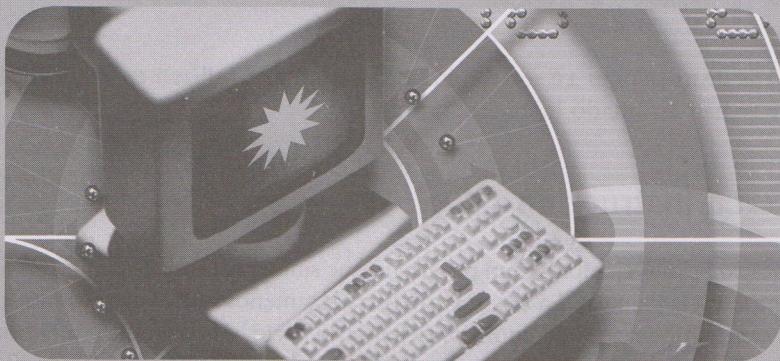


# CAPÍTULO 7



## Selección

En este capítulo conoceremos cómo:

- Utilizar las instrucciones **if** y **switch** para llevar a cabo evaluaciones.
- Manejar múltiples eventos.
- Utilizar los operadores de comparación tales como **>**;
- Utilizar los operadores lógicos **&&**, **||** y **!**;
- Declarar y utilizar datos **boolean**.
- Comparar cadenas.

### ● Introducción

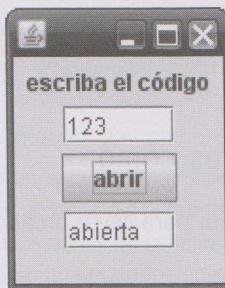
Todos hacemos selecciones en la vida diaria. Usamos un abrigo si llueve. Compramos un CD si tenemos suficiente dinero. Las selecciones también se utilizan mucho en los programas. La computadora evalúa un valor y, de acuerdo con el resultado, toma un curso de acción u otro. Cada vez que el programa tiene una selección de acciones y decide realizar una o la otra, se utiliza una instrucción **if** o **switch** para describir la situación.

Ya hemos visto que un programa computacional consiste en una serie de instrucciones para una computadora. Ésta obedece las instrucciones, una tras otra, en secuencia. Pero algunas veces requerimos que la computadora evalúe ciertos datos y después seleccione una de varias acciones dependiendo del resultado de la evaluación. Por ejemplo, tal vez necesitemos que la computadora evalúe la edad de alguien y después le diga que puede votar o que es demasiado joven. A esto se le conoce como selección y se utiliza una instrucción conocida como **if**, el tema central de este capítulo.

Las instrucciones `if` son tan importantes que se utilizan en todos los lenguajes de programación que se han inventado hasta este momento.

## ● La instrucción `if`

Nuestro primer ejemplo es un programa que simula el candado digital en una bóveda. En la figura 7.1 se muestra la pantalla. La bóveda está cerrada a menos que el usuario introduzca el código correcto en un campo de texto, que en un principio está vacío. Al hacer clic en el botón, el programa convierte el texto introducido en un número y lo compara con el código correcto. Si el código es correcto, se muestra un mensaje.



**Figura 7.1** Pantalla del programa de la bóveda.

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

public class Bóveda extends JFrame implements ActionListener {

    private JLabel greetingLabel;
    private JTextField campoCódigo;
    private JButton botón;
    private JTextField campoTextoResultado;

    public static void main(String[] args) {
        Bóveda demo = new Bóveda();
        demo.setSize(100,150);
        demo.crearGUI();
        demo.setVisible(true);
    }

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

        etiquetaBienvenida = new JLabel("escriba el código");
        ventana.add(etiquetaBienvenida);
```

```

campoCódigo = new JTextField(5);
ventana.add(campoCódigo);

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

campoTextoResultado = new JTextField(5);
ventana.add(campoTextoResultado);

}

public void actionPerformed(ActionEvent event) {
    String cadenaCódigo;
    int código;

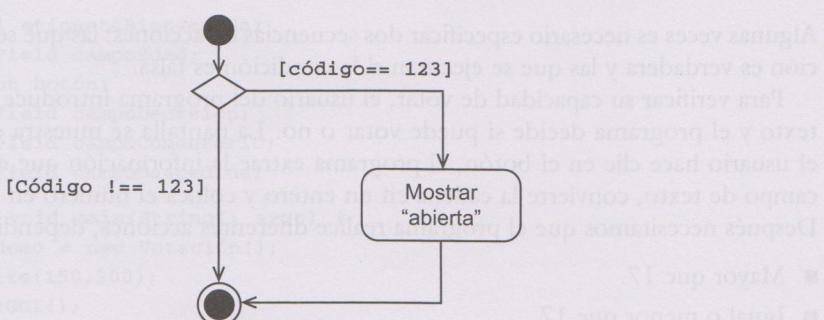
    cadenaCódigo = campoCódigo.getText();
    código = Integer.parseInt(cadenaCódigo);
    if (código == 123) {
        campoTextoResultado.setText("abierta");
    }
}
}

```

La instrucción **if** prueba el valor del número introducido. Si es igual al valor 123, se ejecuta la trucción que está entre las llaves { y }. A continuación, se ejecuta cualquier instrucción que se cuente después de la llave de cierre. Por otro lado, si el número no es igual a 123, se ignora la trucción entre las llaves y se ejecuta cualquier instrucción que esté después de la llave de cierre. Tenga en cuenta que la condición a evaluar se encierra entre paréntesis; ésta es una regla gráical de Java.

Observe también que una prueba de igualdad utiliza el operador == (no el operador =, que se liza para asignación).

Una forma de visualizar a una instrucción **if** es mediante un diagrama de actividades (figura 7.2). uí se muestra la instrucción **if** anterior en forma gráfica. Para utilizar este diagrama, empiece en círculo relleno de la parte superior y siga las flechas. Una decisión se muestra como un diamante,



**Figura 7.2** Diagrama de actividades de una instrucción **if**.

y las dos posibles condiciones se muestran entre corchetes. Las acciones se muestran dentro de cuadros con esquinas redondas y el fin de la secuencia es un círculo con una forma especial en la parte inferior del diagrama.

La instrucción **if** consta de dos partes:

- La condición que se va a evaluar.
- La instrucción o secuencia de instrucciones a ejecutar si la condición es verdadera.

Todos los programas consisten en una secuencia de acciones, y la secuencia evidente en el ejemplo anterior es:

1. Se recibe una pieza de texto del campo de texto.
2. Luego se realiza una evaluación.
3. Si es apropiado, se muestra un mensaje para indicar que la bóveda está abierta.

Es muy frecuente que se lleven a cabo no sólo una acción sino una secuencia completa de acciones si el resultado de la evaluación es verdadero; estas acciones deben ir encerradas entre las llaves.

Cabe mencionar que se aplica sangría a una línea para reflejar la estructura de la instrucción **if**. Sangría significa que se utilizan espacios para desplazar el texto hacia la derecha. Aunque no es esencial el uso de sangrías, es muy conveniente para que el lector (humano) de un programa lo pueda entender con facilidad. Todos los programas bien hechos (cualkiera que sea el lenguaje) tienen sangría y todos los programadores con experiencia la utilizan.

## PRÁCTICA DE AUTOEVALUACIÓN

- 7.1 Mejore la instrucción **if** de manera que borre el número cuando se introduzca el código correcto en el campo de texto.

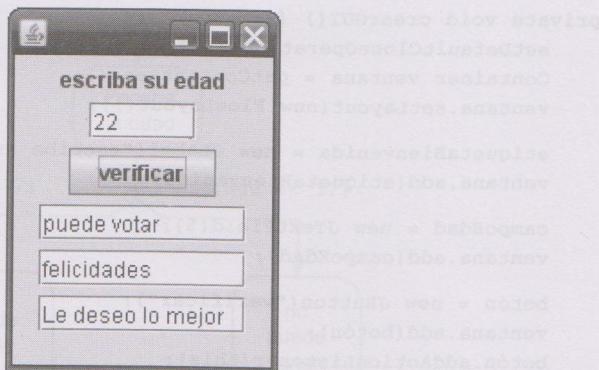
### ● **if...else**

Algunas veces es necesario especificar dos secuencias de acciones: las que se llevan a cabo si la condición es verdadera y las que se ejecutan si la condición es falsa.

Para verificar su capacidad de votar, el usuario del programa introduce su edad en un campo de texto y el programa decide si puede votar o no. La pantalla se muestra en la figura 7.3. Cuando el usuario hace clic en el botón, el programa extrae la información que el usuario introdujo en el campo de texto, convierte la cadena en un entero y coloca el número en la variable llamada **edad**. Después necesitamos que el programa realice diferentes acciones, dependiendo de si el valor es:

- Mayor que 17.
- Igual o menor que 17.

Para ello se utiliza la instrucción **if**:



**Figura 7.3** La pantalla del programa para verificar la capacidad de votar.

```

if (edad > 17) {
    campoDecisión.setText("puede votar");
    campoComentario.setText("felicitaciones");
}
else {
    campoDecisión.setText("no puede votar");
    campoComentario.setText = ("lo siento");
}

```

Los resultados de la prueba se muestran en varios campos de texto. El programa completo es el siguiente:

```

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

public class Votación extends JFrame implements ActionListener {

    private JLabel etiquetaBienvenida;
    private JTextField campoEdad;
    private JButton botón;
    private JTextField campoDecisión;
    private JTextField campoComentario;
    private JTextField campoDespedida;

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

```

```

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

    etiquetaBienvenida = new JLabel("escriba su edad");
    ventana.add(etiquetaBienvenida);

    campoEdad = new JTextField(5);
    ventana.add(campoEdad);

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

    campoDecisión = new JTextField(10);
    ventana.add(campoDecisión);

    campoComentario = new JTextField(10);
    ventana.add(campoComentario);

    campoDespedida = new JTextField(10);
    ventana.add(campoDespedida);
}

public void actionPerformed(ActionEvent event) {
    int edad;

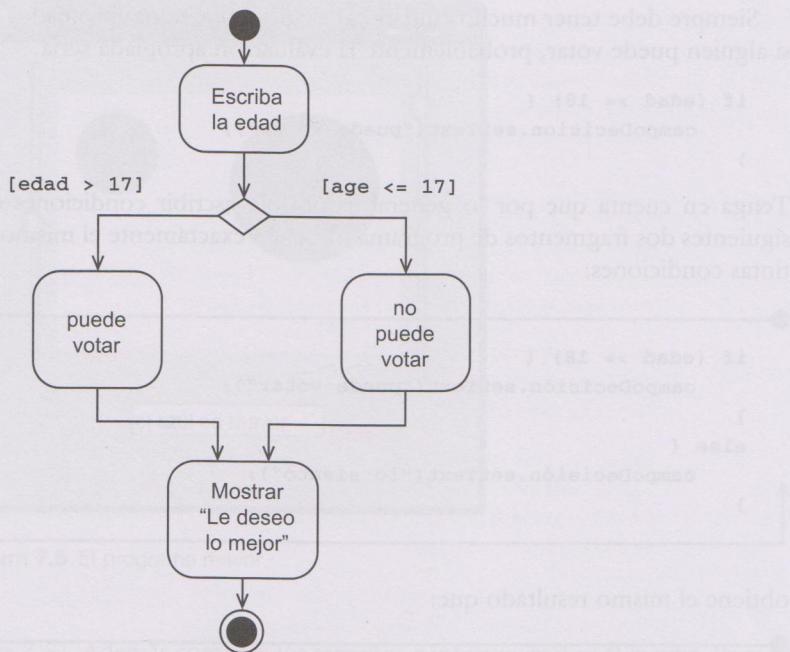
    edad = Integer.parseInt(campoEdad.getText());
    if (edad > 17)
    {
        campoDecisión.setText("puede votar");
        campoComentario.setText("felicidades");
    }
    else
    {
        campoDecisión.setText("no puede votar");
        campoComentario.setText("lo siento");
    }
    campoDespedida.setText("Le deseo lo mejor");
}
}

```



En este programa la instrucción **if** consta de tres partes:

- La condición que se va a evaluar; en este caso, si la edad es mayor que 17.
- La instrucción o secuencia de instrucciones que se ejecutarán si la condición es verdadera, y que deben ir encerradas entre llaves.
- La instrucción o instrucciones a ejecutar si la condición es falsa, y que deben ir encerradas entre llaves.



**Figura 7.4** Diagrama de actividades para una instrucción `if...else`.

El nuevo elemento aquí es la palabra clave `else`, la cual introduce la segunda parte de la instrucción `if`. Observe de nuevo cómo ayuda la sangría a enfatizar la intención del programa.

Podemos visualizar una instrucción `if...else` como un diagrama de actividades, como se muestra en la figura 7.4. Este diagrama muestra la condición que se va a evaluar y las dos acciones separadas.

## ● Operadores de comparación

Los programas anteriores utilizan algunos de los operadores (algunas veces conocidos como relationales) de comparación. He aquí una lista completa de ellos:

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

Aquí podemos ver de nuevo que Java utiliza dos veces el signo “igual que” (`==`) para evaluar si dos cosas son iguales.

Siempre debe tener mucho cuidado al elegir el operador apropiado. En el programa para evaluar si alguien puede votar, probablemente la evaluación apropiada sería:

```
if (edad >= 18) {  
    campoDecisión.setText("puede votar");  
}
```

Tenga en cuenta que por lo general es posible escribir condiciones en una de dos formas. Los siguientes dos fragmentos de programa obtienen exactamente el mismo resultado, pero utilizan distintas condiciones:

```
if (edad >= 18) {  
    campoDecisión.setText("puede votar");  
}  
else {  
    campoDecisión.setText("lo siento");  
}
```

obtiene el mismo resultado que:

```
if (edad < 18) {  
    campoDecisión.setText("lo siento");  
}  
else {  
    campoDecisión.setText("puede votar");  
}
```

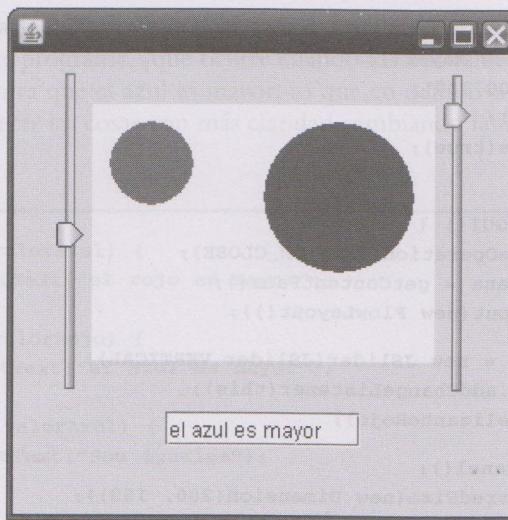
Aunque estos dos fragmentos logren el mismo resultado final, probablemente el primero sea mejor debido a que establece con más claridad la condición de poder votar.

## PRACTICA DE AUTOEVALUACIÓN

7.2 ¿Obtienen las siguientes dos instrucciones **if** el mismo resultado o no?

```
if (edad > 18) {  
    campoDecisión.setText("puede votar");  
}  
if (edad < 18) {  
    campoDecisión.setText("no puede votar");  
}
```

Algunas veces los humanos tienen dificultad para juzgar los tamaños relativos de círculos de distintos colores. El siguiente programa utiliza dos controles deslizables y muestra círculos con



**Figura 7.5** El programa mayor.

tamaños equivalentes (figura 7.5). Además compara los tamaños e informa cuál es el mayor. Primero vamos a escribir el programa mediante la siguiente instrucción **if**:

```
if (valorRojo > valorAzul) {
    campoTexto.setText("el rojo es mayor");
}
else {
    campoTexto.setText("el azul es mayor");
}
```

El método de biblioteca **fillOval** se utiliza para dibujar un círculo relleno cuyo diámetro sea igual al valor obtenido del control deslizable correspondiente. El programa completo es:

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

public class Mayor extends JFrame
    implements ChangeListener {

    private JSlider deslizanteRojo;
    private JPanel panel;
    private JSlider deslizanteAzul;
    private JTextField campoTexto;
```

```
public static void main(String[] args) {
    Mayor demo = new Mayor();
    demo.setSize(300,300);
    demo.crearGUI();
    demo.setVisible(true);
}

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

    deslizanteRojo = new JSlider(JSlider.VERTICAL);
    deslizanteRojo.addChangeListener(this);
    ventana.add(deslizanteRojo);

    panel = new JPanel();
    panel.setPreferredSize(new Dimension(200, 150));
    panel.setBackground(Color.white);
    ventana.add(panel);

    deslizanteAzul = new JSlider(JSlider.VERTICAL);
    deslizanteAzul.addChangeListener(this);
    ventana.add(deslizanteAzul);

    campoTexto = new JTextField(10);
    ventana.add(campoTexto);
}

public void stateChanged(ChangeEvent e) {
    Graphics papel = panel.getGraphics();

    int valorRojo, valorAzul;
    valorRojo = deslizanteRojo.getValue();
    valorAzul = deslizanteAzul.getValue();

    papel.setColor(Color.white);
    papel.fillRect(0, 0, 200, 150);
    papel.setColor(Color.red);
    papel.fillOval(10, 10, valorRojo, valorRojo);
    papel.setColor(Color.blue);
    papel.fillOval(100, 10, valorAzul, valorAzul);

    if (valorRojo > valorAzul) {
        campoTexto.setText("el rojo es mayor");
    }
    else {
        campoTexto.setText("el azul es mayor");
    }
}
```

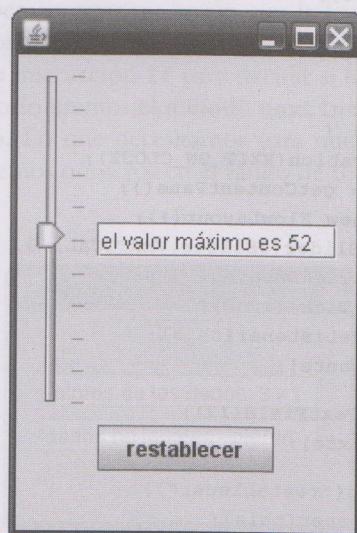
Este programa funciona bien, sólo que ilustra de nuevo la importancia de tener cuidado al usar instrucciones `if`. En este programa, ¿qué ocurre cuando los dos valores son iguales? La respuesta es que el programa encuentra que el azul es mayor; lo que en definitiva no es así. Podríamos mejorar el programa para establecer las cosas con más claridad cambiando la instrucción `if` por el siguiente código:

```
if (valorRojo > valorAzul) {
    campoTexto.setText("el rojo es mayor");
}
if (valorAzul > valorRojo) {
    campoTexto.setText("el azul es mayor");
}
if (valorRojo == valorAzul) {
    campoTexto.setText("Son iguales");
}
```

El siguiente ejemplo es un programa que lleva el registro del valor más grande de un número, a medida que va cambiando. Es un ejercicio común en programación. Algunos termómetros tienen un mecanismo para registrar la temperatura máxima alcanzada. Este programa simula dicho termómetro mediante un control deslizable. Muestra el máximo valor al que está ajustado el control deslizable (vea la figura 7.6).

El programa utiliza una variable llamada `max`, una variable a nivel de clase que contiene el valor de la temperatura más alta alcanzada hasta ese momento. La variable `max` se declara así:

```
private int max = 0;
```



**Figura 7.6** Pantalla del termómetro.

Una instrucción **if** compara el valor actual del control deslizable con el valor de **max** y lo modifica si es necesario:

```
int temp;
temp = deslizante.getValue();
if (temp > max) {
    max = temp;
}
```

He aquí el programa completo:

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

public class Max extends JFrame implements ChangeListener,
    ActionListener {

    private JSlider deslizante;
    private JTextField campoTexto;
    private JButton botón;
    private int max = 0;

    public static void main(String[] args) {
        Max demo = new Max();
        demo.setSize(200,300);
        demo.crearGUI();
        demo.setVisible(true);
    }

    private void crearGUI() {
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        Container ventana = getContentPane();
        ventana.setLayout(new FlowLayout());
        deslizante = new JSlider(JSlider.VERTICAL, 0, 100, 0);
        deslizante.setMajorTickSpacing(10);
        deslizante.setPaintTicks(true);
        deslizante.addChangeListener(this);
        ventana.add(deslizante);

        campoTexto = new JTextField(12);
        ventana.add(campoTexto);

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

    public void stateChanged(ChangeEvent e) {
        if (deslizante.getValue() < max) {
            campoTexto.setText(Integer.toString(deslizante.getValue()));
        } else {
            max = deslizante.getValue();
            campoTexto.setText(Integer.toString(max));
        }
    }

    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == botón) {
            max = deslizante.getValue();
            campoTexto.setText(Integer.toString(max));
        }
    }
}
```

```

public void stateChanged(ChangeEvent e) {
    int temp;
    temp = deslizante.getValue();
    if (temp > max) {
        max = temp;
    }
    mostrar();
}

public void actionPerformed(ActionEvent event) {
    campoTexto.setText("");
    max = 0;
}

private void mostrar() {
    campoTexto.setText("el valor máximo es " + max);
}
}

```

## PRÁCTICA DE AUTOEVALUACIÓN

- 7.3 Escriba un programa que muestre el valor numérico del valor mínimo al que se ajuste el control deslizable.

Ahora veremos un programa que simula la acción de tirar dos dados. La computadora decide los valores de los dados al azar. Debemos crear un botón con la leyenda “tirar”. Al hacer clic en este botón, el programa obtendrá dos números al azar y los utilizará como los valores de los dados (figura 7.7). El programa usa una instrucción **if** para decidir si los valores son iguales.

Para obtener un número aleatorio usamos el método **nextInt** de la clase de biblioteca **Random**. Vimos esta clase en el capítulo 6. Lo que necesitamos para nuestro propósito es un entero en el rango de 1 a 6, por lo que obtenemos números en el rango de 0 a 5 y simplemente les sumamos 1.

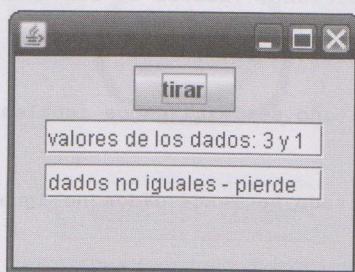


Figura 7.7 Apuestas.

El programa que simula tirar dos dados es el siguiente:

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

public class Apuestas extends JFrame implements ActionListener {
    private JButton botón;
    private JTextField campoTextoValores, campoTextoResultado;
    private Random aleatorio;

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

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

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

        campoTextoValores = new JTextField(14);
        ventana.add(campoTextoValores);

        campoTextoResultado = new JTextField(14);
        ventana.add(campoTextoResultado);

        aleatorio = new Random();
    }

    public void actionPerformed(ActionEvent event) {
        int dado1, dado2;
        dado1 = aleatorio.nextInt(6) + 1;
        dado2 = aleatorio.nextInt(6) + 1;

        campoTextoValores.setText("valores de los dados: "
            + Integer.toString(dado1) + " y "
            + Integer.toString(dado2));
        if (dado1 == dado2) {
            campoTextoResultado.setText("dados iguales - gana");
        }
        else {
            campoTextoResultado.setText("dados no iguales - pierde");
        }
    }
}
```

## Múltiples eventos

La instrucción **if** puede cumplir un importante papel en el manejo de múltiples eventos. Por ejemplo, un programa tiene dos botones con las leyendas feliz y triste, como se muestra en la figura 7.8. Al hacer clic en un botón, el programa muestra una imagen apropiada. Sin importar en cuál de los botones se haga clic, se hace una llamada al mismo método **actionPerformed**. Entonces, ¿cómo podemos diferenciar uno del otro? Se utiliza una instrucción **if** para detectar cuál fue el botón que se oprimió. Después se puede llevar a cabo la acción correspondiente.

Cuando ocurre un evento de botón, el sistema operativo llama al método **actionPerformed**. Esta parte, crucial, del programa es:

```
public void actionPerformed(ActionEvent event) {  
    Graphics papel = panel.getGraphics();  
    Object origen = event.getSource();  
    if (origen == botónFeliz) {  
        imagenFeliz.paintIcon(this, papel, 0, 0);  
    }  
    else {  
        imagenTriste.paintIcon(this, papel, 0, 0);  
    }  
}
```

El parámetro que se pasa a **actionPerformed**, de nombre **event** en el programa, provee información sobre la naturaleza del evento. Al usar el método **getSource** en **event** se devuelve el objeto responsable del evento. Colocamos esto en una variable llamada **origen**, un objeto de la clase **Object**. Después utilizamos una instrucción **if** para comparar este objeto con los posibles candidatos. Así, utilizamos el operador **==** para comparar objetos, no números o cadenas (tenga en cuenta que no es el operador **=**).



Figura 7.8 Múltiples botones.

He aquí el texto completo del programa:

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

public class FelizOTriste extends JFrame implements ActionListener {

    private JButton botónFeliz, botónTriste;
    private JPanel panel;

    private ImageIcon imagenFeliz, imagenTriste;

    public static void main(String[] args) {
        FelizOTriste demo = new FelizOTriste();
        demo.setSize(175,175);
        demo.crearGUI();
        demo.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);

        botónFeliz = new JButton("feliz");
        ventana.add(botónFeliz);
        botónFeliz.addActionListener(this);

        botónTriste= new JButton("triste");
        ventana.add(botónTriste);
        botónTriste.addActionListener(this);

        imagenFeliz = new ImageIcon("feliz.jpg");
        imagenTriste = new ImageIcon("triste.jpg");
    }

    public void actionPerformed(ActionEvent event) {
        Graphics papel = panel.getGraphics();
        Object origen = event.getSource();
        if (origen == botónFeliz) {
            imagenFeliz.paintIcon(this, papel, 0, 0);
        }
        else {
            imagenTriste.paintIcon(this, papel, 0, 0);
        }
    }
}
```

## PRÁCTICA DE AUTOEVALUACIÓN

- 7.4 Un programa muestra tres botones, con las leyendas 1, 2 y 3. Escriba las instrucciones para mostrar el número del botón en el que se hizo clic.

### ● And, or, not

A menudo en programación necesitamos evaluar dos cosas a la vez. Suponga, por ejemplo, que necesitamos evaluar si alguien debe pagar una tarifa reducida por un boleto:

```
if (edad >= 6 && edad < 16) {
    campoTexto.setText("tarifa infantil");
}
```

La palabra `&&` es uno de los operadores lógicos de Java; significa “and” en inglés e “y” en español común.

Se pueden utilizar paréntesis adicionales para mejorar la legibilidad de estas condiciones más complejas. Por ejemplo, podemos replantear la instrucción anterior de la siguiente manera:

```
if ((edad >= 6) && (edad < 16)) {
    campoTexto.setText("tarifa infantil");
}
```

Aunque los paréntesis internos no son esenciales, sirven para diferenciar las dos condiciones que se están evaluando.

Podría verse tentado a escribir:

```
if (edad >= 6 && < 16) // ¡error!
```

pero es incorrecto, ya que las condiciones se tienen que establecer por completo, como se muestra a continuación:

```
if (edad >= 6 && edad < 16) // OK
```

Podríamos utilizar el operador `||` (que significa “or” en inglés y “o” en español) en una instrucción `if` de la siguiente forma:

```
if (edad < 6 || edad >= 60) {
    campoTexto.setText("tarifa reducida");
}
```

aquí se aplica la tarifa reducida cuando las personas son menores de 6 o de 60 en adelante.

El operador `!` significa “not” en inglés (“no” en español) y se utiliza mucho en la programación. He aquí un ejemplo del uso del operador `!`:

```
if (! (edad > 16)) {
    campoTexto.setText("demasiado joven");
}
```

Esto significa: evaluar si la edad es mayor de 16. Si el resultado es verdadero, el `!` lo convierte en falso. Si es falso, el `!` lo convierte en verdadero. Entonces, si el resultado es verdadero, mostrar el mensaje. Desde luego que esto se puede escribir de una manera más sencilla sin el operador `!`.

## PRÁCTICA DE AUTOEVALUACIÓN

**7.5** Rediseñe la instrucción **if** anterior sin utilizar el operador **!**.



Figura 7.9 El programa de los dados.

El siguiente programa ilustra una serie más compleja de evaluaciones. Se lanzan dos dados en un juego de apuestas y el programa tiene que decidir cuál es el resultado. El programa usa dos controles deslizables, cada uno con un rango de 1 a 6 para especificar los valores de cada uno de los dos dados (figura 7.9). El resultado se muestra en dos campos de texto. Para empezar usaremos la regla que establece que sólo una puntuación total de 6 gana.

El programa se muestra a continuación. Cada vez que se mueve uno de los dos controles deslizables, el programa muestra el valor total y utiliza una instrucción **if** para determinar si el jugador ganó.

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

public class Dados extends JFrame implements ChangeListener {
    private JSlider deslizante1, deslizante2;
    private JTextField campoTextoTotal, campoTextoComentario;

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

    private void crearGUI() {
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        Container ventana = getContentPane();
        ventana.setLayout(new FlowLayout());
        deslizante1 = new JSlider(1, 6, 3);
        deslizante1.addChangeListener(this);
        ventana.add(deslizante1);
    }
}
```

```

deslizante2 = new JSlider(1, 6, 3);
deslizante2.addChangeListener(this);
ventana.add(deslizante2);

campoTextoTotal = new JTextField(10);
ventana.add(campoTextoTotal);

campoTextoComentario= new JTextField(10);
ventana.add(campoTextoComentario);
}

public void stateChanged(ChangeEvent e) {
    int dado1, dado2, total;

    dado1 = deslizante1.getValue();
    dado2 = deslizante2.getValue();
    total = dado1 + dado2;
    campoTextoTotal.setText("el total es " + total);
    if (total == 6) {
        campoTextoComentario.setText("usted ha ganado");
    }
    else {
        campoTextoComentario.setText("usted ha perdido");
    }
}
}

```

Ahora vamos a modificar las reglas y a rediseñar el programa. Suponga que cualquier par de valores idénticos gana; por ejemplo, los dos dados con uno, con dos, etc. Entonces la instrucción **if** sería:

```

if (dado1 == dado2) {
    campoTextoComentario.setText("usted ha ganado");
}

```

Ahora supongamos que usted sólo gana si obtiene un total de 2 o 7:

```

if ((total == 2) || (total == 7)) {
    campoTextoComentario.setText("usted ha ganado");
}

```

Observe de nuevo que hemos encerrado cada una de las condiciones entre paréntesis. Estos paréntesis no son estrictamente necesarios en Java, pero ayudan mucho para aclarar el significado de la condición a evaluar.

En la siguiente tabla se sintetizan los operadores **and**, **or** y **not** de Java:

Símbolo	Significado
&&	and
	or
!	not

## PRÁCTICAS DE AUTOEVALUACIÓN

- 7.6** Modifique el programa de los dados para que el jugador gane cuando obtenga un valor total de 2, 5 o 7.
- 7.7** Escriba instrucciones **if** para evaluar si alguien puede obtener un empleo de tiempo completo. La regla es que debe tener 16 o más años y ser menor que 65.

### ● Instrucciones **if** anidadas

Analice el siguiente fragmento de un programa:

```
if (edad < 6) {
    campoTexto.setText("tarifa infantil");
}
else {
    if (edad < 16) {
        campoTexto.setText("tarifa junior");
    }
    else {
        campoTexto.setText("tarifa de adulto");
    }
}
```

Aquí podemos ver que la segunda instrucción **if** está completamente dentro de la primera (la sangría ayuda a mejorar la legibilidad). A esto se le conoce como anidamiento. El anidamiento no es lo mismo que aplicar sangría al código; es sólo que la sangría hace el anidamiento muy evidente.

- El efecto general de esta pieza de código es:
- Si la edad es mayor que 6, se aplica la tarifa infantil.
  - De otra manera, si la edad es menor que 16, se utiliza la tarifa junior.
  - De otra manera, si ninguna de las anteriores condiciones es verdadera, se utiliza la tarifa de adulto.

Es común ver el anidamiento en los programas, pero un programa de este tipo tiene una complejidad que dificulta un poco su comprensión. Podemos rediseñar esta sección del programa en un estilo de instrucciones **if** anidadadas conocidas como instrucciones **else if**, como se muestra en el siguiente ejemplo:

```
if (edad < 6) {
    campoTexto.setText("tarifa infantil");
}
else if (edad < 16) {
    campoTexto.setText("tarifa junior");
}
else {
    campoTexto.setText("tarifa de adulto");
}
```

Esta versión es exactamente la misma que la anterior, sólo que la sangría es distinta y algunos de los pares de llaves se eliminaron. Esto se debe a que la regla establece que cuando hay una sola instrucción a ejecutar, podemos omitir las llaves. Ésta es la única ocasión en que recomendamos omitir las llaves.

He aquí una tercera versión de esta pieza de programa. A menudo es posible escribir un programa de una manera más simple utilizando los operadores lógicos. Por ejemplo, en el siguiente ejemplo se obtiene el mismo resultado que en el ejemplo anterior sin utilizar el anidamiento:

```
if (edad < 6) {
    campoTexto.setText("tarifa infantil");
}
if (edad >= 6) && (edad < 16)) {
    campoTexto.setText("tarifa junior");
}
if (edad >= 16) {
    campoTexto.setText("tarifa de adulto");
}
```

Ahora tenemos tres piezas de código que obtienen el mismo resultado final, dos con anidamiento y uno sin esta técnica. Algunas personas argumentan que es difícil entender el anidamiento, de tal forma que el programa es propenso a errores y por ende es mejor evitar el anidamiento. A menudo es posible evitarlo si utilizamos operadores lógicos.

## PRÁCTICAS DE AUTOEVALUACIÓN

- 7.8** Escriba un programa que reciba un salario mediante un control deslizable y determine cuánto impuesto debe pagar alguien de acuerdo con las siguientes reglas:

Las personas no deben pagar impuesto si ganan hasta \$10,000. Deben pagar el 20% de impuesto si ganan más de \$10,000 y hasta \$50,000. Deben pagar el 90% de impuesto si ganan más de \$50,000. El control deslizable debe tener un rango de 0 a 100,000.

- 7.9** Escriba un programa que genere tres controles deslizables y muestre el mayor de los tres valores.

- 7.10** La agencia de viajes Joven y Bella sólo acepta clientes entre los 18 y 30 años (si es menor de 18 no tiene dinero; si es mayor de 30 tiene demasiadas arrugas). Escriba un programa para evaluar si usted puede irse de vacaciones con esta empresa. La edad se introduce en un campo de texto. El resultado se muestra en un segundo campo de texto cuando se hace clic en un botón.

## ● La instrucción switch

Esta instrucción es otra forma de usar muchas instrucciones **if**. Siempre podrá realizar todo lo que necesite con la ayuda de las instrucciones **if**, pero **switch** puede ser una alternativa más eficiente en circunstancias apropiadas. Por ejemplo, suponga que necesitamos una pieza de código para mostrar el día de la semana como una cadena de texto. Supongamos que el programa representa el día de la semana como una variable **int** llamada **númeroDía**, que tiene uno de los valores del 1 al 7 para representar los días de lunes a domingo. Queremos convertir la versión entera del día en una versión de cadena de texto llamada **nombreDía**. Podríamos escribir la siguiente serie de instrucciones **if**:

```
if (númeroDía == 1) {  
    nombreDía = "Lunes";  
}  
if (númeroDía == 2) {  
    nombreDía = "Martes";  
}  
if (númeroDía == 3) {  
    nombreDía = "Miércoles";  
}  
if (númeroDía == 4) {  
    nombreDía = "Jueves";  
}  
if (númeroDía == 5) {  
    nombreDía = "Viernes";  
}  
if (númeroDía == 6) {  
    nombreDía = "Sábado";  
}  
if (númeroDía == 7) {  
    nombreDía = "Domingo";  
}
```

Aunque la anterior pieza de código está clara y bien estructurada, hay una alternativa que cumple la misma función, en la cual se utiliza la instrucción **switch**:

```
switch (númeroDía) {  
  
    case 1:  
        nombreDía = "Lunes";  
        break;  
  
    case 2:  
        nombreDía = "Martes";  
        break;
```

```

case 3:
    nombreDía = "Miércoles";
    break;

case 4:
    nombreDía = "Jueves";
    break;

case 5:
    nombreDía = "Viernes";
    break;

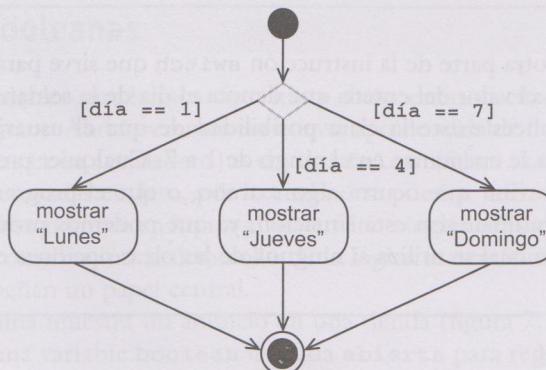
case 6:
    nombreDía = "Sábado";
    break;

case 7:
    nombreDía = "Domingo";
    break;
}

```

Se antepone la palabra **case** a cada uno de los posibles valores. La instrucción **break** transfiere el control al final de la instrucción **switch**, el cual está marcado con una llave de cierre. Esto nos permite ver con más claridad lo que se va a hacer, en contraste con la serie equivalente de instrucciones **if**.

Una instrucción **switch** como la anterior se puede visualizar como el diagrama de actividades de la figura 7.10.



**Figura 7.10** Diagrama de actividades que muestra parte de una instrucción **switch**.

## PRÁCTICA DE AUTOEVALUACIÓN

- 7.11** Escriba un método que convierta los enteros 1, 2, 3 y 4 en las palabras diamantes, corazones, tréboles y espadas, respectivamente.

Puede haber varias instrucciones en cada una de las opciones de una instrucción **switch**. Por ejemplo, una de las opciones podría ser:

```
case 6:
    JOptionPane.showMessageDialog(null, "hurra");
    nombreDía = "Sábado";
    break;
```

Otra característica de la instrucción **switch** es la de agrupar varias opciones, como en el siguiente ejemplo:

```
switch (númeroDía) {
    case 1:
    case 2:
    case 3:
    case 4:
    case 5:
        nombreDía = "entre semana";
        break;

    case 6:
    case 7:
        nombreDía = "fin de semana";
        break;
}
```

La opción **default** es otra parte de la instrucción **switch** que sirve para ciertos casos. Suponga en el ejemplo anterior que el valor del entero que denota el día de la semana se introduce mediante un cuadro de texto. Entonces existe la clara posibilidad de que el usuario introduzca en forma errónea un número que no se encuentre en el rango de 1 a 7. Cualquier programa decente necesita tener esto en cuenta para evitar que ocurra algo extraño, o que el programa falle. La instrucción **switch** es muy buena para lidiar con esta situación, ya que podemos proveer una opción “atrapa todo” o predeterminada, la cual se utiliza si ninguna de las otras opciones es válida:

```
switch (númeroDía) {

    case 1:
        nombreDía = "Lunes";
        break;

    case 2:
        nombreDía = "Martes";
        break;

    case 3:
        nombreDía = "Miércoles";
        break;
```

```

case 4:
    nombreDía = "Jueves";
    break;

case 5:
    nombreDía = "Viernes";
    break;

case 6:
    nombreDía = "Sábado";
    break;

case 7:
    nombreDía = "Domingo";
    break;

default:
    nombreDía = "día ilegal";
    break;
}

```

Si no se escribe una opción **default** como parte de una instrucción **switch** y ninguno de los casos provistos corresponde con el valor actual de la variable, entonces se ignoran todas las opciones.

## ● Variables booleanas

Todos los tipos de variables que hemos visto hasta ahora están diseñados para contener números, cadenas de texto u objetos. Ahora veremos un nuevo tipo de variable llamada **boolean**, la cual sólo puede contener el valor **true** (verdadero) o el valor **false** (falso). Las palabras **boolean**, **true** y **false** son palabras reservadas en Java, por lo cual no se pueden utilizar para ningún otro fin. Este tipo de variable recibió su nombre en honor al matemático inglés del siglo diecinueve George Boole, quien hizo una gran contribución al desarrollo de la lógica matemática, en donde las ideas de verdadero y falso desempeñan un papel central.

El siguiente programa muestra un anuncio en una tienda (figura 7.11). El anuncio dice abierta o cerrada. Se utiliza una variable **boolean** llamada **abierta** para registrar si la tienda está abierta (**true**) o cerrada (**false**). Hay dos botones que permiten al encargado de la tienda cambiar el anuncio a abierta o cerrada. Hay otros dos botones que prenden y apagan el anuncio. El programa muestra letras de tamaño grande mediante el método **setFont**.

La variable **boolean abierta** es una variable a nivel de clase, que al principio es falsa para indicar que la tienda está cerrada:

```
private boolean abierta = false;
```

Al hacer clic en el botón **Abierta**:

```
abierta = true;
```

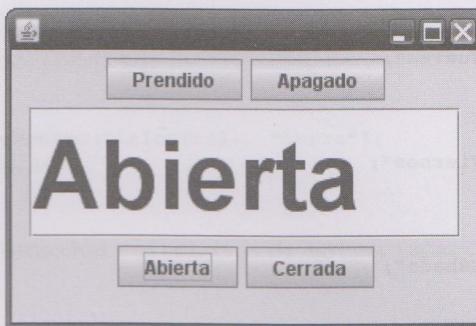


Figura 7.11 El anuncio de una tienda.

Al hacer clic en el botón **Cerrada**:

```
abierta = false;
```

Al hacer clic en el botón **Prendido**, el valor de **abierta** se evalúa mediante una instrucción **if** y se muestra el anuncio apropiado:

```
if (abierta) {
    campoTexto.setText("Abierta");
}
else {
    campoTexto.setText("Cerrada");
}
```

He aquí el programa completo:

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

public class AnuncioTienda extends JFrame implements ActionListener {
    private JButton botónPrendido, botónApagado, botónAbierta, botónCerrada;
    private JTextField campoTexto;
    private boolean prendido = false, abierta = false;
    public static void main(String[] args) {
        AnuncioTienda demo = new AnuncioTienda();
        demo.setSize(300,200);
        demo.crearGUI();
        demo.setVisible(true);
    }
}
```

```
private void crearGUI() {
    setDefaultCloseOperation(EXIT_ON_CLOSE);
    Container ventana = getContentPane();
    ventana.setLayout(new FlowLayout());
    botónPrendido = new JButton("Prendido");
    ventana.add(botónPrendido);
    botónPrendido.addActionListener(this);
    botónApagado = new JButton("Apagado");
    ventana.add(botónApagado);
    botónApagado.addActionListener(this);
    campoTexto = new JTextField(5);
    campoTexto.setSize(5, 100);
    campoTexto.setFont(new Font(null, Font.BOLD, 60));
    ventana.add(campoTexto);
    botónAbierta = new JButton("Abierta");
    ventana.add(botónAbierta);
    botónAbierta.addActionListener(this);
    botónCerrada = new JButton("Cerrada");
    ventana.add(botónCerrada);
    botónCerrada.addActionListener(this);
}

public void actionPerformed(ActionEvent event) {
    Object origen = event.getSource();
    if (origen == botónPrendido) {
        manejarBotónPrendido();
    }
    else if (origen == botónApagado) {
        manejarBotónApagado();
    }
    else if (origen == botónAbierta) {
        manejarBotónAbierta();
    }
    else manejarBotónCerrada();
    dibujarAnuncio();
}

private void manejarBotónPrendido() {
    prendido = true;
}

private void manejarBotónApagado() {
    prendido = false;
}

private void manejarBotónAbierta() {
    abierta = true;
}

private void manejarBotónCerrada() {
    abierta = false;
}
```

```
private void dibujarAnuncio() {  
    if abierta) {  
        campoTexto.setText("Abierta");  
    }  
    else {  
        campoTexto.setText("Cerrada");  
    }  
    if (!prendido) {  
        campoTexto.setText("");  
    }  
}
```

En el programa anterior, una de las instrucciones `if` es como se muestra a continuación, ya que la variable `abierta` puede ser verdadera o falsa, y se puede evaluar en forma directa:

```
if (abierta) {
```

Ésta es la forma más eficiente de evaluar el valor de una variable `boolean`. Se puede replantear en una forma menos concisa:

```
if (abierta == true) {
```

Para resumir, las variables `boolean` se utilizan en la programación para recordar si algo es verdadero o falso, tal vez por un tiempo corto o tal vez durante todo el tiempo que se ejecute el programa.

## PRÁCTICA DE AUTOEVALUACIÓN

- 7.12 El propietario de la tienda necesita un anuncio adicional que diga “OFERTA”. ¿Podemos seguir utilizando una variable `boolean`?

Los métodos pueden usar valores `boolean` como parámetros y como valores de retorno. Por ejemplo, he aquí un método que verifica si tres números están en orden:

```
private boolean enOrden(int a, int b, int c) {  
    if ((a <= b) && (b <= c)) {  
        return true;  
    }  
    else {  
        return false;  
    }  
}
```

## Comparación de cadenas

Hasta ahora hemos visto programas que usan los operadores de comparación (como `>`) para comparar números. Sin embargo, muchos programas necesitan comparar cadenas de texto. En estos casos los operadores de comparación no son apropiados, por lo que en vez de ellos utilizamos el método `equals`.

El programa de la bóveda que vimos antes requería que el usuario introdujera un código numérico. Suponga que ahora el código es alfabético. En este caso, el programa necesita comparar la cadena introducida con el código correcto (por decir, “Bill”). La instrucción `if` apropiada sería:

```
String código;
código = campoCódigo.getText();
if (código.equals("Bill")) {
    campoTextoResultado.setText("abierta");
}
```

Se hace una llamada al método `equals`. El parámetro es la cadena “Bill”. El método devuelve verdadero o falso. Después, la instrucción `if` actúa de manera acorde.

## Fundamentos de programación

Por lo general la computadora lleva a cabo las instrucciones una por una, en secuencia. Una instrucción `if` instruye a la computadora para que evalúe el valor de ciertos datos y después elija una de varias acciones dependiendo del resultado de la evaluación. A este proceso de elección se le conoce algunas veces como selección. A la evaluación de los datos se le conoce como condición. Una vez que se completa una instrucción `if`, la computadora sigue llevando a cabo las instrucciones en secuencia.

## Errores comunes de programación

### Paréntesis

La condición dentro de una instrucción `if` debe ir encerrada entre paréntesis. Por ejemplo:

```
if (a > b) etc.
```

### Igualdad

Si desea evaluar la igualdad, use el operador `==` (y no un solo signo de igual, `=`). El siguiente ejemplo es correcto:

```
if (a == b) etc.
```



*Errores comunes de programación (continuación)*

Por desgracia, un programa que utilice un solo = se compilará correctamente pero funcionará en forma incorrecta.

## Comparación de cadenas

Si desea comparar dos cadenas, debe usar el método `equals` como en el siguiente ejemplo:

```
if (cadena1.equals(cadena2)) etc.
```

## Llaves

Ahora veamos las llaves. Esta instrucción es completamente correcta:

```
if (código == 123)
    campoTextoResultado.setText("abierta");
```

aun cuando se omitieron las llaves que rodean a la instrucción. La regla de Java es que si sólo hay **una** instrucción a ejecutar, entonces no son necesarias las llaves. Sin embargo, esto puede ocasionar molestos problemas de programación, por lo que nuestro mejor consejo es insertar las llaves en todo momento. Hay una excepción a esta sugerencia cuando usamos instrucciones `if` anidadas en el estilo `else if`, como explicamos antes.

## Condiciones compuestas

Tal vez alguna vez llegue a escribir una instrucción `if` de esta forma:

```
if (a > 18 && < 25)
```

lo cual es incorrecto. El signo `&&` debe enlazar dos condiciones completas, de preferencia entre paréntesis para mejorar la claridad, como en el siguiente ejemplo:

```
if ((a > 18) && (a < 25))
```

## switch

La instrucción `switch` es muy útil, pero por desgracia no es tan flexible como quisieramos. Suponga, por ejemplo, que deseamos escribir una pieza de programa para mostrar dos números, con el mayor primero, seguido del menor. Si usamos instrucciones `if` podríamos escribir lo siguiente:

```
if (a > b) {
    campoTexto.setText(Integer.toString(a) + " es mayor que "
        + Integer.toString(b));
}
```

```

if (b > a) {
    campoTexto.setText(Integer.toString(b) + " es mayor que "
                      + Integer.toString(a));
}
if (a == b) {
    campoTexto.setText("son iguales");
}

```

Podríamos vernos tentados a rediseñar esto mediante una instrucción **switch**, como se muestra a continuación:

```

switch (?) {      // ¡cuidado! Java inválido
    case a > b:
        campoTexto.setText(Integer.toString(a) + " es mayor que "
                           + Integer.toString(b));
        break;
    case b > a:
        campoTexto.setText(Integer.toString(b) + " es mayor que "
                           + Integer.toString(a));
        break;
    case a == b:
        campoTexto.setText("son iguales");
        break;
}

```

Pero esto no se permite ya que, como lo indica el signo de interrogación, **switch** sólo trabaja con una sola variable entera como sujeto y **case** no puede utilizar los operadores **>**, **==**, **<**, etc.

## Secretos de codificación

El primer tipo de instrucción **if** tiene la siguiente estructura:

```

if (condición) {
    instrucciones
}

```

El segundo tipo de instrucción **if** tiene la siguiente estructura:

```

if (condición) {
    instrucciones
}
else {
    instrucciones
}

```





### Secretos de codificación (continuación)

La instrucción **switch** tiene la siguiente estructura:

```
switch (variable) {  
    case valor1:  
        instrucciones  
        break;  
    case valor2:  
        instrucciones  
        break;  
    default:  
        instrucciones  
        break;  
}
```

La sección predeterminada (**default**) es opcional.

## Nuevos elementos del lenguaje

- Estructuras de control para decisiones:

```
if, else  
switch, case, break, default
```

- Los operadores de comparación **>**, **<**, **==**, **!=**, **<=** y **>=**.
- Los operadores lógicos **&&**, **||** y **!**.
- Las variables declaradas como **boolean**, que pueden contener el valor **true** o el valor **false**.

## Resumen

Las instrucciones **if** permiten al programador controlar la secuencia de acciones al hacer que el programa lleve a cabo una evaluación. Después de realizar la evaluación, la computadora ejecuta una de varias acciones disponibles. Hay dos variedades de una instrucción **if**:

- **if**
- **if...else**

La instrucción **if** se puede utilizar para identificar el origen de un evento de GUI. El método **getSource** devuelve el objeto que produjo el evento. Este objeto se compara con cada uno de los objetos que pudieron haber causado el evento, mediante el operador de comparación **==**.

La instrucción **switch** ofrece una manera conveniente de llevar a cabo varias evaluaciones. Sin embargo, está restringida a realizar evaluaciones con enteros.

Una variable **boolean** puede contener el valor **true** o el valor **false**. Se puede evaluar mediante una instrucción **if**. Una variable **boolean** es útil en situaciones en las que una variable sólo tiene dos valores significativos.

## Ejercicios

**7.1 Precio de función de cine** Escriba un programa para averiguar cuánto paga una persona por ir a una función de cine. El programa debe recibir una edad desde un control deslizable o un campo de texto y después debe decidir con base en lo siguiente:

- Menor de 5 años, la función es gratis.
- De 5 a 12, paga la mitad de la tarifa.
- De 13 a 54, paga la tarifa completa.
- De 55 o mayor, la función es gratis.

**7.2 El elevador** Escriba un programa para simular un elevador muy primitivo. Para representar al elevador utilice un cuadrado relleno de color negro que se muestre en un panel alto, delgado y blanco. Debe haber dos botones: uno para que se desplace 20 píxeles hacia arriba por el panel y otro para que baje. Despu s mejore el programa para asegurarse de que el elevador no suba o baje demasiado.

**7.3 Ordenamiento** Escriba un programa que reciba n m eros de tres controles deslizables o tres campos de texto y los muestre por tama o num rico en orden de menor a mayor.

**7.4 Apuestas** Un grupo de personas desean apostar sobre el resultado de tres lanzamientos de un dado. Una persona debe apostar \$1 para tratar de adivinar el resultado de los tres lanzamientos. Escriba un programa que utilice el m todo de los n m eros aleatorios para simular tres lanzamientos de un dado y que muestre las ganancias de acuerdo con las siguientes reglas:

- Los tres lanzamientos cayeron en seis: gana \$20.
- Los tres lanzamientos cayeron en el mismo n m ero (pero no en seis): gana \$10.
- Dos de tres lanzamientos cayeron en el mismo n m ero: gana \$5.

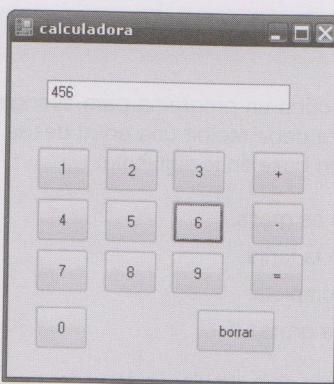
**7.5 B veda de combinaci n digital** Escriba un programa que act e como un candado de combinaci n digital para una b veda. Cree tres botones que representen los n m eros 1, 2 y 3. El usuario debe hacer clic en los botones para tratar de adivinar los n m eros correctos (por ejemplo, 331121). El programa debe permanecer sin hacer nada hasta que se opriman los botones correctos. Despu s debe felicitar al usuario con un mensaje apropiado. Debe haber un bot n para reiniciar.

Mejore el programa de manera que tenga otro bot n para que el usuario pueda modificar la combinaci n de la b veda, siempre y cuando se haya introducido el c digo correcto.

**7.6 Repartir una carta** Escriba un programa con un solo bot n que, al ser oprimido, seleccione al azar una carta. Primero utilice el generador de n m eros aleatorios de la biblioteca para crear un n m ero en el rango de 1 a 4. Despu s convierta el n m ero en un palo (coraz n, diamante, tr bol y espada). Luego utilice el generador de n m eros aleatorios para crear un n m ero aleatorio en el rango de 1 a 13. Convierta el n m ero en un as, un 2, 3, etc., y finalmente muestre el valor de la carta elegida.

Sugerencia: utilice **switch** de la manera m s apropiada.

**7.7 Juego de piedra, papel o tijera** En su forma original, cada uno de los dos jugadores elige al mismo tiempo piedra, papel o tijera. La piedra gana a la tijera, el papel gana a la piedra y la tijera gana al papel. Si ambos jugadores eligen la misma opci n, es un empate. Escriba un programa para jugar este juego. El jugador debe seleccionar uno de tres botones marcados como piedra, papel o tijera. La computadora debe realizar su elecci n al azar mediante el generador de n m eros aleatorios. Tambi n debe decidir y mostrar qui n gana.



**Figura 7.12** La calculadora.

- 7.8 La calculadora** Escriba un programa que simule una calculadora simple de escritorio (figura 7.12), que opere con números enteros. Debe tener un botón para cada uno de los 10 dígitos del 0 al 9. También debe tener un botón para sumar y otro para restar. Además debe contar con un botón **borrar** para borrar la pantalla (un campo de texto) y un botón de igual (=) para obtener la respuesta.

Al oprimir el botón para borrar la pantalla, ésta debe quedar en cero y el total (oculto) debe quedar también en 0.

Al oprimir el botón de un dígito, éste se debe agregar a la derecha de los dígitos que ya se encuentren en la pantalla (en caso de haber alguno).

Al oprimir el botón +, el número en la pantalla se deberá sumar al total (haga lo mismo para el botón -).

Al oprimir el botón = deberá aparecer el valor del total.

- 7.9 Nim** Este juego se juega con cerillos. No importa cuántos cerillos haya. Hay que colocarlos en tres pilas. Tampoco importa cuántos cerillos haya en cada pila. Los jugadores van por turnos. Cada jugador puede quitar cualquier cantidad de cerillos de cualquier pila, pero sólo de una pila. Además, cada jugador debe quitar por lo menos un cerillo. El ganador es la persona que haga que el otro jugador tome el último cerillo.

Escriba un programa para jugar este juego. Al principio la computadora debe repartir tres pilas de cerillos, con un número aleatorio (en el rango de 1 a 200) de cerillos en cada pila. Las tres cantidades se deben mostrar en campos de texto. La computadora debe participar como jugador, debe elegir una pila y una cantidad de cerillos al azar. El otro jugador será el usuario humano, quien debe especificar el número de pila con un botón y la cantidad de cerillos mediante un campo de texto.

También debe haber un botón "nuevo juego".

- 7.10 Gráficos de tortuga** Este tipo de gráficos es una forma de facilitar a los niños el aprendizaje de la programación. Imagine una pluma fija en la panza de una tortuga. A medida que la tortuga se mueve por el piso, la pluma deja un trazo. La tortuga puede avanzar hacia el norte, sur, este y oeste. Además, la tortuga puede recibir órdenes mediante un botón para cada orden, de la siguiente manera:

- Subir pluma.
- Bajar pluma.
- Voltear a la izquierda 90°.
- Voltear a la derecha 90°.
- Avanzar  $n$  píxeles.

En un principio la tortuga se encuentra en la parte superior izquierda del panel y de cara hacia el este. Con base en esto, podemos dibujar un rectángulo si utilizamos la siguiente secuencia de ejemplo:

- |   |  |
|---|--|
| <ol style="list-style-type: none"> <li>1. Bajar pluma.</li> <li>2. Avanzar 20 píxeles.</li> <li>3. Voltear a la derecha 90°.</li> <li>4. Avanzar 20 píxeles.</li> </ol> | <ol style="list-style-type: none"> <li>5. Voltear a la derecha 90°.</li> <li>6. Avanzar 20 píxeles.</li> <li>7. Voltear a la derecha 90°.</li> <li>8. Avanzar 20 píxeles.</li> </ol> |
|---|--|

Escriba un programa que se comporte como la tortuga, con un botón para cada una de las órdenes. Utilice un control deslizable o un cuadro de texto para introducir el número de píxeles ( $n$ ) que debe avanzar la tortuga. La dirección de la tortuga (norte, sur, este, oeste) se debe mostrar en un campo de texto.

## Respuestas a las prácticas de autoevaluación

- 7.1     if (código == 123) {  
    campoTextoResultado.setText("abierta");  
    campoCódigo.setText("");  
}
- 7.2     No, ya que consideran a la edad específica de 18 de manera distinta.
- 7.3     La parte esencial de este programa es:
- ```
if (temp < min) {  

    min = temp;  

}  

campoTexto.setText("El valor mínimo es " + min);
```
- 7.4     public void actionPerformed(ActionEvent event) {  
 Object origen;  
 origen = event.getSource();  
 if (origen == botón1) {  
 campoTexto.setText("botón 1");  
 }  
 if (origen == botón2) {  
 campoTexto.setText("botón 2");  
 }  
 if (origen == botón3) {  
 campoTexto.setText("botón 3");  
 }  
}



*Respuestas a las prácticas de autoevaluación (continuación)*

```
7.5    if (edad <= 16) {  
        campoTexto.setText("demasiado joven");  
    }  
  
7.6    if ((total == 2) || (total == 5) || (total == 7)) {  
        campoTexto.setText("usted ha ganado");  
    }  
  
7.7    if ((edad >= 16) && (edad < 65)) {  
        JOptionPane.showMessageDialog(null, "puede obtener un empleo");  
    }  
  
7.8    int salario, impuestos;  
  
        salario = deslizante.getValue();  
  
        if ((salario > 10000) && (salario <= 50000)) {  
            impuestos = (salario - 10000) / 5;  
        }  
        if (salario > 50000) {  
            impuestos = 8000 + ((salario - 50000) * 9 / 10);  
        }  
        if (salario <= 10000) {  
            impuestos = 0;  
        }  
  
7.9    public void stateChanged(ChangeEvent e) {  
        int a, b, c;  
        int mayor;  
  
        a = deslizante1.getValue();  
        b = deslizante2.getValue();  
        c = deslizante3.getValue();  
  
        if ((a >= b) && (a >= c)) {  
            mayor = a;  
        }  
        else if ((b >= a) && (b >= c)) {  
            mayor = b;  
        }  
        else {  
            mayor = c;  
        }  
        JOptionPane.showMessageDialog(null,  
            "el valor mayor es " + mayor);  
    }
```

7.10

```

int edad;
edad = Integer.parseInt(campoTextoEdad.getText());
if ((edad >= 18) && (edad <= 30)) {
    campoTextoResultado.setText("puede irse de vacaciones");
}

```

7.11

```

private String convertir(int p) {
    String palo;

    switch (s) {
        case 1:
            palo = "diamantes";
            break;
        case 2:
            palo = "corazones";
            break;
        case 3:
            palo = "tréboles";
            break;
        case 4:
            palo = "espadas";
            break;
        default:
            palo = "error";
            break;
    }
    return palo;
}

```

7.12 No, porque ahora hay tres posibles valores, no dos.