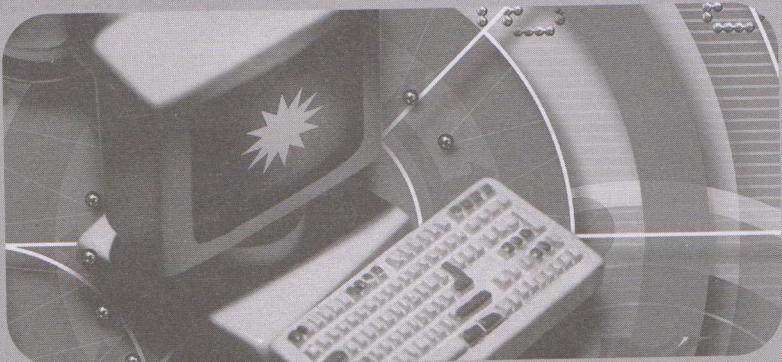


## CAPÍTULO

# 6



## Cómo usar objetos

En este capítulo conoceremos cómo:

- Usar las variables de instancia privadas.
- Usar las clases de biblioteca.
- Usar `new` y los constructores.
- Manejar eventos.
- Usar la clase `Random`.
- Usar las clases etiqueta, campo de texto, panel y botón de Swing.
- Usar las clases control deslizante, temporizador e ícono de imagen de Swing.

### ● Introducción

En este capítulo veremos los objetos a un nivel más detallado. En especial analizaremos el uso de las clases de las bibliotecas de Java. Debe tener en cuenta que, aunque hay muchos cientos de estos objetos, los principios para usarlos son similares.

He aquí una analogía: para leer un libro (cualquiera que sea) hay que abrirlo por su parte frontal, leer una página y después avanzar a la siguiente página. Sabemos qué hacer con un libro. Lo mismo pasa con los objetos. Después de usar unos cuantos de ellos sabemos qué buscar cuando se nos presenta uno nuevo.

## Variables de instancia

Para lidiar con problemas más avanzados necesitamos introducir un nuevo lugar en donde podamos declarar variables. Hasta ahora hemos utilizado las palabras clave `int` y `double` para declarar variables locales dentro de los métodos. Pero las variables locales por sí solas no pueden lidiar con la mayoría de los problemas.

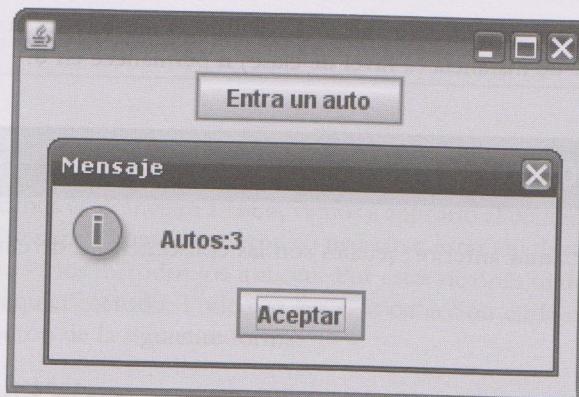
A continuación veremos un programa (`ContadorAutos`, figura 6.1) para ayudar a operar un estacionamiento. Tiene un solo botón, en el que el empleado hace clic a medida que un automóvil entra. El programa lleva la cuenta del número de automóviles en el estacionamiento y lo muestra en un cuadro de mensaje.

En esencia, necesitamos sumar 1 a una variable (a la cual llamaremos `cuentaAutos`) en el método `actionPerformed` asociado con el clic del botón. Sin embargo, es importante mencionar que una variable local (declarada dentro del método `actionPerformed`) **no** funcionará. Las variables locales son temporales; se crean al momento de entrar a un método y se destruyen cuando el método termina. No se preserva el valor que contienen.

He aquí el código correcto (omitimos las partes de la interfaz de usuario en aras de la claridad):

```
public class ContadorAutos extends JFrame
    implements ActionListener {
    private int cuentaAutos = 0;

    public void actionPerformed(ActionEvent event) {
        cuentaAutos = cuentaAutos + 1;
        JOptionPane.showMessageDialog(null, "Autos:" + cuentaAutos);
    }
}
```



**Figura 6.1** Pantalla del programa `ContadorAutos`.

La cuestión aquí es la declaración de la variable `cuentaAutos`:

- Esta variable se declara **fuera** del método, pero **dentro** de la clase `ContadorAutos`. Cualquier método de la clase la puede utilizar (aunque aquí sólo la usamos en `actionPerformed`).
- Se ha declarado como **private**, lo cual significa que cualquier otra clase que pudiéramos llegar a tener no podrá utilizarla. La variable está *encapsulada* o sellada dentro de `ContadorAutos`; es decir, es para que la utilicen los métodos de `ContadorAutos` solamente.
- `cuentaAutos` es un ejemplo de una *variable de instancia*. Pertenece a una instancia de una clase, en vez de pertenecer a un método. Otro término para ella es *variable a nivel de clase*.
- Se dice que `cuentaAutos` tiene *alcance de clase*. El alcance de un elemento es el área del programa en donde se puede utilizar. El otro tipo de alcance que hemos visto es el alcance local, el cual se utiliza con las variables locales que se declaran dentro de los métodos.
- El estilo preferido para las variables de instancia es declararlas como **private**.
- La convención de Java es poner en minúscula la primera letra de una variable de instancia.

El programador tiene la libertad de elegir los nombres para las variables de instancia. Pero, ¿qué pasa si un nombre coincide con el nombre de una variable local, como en el siguiente ejemplo?

```
public class UnaClase {  
    private int n = 8;  
  
    private void miMétodo() {  
        int n;  
        n = 3;           // ¿cuál n?  
    }  
    // otros métodos que omitimos aquí  
}
```

Aunque ambas variables son accesibles (en alcance) dentro de `miMétodo`, la regla es que se elige la variable local. La variable de instancia (a nivel de clase) `n` permanece en 8.

## PRÁCTICA DE AUTOEVALUACIÓN

- 6.1 En la clase `UnaClase` anterior, ¿cuáles son las consecuencias de eliminar la declaración local de `n`?

Las variables de instancia son esenciales, pero no debemos ignorar a las variables locales. Por ejemplo, si una variable se utiliza sólo dentro de un método y no necesitamos mantener su valor entre una llamada y otra, es mejor hacerla local.

He aquí el programa `ContadorAutos` completo, incluyendo el código de la interfaz de usuario:

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class ContadorAutos extends JFrame
    implements ActionListener {

    private int cuentaAutos = 0;

    private JButton botón;

    public static void main(String[] args) {
        ContadorAutos marco = new ContadorAutos();
        marco.setSize(300, 200);
        marco.clearRect();
        marco.setVisible(true);
    }

    private void crearGUI() {
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        Container ventana = getContentPane();
        ventana.setLayout(new FlowLayout());

        botón = new JButton("Entra un auto");
        ventana.add(botón);
        botón.addActionListener(this);
    }

    public void actionPerformed(ActionEvent event) {
        cuentaAutos = cuentaAutos + 1;
        JOptionPane.showMessageDialog(null, "Autos:" + cuentaAutos);
    }
}
```

No caiga en la tentación de enmendar el código de la interfaz de usuario por ahora. Debe estar exactamente como se muestra.

Ahora que introducimos el alcance `private`, vamos a aplicarlo al código de la interfaz de usuario. Los elementos de una ventana (como los botones) necesitan estar ahí durante la vida del programa. Además, es común que varios métodos los utilicen. Por estas razones se declaran como variables de instancia, fuera de cualquier método. Podemos ver esto en acción en la clase `ContadorAutos`, en donde se declara un botón de la siguiente forma:

```
private JButton botón;
```

en la misma área de código que la variable `cuentaAutos`. Más adelante en este capítulo retomaremos la explicación sobre las clases de la interfaz de usuario.

## PRÁCTICA DE AUTOEVALUACIÓN

6.2 ¿Qué hace el siguiente programa? (Omitimos de manera intencional la creación de los objetos de la GUI para que se pueda enfocar en los alcances).

```
private int x = 0;

public void actionPerformed(ActionEvent event) {
    Graphics paper = panel.getGraphics();
    Graphics papel = panel.getGraphics();
    x = x + 10;
}
```

### ● Instanciación: uso de constructores con new

Hasta aquí, hemos escrito programas que utilizan los tipos `int` y `double`. A éstos se les considera como tipos “integrados” o “primitivos”: **no** son instancias de clases (es decir, no son objetos). Recuerde que podemos declararlos y proveer un valor inicial, como en:

```
int n = 3;
```

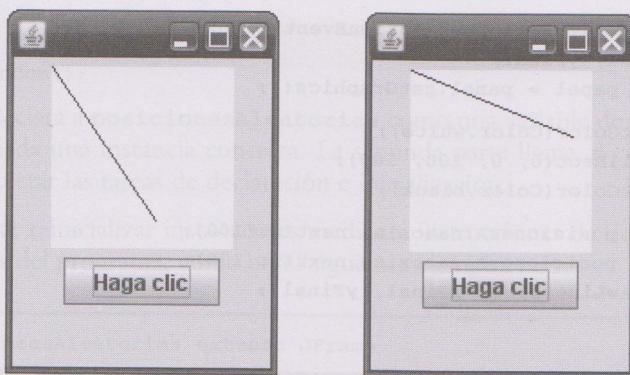
Lo que estamos diciendo es: “crea un nuevo entero llamado `n`, con un valor inicial de 3”.

Sin embargo, hemos usado otros tipos de elementos (como botones y áreas para dibujar gráficos). Éstos **son** instancias de clases. Tenemos que crearlos de una manera especial, mediante la palabra `new`. Al proceso de crear una instancia con `new` se le conoce como *instanciación*.

Para ilustrar el uso de `new` vamos a estudiar la clase `Random` de la biblioteca de Java.

### ● La clase Random

Los números aleatorios son muy útiles en simulaciones y juegos; por ejemplo, podemos proporcionar al jugador una situación inicial distinta cada vez que juegue. Las instancias de la clase `Random` nos proporcionan un “flujo” de números, los cuales podemos obtener uno a la vez mediante el método `nextInt`. A continuación veremos un programa ([LíneasAleatorias](#)) que dibuja una línea aleatoria cada vez que hacemos clic en el botón. Un extremo de la línea está fijo en  $(0, 0)$  y el otro extremo tiene una posición  $x$  e  $y$  aleatoria. Antes de dibujar la línea borramos el área de dibujo, para lo cual dibujamos un rectángulo blanco que abarque toda el área de dibujo (en este ejemplo, de 100 por 100) y después establecemos el color en negro. En la figura 6.2 se muestran dos pantallas y he aquí el código:



**Figura 6.2** Dos pantallas del programa LíneasAleatorias.

```

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.util.*;

public class LíneasAleatorias extends JFrame
    implements ActionListener {

    private Random posicionesAleatorias = new Random();
    private JButton botón;
    private JPanel panel;

    public static void main(String[] args) {
        LíneasAleatorias marco = new LíneasAleatorias();
        marco.setSize(150, 200);
        marco.crearGUI();
        marco.setVisible(true);
    }

    private void crearGUI() {
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        Container ventana = getContentPane();
        ventana.setLayout(new FlowLayout());

        panel = new JPanel();
        panel.setPreferredSize(new Dimension(100,100));
        panel.setBackground(Color.white);
        ventana.add(panel);

        botón = new JButton("Haga clic");
        ventana.add(botón);
        botón.addActionListener(this);
    }
}

```

```

public void actionPerformed(ActionEvent event) {
    int xFinal, yFinal;
    Graphics panel = panel.getGraphics();

    panel.setColor(Color.white);
    panel.fillRect(0, 0, 100, 100);
    panel.setColor(Color.black);

    xFinal = posicionesAleatorias.nextInt(100);
    yFinal = posicionesAleatorias.nextInt(100);
    panel.drawLine(0, 0, xFinal, yFinal);
}
}

```

Para usar `Random` de una manera conveniente, necesitamos esta instrucción de importación:

```
import java.util.*;
```

Si omitiéramos la instrucción de importación tendríamos que hacer referencia a la clase de esta manera:

```
java.util.Random
```

El uso de `import` nos permite ser más directos.

Después debemos declarar e inicializar una instancia de nuestra clase. Esto podemos hacerlo de dos maneras. Uno de los métodos sería utilizar una sola instrucción, como en el siguiente ejemplo:

```
private Random posicionesAleatorias = new Random();
```

Observe que:

- Elegimos el alcance `private` en vez del local.
- Después de `private` indicamos la clase del elemento que vamos a declarar. En este caso el elemento es una instancia de la clase `Random`.
- Elegimos el nombre `posicionesAleatorias` para nuestra instancia. Otros nombres apropiados para nuestra instancia serían `númerosAleatorios`, `aleatorio`. También existe la posibilidad de usar `random` como el nombre de una instancia. Java es susceptible al uso de mayúsculas y minúsculas; la convención es que todos los nombres de las clases empiecen con mayúscula. Por lo tanto, `random` (con minúscula) se puede usar para el nombre de una instancia. En un programa en donde sólo vamos a crear una instancia a partir de una clase, es común usar el mismo nombre que la clase (pero con la primera letra minúscula). En este ejemplo, `posicionesAleatorias` transmite el significado deseado. Utilizamos plural debido a que nuestra instancia puede proveer todos los números aleatorios que necesitemos.
- La palabra `new` se antepone al uso del constructor, que en esencia es un método con el mismo nombre que la clase: `Random`. El uso de `new` crea una nueva instancia de una clase en la memoria de acceso aleatorio, y la asigna a `posicionesAleatorias`.
- Los constructores se pueden sobrecargar, por lo que necesitamos elegir el constructor más conveniente. `Random` tiene dos constructores con distintos parámetros; en este caso es apropiado usar el que no tiene parámetros.
- Podemos considerar que la instrucción consta de dos partes:

```
private Random posicionesAleatorias...
```

y:

```
... = new Random();
```

La primera parte declara a **posicionesAleatorias** como una variable de la clase **Random**, pero aún no tiene asociada una instancia concreta. La segunda parte llama al constructor de la clase **Random** para completar las tareas de declaración e inicialización.

Otra forma de declarar e inicializar instancias es mediante instrucciones de declaración e inicialización en distintas áreas del programa, como en el siguiente ejemplo:

```
public class LíneasAleatorias extends JFrame
    implements ActionListener {

    private Random posicionesAleatorias;
    ...
    private void unMetodo() {
        posicionesAleatorias = new Random();
        ...
    }
}
```

Si importar el método que seleccionemos, debemos tener en cuenta varios puntos:

- La declaración establece la clase de la instancia. En este caso es una instancia de **Random**.
- La declaración establece el alcance de la instancia. En este caso, **posicionesAleatorias** tiene alcance de clase; se puede utilizar en cualquier método de la clase **LíneasAleatorias** en vez de ser local para un método.
- **posicionesAleatorias** es privada. No se puede utilizar en otras clases que no sean nuestra clase **LíneasAleatorias**. Por lo general, todas las variables de instancia las hacemos privadas.
- La forma de declaración de una sola instrucción es conveniente, pero no siempre se utiliza. Algunas veces necesitamos inicializar una variable de instancia en una etapa posterior, con base en los valores calculados por el programa. Esto se debe hacer dentro de un método, aun cuando la declaración se coloque fuera de los métodos. Más adelante en el capítulo mostraremos ejemplos sobre cómo separar la inicialización de la declaración.

## PRÁCTICA DE AUTOEVALUACIÓN

6.3 ¿Cuál es el error en el siguiente fragmento de código?

```
public class SomeClass extends JFrame
    implements ActionListener {

    private Random r;
    r = new Random();
    ...
}
```

Ahora regresemos al programa `LíneasAleatorias`. Hasta aquí sólo hemos creado un objeto; es decir, una instancia de la clase `Random` llamada `posicionesAleatorias`. Aún no hemos creado números aleatorios reales.

Una vez que creamos un objeto con `new`, podemos utilizar sus métodos. La documentación nos indica que hay varios métodos que nos proporcionan un número aleatorio, por lo que elegimos el método que provee enteros y nos permite especificar el rango de los números. Este método se llama `nextInt` (debido a que obtiene el siguiente número aleatorio de una secuencia de números). En nuestro programa colocamos lo siguiente:

```
xFinal = posicionesAleatorias.nextInt(100);
yFinal = posicionesAleatorias.nextInt(100);
```

Se eligió el rango de números aleatorios (100 en este caso) de manera que fuera adecuado para el tamaño del área de dibujo.

Para finalizar, declaramos una instancia de la clase apropiada (`Random`) y utilizamos `new` para crearla e inicializarla. Se pueden combinar o separar estas dos etapas, dependiendo del programa específico en el que estemos trabajando. Después utilizamos el método `nextInt` de la instancia creada.

Vamos a ampliar nuestro estudio sobre la clase `Random`. Analizaremos sus constructores y sus métodos más útiles.

En primer lugar hay dos constructores. En el ejemplo anterior usamos el que no tiene parámetros. Sin embargo, el constructor está sobrecargado: hay otra versión con un solo parámetro. He aquí los dos constructores en uso:

```
private Random random = new Random();
private Random randomIgual = new Random(1000);
```

La primera versión produce una secuencia aleatoria distinta cada vez que ejecutamos el programa. La segunda versión deriva la secuencia aleatoria del número que suministramos, que puede ser cualquier valor. En este caso ocurre la misma secuencia aleatoria cada vez. Podríamos usar esta segunda forma si quisieramos realizar muchas ejecuciones de prueba con la misma secuencia.

La figura 6.3 muestra los métodos más útiles.

Vamos a considerar el método `nextInt` con más detalle. En el programa `LíneasAleatorias` utilizamos las siguientes líneas:

```
xFinal = posicionesAleatorias.nextInt(100);
yFinal = posicionesAleatorias.nextInt(100);
```

<code>nextInt(int n)</code>	Devuelve un valor <code>int &gt;= 0 y &lt; n</code>
<code>nextDouble()</code>	No tiene parámetros. Devuelve un valor <code>double &gt;= 0.0 y &lt; 1.0</code>

Figura 6.3 Métodos de la clase `Random`.

Esto producirá un valor aleatorio en el rango de 0 a 99. El valor 100 nunca ocurrirá. Esto se debe a que la especificación de `nextInt` indica “menor que” en vez de “menor o igual que”. Ésta es una causa común de los errores de programación, debido a que la mayoría de los problemas se declaran con rangos inclusivos, como en “al lanzar un dado se obtiene un número del 1 al 6”, o “las cartas del juego se enumeran del 2 al 10, excluyendo los ases”. Se aplican advertencias similares para `nextDouble`, que nunca producirá un valor exacto de 1.0.

Ahora vamos a escribir un método que simplifica el uso de los números aleatorios. Tiene dos parámetros que nos permiten especificar los valores mínimos y máximos, ambos incluidos, de nuestros números. He aquí el código:

```
private int aleatorioEnRango(int min, int max) {  
    return min+random.nextInt(max-min+1);  
}
```

Para simular el lanzamiento de un dado, podríamos usar lo siguiente:

```
int suertudo;  
suertudo = aleatorioEnRango(1, 6);
```

Al usar una clase, es importante comprender las herramientas que proporcionan sus métodos y constructores. Algunas veces la documentación que se incluye en los sistemas de Java es bastante difícil de comprender, por lo que en el apéndice A sintetizamos todas las clases que utilizamos en este libro.

## PRÁCTICA DE AUTOEVALUACIÓN

- 6.4 ¿Cómo podríamos llamar a `aleatorioEnRango` para obtener una edad aleatoria en el rango de 16 a 59, ambos incluidos?

### ● El método `main` y `new`

Ya hemos visto el uso de `new` para crear una nueva instancia de una clase, la cual posteriormente podemos usar a través de sus métodos. Pero si hacemos memoria y regresamos a los detalles de sus programas, podrá ver que todos ellos son clases de la forma:

```
public class UnNombre...{  
    private declaraciones...  
    una serie de métodos...  
}
```

En sentido informal hemos hablado de “escribir un programa”, aunque de hecho deberíamos decir “escribir una clase”. Pero, ¿es el programa una clase, o una instancia de una clase?

Recuerde que el sistema de Java llama de manera automática al método `main` antes de que ocurra cualquier otra cosa. Analice cualquiera de nuestros métodos `main`. Su primera tarea es usar `new` para crear una instancia de la clase que lo contiene.

Vamos a seguir hablando de programas, ya que esto es más natural. Para responder nuestra pregunta sobre los programas y las clases: un programa en ejecución es una instancia de una clase. Se genera con la instrucción `new` dentro del método `main`.

## ● El kit de herramientas Swing

Al crear Java se incluyó un conjunto de clases que contienen componentes de interfaz de usuario, como botones, barras de desplazamiento, etc. A este conjunto de clases se le denominó Kit de herramientas de Ventanas Abstractas (AWT). Sin embargo, algunos de los componentes eran de una calidad bastante inferior; además, se veían diferente en distintas plataformas debido a que utilizaban los componentes que proveía el sistema operativo en uso. En el apéndice B se incluyen las generalidades sobre el AWT.

Con el propósito de mejorar esta situación, se escribió un conjunto de componentes en Java para proveer más herramientas que se vieran idénticas en cualquier plataforma. A estas clases se les denominó kit de herramientas Swing. En los ejercicios del libro hemos usado con frecuencia la clase `JButton`; la `J` indica que la clase está escrita en Java.

Aunque Swing ofrece más poder, de todas formas necesitamos partes del antiguo AWT, como puede ver en las instrucciones `import` que colocamos en la parte superior de nuestros programas.

## ● Eventos

En secciones anteriores vimos la clase `Random`, cómo crear nuevas instancias y manipularlas con sus métodos. Seguiremos esta metodología para muchas clases. Sin embargo, hay otras clases (como `JButton`) que son distintas, ya que involucran eventos. Vamos a analizar esta clase con detalle y después generalizaremos el uso de las clases, para que usted pueda usar cualquier otra clase con la que se encuentre.

Hemos usado eventos en muchos de los programas que hemos visto. Creamos un botón y colocamos el código en el método `actionPerformed` para responder al evento. Aquí hablaremos sobre los eventos con más detalle.

En Java, los eventos se dividen en categorías. Por ejemplo, tenemos:

- Eventos de “acción”, como hacer clic en un botón.
- Eventos de “cambio”, como ajustar la posición de un control deslizable para cambiar el volumen del altavoz de una computadora.

Recordemos la siguiente línea del programa `ContadorAutos`:

```
public class ContadorAutos extends JFrame  
    implements ActionListener {
```

La palabra clave `extends` expresa herencia, tema que veremos en el capítulo 10. La palabra clave `implements` se puede usar para proveer el manejo de eventos. He aquí una analogía. Suponga que tiene una tarjeta SuperCrédito y que ve un anuncio afuera de una tienda, el cual menciona que se aceptan tarjetas SuperCrédito. Usted supone que al hacer una compra la tienda le proporcionará las herramientas que necesita, como una máquina adecuada para procesar su tarjeta de crédito. En otras palabras, la tienda implementa la interfaz de SuperCrédito. En el capítulo 23 hablaremos sobre las interfaces.

Al usar `implements ActionListener`, estamos declarando que nuestro programa implementa la interfaz `ActionListener`. Para ello debemos proveer un método llamado `actionPerformed`, el cual se llamará cuando ocurra un evento de acción, como el clic de un botón.

Además, tenemos que registrar el programa como un “escuchador” (o “detector”) para los tipos de eventos. A continuación veremos cómo hacer esto.

## ● Creación de un objeto JButton

Aquí vamos a ver el proceso de crear un botón. Al igual que cuando usamos la clase `Random`, debemos declarar e inicializar el objeto antes de usarlo. Además, debemos implementar la interfaz `ActionListener` y registrar el programa como escuchador para los eventos de acción. He aquí otra vez el programa `ContadorAutos` para analizarlo:

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class ContadorAutos extends JFrame
    implements ActionListener {
    private int cuentaAutos = 0;
    private JButton botón;
    public static void main(String[] args) {
        ContadorAutos marco = new ContadorAutos();
        marco.setSize(300, 200);
        marco.crearGUI();
        marco.setVisible(true);
    }
    private void crearGUI() {
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        Container ventana = getContentPane();
        ventana.setLayout(new FlowLayout());
        botón = new JButton("Entra un auto");
        ventana.add(botón);
        botón.addActionListener(this);
    }
    public void actionPerformed(ActionEvent event) {
        cuentaAutos = cuentaAutos + 1;
        JOptionPane.showMessageDialog(null, "Autos:" + cuentaAutos);
    }
}
```

Figura 6.4: captura del programa `ContadorAutos`.

El proceso es el siguiente:

- Primero declaramos que nuestro programa implementa la interfaz `ActionListener`:

```
public class ContadorAutos extends JFrame  
    implements ActionListener {
```

Para ello tenemos que escribir un método llamado `actionPerformed`, como se muestra en el código.

- A continuación declaramos el botón, como una variable de instancia:

```
private JButton botón;
```

- Después creamos el botón y proveemos el texto que aparecerá dentro de él en su constructor. Podemos hacer esto al mismo tiempo que la declaración, pero en este caso elegimos agrupar todo el código relacionado con la inicialización en un método, al cual llamamos `crearGUI`. La inicialización es:

```
botón = new JButton("Entra un auto");
```

- El siguiente paso es agregar un botón a una instancia de la clase `Container` llamada `ventana`:

```
ventana.add(botón);
```

Cabe mencionar que `add` es un método de `ventana` y no de `botón`. El método `add` coloca los elementos en la pantalla, en orden de izquierda a derecha. Los elementos en una fila están centrados. Cuando se llena una fila, empieza de manera automática una nueva fila de objetos (a este esquema se le conoce como “esquema de flujo”). Hay otro esquema conocido como “esquema de borde”, que veremos en el apéndice A).

- A continuación registramos el programa como escuchador para los eventos de acción provenientes del botón:

```
botón.addActionListener(this);
```

- He aquí el método manejador de eventos, en donde colocamos el código para responder al evento:

```
public void actionPerformed(ActionEvent event) {
```

Como puede ver, el proceso es bastante complicado. La buena noticia es que el proceso es casi idéntico cada vez que lo utilizamos.

Hay ciertos elementos arbitrarios que elegimos en el programa. Por ejemplo:

- El nombre de la instancia de `JButton`; elegimos `botón`.
- El texto que se muestra en el botón; elegimos `Entra un auto`.

Hay varias partes esenciales del programa que no podemos cambiar:

- Las instrucciones `import`.
- La instrucción `implements ActionListener`.
- Un método `actionPerformed`.
- El uso de `addActionListener` para registrarse como escuchador del botón.
- El método `main` y las partes de `crearGUI` que establecen el cierre del marco exterior y la disposición de los objetos.

Al decir que no podemos cambiar estas partes, hablamos en serio. No invente sus propios nombres, como `clickPerformed`.

## ● Lineamientos para usar objetos

Ta vimos cómo se pueden incorporar a los programas las instancias de la clase `Random` y la clase `JButton`. Ahora estamos en posición de retroceder y proveerle algunos lineamientos generales. Despúes aplicaremos estos lineamientos a las clases `JLabel`, `JTextField`, `JSlider`, `JPanel`, `Timer` e `ImageIcon`.

He aquí la metodología a seguir:

1. Examine la documentación de la clase en cuestión. Determine la instrucción `import` requerida. Algunas veces habrá que importar clases adicionales, así como la clase en cuestión (como en el caso de `JSlider` que veremos más adelante).
2. Seleccione un constructor para usarlo con `new`.
3. Si la clase es un componente de una interfaz de usuario, agréguela a la ventana.
4. Una vez que haya declarado y creado la instancia con `new`, úsela a través de sus métodos.

Ahora examinaremos otras clases útiles. No hay nuevas herramientas de Java requeridas; hemos visto cómo incorporar clases y crear instancias. Sin embargo, estas clases son muy útiles y aparecen en muchos de los siguientes capítulos.

## ● La clase JLabel

La clase `JLabel` nos permite mostrar texto que no cambia, como instrucciones o indicadores para el usuario. Si deseamos texto que cambie (como el resultado de un cálculo) debemos usar un objeto `JTextField`. El programa `SumarCamposTexto` (figura 6.4) muestra cómo se usa para visualizar el carácter = fijo. Ahora examinemos su uso:

- Es un componente de Swing, por lo que basta con usar nuestra instrucción `import` normal.
- He aquí un ejemplo de su constructor, en el cual proveemos el texto a mostrar:

```
etiquetaIgual = new JLabel(" = ");
```

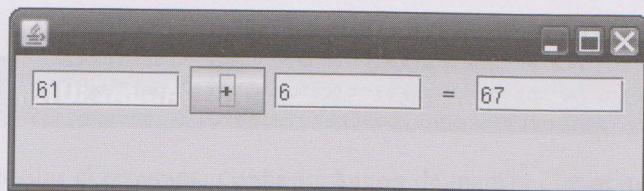


Figura 6.4 Pantalla del programa `SumarCamposTexto`.

- Se agrega a la ventana en forma similar a un botón.
- No produce eventos y es poco probable que el programa lo manipule de nuevo, una vez que se agregue a la ventana.

He aquí el código de **SumarCamposTexto**, que suma dos valores al hacer clic en el botón +:

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class SumarCamposTexto extends JFrame
    implements ActionListener {

    private JTextField campoNúmero1, campoNúmero2, campoSuma;
    private JLabel etiquetaIgual;
    private JButton botónSuma;

    public static void main(String[] args) {
        SumarCamposTexto marco = new SumarCamposTexto();
        marco.setSize(350, 100);
        marco.crearGUI();
        marco.setVisible(true);
    }

    private void crearGUI() {
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        Container ventana = getContentPane();
        ventana.setLayout(new FlowLayout());

        campoNúmero1 = new JTextField(7);
        ventana.add(campoNúmero1);

        botónSuma = new JButton("+");
        ventana.add(botónSuma);
        botónSuma.addActionListener(this);

        campoNúmero2 = new JTextField(7);
        ventana.add(campoNúmero2);

        etiquetaIgual = new JLabel(" = ");
        ventana.add(etiquetaIgual);

        campoSuma = new JTextField(7);
        ventana.add(campoSuma);
    }

    public void actionPerformed(ActionEvent event) {
        int número1 = Integer.parseInt(campoNúmero1.getText());
        int número2 = Integer.parseInt(campoNúmero2.getText());
        campoSuma.setText(Integer.toString(número1 + número2));
    }
}
```

## ● La clase JTextField

La clase **JTextField** provee un área de una sola línea que se puede usar para mostrar o introducir texto. Examinemos su uso:

- Es un componente de Swing, por lo que basta con usar nuestra instrucción **import** normal.
- Cuando el usuario oprime la tecla “Intro” en un campo de texto, se produce un evento de acción. Si deseamos usar este evento, debemos implementar **ActionListener** y proveer un método **actionPerformed**. En nuestro ejemplo utilizaremos un clic de botón en vez de la tecla “Intro” para iniciar el cálculo.
- He aquí ejemplos de sus constructores:

```
campoTexto1 = new JTextField(15);
campoTexto2 = new JTextField(";Hola!", 15);
```

El primero crea un campo de texto vacío con la anchura especificada (en términos de caracteres) y el segundo también nos permite configurar cierto texto inicial. Cabe mencionar que la fuente predeterminada de los campos de texto es proporcional, por lo que **m** ocupa más espacio que **1**. La anchura de un campo de texto se basa en el tamaño de una **m**.

- Se agrega a la ventana en forma similar a un botón.
- Su contenido se puede manipular mediante los métodos **setText** y **getText**, como en el siguiente ejemplo:

```
String s;
s = campoTexto1.getText();
campoTexto1.setText(s);
```

En el programa **SumarCamposTexto**, el usuario introduce dos enteros en los campos de texto de la izquierda. Al hacer clic en el botón, aparece la suma en el tercer campo de texto. Recuerde la forma en que usamos los cuadros de entrada para introducir números en el capítulo 4. De nuevo, necesitamos convertir en entero la cadena introducida en el campo de texto y, a su vez, convertir en cadena nuestro resultado entero para poder mostrarlo en pantalla. Usaremos los métodos **Integer.parseInt** e **Integer.toString**. He aquí el código:

```
int número1 = Integer.parseInt(campoNúmero1.getText());
int número2 = Integer.parseInt(campoNúmero2.getText());
campoSuma.setText(Integer.toString(número1 + número2));
```

### PRÁCTICA DE AUTOEVALUACIÓN

- 6.5** Vuelva a escribir el programa **ContadorAutos** de manera que se muestre el conteo en un campo de texto en vez de un cuadro de mensaje.

## ● La clase JPanel

El panel se puede usar para dibujar o para contener otros objetos, como los botones. Al crear un área de dibujo, por lo general necesitamos especificar su tamaño en píxeles en vez de dejar que Java la trate de la misma forma que a los botones.

He aquí el código estándar que usamos para crear un panel con un tamaño específico:

```
panel = new JPanel();
panel.setPreferredSize(new Dimension(200, 200));
panel.setBackground(Color.white);
ventana.add(panel);
```

Para usar el panel como área de dibujo en vez de como contenedor para otros objetos, escribimos el siguiente código:

```
Graphics papel = panel.getGraphics();
```

## ● La clase Timer

El temporizador crea eventos de acción espaciados en forma regular, lo cual podemos considerar como el tic de un reloj. Podemos iniciar y detener el temporizador, además de controlar su velocidad. A diferencia de un botón, el temporizador no tiene representación en pantalla. He aquí las principales herramientas del temporizador:

- El temporizador crea tics a intervalos regulares. Cada tic es un evento manejado por el método `actionPerformed`.
- Hay que realizar con cuidado la importación. Hay dos clases `Timer` en estas bibliotecas:

```
java.util
javax.swing
```

Nosotros requerimos la que está en la biblioteca de Swing. Si importamos todas las clases de cada biblioteca y después tratamos de declarar una instancia de la clase `Timer` se producirá un error de compilación. He aquí cómo podemos resolver el conflicto:

- La mayoría de nuestros programas importan todas las clases de `javax.swing`. Si el programa no necesita la biblioteca `java.util`, no hay problema. Declaramos un temporizador de la siguiente manera:

```
private Timer temporizador;
```

- Si el programa necesita ambas bibliotecas (como en el programa `GotasDeLluvia` que veremos a continuación), entonces podemos declarar un temporizador así:

```
private javax.swing.Timer temporizador;
```

Debemos recurrir a esta forma larga cada vez que usemos el nombre de la clase `Timer`.

- El temporizador crea eventos de acción, por lo que especificamos `implements ActionListener` y proveemos un método `actionPerformed`.

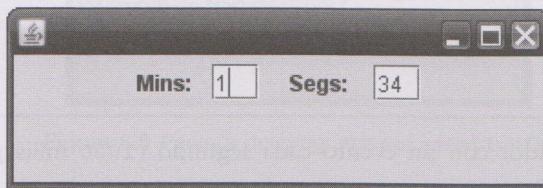
- El constructor para el temporizador requiere dos parámetros:
  - Un entero que especifique el número de milisegundos entre eventos (tics).
  - El programa que se registra para detectar los eventos de acción. Al igual que con los botones, usamos `this`. He aquí un ejemplo:

```
temporizador = new Timer(1000, this);
```

- Podemos iniciar y detener el temporizador mediante los métodos `start` y `stop`.
- El tiempo entre eventos (en milisegundos) se puede modificar mediante `setDelay`, como en:

```
temporizador.setDelay(500);
```

He aquí un programa (`EjemploTimer`) que muestra los minutos y segundos en la pantalla. La figura 6.5 muestra la pantalla resultante y he aquí el código.



**Figura 6.5** Pantalla del programa `EjemploTimer`.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class EjemploTimer extends JFrame
    implements ActionListener {

    private JTextField campoSegs, campoMins;
    private JLabel etiquetaSegs, etiquetaMins;
    private int tics = 0;
    private Timer temporizador;

    public static void main (String[] args) {
        EjemploTimer marco = new EjemploTimer();
        marco.setSize(300,100);
        marco.crearGUI();
        marco.setVisible(true);
    }

    private void crearGUI() {
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        Container ventana = getContentPane();
        ventana.setLayout(new FlowLayout());
```

```

etiquetaMins = new JLabel("Mins: ");
ventana.add(etiquetaMins);

campoMins = new JTextField(2);
ventana.add(campoMins);

etiquetaSegs = new JLabel(" Segs: ");
ventana.add(etiquetaSegs);

campoSegs = new JTextField(2);
ventana.add(campoSegs);

temporizador = new Timer(1000, this);
temporizador.start();

}

public void actionPerformed(ActionEvent event) {
    campoMins.setText(Integer.toString(ticks / 60));
    campoSegs.setText(Integer.toString(ticks % 60));
    ticks = ticks + 1;
}
}

```

Creamos un temporizador con un evento cada segundo (1000 milisegundos). El manejo del evento implica lo siguiente:

- Una variable privada (`ticks`) para contar el número de tics.
- Calcular los minutos mediante una división entre 60.
- Usar % para evitar que el despliegue de los segundos pase de 59.
- Actualizar los campos de texto.
- Incrementar el contador.

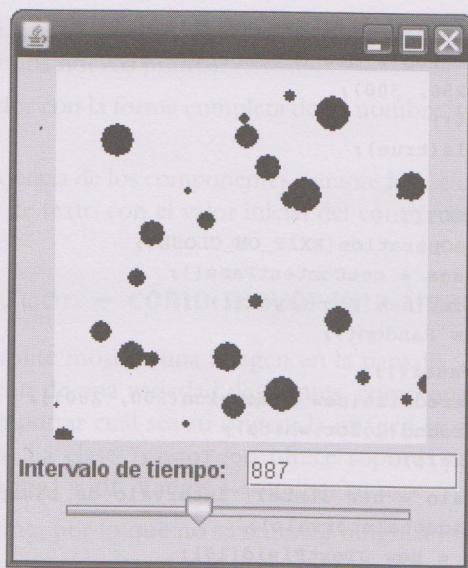
## PRÁCTICA DE AUTOEVALUACIÓN

**6.6** Explique por qué la variable `ticks` no puede ser local.

### ● La clase `JSlider`

Nuestra última clase es la del control deslizable. Esta clase provee un control que se puede arrastrar para seleccionar un valor. Podemos verlo en acción en el programa `GotasDeLluvia` de la figura 6.6. He aquí los puntos principales:

- Necesitamos importar la biblioteca `javax.swing.event` para el manejo de eventos.
- Crea eventos de “cambio”. Necesitamos usar `implements ChangeListener` y proveer un método `stateChanged` para manejar los eventos.
- Para crear un control deslizable se requiere proveer cuatro parámetros al constructor:
  - La orientación del control deslizable, que se especifica mediante `JSlider.VERTICAL` o `JSlider.HORIZONTAL`.



**Figura 6.6** Pantalla del programa **GotasDeLluvia**.

- El valor mínimo del control deslizable.
- El valor máximo del control deslizable.
- La posición inicial del deslizador.
- El valor actual se obtiene mediante el método `getValue`.
- El método `stateChanged` se llama cuando el usuario mueve el deslizador. Se crearán varios eventos al momento de arrastrar el deslizador, pero la llamada final ocurrirá cuando el usuario elija un valor.

He aquí el código de **GotasDeLluvia**. En la figura 6.6 aparece la pantalla resultante:

```

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
import java.util.*;

public class GotasDeLluvia extends JFrame implements
    ActionListener, ChangeListener {
    ...
    
```

```

public static void main (String[] args) {
    GotasDeLluvia marco = new GotasDeLluvia();
    marco.setSize(250, 300);
    marco.crearGUI();
    marco.setVisible(true);
}

private void crearGUI() {
    setDefaultCloseOperation(EXIT_ON_CLOSE);
    Container ventana = getContentPane();
    ventana.setLayout(new FlowLayout());
    aleatorio = new Random();
    panel = new JPanel();
    panel.setPreferredSize(new Dimension(200, 200));
    panel.setBackground(Color.white);
    ventana.add(panel);

    etiquetaIntervalo = new JLabel("Intervalo de tiempo: ");
    ventana.add(etiquetaIntervalo);
    campoIntervalo = new JTextField(10);
    ventana.add(campoIntervalo);

    deslizable = new JSlider(JSlider.HORIZONTAL, 200, 2000, 1000);
    ventana.add(deslizable);
    deslizable.addChangeListener(this);
    campoIntervalo.setText(Integer.toString(deslizable.getValue()));
    temporizador = new javax.swing.Timer(1000, this);
    temporizador.start();
}

public void actionPerformed(ActionEvent event) {
    int x, y, tamaño;
    Graphics papel = panel.getGraphics();
    x=aleatorio.nextInt(200);
    y = aleatorio.nextInt(200);
    tamaño = aleatorio.nextInt(20);
    papel.fillOval(x,y, tamaño, tamaño);
}

public void stateChanged(ChangeEvent e) {
    int timeGap = deslizable.getValue();
    campoIntervalo.setText(Integer.toString(timeGap));
    temporizador.setDelay(timeGap);
}
}

```

El programa simula gotas de lluvia de un tamaño aleatorio que caen en una hoja de papel. El usuario puede modificar el tiempo entre cada gota con sólo mover el deslizador.

Cada vez que ocurre un evento del temporizador, el programa dibuja un círculo de tamaño aleatorio en una posición aleatoria. Al mover el deslizador aparece el valor actual del control en un campo de texto y se modifica la velocidad del temporizador. Descubrimos el rango del control des-

lizable de 200 a 2000 mediante la experimentación. El programa usa la mayoría de las clases que hemos examinado, pero hay dos nuevos puntos:

- Declaramos el temporizador con la forma completa de su nombre, ya que se importó la biblioteca `util` para `Random`.
- Explotamos la interdependencia de los componentes durante la fase de inicialización. Establecemos el valor inicial del campo de texto con el valor inicial del control deslizable.

## ● La clase `ImageIcon` – cómo mover una imagen

La clase `ImageIcon` nos permite mostrar una imagen en la pantalla, como se muestra en la figura 6.7. La imagen puede provenir de una variedad de fuentes, como una cámara digital, un escáner o un paquete de dibujo. Sin importar cuál sea su origen, la imagen se debe almacenar en un archivo antes de mostrarla con Java. La clase `ImageIcon` ofrece soporte para una variedad de tipos de archivos, incluyendo `jpeg` (o `jpg`) y `gif`. Vamos a examinar su uso:

- Es un componente de Swing, por lo que no se requiere ninguna instrucción de importación adicional.
- Su constructor tiene un solo parámetro de cadena, que representa el nombre del archivo que contiene la imagen. He aquí un ejemplo:

```
 ImageIcon miImagen = new ImageIcon("miFoto.jpg");
```

En el ejemplo anterior suponemos que la imagen está guardada en la misma carpeta que el archivo de clase que se va a ejecutar. Si usted utiliza un IDE, explore las carpetas de un proyecto para averiguar en dónde almacena los archivos `.class`.

- Una vez creada la instancia de `ImageIcon` con `new`, podemos usar sus métodos. En nuestro ejemplo usamos el método `paintIcon` para mostrar la imagen, como en:

```
 miImagen.paintIcon(this, papel, 10, 10);
```

Los parámetros de `paintIcon` son:

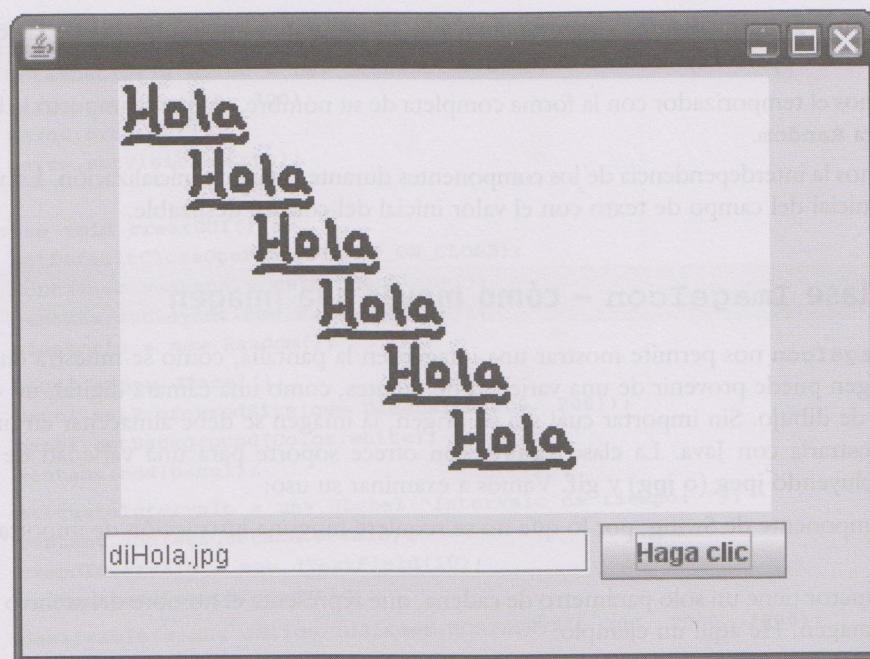
- El objeto actual en ejecución, representado por `this`.
- El área `Graphics` en donde se va a colocar la imagen.
- La posición horizontal (x) de la esquina superior izquierda de la imagen dentro del área de dibujo.
- La posición vertical (y) de la esquina superior izquierda de la imagen dentro del área de dibujo.

Si la imagen es demasiado grande como para caber dentro del área de dibujo, se recorta. Sólo se muestra la parte que se pueda ajustar dentro del área de dibujo.

La figura 6.7 muestra la pantalla de un programa que despliega una pequeña imagen y después la recorre una pequeña distancia en sentido diagonal, cada vez que el usuario hace clic en el botón.

He aquí el código:

```
import java.awt.*;  
import java.awt.event.*;  
import javax.swing.*;
```



**Figura 6.7** Pantalla del programa **ImageDemo**.

```

public class ImageDemo extends JFrame
    implements ActionListener {
    private JButton botón;
    private JPanel panel;
    private JTextField campoArchivo;
    private int posiciónX = 0, posiciónY = 0;

    public static void main (String[] args) {
        DemoImagen marco = new DemoImagen();
        marco.setSize(400, 300);
        marco.crearGUI();
        marco.setVisible(true);
    }

    private void crearGUI() {
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        Container ventana = getContentPane();
        ventana.setLayout(new FlowLayout());

        panel = new JPanel();
        panel.setPreferredSize(new Dimension(300, 200));
        panel.setBackground(Color.white);
        ventana.add(panel);
    }
}

```

```

campoArchivo = new JTextField(20);
ventana.add(campoArchivo);

botón = new JButton("Haga clic");
ventana.add(botón);
botón.addActionListener(this);
}

public void actionPerformed(ActionEvent event) {
    ImageIcon miImagen = new ImageIcon(campoArchivo.getText());
    Graphics papel = panel.getGraphics();
    miImagen.paintIcon(this, papel, posiciónX, posiciónY);
    posiciónX = posiciónX + 30;
    posiciónY = posiciónY + 30;
}
}

```

El programa requiere que el usuario escriba el nombre de un archivo de imagen en un campo de texto. Es necesario colocar la imagen en la misma carpeta que el archivo de clase del programa (puede crear una pequeña imagen con un programa de dibujo, o puede descargar una).

Al hacer clic en el botón, la imagen se vuelve a dibujar en una posición hacia abajo y a la derecha de su posición actual, para lo cual se suma 30 a las variables `posiciónX` y `posiciónY`.

Consideré el alcance de estas variables: su valor se debe preservar entre los clics del botón, por lo que es esencial el alcance privado. Si eligiéramos el alcance local, se perdería su valor cada vez que el programa saliera del método `actionPerformed`.

El programa ofrece la posibilidad de crear animaciones. Si borramos el área de dibujo antes de volver a dibujar y hacemos que el programa vuelva a dibujar la imagen sin esperar a que hagamos clic, lograremos la ilusión de movimiento continuo.

## PRÁCTICA DE AUTOEVALUACIÓN

- 6.7 En el ejemplo `GotasDeLluvia`, la posición actual del deslizador se muestra en un campo de texto. ¿Cuáles serían las consecuencias de modificar la posición inicial del deslizador en la llamada de un constructor de `JSlider`?

## Fundamentos de programación

Durante muchos años el sueño de los programadores ha sido poder crear programas de la misma forma en que se construyen los sistemas de alta fidelidad; es decir, a partir de componentes comerciales tales como bocinas, amplificadores, controles de volumen, etc. Las mejoras en la programación orientada a objetos junto con las extensas bibliotecas de Java hacen que este sueño esté cada vez más cerca de cumplirse.

## Errores comunes de programación

- Si se declara una instancia pero se omite su inicialización con `new`, se producirá un error en tiempo de ejecución del tipo `nullPointerException`, al momento de tratar de utilizar dicha instancia. Los errores en tiempo de ejecución (o “bugs”) son más problemáticos que los errores en tiempo de compilación; son más difíciles de encontrar y son más graves, ya que se detiene la ejecución del programa.
- Los nombres de las clases de GUI de Java empiezan con `J`, como en el caso de `JButton`. Hay clases con nombres similares en la biblioteca AWT, pero sin la `J` (como `Button`). Se produce un error en tiempo de ejecución si utilizamos estas clases. Recuerde la `J`.

## Secretos de codificación

- Las variables de instancia se declaran fuera de los métodos, mediante `private`, como en el siguiente ejemplo:

```
private int suVariable;
private Random miVariable = new Random();
```

- Las variables de instancia se pueden inicializar al momento de su declaración o dentro de un método.

## Nuevos elementos del lenguaje

- Las variables de instancia `private`.
- El uso de `new` para crear instancias.
- La instrucción `import` para facilitar el uso de las bibliotecas.
- Las clases `JButton`, `JLabel`, `JTextField`, `Random`, `JSlider`, `ImageIcon` y `Timer`.

## Resumen

El sistema de Java tiene una gran cantidad de clases que podemos (y debemos) usar. No escriba su propio código sin antes investigar las bibliotecas.

## Ejercicios

- 6.1 Escriba un programa que calcule el área de un rectángulo. Las dimensiones se deben introducir mediante campos de texto y tiene que mostrar el resultado en un campo de texto. Asegúrese de que los campos de entrada estén identificados con claridad.

**6.2** Escriba un programa que produzca un número aleatorio entre 200 y 400 cada vez que se haga clic en un botón. El programa deberá mostrar este número, junto con la suma y el promedio de todos los números hasta ese momento. A medida que usted oprima el botón una y otra vez, el promedio deberá convergir hacia 300. Si no lo hace, entonces sospecharemos del generador de números aleatorios, ¡así como sospecharíamos de una moneda que al tirarla cayera en cara durante 100 veces seguidas!

**6.3** (a) Escriba un programa que convierta los grados Celsius en grados Fahrenheit. El usuario debe introducir el valor Celsius en un campo de texto; utilice valores enteros. Al hacer clic en un botón deberá aparecer el valor Fahrenheit en otro campo de texto. La fórmula de conversión es:

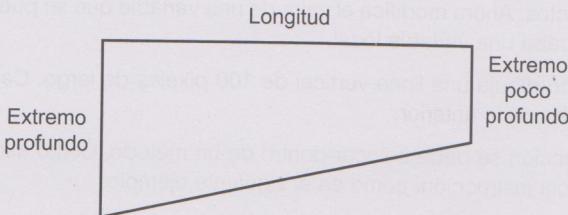
$$f = (c * 9) / 5 + 32;$$

- (b) Modifique el programa de manera que se introduzca el valor Celsius mediante un control deslizable, el cual deberá tener su valor mínimo en 0 y su valor máximo en 100.
- (c) Represente ambas temperaturas como rectángulos largos y delgados que se dibujen después de cada evento de "cambio". Recuerde borrar el área de dibujo y restablecer el color en cada ocasión.

**6.4** Escriba un programa que calcule el volumen de una alberca y muestre su sección transversal en un cuadro de imagen. La anchura de la alberca está fija en 5 metros y la longitud está fija en 20 metros. El programa debe tener dos controles deslizables: uno para ajustar la profundidad del extremo profundo y otro para ajustar la profundidad del extremo poco profundo. La profundidad mínima de cada extremo debe ser de 1 metro. Vuelva a dibujar la alberca en el método **stateChanged**. Seleccione valores apropiados para los valores mínimo y máximo del control deslizable en tiempo de diseño. La fórmula del volumen es:

$$v = \text{profundidadPromedio} * \text{anchura} * \text{longitud};$$

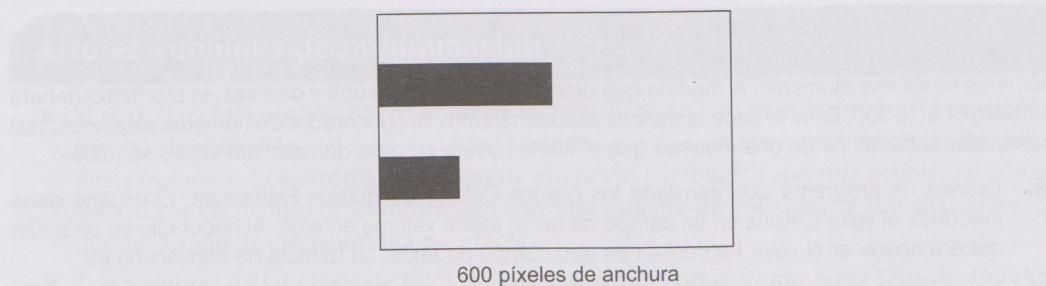
La figura 6.8 muestra la sección transversal de la alberca



**Figura 6.8** Sección transversal de una alberca.

**6.5** Escriba un programa que muestre los minutos y segundos cambiantes, representándolos mediante dos rectángulos largos: haga que la anchura máxima de los rectángulos sea de 600 píxeles para simplificar la aritmética (10 píxeles por cada minuto y cada segundo). Establezca el tamaño del marco en 700 píxeles de ancho y la anchura preferida del panel de dibujo en 700 píxeles. Vuelva a dibujar los dos rectángulos cada segundo. La figura 6.9 muestra una representación de 30 minutos y 15 segundos.

El programa debe contar en segundos con un temporizador y mostrar el total de segundos, además del tiempo en minutos y segundos. Para agilizar la prueba del programa reduzca el intervalo de tiempo de 1000 milisegundos a 200, por ejemplo.



**Figura 6.9** Despliegue del tiempo para 30 minutos con 15 segundos.

- 6.6** Esta pregunta lo llevará a través del proceso de escribir un juego de geometría:
- Escriba un programa con dos controles deslizables que controlen la posición horizontal y vertical de un círculo que tenga 200 píxeles de diámetro. Borre la pantalla y vuelva a dibujar el círculo en el método `stateChanged`.
  - Agregue un tercer control deslizable para controlar el diámetro del círculo.
  - Lo que sigue es un juego basado en el hecho matemático de que se puede dibujar un círculo en base a tres puntos cualesquiera. En un principio el programa debe mostrar tres puntos (cada uno de ellos es un pequeño círculo relleno). Algunas posiciones iniciales convenientes son (100, 100), (200, 200) y (200, 100), pero puede agregar un pequeño número aleatorio a estas posiciones para obtener más variedad. El jugador debe manipular el círculo hasta que éste pase por cada uno de los puntos.

## Respuestas a las prácticas de autoevaluación

- 6.1** El programa se compilará y ejecutará de todas formas, pero es muy probable que produzca resultados incorrectos. Ahora modifica el valor de una variable que se puede usar en otros métodos. Antes modificaba una variable local.
- 6.2** Cada clic del botón dibuja una línea vertical de 100 píxeles de largo. Cada línea se localiza 10 píxeles a la derecha de la anterior.
- 6.3** La segunda instrucción se debe colocar dentro de un método. Como alternativa se podría usar la forma de una sola instrucción, como en el siguiente ejemplo:

```
private Random r = new Random();
```

- 6.4** int edad = aleatorioEnRango(16, 59);

- 6.5** Agregamos un campo de texto a la ventana mediante la misma codificación que en el programa `SumarCamposTexto`. Un nombre apropiado sería `campoConteo`. En vez de mostrar la respuesta en un cuadro de mensaje, usamos lo siguiente:

```
campoConteo.setText(Integer.toString(conteoAutos));
```

- 6.6** Las variables locales se crean de nuevo al entrar a su método y sus valores se borran cuando el método termina. Si `tics` fuera local no se mantendría el conteo.
- 6.7** No hay consecuencias, ya que el valor del campo de texto se inicializa con el valor actual del control deslizable, sin importar cuál sea.