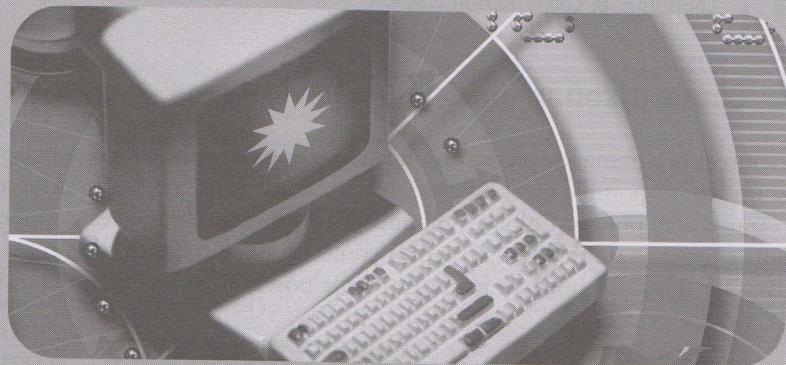


CAPÍTULO 9



Cómo escribir clases

En este capítulo conoceremos cómo:

- Escribir una clase.
- Escribir métodos **public**.
- Utilizar variables **private** dentro de una clase.
- Escribir métodos constructores.

● Introducción

En capítulos anteriores vimos cómo utilizar las clases de biblioteca. En este capítulo veremos cómo escribir nuestras propias clases. Una clase describe cualquier cantidad de objetos similares que se pueden fabricar a partir de ella, mediante la palabra clave **new**.

En este capítulo veremos que una clase consiste por lo general en:

- Datos **private** (variables) que contienen información sobre el objeto.
- Métodos **public** que el usuario del objeto puede llamar para llevar a cabo funciones útiles.
- Opcionalmente uno o más métodos constructores, que se utilizan al momento de crear un objeto. Estos objetos se usan para llevar a cabo cualquier inicialización, como asignar valores iniciales a las variables dentro del objeto.
- Métodos **private** que se utilizan sólo dentro del objeto y son inaccesibles desde su exterior.

● Cómo diseñar una clase

Cuando el programador piensa en un nuevo programa, tal vez surja la necesidad de tener un objeto que no esté disponible en la biblioteca de clases de Java. Primero vamos a utilizar un programa para mostrar y manipular un globo simplificado, y representaremos este globo como un objeto. El programa simplemente muestra un globo como un círculo dentro de un panel, como se muestra en la figura 9.1. Hay dos botones para cambiar la posición y el tamaño del globo.

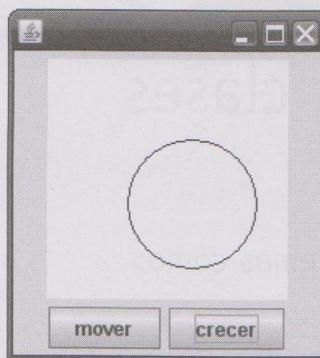


Figura 9.1 El programa del globo.

Construiremos este programa a partir de dos objetos y, por ende, de dos clases:

- La clase **Globo** representa el globo. Esta clase proveerá los métodos llamados `moverDerecha` y `cambiarTamaño`, cuyas funciones son obvias.
- La clase **UsarGlobo** provee la GUI para el programa. Esta clase utilizará la clase **Globo** según sea necesario.

Ambas clases se muestran en el diagrama de clases de UML de la figura 9.2. Cada una de las clases se representa mediante un rectángulo. Una relación entre las clases se muestra como una línea que une ambas clases. En este caso la relación se muestra como una anotación arriba de la línea: la clase **UsarGlobo** usa a la clase **Globo**.

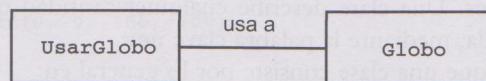


Figura 9.2 Diagrama de clases que muestra las dos clases en el programa del globo.

Primero completaremos la clase **UsarGlobo** y después escribiremos la clase **Globo**. El código completo para **UsarGlobo** es:

```

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

```
public class UsarGlobo extends JFrame
    implements ActionListener {

    private JButton botónCrecer, botónMover;
    private JPanel panel;
    private Globo globo;

    public static void main(String[] args) {
        UsarGlobo demo = new UsarGlobo();
        demo.setSize(200, 220);
        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(150, 150));
        panel.setBackground(Color.white);
        ventana.add(panel);

        botónMover = new JButton("mover");
        ventana.add(botónMover);
        botónMover.addActionListener(this);

        botónCrecer = new JButton("crecer");
        ventana.add(botónCrecer);
        botónCrecer.addActionListener(this);

        globo = new Globo();
    }

    public void actionPerformed(ActionEvent event) {
        Graphics papel = panel.getGraphics();
        if (event.getSource() == botónMover) {
            globo.moverDerecha(20);
        } else {
            globo.cambiarTamaño(20);
        }
        papel.setColor(Color.white);
        papel.fillRect(0, 0, 150, 150);
        globo.mostrar(papel);
    }
}
```

En la parte superior de la clase **UsarGlobo** declaramos las variables de instancia en la forma usual, incluyendo una variable llamada **globo**:

```
private Globo globo;
```

Dentro de la clase **UsarGlobo** realizamos la inicialización necesaria, incluyendo la creación de una nueva instancia de la clase **Globo**. Éste es el punto crucial en el que creamos un objeto a partir de nuestra propia clase.

```
globo = new Globo();
```

Ahora viene el código para responder a los eventos de clic de botón. Si se hace clic en el botón **mover**, entonces se hace una llamada al método **moverDerecha**. Si se hace clic en el otro botón se hace una llamada a **cambiarTamaño**.

```
public void actionPerformed(ActionEvent event) {
    Graphics papel = panel.getGraphics();
    if (event.getSource() == botónMover) {
        globo.moverDerecha(20);
    }
    else {
        globo.cambiarTamaño(20);
    }
    papel.setColor(Color.white);
    papel.fillRect(0, 0, 150, 150);
    globo.mostrar(papel);
}
```

Éste es todo el código de la clase **UsarGlobo**. Escribir este código nos ayuda a entender la forma en que se utiliza un objeto globo, al tiempo que nos permite ver qué métodos necesita proveer la clase **Globo**, así como la naturaleza de sus parámetros. Esto nos conduce a escribir el código para la clase **Globo**:

```
public class Globo {
    private int x = 50;
    private int y = 50;
    private int diámetro = 20;

    public void moverDerecha(int pasoX) {
        x = x + pasoX;
    }

    public void cambiarTamaño(int cambio) {
        diámetro = diámetro + cambio;
    }

    public void mostrar(Graphics papel) {
        papel.setColor(Color.black);
        papel.drawOval(x, y, diámetro, diámetro);
    }
}
```

El encabezado de una clase empieza con la palabra clave **class** y proporciona el nombre de la clase, seguido de una llave. La clase completa termina con una llave. Una clase se etiqueta como **public** con la finalidad de que se pueda utilizar ampliamente. La convención de Java (y de la mayoría de los lenguajes OO) es que los nombres de las clases empiecen con mayúscula. El cuerpo de una clase consiste en las declaraciones de las variables y los métodos. Observe cómo se mejora la legibilidad de la clase utilizando sangría y líneas en blanco. En las siguientes secciones del capítulo veremos con detalle cada uno de los ingredientes de la clase anterior.

En resumen, la estructura general de una clase es:

```
public class Globo {  
    // variables de instancia  
    // métodos  
}
```

Ahora que hemos escrito la clase **Globo**, podemos crear cualquier número de instancias de ella. Ya hemos creado un objeto mediante el siguiente código:

```
globo = new Globo();
```

pero además podemos hacer lo siguiente:

```
Globo globo2 = new Globo();
```

● Clases y archivos

Cuando un programa consta sólo de una clase, ya hemos visto que el código fuente de Java se debe colocar en un archivo que tenga el mismo nombre de la clase, pero con la extensión **.java**. Por ejemplo, una clase llamada **Juego** va en un archivo llamado **Juego.java** y el encabezado de la clase es:

```
public class Juego etc.
```

Las instrucciones **import** deben ir antes de este encabezado. El compilador traduce el código de Java en código de bytes y después lo coloca en un archivo llamado **Juego.class**.

Cuando un programa consta de dos o más clases, hay dos metodologías distintas para colocar las clases en archivos:

1. Colocar todas las clases en un solo archivo.
2. Colocar cada clase en su propio archivo.

Ahora bien, los detalles sobre el uso de cada una de estas metodologías dependerán del sistema de desarrollo que usted utilice. Pero he aquí algunos escenarios comunes.

Un solo archivo

Para adoptar esta metodología:

1. Coloque todas las clases en un archivo.
2. Declare como **public** la clase que contiene el método **main**.

3. Declare todas las demás clases de manera que no sean **public**; es decir, sin descripción de acceso.
4. Haga el nombre del archivo igual al nombre de la clase **public**.
5. Coloque las instrucciones **import** al principio del archivo. Éstas aplican para todas las clases en el archivo.

Por ejemplo, el archivo **UsarGlobo.java** contiene ambas clases:

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

public class UsarGlobo extends JFrame
    implements ActionListener {
    // cuerpo de la clase UsarGlobo
}

class Globo {
    // cuerpo de la clase Globo
}
```

Esta metodología tiene la ventaja de que todas las clases están en un solo lugar. Además, las instrucciones **import** sólo se necesitan una vez.

Archivos separados

Para adoptar esta metodología:

1. Coloque cada clase en un archivo por sí sola. La mayoría de los entornos de desarrollo interactivos tienen una nueva herramienta de clases.
2. Declare todas las clases como **public**.
3. Haga cada nombre de archivo igual al nombre de la clase que contiene.
4. Coloque las instrucciones **import** apropiadas al inicio de cada clase.
5. Coloque todos los archivos en la misma carpeta.

Por ejemplo, el archivo **UsarGlobo.java** contiene:

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

public class UsarGlobo extends JFrame
    implements ActionListener {
    // cuerpo de la clase UsarGlobo
}
```

Tu segundo archivo llamado **Globo.java** contiene:

```
import java.awt.*;
public class Globo {
    // cuerpo de la clase Globo
}
```

Esta metodología tiene la ventaja de que las clases están en distintos archivos y, por lo tanto, pueden utilizarse con más facilidad. Utilizamos esta metodología en todo el libro.

Es imprescindible compilar los archivos en orden de dependencia. La clase **UsarGlobo** utiliza un objeto **Globo** y por lo tanto debemos compilar primero la clase **Globo**. Un IDE hará esto de manera automática.

● Variables private

Un globo tiene ciertos datos asociados: su tamaño (diámetro) y su posición (como coordenadas *x, y*). Un objeto globo debe recordar estos valores. Estos datos se guardan en variables que se describen de la siguiente forma:

```
private int x = 50;
private int y = 50;
private int diámetro = 20;
```

Las variables **diámetro**, **x** y **y** se declaran en la parte superior de la clase. Cualquier instrucción dentro de la clase puede acceder a ellas. Se denominan *variables a nivel de clase* o *variables de instancia*.

Hemos agregado la palabra clave **private** a la palabra que normalmente se utiliza para introducir las variables (como **int**). Las variables a nivel de clase casi siempre se declaran como **private**. Esto significa que son accesibles desde cualquier parte dentro de la clase, pero son inaccesibles desde el exterior.

Aunque podríamos describir estas variables como **public**, por lo general se considera una mala práctica. Es mejor dejarlas como **private** y usar métodos para acceder a sus valores desde el exterior de la clase. Pronto veremos cómo hacerlo.

PRÁCTICA DE AUTOEVALUACIÓN

- 9.1 Extienda el objeto globo de manera que tenga una variable que describa el color del globo.

● Métodos public

Ciertas características de un objeto necesitan estar públicamente disponibles para otras piezas del programa. Esto incluye aquellos métodos que, después de todo, han sido diseñados para que otros métodos los utilicen. Como hemos visto, un globo tiene asociadas ciertas acciones; por ejemplo, para cambiar su tamaño. Estas acciones se escriben como métodos. Para cambiar el tamaño utilizamos el siguiente código:

```
public void cambiarTamaño(int cambio) {
    diámetro = diámetro + cambio;
}
```

Para indicar que el método está públicamente disponible para los usuarios de la clase, debemos anteponer al encabezado del método la palabra clave **public** de Java.

Ahora vamos a escribir el método para mover un globo:

```
public void moverDerecha(int pasoX) {
    x = x + pasoX;
}
```

Para completar la clase proveeremos un método para que un globo se muestre a sí mismo cada vez que se lo soliciten:

```
public void mostrar(Graphics papel) {
    papel.setColor(Color.black);
    papel.drawOval(x, y, diámetro, diámetro);
}
```

Ahora hemos marcado con toda claridad la diferencia entre los elementos públicamente disponibles y los que son privados. Éste es un importante ingrediente de la filosofía de la POO. Los datos (variables) y las acciones (métodos) están agrupados de manera conjunta, pero de tal forma que se oculta parte de la información al mundo exterior. Por lo general, son los datos los que se ocultan al resto del mundo. Esto se denomina *encapsulamiento* u *ocultamiento de información*.

Así, una clase generalmente consta de:

- Métodos **public**.
- Variables **private**.

PRÁCTICAS DE AUTOEVALUACIÓN

- 9.2 Escriba un método que mueva un globo hacia arriba cierta cantidad de espacio que se proporcionará como el parámetro. Asigne a este método el nombre **moverArriba**.
- 9.3 Escriba un método que permita cambiar el color de un globo.
- 9.4 Reescriba el método **mostrar** de manera que muestre un globo coloreado.

Una clase (u objeto) tiene la estructura general que se muestra en la figura 9.3. Ésta es la vista del programador que la escribió; consta de variables y métodos. La forma en que los usuarios ven a un objeto se muestra en la figura 9.4. Los usuarios, a los cuales provee un servicio, ven al objeto en forma muy distinta. Únicamente los elementos **public** (por lo general los métodos) son visibles; todo lo demás está oculto dentro de una caja impenetrable.

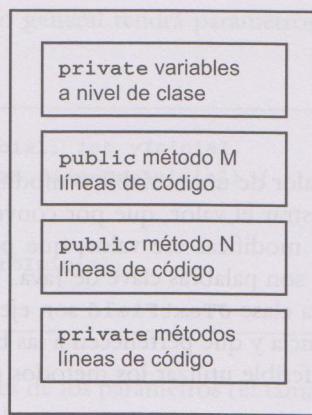


Figura 9.3 Estructura de un objeto o clase en la forma en que lo ve el programador que lo escribió.

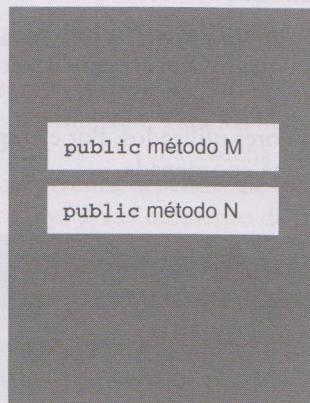


Figura 9.4 Estructura de un objeto o clase en la forma en que lo ven sus usuarios.

● Los métodos get y set

Es muy mala práctica permitir el acceso externo a las variables dentro de un objeto. Por ejemplo, suponga que una clase necesita conocer la coordenada *x* de un globo. Sería muy tentador declarar simplemente el valor *x* como **public**. Así el usuario del objeto sólo tendría que hacer referencia al valor como *globo.x*. Esto es factible pero es un mal diseño. Lo que debemos hacer es controlar el acceso a las variables ofreciendo métodos que tengan acceso a ellas. Con esta finalidad, proveemos un método llamado **getX** como parte de la clase **Globo**. Un usuario podría utilizarlo de la siguiente forma:

```
int posición = globo.getX();
```

He aquí el código para el método **getx**:

```
public int getX() {
    return x;
}
```

En general, los usuarios necesitan leer el valor de una variable o modificarlo, o ambas cosas. Por lo tanto, requerimos un método para suministrar el valor, que por convención se denomina método *get*; además necesitamos un método para modificar ese valor, que por convención se denomina método *set*. Las palabras “*get*” y “*set*” no son palabras clave de Java.

Los métodos **getText** y **setText** de la clase **JTextField** son ejemplos comunes de métodos *get* y *set* que se utilizan con mucha frecuencia y que pertenecen a las bibliotecas de Java.

Hay varias razones por las cuales es preferible utilizar los métodos para controlar el acceso a las variables:

- La clase puede ocultar la representación interna de los datos a los usuarios, sin dejar de mantener la interfaz externa. Por ejemplo, el autor de la clase **Globo** podría optar por guardar las coordenadas del punto central de un globo, pero proveer a los usuarios las coordenadas de la esquina superior izquierda de un cuadrado circundante.
- El autor de la clase puede optar por restringir el acceso a los datos. Por ejemplo, la clase podría restringir el valor de la coordenada *x* para que tuviera acceso de sólo lectura (*get*), deshabilitando el acceso de escritura (*set*).
- La clase puede validar o comprobar los valores utilizados. Por ejemplo, podría ignorar un intento de proporcionar un valor negativo para una coordenada.

PRÁCTICA DE AUTOEVALUACIÓN

- 9.5** Escriba un método para permitir que un usuario sólo tenga acceso *get* a la coordenada *y* de un globo.

● Constructores

Al crear un objeto globo, su posición y tamaño deben recibir valores. A esto se le conoce como inicializar las variables. Hay dos formas de inicializar variables. Una de ellas es incluir la inicialización como parte de la declaración de las variables a nivel de clase. Por ejemplo:

```
private int x = 50;
private int y = 50;
private int diámetro = 20;
```

Otra manera de inicializar un objeto es escribir un método especial para realizar la inicialización. A éste se le conoce como *método constructor* o simplemente *constructor* (ya que está involucrado en la construcción del objeto). Este método siempre debe tener el mismo nombre que la clase. No tiene

valor de retorno, pero por lo general tendrá parámetros. He aquí un método constructor para la clase **Globo**:

```
public Globo(int xInicial, int yInicial,  
            int diámetroInicial) {  
    x = xInicial;  
    y = yInicial;  
    diámetro = diámetroInicial;  
}
```

Este método asigna los valores de los parámetros (el tamaño y la posición) a las variables apropiadas dentro del objeto. Un método constructor tal como éste se escribe en la parte superior de la clase, después de las declaraciones de las variables a nivel de clase. Los constructores se etiquetan como **public** debido a que se debe acceder a ellos desde el exterior de la clase. Observe que el constructor no tiene un valor de retorno, indicado por la palabra clave **void**.

El método constructor anterior se utilizaría en forma similar al siguiente ejemplo:

```
Globo globo = new Globo(10, 10, 50);
```

Si el programador no inicializa una variable en forma explícita, el sistema de Java le proporciona a cada una de éstas un valor predeterminado. Para los números es cero, para las variables **boolean** es **false**, para las variables **String** es **""** (una cadena vacía) y para cualquier objeto es **null**. Sin embargo, se considera una mala práctica depender de este método de inicialización de variables. Es mejor hacerlo en forma explícita, ya sea al momento de declarar la información o mediante instrucciones dentro de un constructor.

Otras de las acciones que podría realizar un método constructor incluyen la creación de otros objetos que el objeto en cuestión utilice, o abrir un archivo que el objeto utilice.

Si una clase no tiene un constructor explícito, entonces se asume que tiene un solo constructor con cero parámetros, al cual se le conoce como constructor predeterminado o constructor con cero argumentos.

Varios constructores

Una clase puede tener ninguno, uno o varios métodos constructores. Si tiene uno o más constructores, por lo general llevan parámetros y se deben llamar con los parámetros apropiados. Por ejemplo, en la clase **Globo** podemos escribir los siguientes dos constructores:

```
public Globo(int xInicial, int yInicial,  
            int diámetroInicial) {  
    x = xInicial;  
    y = yInicial;  
    diámetro = diámetroInicial;  
}
```

```
public Globo(int xInicial, int yInicial) {
    x = xInicial;
    y = yInicial;
    diámetro = 20;
}
```

Lo cual nos permitiría crear objetos globo en cualquiera de las dos formas siguientes:

```
Globo globo1 = new Globo(10, 10, 50);
Globo globo2 = new Globo(10, 10);
```

pero no podríamos hacer esto:

```
Globo globo3 = new Globo();
```

Por lo tanto, si escribe varios constructores pero de todas formas necesita un constructor sin parámetros, debe escribirlo en forma explícita, por ejemplo:

```
public Globo() {
    x = 50;
    y = 50;
    diámetro = 20;
}
```

Ahora tenemos tres constructores para la clase **Globo** y he aquí cómo podríamos usarlos para crear tres objetos distintos a partir de la misma clase:

```
Globo globo1 = new Globo(10, 10, 50);
Globo globo2 = new Globo(10, 10);
Globo globo3 = new Globo();
```

PRÁCTICA DE AUTOEVALUACIÓN

9.6 Escriba un método constructor para crear un nuevo globo, especificando sólo el diámetro.

Métodos **private**

El fin de escribir una clase es permitir la creación de objetos que presenten herramientas útiles a otros objetos. Estas herramientas son los métodos **public** que ofrece el objeto. Pero a menudo una clase tiene métodos que no necesitan hacerse **public** y, de hecho, todos los métodos de los programas que hemos visto en el libro hasta este momento son **private**.

He aquí una clase **Pelota** que representa una pelota que se puede animar de manera que rebote alrededor de un panel. Utiliza métodos **private** así como un método **public** y un constructor. Usa los métodos **private** como forma de aclarar lo que de otra manera sería una pieza compleja de programa. El método **public animar** se llama a intervalos regulares frecuentes para volver a dibujar una imagen. Éste a su vez llama a los métodos **private mover**, **rebotar**, **eliminar** y

dibujar. Hemos creado métodos **private** con la finalidad de apoyar a los métodos **public** de la clase. En este ejemplo, los métodos **private** no utilizan parámetros pero, en general, los métodos **private** tienen parámetros.

```

import java.awt.*;
import javax.swing.*;
public class Pelota {
    private JPanel panel;
    private int x = 7, cambioX = 7;
    private int y = 0, cambioY = 2;
    private final int diámetro = 10;
    private final int anchura = 100, altura = 100;

    public Pelota(JPanel elPanel) {
        panel = elPanel;
    }

    public void animar() {
        eliminar();
        mover();
        rebotar();
        dibujar();
    }

    private void mover() {
        x = x + cambioX;
        y = y + cambioY;
    }

    private void rebotar() {
        if (x <= 0 || x >= anchura)
            cambioX = -cambioX;
        if (y <= 0 || y >= altura)
            cambioY = -cambioY;
    }

    private void dibujar() {
        Graphics papel = panel.getGraphics();
        papel.setColor(Color.red);
        papel.fillOval(x, y, diámetro, diámetro);
    }

    private void eliminar() {
        Graphics papel = panel.getGraphics();
        papel.setColor(Color.white);
        papel.fillOval (x, y, diámetro, diámetro);
    }
}

```

Para llamar a un método desde el interior del objeto, tenemos que hacer lo siguiente:

```
    mover();
```

proporcionando el nombre del método y sus parámetros de la manera usual. Si en realidad queremos enfatizar cuál objeto se está usando, podríamos escribir el siguiente código equivalente:

```
this.mover();
```

al utilizar la palabra clave **this** estamos indicando que es el objeto actual.

Dependiendo de su tamaño y complejidad, una clase podría tener varios métodos **private**. Su propósito es clarificar y simplificar el funcionamiento de la clase.

● Reglas de alcance

En la programación, el término *accesibilidad* (algunas veces conocido como *reglas de alcance* o *visibilidad*) se refiere a las reglas para acceder a las variables y los métodos. Para los humanos, las reglas de accesibilidad son como la regla que establece que en Australia debemos conducir por la izquierda, o la regla que establece que sólo podemos entrar a la casa de alguien por la puerta delantera. En un programa, el compilador se encarga de hacer cumplir al pie de la letra las reglas de este tipo, para evitar un acceso deliberado o erróneo a la información protegida. Las reglas de accesibilidad restringen al programador, pero le ayudan a organizar un programa en forma clara y lógica. Las reglas de accesibilidad asociadas con las clases y los métodos permiten al programador encapsular las variables y los métodos en forma conveniente.

El programador puede describir cada variable y método como **public** o **private**. Dentro de una clase, cualquier instrucción en cualquier parte de la clase puede invocar a cualquier método, ya sea **public** o **private**. Además, cualquier instrucción puede hacer referencia a cualquier variable de instancia. La excepción es que las variables locales (las que se declaran dentro de un método) sólo pueden ser utilizadas por las instrucciones que están dentro del método.

Cuando una clase hace referencia a otra, sólo aquellos métodos y variables etiquetados como **public** se pueden utilizar fuera de la clase. Todos los demás elementos son inaccesibles. Es una buena práctica de diseño minimizar el número de métodos que sean **public**, restringiéndolos de manera que sólo se ofrezcan los servicios de la clase. También es buena práctica nunca (o muy pocas veces) hacer las variables **public**. Si debemos inspeccionar o modificar una variable, debe haber métodos para realizar este trabajo.

En resumen, una variable o un método dentro de una clase se pueden describir como:

- public** – se puede utilizar en cualquier parte (desde el interior de la clase o desde cualquier otra clase).
- private** – se puede utilizar sólo dentro de la clase.

Además, sólo se puede acceder a las variables locales (las variables que se declaran dentro de un método) desde el interior del método.

Las clases se etiquetan como **public** para poder usarlas en donde sea posible. Los constructores se etiquetan como **public** debido a que necesitamos llamarlos desde el exterior de la clase.

En el capítulo 10 volveremos a ver las reglas de alcance, cuando estudiemos el tema de la herencia.

● Operaciones sobre los objetos

Muchos de los objetos que se utilizan en los programas de Java se deben declarar como instancias de clases, pero algunos no. Las variables que se declaran como `int`, `boolean` y `double` se llaman *tipos primitivos*. Vienen incluidos como parte del lenguaje Java. Mientras que los nombres de las clases por lo general empiezan con letra mayúscula, los nombres de los tipos primitivos empiezan con minúscula. Al declarar una de estas variables la podemos usar de inmediato. Por ejemplo:

```
int número;
```

declara la variable `número` y la crea a la vez. Por el contrario, al crear cualquier otro objeto tenemos que utilizar de manera explícita la palabra clave `new`. Por ejemplo:

```
Globo globo = new Globo(10, 20, 50);
```

Entonces, las variables en Java pueden ser:

1. Tipos primitivos tales como `int`, `boolean` y `double`.
2. Objetos creados de manera explícita a partir de clases, mediante `new`.

Las variables que se declaran de un tipo primitivo incluyen toda una diversidad de cosas que podemos hacer con ellas. Por ejemplo, con las variables de tipo `int` podemos:

- Declarar variables.
- Asignar valores mediante `=`.
- Realizar operaciones aritméticas.
- Comparar mediante `==`, `<`, etc.
- Usarlas como parámetro o como valor de retorno.

No necesariamente podemos hacer todas estas cosas con los objetos. Muchas de las cosas que un programa de Java utiliza son objetos pero, como hemos visto, no todo es un objeto. Y es tentador suponer que sea posible utilizar todas estas operaciones con cualquier objeto, pero no es el caso. ¿Qué podemos hacer con un objeto? La respuesta es que al escribir una clase, definimos el conjunto de operaciones que se pueden realizar sobre los objetos de ese tipo. Por ejemplo, con la clase `Globo` hemos definido las operaciones `cambiarTamaño`, `mover` y `mostrar`. No debe suponer que podemos hacer cualquier otra cosa con un globo. Sin embargo, puede suponer con confianza que puede hacer lo siguiente con cada objeto:

- Crearlo mediante `new`.
- Usarlo como parámetro y como valor de retorno.
- Asignarlo a una variable de la misma clase mediante `=`.
- Usar los métodos que se proporcionan como parte de su clase.

PRÁCTICA DE AUTOEVALUACIÓN

- 9.7 Sugiera una lista de operaciones que se puedan realizar con un objeto de la clase `Globo` y mencione ejemplos de cómo usarlas.

● Destrucción de objetos

Ya vimos cómo crear objetos mediante la poderosa palabra `new`. Pero, ¿cómo mueren? Una respuesta obvia y acertada es que esto ocurre cuando el programa deja de ejecutarse. También pueden morir cuando el programa deja de utilizarlos. Por ejemplo, si hacemos lo siguiente para crear un nuevo objeto:

```
Globo globo;  
globo = new Globo(20, 100, 100);
```

y después hacemos lo siguiente:

```
globo = new Globo(40, 200, 200);
```

lo que ocurre es que el primer objeto creado con `new` tuvo una vida muy corta, ya que murió cuando el programa dejó de tener conocimiento de él y su valor fue usurpado por el objeto más reciente.

Cuando se destruye un objeto, se reclama la memoria que se utilizaba para almacenar los valores de sus variables y cualquier otro recurso para que el sistema en tiempo de ejecución la utilice en otros procesos. A esto se le conoce como *recolección de basura*. En Java la recolección de basura es automática (en algunos otros lenguajes como C++ no es automática, por lo que el programador tiene que llevar cuenta de los objetos que ya no son necesarios).

Por último, podemos destruir un objeto al asignarle el valor `null`; por ejemplo:

```
globo = null;
```

La palabra clave `null` de Java describe a un objeto no existente (no instanciado).

● Métodos static

Algunos métodos no necesitan un objeto para trabajar. La función matemática de raíz cuadrada es un ejemplo. Los métodos matemáticos como la raíz cuadrada (`sqrt`) y el seno de un ángulo (`sin`) se proporcionan dentro de una clase de biblioteca llamada `Math`. Para usarlas en un programa escribimos instrucciones como:

```
double x, y;  
x = Math.sqrt(y);
```

En esta instrucción hay dos variables `double` llamadas `x` y `y`, pero no hay objetos. Tenga en cuenta que `Math` es el nombre de una clase y no un objeto. El método de raíz cuadrada `sqrt` actúa sobre su parámetro `y`. La pregunta es: si `sqrt` no es método de un objeto, entonces ¿qué es? La respuesta es que los métodos como éste forman parte de una clase, pero se describen como `static`. Al uti-

llamar uno de estos métodos debemos anteponer a su nombre el nombre de la clase a la que pertenece (en vez del nombre de un objeto).

La clase **Math** tiene la siguiente estructura (el código que se muestra está incompleto). Los métodos se etiquetan como **static**:

```
public class Math {
    public static double sqrt(double x) {
        // cuerpo de sqrt
    }

    public static double sin(double x) {
        // cuerpo de sin
    }
}
```

El método **parseInt** es otro ejemplo de un método **static**, el cual se encuentra dentro de la clase **Integer**. El método **main** que aparece en el encabezado de todas las aplicaciones de Java también es un método **static**.

¿Cuál es la finalidad de los métodos **static**? En la POO todo se escribe como parte de una clase; nada existe que no esté dentro de las clases. Si pensamos en la clase **Globo**, ésta contiene variables **private** como **x** y **y** que registran el estado de un objeto. Pero algunos métodos, como **sqrt**, no involucran un estado. Sin embargo, los métodos independientes como **sqrt** que obviamente no forman parte de una clase deben obedecer la regla central de la POO: tienen que formar parte de una clase. He aquí la razón de los métodos **static**. Es común que los programadores utilicen los métodos **static** de biblioteca, pero es muy poco usual que los programadores novatos los escriban.

A los métodos **static** también se les denomina métodos de *clase*, ya que pertenecen a la clase y no a un objeto creado a partir de la clase.

PRÁCTICA DE AUTOEVALUACIÓN

- 9.8** El método **static** llamado **max**, que está dentro de la clase **Math**, encuentra el máximo de sus dos parámetros **int**. Escriba un ejemplo de invocación a **max**.

Variables static

Una variable que se declara en el encabezado de una clase se puede describir como estática, o **static**. Esto significa que pertenece a la clase y no a los objetos individuales que se crean como instancias de esa clase.

Como ejemplo, la clase **Math** contiene una variable estática, la constante matemática **pi**. Esta variable se referencia como **Math.PI**, en donde se antepone de nuevo el nombre de la clase al nombre de la variable. Es muy inusual en la POO que haya valores de datos públicos como éste, ya que por lo general las variables se etiquetan como **private** en virtud del ocultamiento de la informa-

ción. El acceso a las constantes matemáticas es una excepción a esta regla general. La variable `PI` se declara de la siguiente forma dentro de la clase `Math`:

```
public class Math {
    public final static double PI = 3.142;
    // resto de la clase Math
}
```

(con la excepción de que al valor de `pi` se le asigne una mayor precisión).

La clase `Color` contiene otros ejemplos de variables `static`. Esta clase provee variables que se refencian como `Color.black`, `Color.white`, etcétera.

La descripción `static` no significa que una variable `static` no se pueda modificar. Significa que, a diferencia de las variables no estáticas, no se crea una copia de la variable cuando se crea un objeto a partir de la clase. La descripción `static` implica unicidad; es decir, sólo hay una copia de esta variable para toda la clase, en vez de una copia para cada instancia de la clase.

A las variables `static` se les conoce algunas veces como variables de clase, pero no son lo mismo que las variables a nivel de clase.

Fundamentos de programación

La POO trata acerca de cómo construir programas a partir de objetos. Un *objeto* es una combinación de ciertos datos (variables) y ciertas acciones (métodos) que desempeñan una función útil en un programa. El programador diseña un objeto de manera que los datos y las acciones estén muy relacionados entre sí, en vez de agruparlos al azar.

Al igual que en la mayoría de los lenguajes de POO, en Java no es posible escribir instrucciones que describan a un objeto en forma directa. En vez de ello, el lenguaje hace que el programador defina a todos los objetos de la misma clase. Por ejemplo, si necesitamos un objeto botón creamos una instancia de la clase `JButton`. Si necesitamos un segundo botón, creamos una segunda instancia de esta misma clase. La descripción de la estructura de todos los posibles botones se llama *clase*. Una clase es la plantilla o el plan maestro para fabricar cualquier número de objetos; una clase es la generalización de un objeto.

La idea de clases es común en la mayoría de las actividades de diseño. Por lo general, antes de construir algo real hay que crear un diseño para el objeto. Esto se aplica en el diseño automotriz, la arquitectura, la construcción —incluso en las bellas artes. Primero se bosqueja una especie de plano, a menudo en papel y algunas veces en la computadora. Comúnmente se le denomina plano de construcción. Dicho diseño especifica por completo el objeto deseado, de manera que si el diseñador es arrollado por un autobús alguien más pueda llevar a cabo la construcción del objeto. Una vez diseñado, se pueden construir varios objetos idénticos; piense en automóviles, libros o computadoras. De esta manera, el diseño especifica la composición de uno o cualquier cantidad de objetos. Lo mismo se aplica en la POO: una clase es el plano para cualquier número de objetos idénticos. Una vez que especificamos una clase, podemos construir cualquier número de objetos con el mismo comportamiento.

Si analizamos de nuevo la clase de biblioteca `JButton`, lo que tenemos es la descripción de la apariencia de cada objeto botón. Los botones sólo difieren en sus propiedades individuales, como sus posiciones en la pantalla. Por ello, en la POO una clase es la especificación para cualquier cantidad de objetos que son iguales. Una vez que se describe una clase, se puede construir un

objeto específico creando una *instancia* de esa clase. Es algo así como decir que hemos tenido una instancia de gripe en el hogar. O que este Ford Modelo T es una instancia del diseño del Ford Modelo T. Su propia cuenta bancaria es una instancia de la clase cuenta bancaria.

Un objeto constituye un agrupamiento lógico de variables y métodos. Forma un módulo independiente que se puede utilizar y entender con facilidad. El principio del ocultamiento de información o encapsulamiento implica que los usuarios de un objeto tienen una vista restringida del mismo. Un objeto proporciona un conjunto de servicios en forma de métodos `public` que otros pueden utilizar. El resto del objeto, sus variables y las instrucciones que implementan a los métodos están ocultos. Esto mejora la abstracción y la modularidad.

En computación a una clase se le denomina algunas veces tipo de datos abstracto (ADT). Un tipo de datos es algo así como una variable, por ejemplo `int`, `double` o `boolean`. Estos tipos primitivos son tipos integrados en el lenguaje Java y están disponibles para utilizarlos de inmediato. Hay un conjunto de operaciones asociado a cada uno de estos tipos. Por ejemplo, con un `int` podemos realizar operaciones de asignación, suma, resta, etcétera. La clase `Globo` que vimos antes es un ejemplo de ADT. Define ciertos datos (variables), junto con una colección de operaciones (métodos) que pueden realizar operaciones con los datos. La clase presenta una abstracción de un globo; los detalles concretos de la implementación están ocultos.

Ahora podemos entender por completo la estructura general de un programa. Todo programa de Java tiene un encabezado similar al siguiente (pero con el nombre de la clase apropiada):

```
public class UsarGlobo
```

Ésta es una descripción de una clase llamada `UsarGlobo` ya que, como todo lo demás en Java, un programa es una clase. Al iniciar un programa se hace una llamada al método `main` estático (`static`).

Errores comunes de programación

Algunas veces los principiantes quieren codificar un objeto directamente. Esto no es posible; debemos declarar una clase y después crear una instancia de esa clase.

No olvide inicializar las variables de instancia de manera explícita, por medio de un método constructor o como parte de la misma declaración; además, no se confie de la inicialización predeterminada de Java.

Si usted declara:

```
Globo globo;
```

y después ejecuta la siguiente línea de código:

```
globo.mostrar(papel);
```

su programa terminará con un mensaje de error indicando que hay una excepción de apuntador a `null`. Esto se debe a que declaró un objeto pero no lo creó (con `new`). El objeto `globo` no existe. Dicho en forma más precisa, tiene el valor `null` —que significa lo mismo. En la programación elemental casi nunca es necesario utilizar el valor `null`, a menos que olvidemos utilizar `new`.

Secretos de codificación

- Una clase tiene la siguiente estructura:

```
public class NombreClase {  
    // declaraciones de las variables de instancia  
    // declaraciones de los métodos  
}
```

- Las variables y los métodos se pueden describir como **public** o **private**.
- Uno o más de los métodos en una clase pueden tener el mismo nombre que la clase. Se puede llamar a cualquiera de estos métodos constructores (con los parámetros apropiados) para inicializar el objeto al momento de crearlo.
- La declaración de un método **public** tiene la siguiente estructura:

```
public void nombreMétodo(parámetros) {  
    // cuerpo  
}
```

- Un método estático lleva el prefijo **static** en su encabezado.
- Para llamar a un método **static** de una clase:

```
NombreClase.nombreMétodo(parámetros);
```

Nuevos elementos del lenguaje

- **class** – aparece en el encabezado de una clase.
- **public** – la descripción de una variable o un método que se pueden utilizar desde cualquier parte.
- **private** – la descripción de una variable o un método que sólo se pueden utilizar dentro de la clase.
- **new** – se utiliza para crear una nueva instancia de una clase (un nuevo objeto).
- **this** – el nombre del objeto actual.
- **null** – el nombre de un objeto que no existe.
- **static** – la descripción que se adjunta a una variable o un método que pertenece a una clase como un todo y no a una instancia creada como un objeto a partir de la clase.

Resumen

- Un objeto es una colección de datos junto con las acciones (métodos) asociadas que pueden actuar sobre esos datos. Los programas de Java se construyen como una variedad de objetos.
- Una clase es la descripción de cualquier número de objetos.
- Por lo general, una clase consiste en declaraciones de variables privadas, seguidas de algunos métodos públicos.
- Los elementos de una clase se pueden declarar como **private** o **public**. Un elemento **private** sólo se puede utilizar dentro de la clase. Un elemento **public** se puede utilizar en cualquier parte (dentro o fuera de la clase). Al diseñar un programa de Java se evitan las variables **public** para mejorar el ocultamiento de información.
- El programador puede escribir métodos para inicializar un objeto al crearlo. A estos métodos se les denomina constructores y tienen el mismo nombre que la clase.
- La descripción **static** significa que la variable o el método pertenecen a la clase y no a objetos específicos.

Ejercicios

- 9.1 Globos** Agregue varios datos más a la clase **Globo**: una variable **String** que guarde el nombre del globo y una variable **Color** que describa su color. Agregue código para inicializar estos valores mediante un método constructor. Agregue el código para mostrar el balón de color y su nombre.
Mejore el programa del globo con botones que muevan el globo a la izquierda, a la derecha, hacia arriba y hacia abajo.
- 9.2 Termómetro** Algunos termómetros registran las temperaturas máxima y mínima que se han alcanzado. Escriba un programa que simule un termómetro mediante un control deslizable y que muestre en campos de texto los valores máximo y mínimo a los que se ajustó el control deslizable. Escriba como una clase separada la pieza de programa que recuerde los valores mayor y menor, y que compare nuevos valores. Esta clase debe tener los métodos **setNuevoValor**, **getMenorValor** y **getMayorValor**.
- 9.3 Cuenta bancaria** Escriba un programa que simule una cuenta bancaria. Un botón permite hacer un depósito en la cuenta. El monto se introduce en un campo de texto. Un segundo botón permite hacer un retiro. El monto (el saldo) y el estado de la cuenta se deben mostrar en forma continua: el estado puede ser OK o sobregiro. Cree una clase llamada **Cuenta** para representar cuentas bancarias. Debe tener los métodos **depositar** y **retirar**, **getSaldoActual** y **setSaldoActual**.
- 9.4 Tanteador** Diseñe y escriba una clase que actúe como tanteador para un juego de computadora. Debe mantener un solo entero: la puntuación. Además debe proporcionar un método para inicializar la puntuación en cero, un método para incrementar la puntuación, un método para reducir la puntuación

y otro método para devolver la puntuación. Escriba un programa para crear un solo objeto y utilizarlo. Siempre debe aparecer la puntuación actual en pantalla, dentro de un campo de texto. Debe haber botones para incrementar, reducir e inicializar la puntuación con base en una cantidad introducida en un campo de texto.

- 9.5 Dados** Diseñe y escriba una clase que actúe como un dado, el cual se puede lanzar para obtener un valor del 1 al 6. Escriba la clase de manera que en un principio se obtenga el valor 6. Escriba un programa para crear un objeto dado y utilizarlo. La pantalla debe mostrar un botón que al oprimirlo haga que se lance el dado y se muestre su valor.

Después modifique la clase dado de manera que proporcione un valor que sea un punto mayor al que tenía la última vez que se lanzó; por ejemplo, 4 cuando era un 3. Cuando el último valor sea 6, el nuevo valor será 1.

Luego modifique la clase para que utilice el generador de números aleatorios de la biblioteca.

Algunos juegos como el backgammon necesitan dos dados. Escriba instrucciones de Java para crear dos instancias del objeto dado, lanzar los dados y mostrar los resultados.

- 9.6 Generador de números aleatorios** Escriba su propio generador de números aleatorios como una clase que utilice una fórmula para obtener el siguiente número pseudoaleatorio a partir del anterior. Un programa de números aleatorios funciona empezando con cierto valor de "semilla". A partir de ese momento, el número aleatorio actual se utiliza como base para obtener el siguiente número, para lo cual realizamos ciertos cálculos con el número actual para convertirlo en algún otro número (aparentemente aleatorio). Una buena fórmula que podemos usar para los enteros es:

```
siguienteA = ((antiguoA * 25173) + 13849) % 65536;
```

esta fórmula produce números en el rango de 0 a 65,535. Los números específicos en esta fórmula han demostrado producir buenos resultados tipo aleatorios.

Al principio haga que el valor de semilla sea 1. Después, en un programa más sofisticado, obtenga la parte de los milisegundos del tiempo mediante el uso de la clase de biblioteca **Calendar** (vea el apéndice A) para que actúe como semilla.

- 9.7 Estacionamiento** Un programa provee dos botones. El asistente del estacionamiento hace clic en un botón cuando un automóvil entra al lote y presiona el otro botón cuando sale un automóvil. Si un automóvil intenta entrar cuando el lote ya está lleno, se muestra una advertencia en un panel de opción.

Implemente el conteo de automóviles y sus operaciones como una clase. Debe proveer un método llamado **entrar**, el cual debe incrementar el conteo; además, debe incluir un método llamado **salir**, que disminuya el conteo. Un tercer método (llamado **lleno**) debe devolver un valor **boolean** que especifique si el lote está lleno o no.

- 9.8 Números complejos** Escriba una clase llamada **Complejo** para representar números complejos (junto con sus operaciones). Un número complejo consta de dos partes: una parte real (un **double**) y una parte imaginaria (un **double**). El método constructor debe crear un nuevo número complejo mediante los valores **double** que se proporcionen como parámetros, de la siguiente forma:

```
Complejo c = new Complejo(1.0, 2.0);
```

Escriba los métodos **getReal** y **getImaginaria** para obtener la parte real y la parte imaginaria de un número complejo, las cuales se deben utilizar de la siguiente manera:

```
double x = c.getReal();
```

Escriba un método llamado **suma** para sumar dos números complejos y devolver su suma. La parte real es la suma de las dos partes reales. La parte imaginaria es la suma de las dos partes imaginarias. Una llamada al método sería así:

```
Complejo c = c1.suma(c2);
```

Utilice dos campos de texto para introducir los valores de **c1** y haga lo mismo para **c2**. También muestre los valores de **c** en dos campos de texto.

Escriba un método llamado **prod** para calcular el producto de dos números complejos. Si un número tiene los componentes x_1 y y_1 , y el segundo número tiene los componentes x_2 y y_2 :

La parte real del producto = $x_1 \times x_2 - y_1 \times y_2$

La parte imaginaria del producto = $x_1 \times y_2 + x_2 \times y_1$

Respuestas a las prácticas de autoevaluación

- 9.1

```
private Color color;
```
- 9.2

```
public void moverArriba(int cantidad) {
            coordY = coordY - cantidad;
        }
```
- 9.3

```
public void cambiarColor(Color nuevoColor) {
            color = nuevoColor;
        }
```
- 9.4

```
public void mostrar(Graphics papel) {
            papel.setColor(color);
            papel.drawOval(x, y, diámetro, diámetro);
        }
```
- 9.5

```
public int getY() {
            return y;
        }
```
- 9.6

```
public Globo(int diámetroInicial) {
            diámetro = diámetroInicial;
        }
```
- 9.7 Los métodos son: **cambiarColor**, **moverIzquierda**, **moverDerecha**, **cambiarTamaño**, **mostrar**, **getX**, **getY**.
Algunos ejemplos son:

```
globo.cambiarColor(Color.red);
globo.moverIzquierda(20);
globo.moverDerecha(50);
globo.cambiarTamaño(10);
globo.mostrar(papel);
int x = globo.getX();
int y = globo.getY();
```
- 9.8

```
int x;
x = Math.max(7, 8);
```