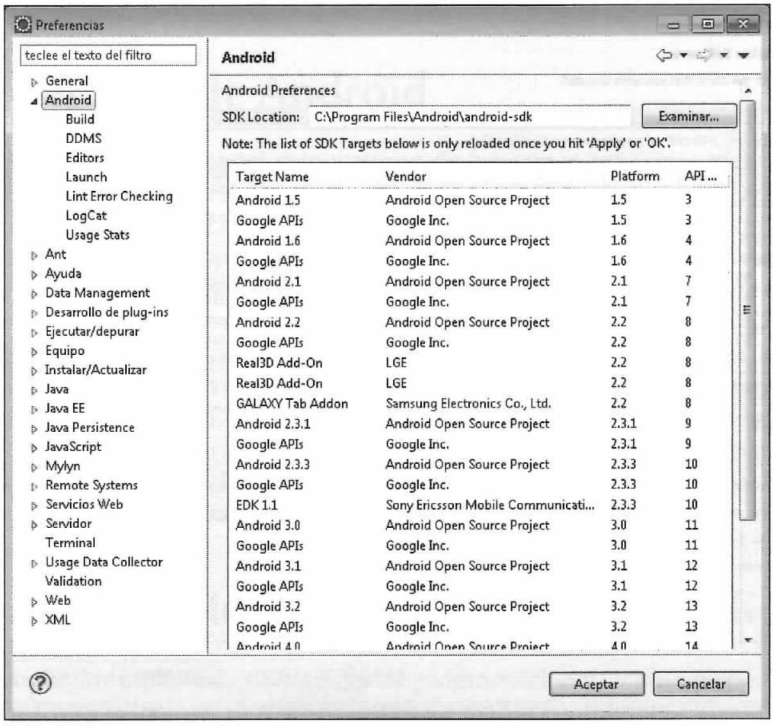


Figura C.2. Configuración de preferencias de Android en Eclipse.

Estos destinos son las versiones de Android para las que puede crear aplicaciones con el SDK. Las aplicaciones de Android deben indicar la versión más antigua para la que se han creado.

Pulse **Aceptar** para cerrar el cuadro de diálogo y guardar sus preferencias.

Una vez instalado el complemento de Android y tras localizar correctamente el SDK, aparecerán nuevos comandos de menú en Eclipse, como Ventana>Android SDK Manager (Administrador del SDK) y Ventana>AVD Manager (Administrador de DVA). En caso de que no aparezcan, cierre y reinicie Eclipse.

Puede usar el administrador para mantener actualizado el SDK. Seleccione Ventana>Android SDK Manager para abrir el administrador.

Marque Installed (Instalados) para ver los componentes del SDK instalados en su equipo. Marque Updates/New (Actualizaciones/Novedades) para ver lo que no está instalado.

Marque la casilla de uno de los paquetes. Eclipse comprueba su contenido y muestra casillas de verificación para los elementos concretos que debe instalar (véase la figura C.3).

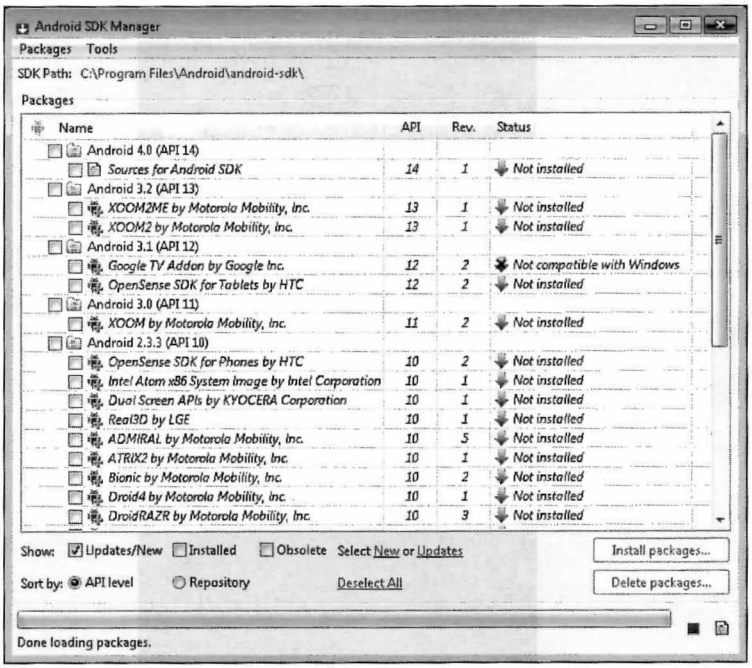


Figura C.3. Instalación de nuevos paquetes para el SDK de Android.

Una vez marcadas las casillas de los paquetes que desea instalar, haga clic en **Install Selected** (Instalar seleccionados).

Debe comprobar periódicamente si hay nuevas actualizaciones. Android se desarrolla a un ritmo frenético debido a la aparición de nuevos teléfonos y dispositivos en el mercado que el SDK debe admitir.

Configurar su teléfono

El SDK de Android incluye un emulador que actúa como un teléfono Android y que puede ejecutar las aplicaciones que diseñe. Resulta muy útil para probar las aplicaciones pero llegará el momento de ver cómo funcionan en un teléfono real (u otro dispositivo).

Puede implementar las aplicaciones que cree con el SDK en un dispositivo Android a través de una conexión USB a su equipo. Puede usar el mismo cable que utiliza para transferir archivos desde y hacia el dispositivo.

Antes de conectar el cable, debe habilitar la depuración USB en el teléfono. Siga los pasos descritos a continuación:

1. En la pantalla Inicio del teléfono, seleccione Menú>Ajustes para abrir la aplicación Ajustes.
2. Seleccione Aplicaciones>Desarrollo y marque la casilla Depuración de USB como se muestra en la figura C.4.

Figura C.4. Un teléfono Android conectando el modo de depuración USB.

Otros dispositivos pueden incluir esta opción en otra parte de sus ajustes, con un nombre como modo de conexión USB, depuración de USB o similar. El sitio de Android (`http://developer.android.com`) incluye documentación para configurarlo en distintos dispositivos Android.

Conecte el cable USB a su equipo y a su teléfono. Aparecerá un icono de Android con forma de insecto en la parte superior del dispositivo, junto a la hora y los iconos de cobertura y batería.

Arrastre la barra hacia abajo. Verá los mensajes Dispositivo de depuración USB Conectado y USB Conectado

Este proceso configura el teléfono, pero también debe configurar el ordenador para poder conectarlo al dispositivo. Si nunca antes se ha conectado al teléfono con un cable USB, consulte la documentación del teléfono. Puede que tenga que instalar un controlador desde un CD o desde el sitio Web del fabricante.

En Windows, el administrador del SDK, que se ejecuta desde Eclipse, le permite descargar un paquete de controladores USB para distintos teléfonos, además de otros paquetes relacionados con su dispositivo.

En el capítulo 24 usamos las herramientas de desarrollo de Android para crear y ejecutar una aplicación de Android. Si ha configurado todo correctamente, podrá ejecutarla en el emulador y en un teléfono Android.

Índice alfabético

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

PROGRAMACIÓN

Java, que comenzó como un programa que se ejecutaba incrustado en los navegadores, hoy en día se encuentra en los servidores de algunos de los portales más grandes de la Web, dirigiendo aplicaciones dinámicas con complejas bases de datos relacionales. Este lenguaje de programación orientado a objetos, permite desarrollar tanto aplicaciones Web, como de escritorio e incluso trabajar con Android.

El libro tiene un enfoque paso a paso, que permite dominar los conceptos y las tecnologías más importantes. Abarca la última versión de Java 7, siendo útil a todos los usuarios: a los que no son programadores, a los programadores noveles y para aquellos que tienen experiencia y desean actualizarse. Aprenda a crear aplicaciones de Java con herramientas de edición visual gratuitas como *NetBean*.

Los textos van acompañados de descriptivas imágenes e instrucciones que le muestran cómo programar. Para probar sus conocimientos, realice los ejercicios prácticos que encontrará al final de cada capítulo. Con un día que le dedique al libro, podrá ponerse manos a la obra y empezar a crear sus propios programas.

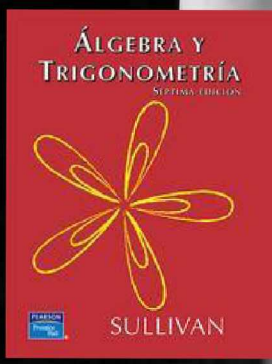
Java 7

- Configurar el entorno de programación de Java.
- Crear su primer programa funcional en cuestión de minutos.
- Controlar decisiones y comportamientos de un programa.
- Diseñar sencillas interfaces de usuario.
- Crear programas Web interactivos.
- Usar subprocesos para diseñar programas con mayor capacidad de respuesta.
- Leer y escribir archivos de datos en XML.
- Crear servicios Web flexibles y compatibles con JAX-WS.
- Usar Java para c[illegible]una aplicación de Android.

Del original:

Sams Teach Yourself Java in 24 Hours SAMS

Rogers Cadenhead @rcade es escritor, programador informático y desarrollador Web. Ha escrito 21 libros sobre Java y temas relacionados con Internet. Es el responsable de *Drudge Retort* y de otros conocidos sitios Web que reciben más de veinte millones de visitas al año.



ANAYA MULTIMEDIA
http://www.anayamultimedia.es