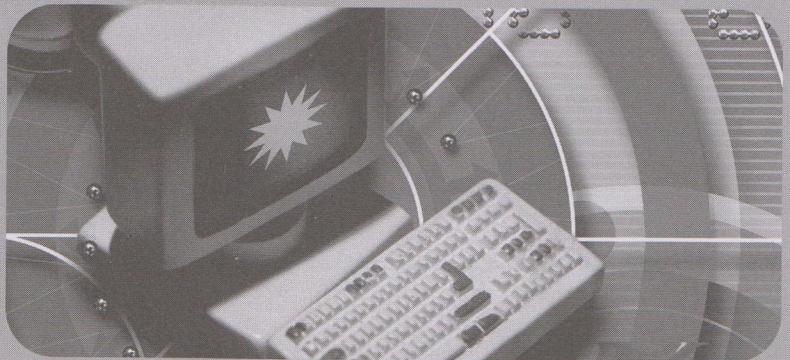


# CAPÍTULO 8



## Repetición

En este capítulo conoceremos cómo:

- Realizar repeticiones mediante instrucciones `while`.
- Realizar repeticiones mediante instrucciones `for`.
- Utilizar los operadores lógicos `&&`, `||` y `!` en ciclos.
- Realizar repeticiones mediante la instrucción `do`.

### ● Introducción

Nosotros los humanos estamos acostumbrados a realizar cosas una y otra vez; por ejemplo: comer, dormir y trabajar. Las computadoras realizan las repeticiones en forma similar. Algunos ejemplos son:

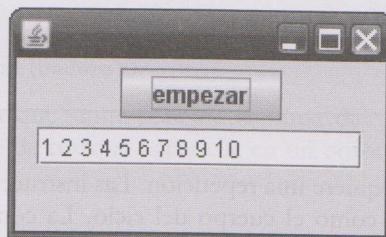
- Sumar una lista de números.
- Buscar cierta información en un archivo.
- Resolver una ecuación matemática en forma iterativa, obteniendo cada vez mejores aproximaciones.
- Hacer que una imagen gráfica se mueva en la pantalla (animación).

Ya hemos visto que una computadora lleva a cabo una secuencia de instrucciones. Ahora veremos cómo repetir una secuencia de instrucciones cierto número de veces. Parte del poder de las computadoras surge de su capacidad para realizar repeticiones con extrema rapidez. En el lenguaje de la programación, a una repetición se le conoce como ciclo.

Hay dos formas principales en las que el programador de Java puede instruir a la computadora para que realice una repetición: `while` y `for`. Podemos utilizar cualquiera de estas dos instrucciones para llevar a cabo repeticiones pero hay diferencias entre ellas, como veremos a continuación.

## while

Primero vamos a utilizar un ciclo para mostrar los enteros del 1 al 10 (figura 8.1) en un cuadro de texto, para lo cual usaremos el siguiente programa:



**Figura 8.1** Pantalla de los números del 1 al 10.

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

public class UnoAlDiez extends JFrame implements ActionListener {

    private JButton botón;
    private JTextField campoTexto;

    public static void main(String[] args) {
        UnoAlDiez demo = new UnoAlDiez();
        demo.setSize(200, 120);
        demo.crearGUI();
        demo.setVisible(true);
    }

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

        botón = new JButton("empezar");
        ventana.add(botón);
        botón.addActionListener(this);

        campoTexto = new JTextField(15);
        ventana.add(campoTexto);
    }

    public void actionPerformed(ActionEvent e) {
        String texto = campoTexto.getText();
        if (botón.getText().equals("empezar")) {
            texto += "1 2 3 4 5 6 7 8 9 10";
            campoTexto.setText(texto);
            botón.setText("parar");
        } else if (botón.getText().equals("parar")) {
            campoTexto.setText("");
            botón.setText("empezar");
        }
    }
}
```

```

public void actionPerformed(ActionEvent event) {
    int número;
    String unoAlDiez = "";
    número = 1;
    while (número <= 10) {
        unoAlDiez = unoAlDiez + Integer.toString(número) + " ";
        número++;
    }
    campoTexto.setText(unoAlDiez);
}

```



La palabra **while** indica que se requiere una repetición. Las instrucciones encerradas entre las llaves se repiten, y a esto se le conoce como el cuerpo del ciclo. La condición entre paréntesis que va inmediatamente después de la palabra **while** controla el ciclo. Si la condición es verdadera, el ciclo continúa. Si es falsa, el ciclo termina y el control se transfiere de vuelta a la instrucción que esté después de la llave de cierre. En este caso el ciclo continúa mientras que cada **número** sea menor o igual que 10.

Antes de iniciar el ciclo, el valor de **número** se hace igual a 1. Al final de cada ciclo, el valor de **número** se incrementa en 1 mediante el operador **++** que vimos en un capítulo anterior. Así, **número** recibe los valores 1, 2, 3, ... hasta el 10.

En un principio la cadena **unoAlDiez** está vacía. Cada vez que se repite el ciclo, se agrega un número (y un espacio) a la cadena mediante el operador de cadena **+**.

El fragmento del programa anterior utiliza el operador menor o igual que (**<=**). Éste es uno de varios operadores de comparación disponibles, los mismos que utilizamos en las instrucciones **if**. He aquí de nuevo la lista completa de los operadores de comparación (a los que se les conoce también como operadores relacionales):

Símbolo	Significado
<b>&gt;</b>	mayor que
<b>&lt;</b>	menor que
<b>==</b>	Igual que
<b>!=</b>	no es igual que
<b>&lt;=</b>	menor o igual que
<b>&gt;=</b>	mayor o igual que

La sangría que se aplica en las instrucciones dentro del ciclo nos ayuda a ver su estructura.

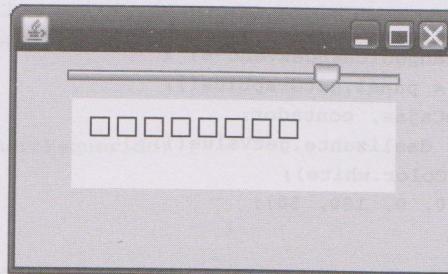
Si su sistema de desarrollo provee un depurador, puede usarlo para seguir con más facilidad la ejecución de este ciclo.

## PRÁCTICAS DE AUTOEVALUACIÓN

**8.1** ¿Qué hace el siguiente fragmento de programa?

```
String cadena = "";
int número = 0;
while (número <= 5) {
    cadena = cadena + Integer.toString(número * número) + " ";
    número++;
}
areaTexto.setText(cadena);
```

**8.2** Escriba un programa que sume (calcule la suma de) los números del 1 al 100 y la muestre en un campo de texto al hacer clic en un botón.



**Figura 8.2** Repetición de cajas utilizando `while`.

El siguiente programa utiliza un ciclo `while` para mostrar una fila de cajas (figura 8.2). El número de cajas se determina con base en el valor seleccionado en un control deslizable. Cada vez que se desplaza el deslizador se genera un evento y el programa muestra el número equivalente de cajas. Para hacer esto necesitamos un contador. El contador, que en un principio es igual a 1, se incrementa en 1 cada vez que se muestra una caja. Necesitamos repetir el proceso de mostrar una caja adicional hasta que el contador llegue al total deseado, para lo cual utilizaremos un ciclo `while` como se aprecia a continuación:

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

public class Cajas extends JFrame implements ChangeListener {
    private JSlider deslizante;
    private JPanel panel;
```

```

public static void main(String[] args) {
    Cajas demo = new Cajas();
    demo.setSize(250,150);
    demo.crearGUI();
    demo.setVisible(true);
}

private void crearGUI() {
    setDefaultCloseOperation(EXIT_ON_CLOSE);
    Container ventana = getContentPane();
    ventana.setLayout(new FlowLayout());
    deslizante = new JSlider(0, 10, 1);
    ventana.add(deslizante);
    deslizante.addChangeListener(this);

    panel = new JPanel();
    panel.setPreferredSize(new Dimension(180, 50));
    panel.setBackground(Color.white);
    ventana.add(panel);
}

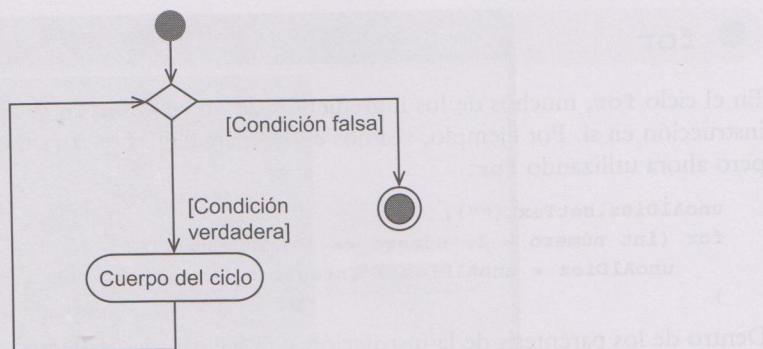
public void stateChanged(ChangeEvent e) {
    Graphics papel = panel.getGraphics();
    int x, númeroDeCajas, contador;
    númeroDeCajas = deslizante.getValue();
    papel.setColor(Color.white);
    papel.fillRect(0, 0, 180, 50);
    x = 10;
    contador = 1;
    papel.setColor( Color.black);
    while (contador <= númeroDeCajas) {
        papel.drawRect(x, 10, 10, 10);
        x = x + 15;
        contador++;
    }
}
}

```



Este programa dibujará todas las cajas que necesitemos. Imagine cuántas instrucciones tendríamos que escribir para mostrar 100 cajas, si no pudiéramos utilizar una instrucción `while`. Tenga presente también que este programa dibujará cero cajas si es lo que el usuario selecciona con el deslizador. Por ende, la instrucción `while` es completamente flexible: nos proporciona tantas o tan pocas repeticiones como requiramos.

Una forma de visualizar un ciclo `while` es por medio de un diagrama de actividades, como se muestra en la figura 8.3. Por lo general la computadora lleva a cabo las instrucciones en secuencia de arriba hacia abajo, como lo indican las flechas. Un ciclo `while` implica que se debe evaluar la condición antes de ejecutar el ciclo, y se debe evaluar otra vez antes de que el ciclo se repita. Si la condición es verdadera, se ejecuta el ciclo. Cuando por fin la condición es falsa, el cuerpo del ciclo deja de ejecutarse y la repetición termina.



**Figura 8.3** Diagrama de actividades del ciclo `while`.

Es conveniente tener mucho cuidado al escribir un ciclo `while` para asegurarnos de que el conteo se realice en forma apropiada. Un error común es hacer que el ciclo se repita demasiadas veces o muy pocas veces. Esto se conoce comúnmente como un error de “desplazamiento por uno” (*off by one*). Algunas veces un ciclo se escribe de manera que empiece con un contador de 0 y la evaluación es para verificar si la condición es menor que el número requerido, como se muestra a continuación:

```

contador = 0;
while (contador < numeroRequerido) {
    // cuerpo
    contador++;
}
  
```

También podemos escribir el ciclo de manera que empiece con un contador de 1 y la evaluación sea para verificar si la condición es menor o igual al número requerido, como se muestra a continuación:

```

contador = 1;
while (contador <= numeroRequerido) {
    // cuerpo
    contador++;
}
  
```

Utilizaremos ambos estilos en este libro.

## PRÁCTICAS DE AUTOEVALUACIÓN

- 8.3 **Barras de la prisión** Escriba un programa para dibujar cinco líneas verticales paralelas.
- 8.4 **Tablero de ajedrez** Escriba un programa para dibujar un tablero de ajedrez con nueve líneas verticales con una separación de 10 píxeles, y nueve líneas horizontales con una separación de 10 píxeles.
- 8.5 **Cuadrados de números** Escriba un programa para mostrar los números del 1 al 5 y sus cuadrados, un número (y su cuadrado) por línea en un área de texto. Use la cadena “\n” para avanzar al inicio de una nueva línea.

## ● for

En el ciclo **for**, muchos de los ingredientes de un ciclo **while** se agrupan en el encabezado de la instrucción en sí. Por ejemplo, veamos el programa anterior para mostrar los números del 1 al 10, pero ahora utilizando **for**:

```
unoAlDiez.setText("");
for (int número = 1; número <= 10; número++) {
    unoAlDiez = unoAlDiez + Integer.toString(número) + " ";
```

Dentro de los paréntesis de la instrucción **for** hay tres ingredientes separados por signos de punto y coma:

- Una instrucción inicial. Se lleva a cabo sólo una vez, antes de que inicie el ciclo.  
Ejemplo: `int número = 1`
- Una condición. Se evalúa antes de cualquier ejecución del ciclo.  
Ejemplo: `número <= 10`
- Una instrucción final. Ésta se lleva a cabo al final de cada ciclo, justo antes del final de cada ciclo.  
Ejemplo: `número++`

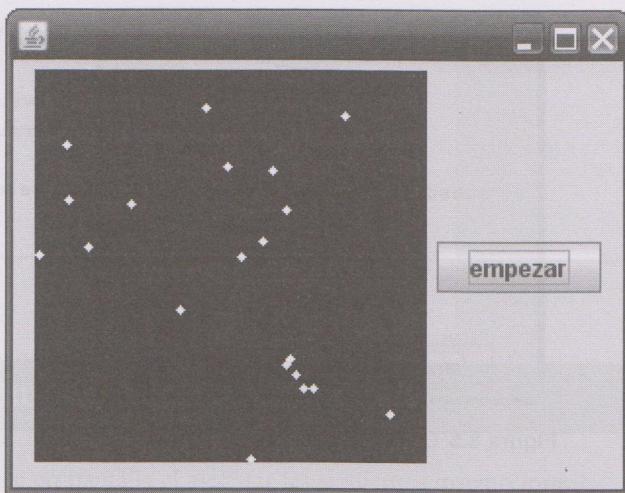
La condición determina si el ciclo **for** se ejecuta o completa de la siguiente manera:

- Si la condición es verdadera, se ejecuta el cuerpo del ciclo.
- Si la condición es falsa, el ciclo termina y se ejecutan las instrucciones que van después de la llave de cierre.

Tenga en cuenta que puede escribir una declaración completa de una variable dentro del encabezado de una instrucción **for** junto con su inicialización, y esto es algo que se hace con frecuencia. Esta variable se puede utilizar a lo largo del cuerpo de la instrucción **for**.

He aquí otro programa de ejemplo que utiliza un ciclo **for** para mostrar 20 círculos pequeños en coordenadas aleatorias en un panel de color negro, como el cielo de la noche (figura 8.4).

```
public void actionPerformed(ActionEvent event) {
    Graphics papel = panel.getGraphics();
    papel.setColor(Color.black);
    papel.fillRect(0, 0, 200, 200);
    papel.setColor(Color.white);
    for (int contador = 0; contador < 20; contador++) {
        int x, y, radio;
        x = aleatorio.nextInt(200);
        y = aleatorio.nextInt(200);
        radio = 5;
        papel.fillOval(x, y, radio, radio);
    }
}
```



**Figura 8.4** El programa de las estrellas.

Siempre es posible volver a codificar un ciclo `for` como un ciclo `while` y viceversa. Pero por lo general uno de los dos será el más claro. El ciclo `for` se utiliza comúnmente cuando contamos, sumando o restando un valor fijo en cada paso.

Además, se considera un estilo pobre terminar el ciclo antes de completar el patrón descrito en el encabezado de un ciclo `for`.

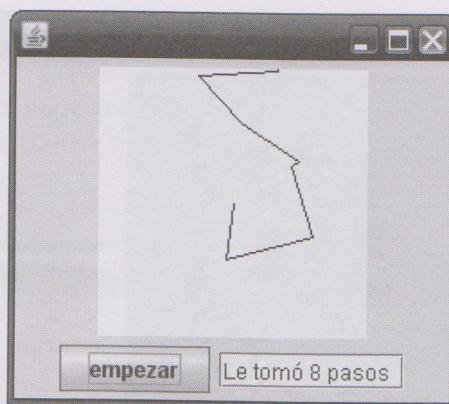
## PRÁCTICA DE AUTOEVALUACIÓN

**8.6** Vuelva a codificar el programa de las estrellas; esta vez utilice `while` en vez de `for`.

### ● And, or, not

En ocasiones, la condición que controla un ciclo es más compleja y necesitamos los operadores lógicos `and`, `or` y `not`. Usted podría utilizar estos operadores en su vida diaria si quisiera decir: “Voy a dar un paseo hasta que empiece a llover o sean las 5 de la tarde”. Ya vimos estos operadores en el capítulo 7, en la sección sobre el uso de la instrucción `if` en las decisiones. Estos operadores son:

Símbolo	Significado
<code>&amp;&amp;</code>	and
<code>  </code>	or
<code>!</code>	not



**Figura 8.5** Caminata aleatoria.

Si quisiéramos describir cuánto tiempo vamos a caminar mediante una instrucción `while`, diríamos: “Mientras que no llueva y no sean las 5 de la tarde, voy a seguir caminando”. Observe que hay un “no” (`not`) antes de cada una de las dos condiciones (llueva, 5 de la tarde) y están enlazadas por un “y” (`and`). Esto es lo que ocurre por lo general cuando usamos un ciclo con una instrucción `while`; es importante tener mucho cuidado en escribir la condición con mucha claridad.

En el siguiente programa utilizamos estos operadores. Una persona ebria intenta llegar a cualquiera de las paredes de un cuarto (representado como un panel). Al principio la persona está en el centro del panel. Da pasos de un tamaño aleatorio y en dirección aleatoria hasta llegar a una pared (figura 8.5).

La posición de la persona en cualquier instante se especifica mediante coordenadas  $x$  y  $y$ . El ciclo continúa hasta llegar a uno de los cuatro lados del panel. Por ende, la instrucción `for` implica cuatro condiciones, enlazadas por operadores `&&`. Como verá, ésta es una combinación de condiciones moderadamente compleja. Una combinación como ésta puede ser difícil de escribir y comprender.

```
public void actionPerformed(ActionEvent event) {
    Graphics papel = panel.getGraphics();
    int x, y, pasoX, pasoY, nuevaX, nuevaY, pasos;
    papel.setColor(Color.white);
    papel.fillRect(0, 0, anchuraPapel, alturaPapel);
    x = anchuraPapel / 2;
    y = alturaPapel / 2;
    for (pasos = 0;
        x < anchuraPapel && x > 0
        &&
        y < alturaPapel && y > 0;
        pasos++) {
        pasoX = aleatorio.nextInt(100) - 50;
        pasoY = aleatorio.nextInt(100) - 50;
        nuevaX = x + pasoX;
        nuevaY = y + pasoY;
```

```

papel.setColor(Color.black);
papel.drawLine(x, y, nuevaX, nuevaY);

x = nuevaX;
y = nuevaY;
}
campoTexto.setText("Le tomó " + pasos + " pasos");
}

```

## PRÁCTICA DE AUTOEVALUACIÓN

- 8.7 ¿Qué aparece en pantalla al ejecutar las siguientes instrucciones?

```

int n, m;
n = 10;
m = 5;
while ((n > 0) || (m > 0)) {
    n = n - 1;
    m = m - 1;
}
 JOptionPane.showMessageDialog(null,
        ("n = " + Integer.toString(n) +
        " m = " + Integer.toString(m)));

```

### do...while

Si utiliza las instrucciones **while** o **for**, la evaluación siempre se realiza al principio de la repetición. El ciclo **do** es una estructura alternativa en la cual la evaluación se lleva a cabo al final de cada repetición. Esto significa que el ciclo siempre se repite por lo menos una vez. Ilustraremos el uso del ciclo **do** escribiendo piezas de programas para mostrar los números del **0** al **9** en un campo de texto, utilizando las tres estructuras de ciclo disponibles. El texto se acumula en una cadena de texto.

Si usamos **while**:

```

int contador;
String cadena = "";
contador = 0;
while (contador <= 9) {
    cadena = cadena + Integer.toString(contador) + " ";
    contador++;
}

```

Si usamos **for**:

```
String cadena = "";
for (int contador = 0; contador <= 9; contador++) {
    cadena = cadena + Integer.toString(contador) + " ";
}
```

Si usamos **do** (la evaluación se hace al final del ciclo):

```
int contador;
String cadena = "";
contador = 0;
do {
    cadena = cadena + Integer.toString(contador) + " ";
    contador++;
}
while (contador < 10);
```

Ahora veremos un ejemplo en donde un ciclo necesita ejecutarse por lo menos una vez. En una lotería los números se seleccionan al azar, pero no puede haber dos números iguales. Consideremos una lotería muy pequeña en la que sólo se seleccionan dos números. Al principio obtenemos el primer número aleatorio. Ahora necesitamos un segundo número, pero éste no debe ser igual al primero. Entonces seleccionamos repetidas veces un segundo número hasta que no sea igual al primero. He aquí el programa:

```
int número1, número2;
Random aleatorio = new Random();

número1 = aleatorio.nextInt(10) + 1;
do {
    número2 = aleatorio.nextInt(10) + 1;
} while (número1 == número2);

campoTexto.setText("los números son "
    + número1 + " y " + número2);
```

## Ciclos anidados

Un ciclo anidado es un ciclo dentro de otro ciclo. Suponga, por ejemplo, que deseamos mostrar la pantalla de la figura 8.6, que es un bloque de apartamentos dibujados en forma muy rudimentaria. El tamaño del edificio se determina con base en la posición de los controles deslizables. Suponga que hay cuatro pisos, cada uno con cinco apartamentos, los cuales aparecen como rectángulos. El ciclo para dibujar un piso individual tiene la siguiente estructura:

```
for (int apartamento = 1; apartamento <= 5; apartamento++) {  
    // código para dibujar un apartamento  
}
```

y el ciclo para dibujar varios pisos tiene la siguiente estructura:

```
for (int piso = 1; piso <= 3; piso++) {  
    // código para dibujar un piso  
}
```

Lo que necesitamos es encerrar el primer ciclo dentro del segundo, de manera que los ciclos estén anidados. Podemos usar controles deslizables para establecer el número de apartamentos por piso y el número de pisos. Cada vez que modifiquemos cualquiera de los controles deslizables se producirá un evento y se ejecutarán las siguientes instrucciones:

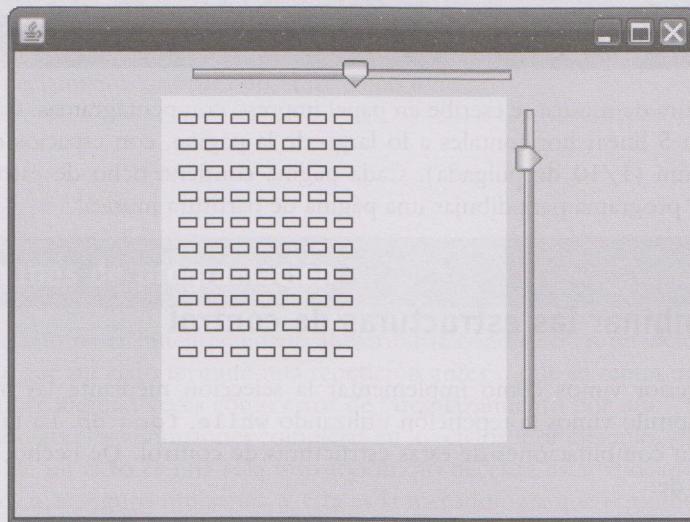


Figura 8.6 Pantalla del bloque de apartamentos.

```

int pisos, apartamentos;
int x, y;
y = 10;
papel.setColor(Color.white);
papel.fillRect(0, 0, 200, 200);

apartamentos = deslizante1.getValue();
pisos = deslizante2.getValue();
for (int piso = 0; piso <= pisos; piso++) {
    x = 10;
    for (int contador = 0; contador <= apartamentos; contador++) {
        papel.setColor(Color.black);
        papel.drawRect(x, y, 10, 5);
        x = x + 15;
    }
    y = y + 15;
}

```

En el ejemplo anterior podemos ver que la sangría ayuda de manera considerable a comprender el funcionamiento del programa. Siempre es posible rediseñar los ciclos anidados utilizando métodos, que algunas veces son más claros. En el capítulo 19 exploraremos esto con más detalle, cuando hablemos sobre el estilo.

## PRÁCTICA DE AUTOEVALUACIÓN

- 8.8** Una partitura de música se escribe en papel impreso con pentagramas. Cada pentagrama consiste en 5 líneas horizontales a lo largo de la página, con espacios de aproximadamente 2 mm (1/10 de pulgada). Cada página contiene ocho de estos pentagramas. Escriba un programa para dibujar una página de partitura musical.

## ● Cómo combinar las estructuras de control

En el capítulo anterior vimos cómo implementar la selección mediante las instrucciones **if** y **switch**; en este capítulo vimos la repetición utilizando **while**, **for** y **do**. La mayoría de los programas consisten en combinaciones de estas estructuras de control. De hecho, la mayoría de los programas constan de:

- Secuencias.
- Ciclos.
- Selecciones.
- Llamadas a métodos de biblioteca.
- Llamadas a métodos que nosotros, los programadores, escribimos.

## Fundamentos de programación

Las repeticiones se utilizan mucho en la programación. Hay tres variedades de instrucciones de ciclos en Java: **while**, **for** y **do**. Entonces, ¿cuál debemos usar?

Las instrucciones **while** y **for** son similares, pero **for** se utiliza generalmente cuando hay un contador asociado con el ciclo. Por ende, el prototipo del ciclo **for** tiene la siguiente estructura:

```
for (int contador = 0; contador <= valorFinal; contador++) {
    // cuerpo del ciclo
}
```

El contador tiene un valor inicial, un valor final y se incrementa cada vez que se repite el ciclo.

El ciclo **while** se utiliza cuando no se puede calcular el número de repeticiones por adelantado; el ciclo continúa mientras que cierta condición sea verdadera. Un ejemplo es el programa de caminata aleatoria que vimos antes, en donde la repetición continúa hasta que la persona llega a la pared. El prototipo del ciclo **while** tiene la siguiente estructura:

```
while (condición) {
    // cuerpo del ciclo
}
```

El ciclo **do** se utiliza cuando la evaluación para que termine la repetición se necesita hacer al final del ciclo; es importante recordar que un ciclo **do** siempre se lleva a cabo por lo menos una vez.

Los ciclos demuestran su valor en los programas que procesan colecciones de datos. Más adelante en el libro veremos varias colecciones, incluyendo listas, cadenas de texto, archivos y arreglos. También conoceremos una variedad del ciclo **for** (conocida como ciclo **for** mejorado) que es especialmente útil para procesar colecciones.

## Errores comunes de programación

Siempre es necesario tener mucho cuidado al escribir la condición en un ciclo. Un error muy común es hacer que un ciclo termine una repetición antes o que se repita demasiadas veces. A esto se le conoce algunas veces como error de “desplazamiento por uno” (*off by one*).

Tenga cuidado con las condiciones complejas en los ciclos. Por ejemplo, ¿necesita `||` o `&&`?

Si el cuerpo de un ciclo es una sola instrucción, no necesita estar rodeado de llaves. Pero por lo general es más seguro utilizarlas, y ésta es la metodología que seguimos en este libro.

## Secretos de codificación

El ciclo **while** tiene la siguiente estructura:

```
while (condición) {  
    instrucción(es)  
}
```

en donde la condición se evalúa antes de cualquier repetición del ciclo. Si es verdadera, el ciclo continúa. Si es falsa, el ciclo termina.

El ciclo **for** tiene la siguiente estructura:

```
for (acción inicial; condición; acción) {  
    instrucción(es)  
}
```

en donde:

- **acción inicial** se lleva a cabo una vez, antes de ejecutar el ciclo.
- **condición** se evalúa antes de cada repetición. Si es verdadera, se repite el ciclo; si es falsa, el ciclo termina.
- **acción** se lleva a cabo al final de cada repetición.

El ciclo **do** tiene la siguiente estructura:

```
do {  
    instrucción(es)  
}  
while (condición)
```

La evaluación se realiza después de cada repetición.

## Nuevos elementos del lenguaje

Las estructuras de control para repetición:

- **while**
- **for**
- **do**

- Llamadas a métodos de biblioteca.
- Llamadas a métodos que nosotros, los programadores, escribimos.

## Resumen

- En programación, a una repetición se le conoce como ciclo.
- Hay tres formas en Java para indicar a la computadora que realice un ciclo: `while`, `for` y `do`.
- La instrucción `for` se utiliza cuando queremos describir las características principales del ciclo dentro de la misma instrucción del ciclo.
- La instrucción `do` se utiliza cuando se debe evaluar una condición al final de un ciclo y/o cuando el ciclo se debe realizar por lo menos una vez.

## Ejercicios

- 8.1 Cubos** Escriba un programa que utilice un ciclo para mostrar los números enteros del 1 al 10 junto con los cubos de cada uno de sus valores.
- 8.2 Números aleatorios** Escriba un programa para mostrar 10 números aleatorios mediante un ciclo. Use la clase de biblioteca `Random` para obtener números aleatorios enteros en el rango de 0 a 9. Muestre los números en un campo de texto.
- 8.3 La vía láctea** Escriba un programa que dibuje 100 círculos en un panel en posiciones aleatorias y con diámetros aleatorios de hasta 10 píxeles.
- 8.4 Escalones** Escriba un programa para dibujar una serie de escalones hechos a partir de ladrillos, como se muestra en la figura 8.7. Use el método de biblioteca `drawRect` para dibujar cada ladrillo.

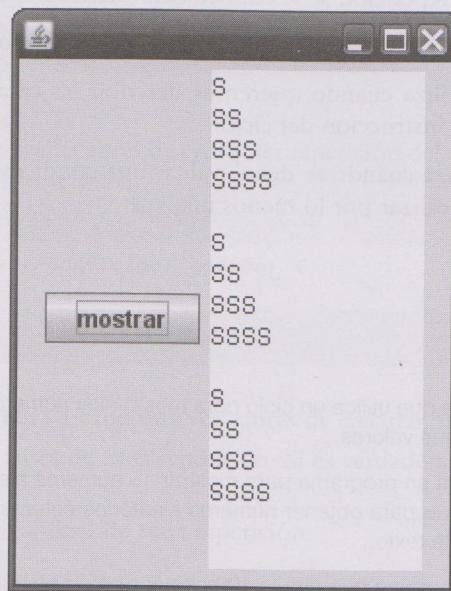


**Figura 8.7** Escalones.

- 8.5 Suma de los enteros** Escriba un programa que sume los números del 0 al 39 utilizando un ciclo. Compruebe que ha obtenido la respuesta correcta, utilizando la fórmula para la suma de los números de 0 a  $n$ :

$$\text{suma} = n \times (n + 1)/2$$

**8.6 Patrón de diente de sierra** Escriba un programa para mostrar un patrón de diente de sierra en un cuadro de texto, como se muestra en la figura 8.8. El programa debe utilizar la cadena “`\n`” para obtener una nueva línea.



**Figura 8.8** Patrón de diente de sierra.

**8.7 Tabla de multiplicación** Escriba un programa para mostrar una tabla de multiplicación, como la que utilizan los niños pequeños. Por ejemplo, la tabla para los números hasta el 4 se muestra en la figura 8.9. Además de utilizar la cadena “`\n`” para obtener una nueva línea, el programa debe usar la cadena “`\t`” para avanzar el tabulador a la siguiente posición, de manera que la información se muestre en columnas ordenadas.

	1	2	3	4
1	1	2	3	4
2	2	4	6	8
3	3	6	9	12
4	4	8	12	16

**Figura 8.9** Tabla de multiplicación.

El programa debe ser capaz de mostrar una tabla de cualquier tamaño, el cual se debe especificar introduciendo un entero en un campo de texto. Use el método `setTabSize` para controlar la distribución. Una llamada típica sería:

```
areaTexto.setTabSize(4);
```

- 8.8 Suma de series** Escriba un programa para calcular y mostrar la suma de las series:

$$1 - 1/2 + 1/3 - 1/4 + \dots$$

hasta llegar a un término que sea menor a 0.0001 (la respuesta es aproximadamente 0.6936).

- 8.9 Canción de cuna** Escriba un programa para mostrar todos los versos de una canción de cuna en un área de texto con una barra de desplazamiento vertical. El primer verso es:

```
10 botellas verdes, colgando en la pared,  
10 botellas verdes, colgando en la pared,  
Si 1 botella verde se fuera a caer,  
Habría 9 botellas verdes, colgando en la pared.
```

En los versos subsiguientes hay cada vez menos botellas, a medida que se caen de la pared.

## Respuestas a las prácticas de autoevaluación

- 8.1** Muestra los números 0 1 4 9 16 25.

- 8.2** La parte central del programa es:

```
int número;  
int suma;  
  
suma = 0;  
número = 1;  
while (número <= 100) {  
    suma = suma + número;  
    número++;  
}  
campoTexto.setText("La suma es " + Integer.toString(suma));
```

Este programa utiliza una técnica de programación común: un total acumulado. Al principio el valor de la variable `suma` es igual a cero. Cada vez que se repite el ciclo, el valor de `número` se suma al valor de `suma` y el resultado se coloca nuevamente en `suma`.

```
8.3    int x, contador, númeroDeBarras;  
númeroDeBarras = 5;  
x = 10;  
contador = 1;  
while (contador <= númeroDeBarras) {  
    papel.drawLine(x, 10, x, 100);  
    x = x + 15;  
    contador++;  
}
```



 Respuestas a las prácticas de autoevaluación (continúa)

```

8.4 int x, y, contador;
x = 10;
contador = 1;
while (contador <= 9) {
    papel.drawLine(x, 10, x, 90);
    x = x + 10;
    contador++;
}
y = 10;
contador = 1;
while (contador <= 9) {
    papel.drawLine(10, y, 90, y);
    y = y + 10;
    contador++;
}

8.5 int número = 1;
while (número <= 5) {
    areaTexto.append(Integer.toString(número) + " "
        + Integer.toString(número * número) + "\n");
    número++;
}

8.6 La esencia del ciclo es:

int contador = 0;
while (contador < 20) {
    // cuerpo del ciclo
    contador++;
}

8.7 n = 0 y m = 5.

8.8 int y;
papel.setColor(Color.white);
papel.fillRect(0, 0, 150, 100);

y = 10;
for (int pentagramas = 1; pentagramas <= 8; pentagramas++) {
    for (int líneas = 1; líneas <= 5; líneas++) {
        papel.setColor(Color.black);
        papel.drawLine(10, y, 90, y);
        y = y + 2;
    }
    y = y + 5;
}

```