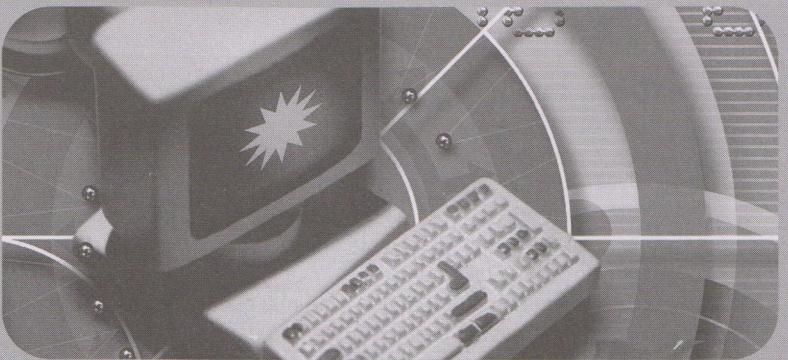


CAPÍTULO

5



Métodos y parámetros

En este capítulo conoceremos cómo:

- Escribir métodos.
- Utilizar parámetros formales y actuales.
- Pasar parámetros a los métodos.
- Usar `return` en los métodos.
- Sobrecargar métodos.

● Introducción

Los programas grandes pueden ser complejos, difíciles de comprender y de depurar. La técnica más importante para reducir la complejidad es dividir un programa en secciones (relativamente) aisladas. Esto nos permite enfocarnos en una sección aislada sin las distracciones del programa completo. Además, si la sección tiene un nombre, podemos “llamarla” o “invocarla” (hacer que se utilice) con sólo utilizar su nombre. En cierta forma, nos permite pensar a un nivel más alto. En Java, a dichas secciones se les conoce como métodos. En el capítulo 3 utilizamos una buena cantidad de métodos gráficos predefinidos para dibujar figuras en la pantalla.

Veamos de nuevo el método `drawRect`, que podemos llamar con cuatro parámetros de la siguiente forma:

```
papel.drawRect(10, 20, 60, 60);
```

En primer lugar, al utilizar parámetros (los elementos entre paréntesis) podemos controlar el tamaño y la posición del rectángulo. Esto asegura que `drawRect` sea lo bastante flexible como para funcionar en diversas circunstancias. Los parámetros modifican sus acciones.

En segundo lugar, cabe mencionar que si no existiera `drawRect` de todas formas podríamos producir un rectángulo mediante cuatro llamadas a `drawLine`. Agrupar las cuatro instrucciones `drawLine` dentro de un método denominado `drawRect` sería una idea sensata, ya que permite al programador pensar a un nivel más alto.

Cómo escribir sus propios métodos

En esta sección veremos cómo crear nuestros propios métodos. Empezaremos con un pequeño ejemplo para simplificar las cosas, y después veremos un ejemplo más práctico.

La Compañía Mundial de Cajas de Cartón tiene un logotipo que consiste en tres cuadrados, uno dentro de otro, como se muestra en la figura 5.1. La empresa desea utilizar el logotipo en varias posiciones en la pantalla, como se muestra en la figura 5.2. He aquí el código para dibujar dos logotipos idénticos en las posiciones (10, 20) y (100, 100):

```
// dibuja el logotipo en la esquina superior izquierda
papel.drawRect(10, 20, 60, 60);
papel.drawRect (10, 20, 40, 40);
papel.drawRect (10, 20, 20, 20);

// dibuja el logotipo en la esquina inferior derecha
papel.drawRect (100, 100, 60, 60);
papel.drawRect (100, 100, 40, 40);
papel.drawRect (100, 100, 20, 20);
```

Observe que los cuadrados son de 20, 40 y 60 píxeles, y sus esquinas superiores izquierdas están en el mismo punto. Si analiza el código observará que en esencia se repiten las tres instrucciones para dibujar un logotipo, excepto por la posición de la esquina superior izquierda del logotipo. Vamos a agrupar esas tres instrucciones para formar un método, de manera que se pueda dibujar un logotipo con una sola instrucción.

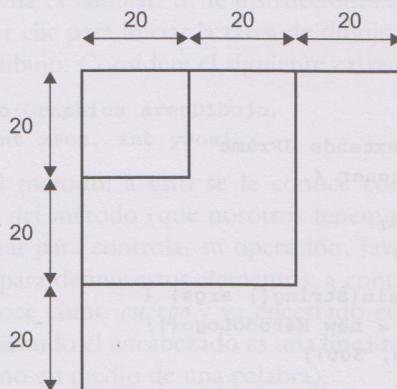


Figura 5.1 El logotipo de la empresa.

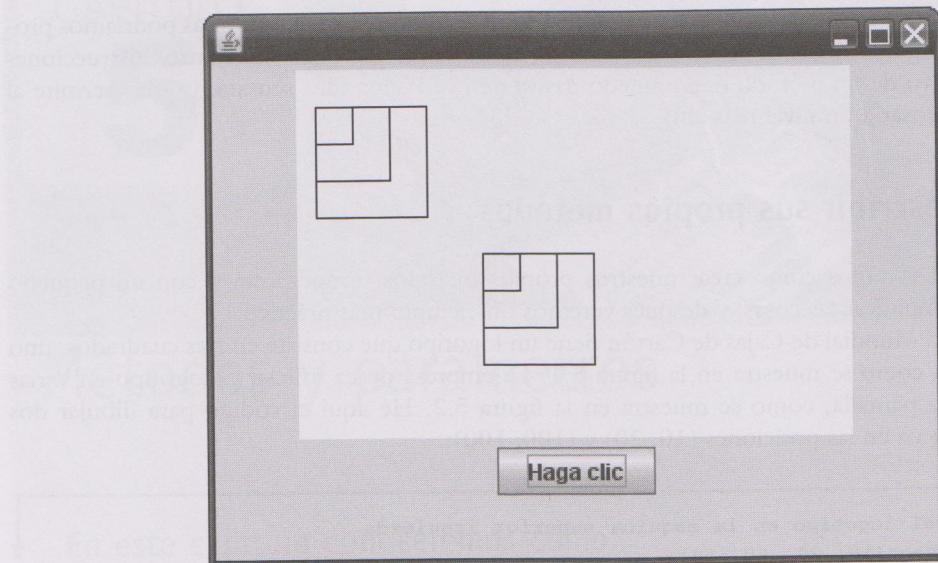


Figura 5.2 Pantalla del programa MétodoLogo.

● Nuestro primer método

He aquí un programa completo llamado **MétodoLogo**. Este programa muestra cómo crear y utilizar un método, al cual llamaremos **dibujarLogo**. La convención de estilo de Java es empezar los nombres de los métodos con minúscula.

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

public class MétodoLogo extends JFrame
    implements ActionListener {

    private JButton botón;
    private JPanel panel;

    public static void main(String[] args) {
        MétodoLogo marco = new MétodoLogo();
        marco.setSize(350, 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);  
  
    botón = new JButton("Haga clic");  
    ventana.add(botón);  
    botón.addActionListener(this);  
}  
  
public void actionPerformed(ActionEvent event) {  
    Graphics papel = panel.getGraphics();  
    dibujarLogo(papel, 10, 20);  
    dibujarLogo(papel, 100, 100);  
}  
  
private void dibujarLogo(Graphics áreaDibujo,  
    int xPos, int yPos) {  
    áreaDibujo.drawRect(xPos, yPos, 60, 60);  
    áreaDibujo.drawRect(xPos, yPos, 40, 40);  
    áreaDibujo.drawRect(xPos, yPos, 20, 20);  
}
```

El programa tiene un panel para dibujar y un botón. La interfaz de usuario es idéntica a nuestros programas de dibujo del capítulo 3. Al hacer clic en el botón se dibujan dos logotipos, como se muestra en la figura 5.2.

El concepto de los métodos y parámetros es una importante habilidad que los programadores necesitan dominar. Ahora analizaremos el programa con detalle.

La forma general del programa es familiar: tiene instrucciones `import` en la parte superior y un botón en el que podemos hacer clic para iniciar la tarea de dibujo. La parte `actionPerformed` es la que se encarga de iniciar el dibujo. Considere el siguiente extracto:

```
private void dibujarLogo(Graphics áreaDibujo,  
    int xPos, int yPos) {
```

Aquí se declara (introduce) el método; a esto se le conoce como encabezado del método. El encabezado declara el nombre del método (que nosotros tenemos la libertad de elegir) y los elementos que se deben suministrar para controlar su operación. Java utiliza los términos *parámetros actuales* y *parámetros formales* para definir estos elementos; a continuación hablaremos sobre ellos. Al resto del método se le conoce como *cuerpo* y va encerrado entre los caracteres { y }; aquí es donde se realiza el trabajo. A menudo el encabezado es una línea tan extensa que podemos dividirla en puntos adecuados (aunque no en medio de una palabra).

Una importante decisión que debe tomar el programador es el lugar desde donde se puede llamar al método. Hay dos opciones principales:

- El método sólo se puede llamar desde el interior del programa actual. En este caso utilizamos la palabra clave **private**.
- El método se puede llamar desde otro programa. En este caso utilizamos la palabra clave **public**. Los métodos como **drawRect** son ejemplos de métodos que se han declarado como **public**, puesto que son de uso general. (Para crear métodos **public** se requiere un conocimiento más profundo de los conceptos orientados a objetos; hablaremos sobre esto con más detalle en el capítulo 9).

Otras decisiones que debe tomar el programador son:

- ¿Realizará el método una tarea sin necesidad de producir un resultado? En este caso, utilizamos la palabra clave **void** después de **private**.
- ¿Calculará el método un resultado y lo devolverá a la sección de código que lo llamó (invocó)? En este caso tenemos que declarar el tipo del resultado, en vez de usar **void**. Más adelante en el capítulo veremos cómo hacerlo.

En el caso de **dibujarLogo**, su tarea es dibujar líneas en la pantalla, no proveer la respuesta a un cálculo. Por ende, utilizamos **void**.

● Cómo llamar a un método

Para llamar a un método privado en Java tiene que indicar su nombre, junto con una lista de parámetros entre paréntesis. En nuestro programa la primera llamada es:

```
dibujarLogo(papel, 10, 20);
```

Esta instrucción tiene dos efectos:

- Los valores de los parámetros se transfieren al método de manera automática. Más adelante hablaremos sobre esto con mayor detalle.
- El programa salta al cuerpo del método (las instrucciones después del encabezado) y ejecuta las instrucciones. Cuando termina con todas las instrucciones y llega al carácter }, la ejecución continúa en el punto desde el que se hizo la llamada al método.

Después se lleva a cabo la segunda llamada:

```
dibujarLogo(papel, 100, 100);
```

En la figura 5.3 se ilustra este proceso. Hay dos llamadas que producen dos logotipos.

● Cómo pasar parámetros

Es imprescindible comprender de la mejor forma posible cómo se transfieren (pasan) los parámetros a los métodos. En nuestro ejemplo el concepto se muestra en las siguientes líneas:

```
dibujarLogo(papel, 10, 20);
private void dibujarLogo(Graphics áreaDibujo,
                           int xPos, int yPos) {
```

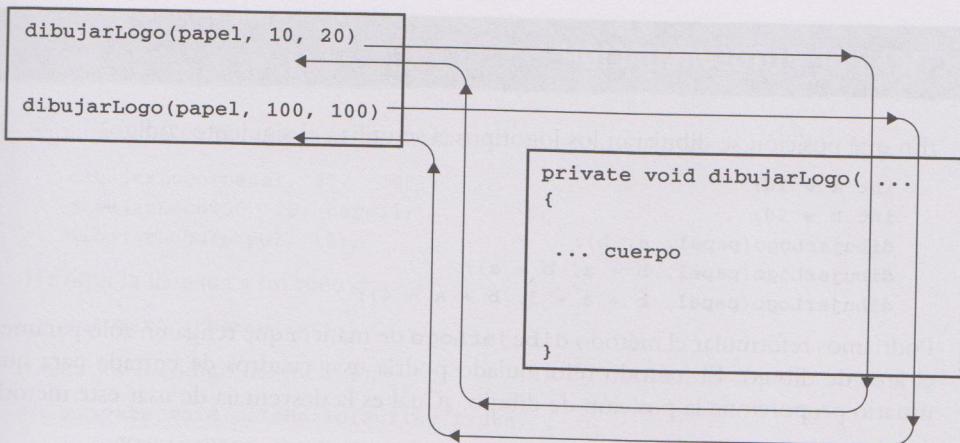


Figura 5.3 Ruta de ejecución de dos llamadas.

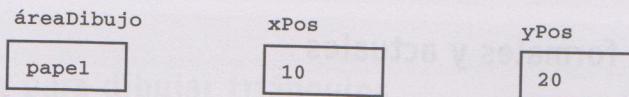


Figura 5.4 Proceso de transferencia de los parámetros actuales hacia los parámetros formales.

El área en la que nos debemos enfocar es en las dos listas de elementos entre paréntesis. En la llamada al método, los elementos se denominan *parámetros actuales*. En el encabezado del método, los elementos se denominan *parámetros formales*. Para aclarar esta situación, vamos a extraer los parámetros actuales y formales:

Parámetros actuales:	papel	10	20
Parámetros formales:	áreaDibujo	xPos	yPos

Recordemos la comparación que hicimos de una variable con una caja. Dentro del método hay un conjunto de cajas vacías (los parámetros) que esperan la transferencia de los valores de los parámetros. Después de la transferencia tenemos la situación que se muestra en la figura 5.4. No tenemos valores numéricos que podamos pasar al área de dibujo, por lo que nos enfocaremos en la forma en que se pasan las coordenadas.

La transferencia se realiza en un orden de izquierda a derecha. La llamada debe proporcionar el número y tipos correctos de los parámetros. Si el que hace la llamada (el usuario) envía de manera accidental los parámetros en el orden incorrecto, ¡el proceso de la transferencia no los regresará a su orden correcto! Cuando se ejecuta el método `dibujarLogo`, los valores antes mencionados controlan el proceso de dibujo. Aunque en el ejemplo anterior llamamos al método con números, podemos utilizar expresiones (es decir, incluir variables y cálculos) como en el siguiente ejemplo:

```

int x = 6;
dibujarLogo (papel, 20 + 3, 3 * 2 + 1); // 23 y 7
dibujarLogo(papel, x * 4, 20); // 24 y 20

```

PRÁCTICAS DE AUTOEVALUACIÓN

5.1 ¿En qué posición se dibujarán los logotipos si se utiliza el siguiente código?

```
int a = 10;
int b = 20;
dibujarLogo(papel, a, b);
dibujarLogo(papel, b + a, b - a);
dibujarLogo(papel, b + a - 3, b + a - 4);
```

5.2 Podríamos reformular el método `dibujarLogo` de manera que tenga un solo parámetro: el área de dibujo. El método reformuladoaría usar cuadros de entrada para que el usuario proporcione la posición de dibujo. ¿Cuál es la desventaja de usar este método?

● Parámetros formales y actuales

Hay dos listas entre paréntesis involucradas en nuestro estudio, por lo cual es importante aclarar el propósito de cada lista:

- El programador que escribe el código debe elegir qué elementos solicitará el método por medio de parámetros formales. Por ende, en el método `dibujarLogo` las medidas de los cuadrados anidados siempre se establecen en 20, 40 y 60, por lo que el que hace la llamada al método no necesita suministrar estos datos. Sin embargo, tal vez sí requiera variar la posición del logotipo, por lo cual hemos convertido estos elementos en parámetros.
- El escritor del método debe elegir el nombre de cada parámetro formal. Si se utilizan nombres similares en otros métodos no hay ningún problema, puesto que cada método tiene su propia copia de sus parámetros. En otras palabras, el escritor tiene la libertad de elegir cualquier nombre.
- Se debe proporcionar el tipo de cada parámetro formal y debe anteponerse al nombre del parámetro. Los tipos dependen del método en particular. Se utiliza una coma para separar un parámetro de otro. En el encabezado de `dibujarLogo` podrá ver esta disposición.
- El que hace la llamada debe proveer una lista de parámetros actuales entre paréntesis. Los parámetros deben estar en el orden que requiere el método y deben ser del tipo correcto.

Los dos beneficios de utilizar un método para dibujar el logotipo son que evitamos duplicar las tres instrucciones `drawRect` cuando se requieren varios logotipos, y que al dar un nombre a esta tarea podemos pensar a un nivel más alto.

Por último, estamos conscientes de que tal vez usted desee aplicar en otros lenguajes las habilidades de programación que aprendió aquí. Los conceptos son similares, pero la terminología es distinta: en muchos lenguajes se utiliza el término *argumento* en vez de *parámetro*. Otra de las terminologías implica el término *invocar* en vez de *llamar*.

PRÁCTICAS DE AUTOEVALUACIÓN

5.3 Explique cuál es el error en estas llamadas:

```
dibujarLogo(papel, 50, "10");
dibujarLogo(50, 10, papel);
dibujarLogo(papel, 10);
```

5.4 He aquí la llamada a un método:

```
soloHazlo("Naranjas");
```

y he aquí el código del método:

```
private void soloHazlo(String fruta) {
    JOptionPane.showMessageDialog(null, fruta);
}
```

¿Qué ocurre cuando se hace una llamada a este método?

● Un método para dibujar triángulos

Para describir más características de los métodos debemos crear uno más útil, al cual llamaremos **dibujarTriángulo**. Como **nosotros** vamos a escribir el método (en vez de utilizar uno pre-definido), podemos elegir qué tipo de triángulo se va a dibujar y los parámetros que deseamos que proporcione el que haga la llamada.

En este caso vamos a dibujar un triángulo recto orientado hacia la derecha, como se muestra en la figura 5.5(a).

Para elegir los parámetros tenemos varias posibilidades: por ejemplo, podríamos requerir que quien haga la llamada proporcione las coordenadas de las tres esquinas. Sin embargo, vamos a optar por los siguientes parámetros:

- El área de dibujo, como en el método anterior.
- Las coordenadas del punto superior del triángulo.
- La anchura del triángulo.
- La altura del triángulo.

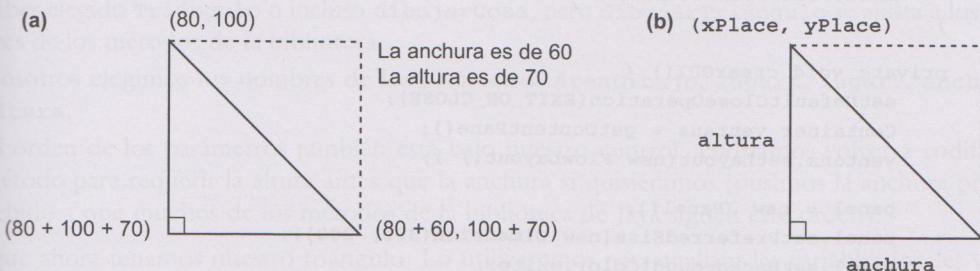


Figura 5.5 (a) Cálculos de las coordenadas de un triángulo. (b) Parámetros formales para **dibujarTriángulo**.

Otra manera de considerar a estas coordenadas es que especifican la posición de un rectángulo circundante para nuestro triángulo recto.

En la figura 5.5(b) se muestra un triángulo identificado con los parámetros.

Podemos dibujar las líneas en cualquier orden. Examinemos el proceso de dibujo con números primero. Como un ejemplo, dibujaremos un triángulo con la esquina superior en (80, 100), con una anchura de 60 y una altura de 70. En la figura 5.5(a) se muestran los cálculos. El proceso es el siguiente:

1. Dibujar de (80, 100) hasta (80, 100 + 70). Recuerde que la coordenada *y* se incrementa hacia abajo.
2. Dibujar de (80, 100 + 70) hasta (80 + 60, 100 + 70).
3. Dibujar de la esquina superior (80, 100) en sentido diagonal hasta (80 + 60, 100 + 70).

Asegúrese de seguir el proceso anterior; tal vez sea conveniente que lo dibuje en papel.

Observe que en nuestra explicación no simplificamos los cálculos: dejamos 100 + 70 en su forma original, en vez de usar 170. Al llegar a la codificación, la posición del triángulo y su tamaño se pasarán como parámetros separados.

He aquí un programa completo llamado **MétodoTriángulo**. Este programa contiene un método llamado **dibujarTriángulo**. También contiene el método **dibujarLogo** para ilustrar que un programa puede contener muchos métodos.

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

public class MétodoTriángulo extends JFrame
    implements ActionListener {

    private JButton botón;
    private JPanel panel;

    public static void main(String[] args) {
        MétodoTriángulo marco = new MétodoTriángulo();
        marco.setSize(350, 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);
    }
}
```

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

public void actionPerformed(ActionEvent event) {
    Graphics papel = panel.getGraphics();
    dibujarLogo(papel, 10, 20);
    dibujarLogo(papel, 100, 100);
    dibujarTriángulo(papel, 100, 10, 40, 40);
    dibujarTriángulo(papel, 10, 100, 20, 60);
}

private void dibujarLogo(Graphics áreaDibujo,
                          int xPos, int yPos) {
    áreaDibujo.drawRect(xPos, yPos, 60, 60);
    áreaDibujo.drawRect(xPos, yPos, 40, 40);
    áreaDibujo.drawRect(xPos, yPos, 20, 20);
}

private void dibujarTriángulo(Graphics áreaDibujo,
                               int lugarX,
                               int lugarY,
                               int anchura,
                               int altura) {
    áreaDibujo.drawLine(lugarX, lugarY,
                        lugarX, lugarY + altura);
    áreaDibujo.drawLine(lugarX, lugarY + altura,
                        lugarX + anchura, lugarY + altura);
    áreaDibujo.drawLine(lugarX, lugarY,
                        lugarX + anchura, lugarY + altura);
}
}
```

El programa tiene un botón y un panel para dibujar. Haga clic en el botón y se dibujarán dos logos y dos triángulos. En la figura 5.6 se muestra el resultado.

Veamos algunos detalles sobre la codificación del método `dibujarTriángulo`:

- Decidimos llamarlo `dibujarTriángulo`, pero podemos elegir cualquier otro nombre. Pudimos haber elegido `Triángulo` o incluso `dibujarCosa`, pero `dibujarTriángulo` se ajusta a los nombres de los métodos de la biblioteca.
- Nosotros elegimos los nombres de los parámetros `áreaDibujo`, `lugarX`, `lugarY`, `anchura` y `altura`.
- El orden de los parámetros también está bajo nuestro control. Podríamos volver a codificar el método para requerir la altura antes que la anchura si quisieramos (pusimos la anchura primero debido a que muchos de los métodos de la biblioteca de Java siguen este orden).

Así que ahora tenemos nuestro triángulo. Lo utilizaremos para analizar las variables locales y también para mostrar cómo puede ser la base para métodos más poderosos.

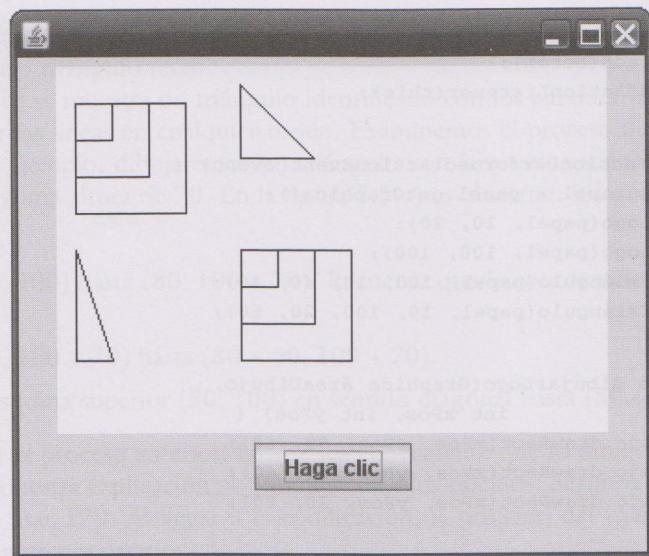


Figura 5.6 Pantalla del programa `MétodoTriángulo`.

● Variables locales

Dé un vistazo a la siguiente versión modificada de `dibujarTriángulo`, a la cual hemos llamado `dibujarTriángulo2`:

```
private void dibujarTriángulo2(Graphics áreaDibujo,
int lugarX,
int lugarY,
int anchura,
int altura) {
    int esquinaDerechaX, esquinaDerechaY;
    esquinaDerechaX = lugarX + anchura;
    esquinaDerechaY = lugarY + altura;
    áreaDibujo.drawLine(lugarX, lugarY,
        lugarX, esquinaDerechaY);
    áreaDibujo.drawLine(lugarX, esquinaDerechaY,
        esquinaDerechaX, esquinaDerechaY);
    áreaDibujo.drawLine(lugarX, lugarY,
        esquinaDerechaX, esquinaDerechaY);
}
```

La llamada a este método se hace de la misma forma que con `dibujarTriángulo`, pero en su interior utiliza dos variables llamadas `esquinaDerechaX` y `esquinaDerechaY`, las cuales se introdujeron para simplificar los cálculos. Vea la forma en que se utilizan para hacer referencia al punto del triángulo que está más a la derecha. Estas variables sólo existen dentro de `dibujarTriángulo2`.

Son locales para el método (en la terminología se dice que tienen *alcance local*). Si existen variables del mismo nombre dentro de otros métodos, entonces no hay conflicto ya que cada método utiliza su propia copia. Otra forma de ver esto es que cuando los programadores crean métodos, pueden inventar las variables locales sin tener que revisar los nombres de las variables de los demás métodos.

La función que desempeñan las variables locales es ayudar al método a realizar su trabajo, sin importar lo que haga. Las variables tienen un alcance limitado, ya que están restringidas a su propio método. Su existencia es temporal: se crean al momento de llamar al método y se destruyen cuando el método termina de ejecutarse.

● Conflictos de nombres

En Java, el creador de un método tiene la libertad de elegir nombres apropiados para las variables locales y los parámetros. Pero, ¿qué ocurre si se eligen nombres que estén en conflicto con otras variables? Podríamos tener lo siguiente:

```
private void metodoUno(int x, int y) {  
    int z = 0;  
    // código...  
}  
  
private void metodoDos(int z, int x) {  
    int w = 1;  
    // código...  
}
```

Suponga que dos personas escribieron estos métodos. `metodoUno` tiene los parámetros `x` e `y`; además declara una variable tipo entero llamada `z`. Estos tres elementos son locales para `metodoUno`. En `metodoDos` el programador ejerce su derecho a nombrar los elementos locales y opta por usar `z`, `x` y `w`. El conflicto de nombres por la `x` (y la `z`) no representa un problema, ya que Java considera que la `x` de `metodoUno` y la `x` de `metodoDos` son distintas.

PRÁCTICA DE AUTOEVALUACIÓN

5.5 Considere la siguiente llamada a un método:

```
int a = 3;  
int b = 8;  
hacerAlgo(a, b);  
JOptionPane.showMessageDialog(null, Integer.toString(a));
```

y he aquí el método en sí:

```
private void hacerAlgo(int x, int y) {  
    int a = 0;  
    a = x + y;  
}
```

¿Qué se muestra en el cuadro de mensaje?

Vamos a ver un resumen de las herramientas para trabajar con métodos que hemos visto hasta ahora (más adelante incluiremos la instrucción `return`).

- La forma general de la declaración de un método que no produce un resultado y recibe los parámetros por valor es:

```
private void unNombre(lista de parámetros formales) {  
    cuerpo  
}
```

El programador elige el nombre del método.

- La lista de parámetros formales es una lista de tipos y nombres. Si un método no necesita parámetros, utilizamos paréntesis vacíos para la lista de parámetros al momento de declararlo, y utilizamos paréntesis vacíos para la lista de parámetros actuales al momento de llamarlo.

```
private void miMétodo() {  
    cuerpo  
}
```

y la llamada al método es:

```
miMétodo();
```

- Una clase puede contener cualquier cantidad de métodos, en cualquier orden. En este capítulo nuestros programas sólo contienen una clase. La esencia de su distribución es:

```
public class UnaClase... {  
  
    public static void main(lista de parámetros...) {  
        cuerpo  
    }  
  
    private void unNombre(lista de parámetros...) {  
        cuerpo  
    }  
  
    private void otroNombre(lista de parámetros...) {  
        cuerpo  
    }  
}
```

En el capítulo 9 veremos cómo usar las palabras clave `public` y `class`. Por ahora sólo basta con tener en cuenta que una clase puede agrupar a una serie de métodos.

● Métodos para manejar eventos y main

Una clase contiene un conjunto de métodos. Algunos de ellos los escribimos nosotros (como `dibujarLogo`) y los llamamos de manera explícita. Sin embargo, hay otros métodos que creamos pero no llamamos, como `main` y `actionPerformed`.

Si nunca llamáramos a un método, no tendría efecto. Sin embargo, los anteriores métodos sí son llamados, sólo que desde el sistema en tiempo de ejecución de Java en vez de llamarlos en forma explícita desde nuestro programa.

- Cuando un programa empieza a ejecutarse, se hace una llamada de manera automática al método `main` antes de que ocurra cualquier otra cosa. Su tarea principal es llamar a algunos métodos que crean la GUI mediante la adición (por ejemplo) de botones a la ventana.
- El sistema en tiempo de ejecución de Java llamará al método `actionPerformed` cada vez que se haga clic en un botón. Este proceso no es totalmente automático: el programa tiene que declarar que va a “escuchar” o estar atento a los clics de los botones. En el capítulo 6 hablaremos sobre esto.

● La instrucción `return` y los resultados

En nuestros ejemplos anteriores de parámetros formales y actuales pasamos valores hacia los métodos para que éstos los utilicen. Sin embargo, con frecuencia es necesario codificar métodos que realicen un cálculo y envíen un resultado al resto del programa, de manera que este resultado se pueda utilizar en cálculos posteriores. En este caso podemos utilizar la instrucción `return`. Veamos a continuación un método que calcula el área de un rectángulo, dados sus dos lados como parámetros de entrada. He aquí un programa completo llamado `MétodoÁrea`, el cual muestra al método y una llamada:

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

public class MétodoÁrea extends JFrame
    implements ActionListener {

    private JButton botón;
    private JPanel panel;

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

    private void createGUI() {
        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);

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

```

public void actionPerformed(ActionEvent event) {
    int a;
    a = áreaRectángulo(10, 20);
    JOptionPane.showMessageDialog(null, "El área es: " + a);
}

private int áreaRectángulo(int longitud, int anchura) {
    int área;
    área = longitud * anchura;
    return área;
}

```



Hay varias características nuevas en este ejemplo que van de la mano.

Considere el encabezado del método:

```
private int áreaRectángulo(int longitud, int anchura) {
```

En vez de `void` especificamos el tipo de elemento que el método debe regresar al que hizo la llamada. En este caso, y como estamos multiplicando dos valores `int`, la respuesta también es de tipo `int`.

La elección del tipo de este elemento depende del problema. Por ejemplo, podría ser entero o cadena, pero también podría ser un objeto más complicado, tal como un botón. El programador que escribe el método elige el tipo de valor que se va a devolver.

Para devolver un valor del método utilizamos la instrucción `return` de la siguiente manera:

```
return expresión;
```

La expresión podría ser un número, una variable o un cálculo (o incluso la llamada a un método), pero debe ser del tipo correcto, según lo especificado en la declaración del método (su encabezado). Además, la instrucción `return` hace que el método actual deje de ejecutarse y regresa de inmediato al lugar en el que se encontraba dentro del método que hizo la llamada. Ahora veamos cómo se puede llamar a un método que devuelve un resultado.

La siguiente es una manera de cómo **no** llamar a dicho método. No se deben utilizar como instrucciones completas; por ejemplo:

```
áreaRectángulo(10, 20); // no
```

El programador debe asegurarse de “consumir” o “utilizar” el valor devuelto. He aquí una metodología para comprender cómo devolver valores: imagine que se elimina la llamada al método (el nombre y la lista de parámetros) y se sustituye por el resultado devuelto. Si el código resultante tiene sentido, entonces Java le permitirá realizar dicha llamada. Vea el siguiente ejemplo:

```
respuesta = áreaRectángulo(30, 40);
```

El resultado es 1200, y si sustituimos la llamada por este resultado, obtenemos lo siguiente:

```
respuesta = 1200;
```

Esto es código válido en Java, pero si utilizamos:

```
áreaRectángulo(30, 40);
```

al sustituir el resultado devuelto se produciría una instrucción de Java que sólo tendría un número:

```
1200;
```

lo cual no tiene significado (aunque en el sentido estricto, el compilador de Java permitirá que se ejecute la llamada anterior a `áreaRectángulo`. Sin embargo, no tiene caso ignorar el resultado devuelto de un método, cuyo propósito principal es precisamente devolver dicho resultado).

He aquí algunas otras formas válidas en las que podríamos consumir el resultado:

```
int n;
n = áreaRectángulo(10, 20);
JOptionPane.showMessageDialog(null, "el área es de " +
    áreaRectángulo(3, 4));
n = áreaRectángulo(10, 20) * áreaRectángulo(7, 8);
```

PRÁCTICA DE AUTOEVALUACIÓN

- 5.6 Utilice lápiz y papel para trabajar con las instrucciones anteriores; sustituya los resultados por las llamadas.

Para completar nuestra explicación sobre `return` cabe mencionar que se puede utilizar con métodos `void`. En este caso, debemos utilizar `return` sin especificar un resultado, como en el siguiente ejemplo:

```
private void demo(int n) {
    // hacer algo
    return;
    // hacer otra cosa
}
```

Esto se puede utilizar cuando queremos que el método termine en una instrucción que no sea la última.

Ahora veamos una manera alternativa de codificar nuestro ejemplo del área:

```
private int áreaRectángulo2(int longitud, int anchura) {
    return longitud * anchura;
}
```

Debido a que podemos utilizar `return` con expresiones, hemos omitido la variable `área` en `áreaRectángulo2`.

Dichas reducciones en el tamaño del programa no siempre son beneficiosas, ya que al reducir los nombres representativos se puede disminuir la claridad, lo que por ende puede provocar que se requiera más tiempo de depuración y prueba.

PRÁCTICA DE AUTOEVALUACIÓN

- 5.7 El siguiente método se llama `doble` y devuelve el doble del valor de su parámetro `int`:

```
private int doble(int n) {
    return 2 * n;
}
```

Dadas las siguientes llamadas al método:

```
int n = 3;
int r;
r = doble(n);
r = doble(n + 1);
r = doble(n) + 1;
r = doble(3 + 2 * n);
r = doble(doble(n));
r = doble(doble(n + 1));
r = doble(doble(n) + 1);
r = doble(doble(doble(n))));
```

Indique el valor devuelto para cada llamada.

● Cómo construir métodos a partir de otros métodos: dibujarCasa

Como ejemplo de métodos que utilizan otros métodos, vamos a crear un método que dibuja una casa sencilla con una sección transversal, la cual se muestra en la figura 5.7. La altura del techo es la misma que la altura de las paredes, y la anchura de las paredes es la misma que la anchura del techo. Vamos a utilizar los siguientes parámetros `int`:

- La posición horizontal del punto superior derecho del techo.
- La posición vertical del punto superior derecho del techo.

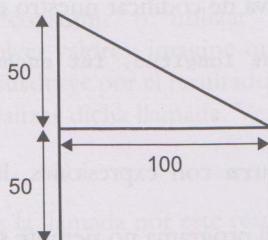


Figura 5.7 Una casa cuya anchura es de 100 y la altura del techo es de 50.

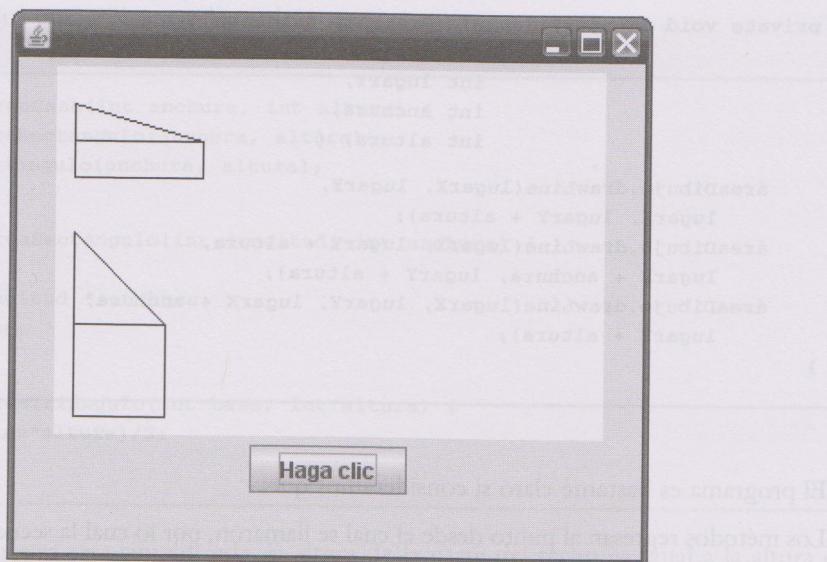


Figura 5.8 Pantalla del programa `DemoCasa`.

- La altura del techo (excluyendo la pared).
- La anchura de la casa. El triángulo para el techo y el rectángulo para las paredes tienen la misma anchura.

Utilizaremos el método `drawRect` de la biblioteca de Java y utilizaremos nuestro propio método `dibujarTriángulo`.

He aquí el nuevo código. La configuración de la interfaz de usuario es idéntica a la de nuestro programa anterior. La imagen resultante se muestra en la figura 5.8.

```
public void actionPerformed(ActionEvent event) {
    Graphics papel = panel.getGraphics();
    dibujarCasa(papel, 10, 20, 70, 20);
    dibujarCasa(papel, 10, 90, 50, 50);
}

private void dibujarCasa(Graphics áreaDibujo,
    int techoSupX,
    int techoSupY,
    int anchura,
    int altura) {

    dibujarTriángulo(áreaDibujo, techoSupX, techoSupY, anchura, altura);
    áreaDibujo.drawRect(techoSupX,
        techoSupY + altura, anchura, altura);
}
```

```

private void dibujarTriángulo(Graphics áreaDibujo,
    int lugarX,
    int lugarY,
    int anchura,
    int altura) {
    áreaDibujo.drawLine(lugarX, lugarY,
        lugarX, lugarY + altura);
    áreaDibujo.drawLine(lugarX, lugarY + altura,
        lugarX + anchura, lugarY + altura);
    áreaDibujo.drawLine(lugarX, lugarY, lugarX + anchura,
        lugarY + altura);
}

```

El programa es bastante claro si consideramos que:

- Los métodos regresan al punto desde el cual se llamaron, por lo cual la secuencia de llamadas es:
 - `actionPerformed` llama a `dibujarCasa`.
 - `dibujarCasa` llama a `drawRect`.
 - `dibujarCasa` llama a `dibujarTriángulo`.
 - `dibujarTriángulo` llama a `drawLine` (tres veces).
- Los parámetros pueden ser expresiones, por lo que se evalúa `lugarY + altura` y después se pasa el resultado a `drawLine`.
- Las variables `anchura` y `altura` de `dibujarCasa` y las variables `anchura` y `altura` de `dibujarTriángulo` son totalmente distintas. Sus valores se almacenan en diferentes lugares.

Aquí podemos ver que lo que hubiera podido ser un programa más grande se ha escrito como un programa corto, dividido en métodos y con nombres representativos. Esto ilustra el poder que se obtiene al utilizar métodos.

● Cómo construir métodos a partir de otros métodos: `áreaCasa`

Ahora veamos un método que calcula y devuelve el área de la sección transversal de nuestra casa. Vamos a utilizar nuestro método `áreaRectángulo` existente y escribiremos un método `áreaTriángulo` basado en lo siguiente:

```
área = (base * altura) / 2;
```

He aquí una llamada a `áreaCasa`:

```

int área = áreaCasa(10, 10);
JOptionPane.showMessageDialog(null,
    "El área de la casa es de " + área);

```

He aquí los métodos que utiliza esta llamada:

```
private int áreaCasa(int anchura, int altura) {  
    return áreaRectángulo(anchura, altura) +  
        áreaTriángulo(anchura, altura);  
}  
  
private int áreaRectángulo(int longitud, int anchura) {  
    int área;  
    área = longitud * anchura;  
    return área;  
}  
  
private int áreaTriángulo(int base, int altura) {  
    return (base*altura)/2;  
}
```

Recuerde que en nuestra casa simplificada, la altura de la parte del techo es igual a la altura de la pared, por lo que sólo necesitamos pasar la anchura y la altura. He aquí cómo funcionan los métodos en conjunto:

- `áreaCasa` llama a `áreaRectángulo` y a `áreaTriángulo`.
- `áreaCasa` suma los dos resultados que se devuelven.
- `áreaCasa` devuelve el resultado final.

Cabe mencionar que `áreaTriángulo` obtiene sus entradas por medio de parámetros en vez de solicitarlas al usuario por medio de cuadros de entrada. Esto lo hace flexible. Por ejemplo, en este caso los valores de sus parámetros se crean en `áreaCasa`.

Hasta ahora, en nuestro estudio sobre los métodos hemos visto lo siguiente:

- Pasar parámetros a un método.
- Pasar un valor fuera de un método mediante `return`.

● La palabra clave `this` y los objetos

Probablemente esté leyendo este libro debido a que Java es un lenguaje orientado a objetos, pero tal vez se esté preguntando por qué no hemos mencionado a los objetos en este capítulo. La verdad es que los métodos y los objetos tienen una conexión vital. Al ejecutar programas pequeños en Java, estamos ejecutando una instancia de una clase; es decir, un objeto. Este objeto contiene métodos (como `dibujarLogo`).

Cuando un objeto llama a un método que está declarado en su interior, podemos simplificar la llamada utilizando:

```
dibujarLogo(papel, 50, 10);
```

podemos utilizar la notación de objetos completa, como en el siguiente ejemplo:

```
this.dibujarLogo(papel, 50, 10);
```

`this` es una palabra clave de Java y representa al objeto actual en ejecución. Por lo tanto, ha estado utilizando la programación orientada a objetos sin darse cuenta de ello. He aquí algunos ejemplos:

```
// funciona como se esperaba  
papel.drawLine(10, 10, 100, 100);  
  
// error de compilación  
this.drawLine(10, 10, 100, 100);
```

En el ejemplo anterior se detectará un error, ya que estamos pidiendo a Java que localice el método `drawLine` dentro del objeto actual. De hecho, `drawLine` existe fuera del programa en la clase `Graphics`.

Sobrecarga de métodos

Nuestro método `áreaTriángulo` es útil en cuanto a que puede trabajar con parámetros actuales que tengan cualquier nombre. La desventaja es que deben ser enteros. Pero ¿qué tal si quisieramos trabajar con dos variables `double`? Podríamos codificar *otro* método:

```
private double áreaTriánguloDouble(double base, double altura) {  
    return 0.5 * (base * altura);  
}
```

Sin embargo, sería conveniente utilizar el mismo nombre para ambos métodos; en Java podemos hacerlo. He aquí cómo codificaríamos las declaraciones de estos métodos:

```
private int áreaTriángulo(int base, int altura) {  
    return (base * altura) / 2;  
}  
  
private double áreaTriángulo(double base, double altura) {  
    return (base * altura) * 0.5;  
}  
  
private double áreaTriángulo(double lado1, double lado2,  
                           double angulo) {  
    return 0.5 * lado1 * lado2 * Math.sin(angulo);  
}
```

Junto a las versiones `int` y `double` colocamos una versión adicional, para aquellos que estén familiarizados con la trigonometría. Aquí el área se calcula con base en la longitud de dos lados y el seno del ángulo entre ellos, en radianes (`Math.sin` provee la función seno). Esta versión de `áreaTriángulo` tiene tres parámetros `double`. Aquí llamamos a los tres métodos y mostramos los resultados en cuadros de mensaje:

```
public void actionPerformed(ActionEvent event) {  
    double da = 9.5, db = 21.5;  
    int ia = 10, ib = 20;  
    double angulo = 0.7;  
    JOptionPane.showMessageDialog(null,  
        "El área del triángulo es de "+áreaTriángulo(ia, ib));  
    JOptionPane.showMessageDialog(null,  
        "El área del triángulo es de "+áreaTriángulo(da, db));  
    JOptionPane.showMessageDialog(null,  
        "El área del triángulo es de "+áreaTriángulo(da, db, angulo));  
}
```

¿Cómo decide Java cuál método utilizar? Hay tres métodos llamados `áreaTriángulo`, por lo que Java busca además del nombre el número de parámetros actuales en la llamada y sus tipos. Por ejemplo, si llamamos a `áreaTriángulo` con dos parámetros `double`, busca la declaración apropiada de `áreaTriángulo` con dos (y sólo dos) parámetros `double`. El código que contienen los métodos puede ser diferente; es el número de parámetros y sus tipos lo que determina a cuál método llamar. A la combinación del número de parámetros y sus tipos se le conoce como la *firma* del método.

Si el método devuelve un resultado, el tipo de este valor de retorno no participa a la hora de determinar cuál método se va a llamar; es decir, son los tipos de los parámetros del método los que deben ser distintos.

Lo que hemos hecho aquí se denomina *sobrecargar* un método. Hemos sobrecargado el método `áreaTriángulo` con varias posibilidades.

Por lo tanto, si usted escribe métodos que realicen tareas similares pero tengan distintos números y/o tipos de parámetros, es conveniente utilizar la sobrecarga y elegir el **mismo** nombre en vez de inventar un nombre artificialmente distinto.

PRÁCTICA DE AUTOEVALUACIÓN

- 5.8 Escriba dos métodos, ambos llamados `sumarNúmeros`. Uno debe sumar dos enteros y devolver su suma. El otro debe sumar tres enteros y devolver la suma. Provea ejemplos de cómo llamar a sus dos métodos.

Fundamentos de programación

- Un método es una sección de código que tiene asignado un nombre. Para llamar al método usamos su nombre.
- Podemos codificar métodos `void` o métodos que devuelvan un resultado.
- Podemos pasar parámetros a un método.
- Si podemos identificar una tarea bien definida en nuestro código, podemos separarla y escribirla como un método.

Errores comunes de programación

- El encabezado del método debe incluir los nombres de los tipos. El siguiente código está mal:

```
private void métodoUno(x) {    // incorrecto
```

Debemos utilizar algo como esto:

```
private void métodoUno(int x) {
```

- La llamada a un método no debe incluir los nombres de los tipos. Por ejemplo, en vez de:

```
métodoUno(int y);    //
```

debemos usar:

```
métodoUno(y);
```

- Al llamar a un método debemos suministrar el número correcto de parámetros y los tipos correctos de éstos.
- Siempre debemos consumir un valor devuelto de alguna forma. El siguiente estilo de llamada no consume un valor devuelto:

```
unMétodo(e, f);    //
```

Secretos de codificación

- El patrón general para los métodos toma dos formas. En primer lugar, para un método que no devuelve un resultado, la forma de declararlo es:

```
private void nombreMétodo(lista de parámetros formales) {  
    ... cuerpo  
}
```

y debemos invocar el método mediante una instrucción, como en el siguiente ejemplo:

```
nombreMétodo(lista de parámetros actuales);
```

- Para un método que devuelve un resultado, la forma de declararlo es:

```
private tipo nombreMétodo(lista de parámetros actuales) {  
    ... cuerpo  
}
```

Se puede especificar cualquier tipo o clase para el valor devuelto. Podemos invocar el método como parte de una expresión, como en el siguiente ejemplo:

```
n = nombreMétodo(a, b);
```

- El cuerpo de un método que devuelve un resultado debe incluir una instrucción `return` con el tipo de valor correcto.
- Cuando un método no tiene parámetros debemos usar paréntesis vacíos () tanto en la declaración como en la llamada.
- El escritor del método crea la lista de parámetros formales. Cada parámetro necesita un nombre y un tipo.
- El que hace la llamada al método escribe la lista de parámetros actuales. Esta lista consta de una serie de elementos en el orden correcto y con los tipos correctos. A diferencia de los parámetros dentro del encabezado de un método, aquí no se especifican los nombres de los tipos.

Nuevos elementos del lenguaje

- La declaración de métodos privados.
- La llamada (o invocación) a un método, que consta del nombre del método y sus parámetros.
- El uso de `return` para salir de un método que no sea `void` y devolver al mismo tiempo un valor.
- El uso de `return` para salir de un método `void`.
- El uso de la sobrecarga.
- El uso de `this` para representar al objeto actual.

Resumen

- Los métodos contienen subtareas de un programa.
- Podemos pasar parámetros a los métodos.
- Utilizar un método es algo conocido como *llamar* (o *invocar*) al método.
- Los métodos que no son `void` devuelven un resultado.

Ejercicios

Para los ejercicios que dibujan figuras, base su código en el programa **MétodoLogo**. Para los programas que sólo necesitan cuadros de mensaje y entrada, base su código en el programa **ÁreaRectángulo**.

El primer grupo de problemas sólo requiere de métodos **void**.

- 5.1 Escriba un método llamado **mostrarNombre** con un parámetro de cadena. Debe mostrar el nombre suministrado en un cuadro de mensaje. Para probar el programa, introduzca un nombre mediante un cuadro de entrada y después llame a **mostrarNombre**.
- 5.2 Escriba un método llamado **mostrarNombres** con dos parámetros de cadena que representen su primer nombre y su apellido paterno. El método debe mostrar su primer nombre en un cuadro de mensaje y después mostrar su apellido paterno en otro cuadro de mensaje.
- 5.3 Escriba un método llamado **mostrarIngresos** con dos parámetros enteros que representen el salario de un empleado y el número de años trabajados. El método debe mostrar el total de ingresos obtenidos por el empleado en un cuadro de mensaje, suponiendo que haya obtenido la misma cantidad de ingresos cada año. El programa deberá obtener los valores mediante cuadros de entrada antes de llamar a **mostrarIngresos**.
- 5.4 Codifique un método que dibuje un círculo, dadas las coordenadas de su centro y su radio. Su encabezado deberá ser el siguiente:

```
private void dibujarCírculo(Graphics áreaDibujo,
                               int xCentro, int yCentro, int radio)
```

- 5.5 Codifique un método llamado **dibujarCalle** que dibuje una calle llena de casas, para lo cual debe utilizar el método **dibujarCasa** que utilizamos en este capítulo. Para los fines de este ejercicio una calle contiene cuatro casas y debe haber un espacio de 20 píxeles entre cada casa. Los parámetros deben proporcionar la ubicación y el tamaño de la casa de más a la izquierda, y deben ser idénticos a los de **dibujarCasa**.
- 5.6 Codifique un método que se llame **dibujarCalleEnPerspectiva** y tenga los mismos parámetros que el método del ejercicio 5.5. Sin embargo, cada casa debe ser un 20% más pequeña que la casa a su izquierda.

Los siguientes programas requieren métodos que devuelvan un resultado.

- 5.7 Escriba un método que devuelva el equivalente en pulgadas de su parámetro en centímetros. La siguiente es una llamada de ejemplo:

```
double pulgadas = equivalentePulgadas(10.5);
```

Multiplique los centímetros por **0.394** para calcular las pulgadas.

- 5.8 Escriba un método que devuelva el volumen de un cubo, dada la longitud de uno de sus lados. La siguiente es una llamada de ejemplo:

```
double vol = volumenCubo(1.2);
```

- 5.9 Escriba un método que devuelva el área de un círculo, dado su radio como parámetro. La siguiente es una llamada de ejemplo:

```
double a = áreaCírculo(1.25);
```

El área de un círculo se obtiene en base a la fórmula **Math.PI * r * r**. Aunque podríamos utilizar un número como **3.14**, **Math.PI** nos proporciona un valor más preciso.

- 5.10 Escriba un método llamado **segsEn** que acepte tres enteros, los cuales representarán el tiempo en horas, minutos y segundos. Debe devolver el tiempo total en segundos. La siguiente es una llamada de ejemplo:

```
int totalSegs = segsEn(1, 1, 2); // devuelve 3662
```

- 5.11 Escriba un método que devuelva el área de un cilindro sólido. Decida qué parámetros utilizar. Debe llamar al método **áreaCírculo** del ejercicio 5.9 para que le ayude a calcular el área de las partes superior e inferior (la circunferencia de un círculo se obtiene mediante la fórmula $2 * \text{Math.PI} * r$).

- 5.12 Escriba un método llamado **incremento** que sume 1 a su parámetro entero. La siguiente es una llamada de ejemplo:

```
int n = 3;
int a = incremento(n); // devuelve 4
```

- 5.13 Escriba un método llamado **resultadoCincoAños** con dos parámetros:

- Una cantidad **double** que se invierte al principio.
- Una tasa de interés **double** (por ejemplo, **1.5** especifica el 1.5% de interés anual).

El método debe devolver la cantidad después de cinco años de interés compuesto.

Sugerencia: después de un año, la nueva cantidad es:

```
cantidad = cantidad * (1 + interes / 100);
```

- 5.14 Escriba un método llamado **diferenciaDeTiempoEnSegs** con seis parámetros y que devuelva un resultado entero. Debe recibir dos tiempos en horas, minutos y segundos; además debe devolver la diferencia entre esos dos tiempos en segundos. Para resolver este problema llame al método **segsEn** (ejercicio 5.10) desde el interior de su método **diferenciaDeTiempoEnSegs**.

Los siguientes problemas son acerca de la sobrecarga de métodos:

- 5.15 Utilice cualquier programa que contenga el método **segsEn**. Agregue un método que también se llame **segsEn** y que tenga sólo dos argumentos, uno para los minutos y otro para los segundos.

- 5.16 Localice el método **dibujarCírculo** que escribió en el ejercicio 5.4. Agregue otro método **dibujarCírculo** a su programa **dibujarCírculo** con los siguientes parámetros:

- Un área de dibujo.
- La posición **x** e **y** del centro.

Se debe dibujar un círculo con un radio de **50** en el punto especificado.

Respuestas a las prácticas de autoevaluación

- 5.1 En (10, 20), (30, 10) y (27, 26).
- 5.2 Ahora el método es más inflexible. Los cuadros de entrada aparecerían cada vez que llamáramos al método. En la versión original del método, el que hace la llamada puede obtener valores para la posición en una variedad de formas antes de pasárselos como parámetros.
- 5.3 En la primera llamada no se deben utilizar las comillas, ya que indican una cadena y no un entero.

En la segunda llamada el orden debe ser:

papel, 50, 10

En la tercera llamada falta un parámetro.

- 5.4 Aparecerá un cuadro de mensaje mostrando **Naranjas**.
- 5.5 El cuadro de mensaje muestra el valor original de **a**, que es 3. La **a** que se convierte en 0 dentro del método es una variable local.
- 5.6 He aquí las etapas para sustituir una llamada por su resultado. Para:

n = áreaRectángulo(10, 20);

tenemos:

n = 200;

Para la línea:

```
JOptionPane.showMessageDialog(null, "el área es de " +  
    áreaRectángulo(3, 4));
```

tenemos las siguientes etapas:

```
JOptionPane.showMessageDialog(null, "el área es de " + 12);  
JOptionPane.showMessageDialog(null, "el área es de 12");
```

Para la línea:

n = áreaRectángulo(10, 20) * áreaRectángulo(7, 8);

tenemos las etapas:

```
n = 200 * 56;  
n = 11200;
```

- 5.7 Los valores que recibe **r** son:

6
8
7
18
12
16
14
24

```
5.8    private int sumarNúmeros(int a, int b) {  
        return a + b;  
    }  
  
    private int sumarNúmeros(int a, int b, int c) {  
        return a + b + c;  
    }
```

Puede llamar a los métodos anteriores así:

```
public void actionPerformed(ActionEvent event) {  
    int suma2, suma3;  
    int x = 22, y = 87, z = 42;  
    suma2 = sumarNúmeros(x, y);  
    suma3 = sumarNúmeros(x, y, z);  
}
```