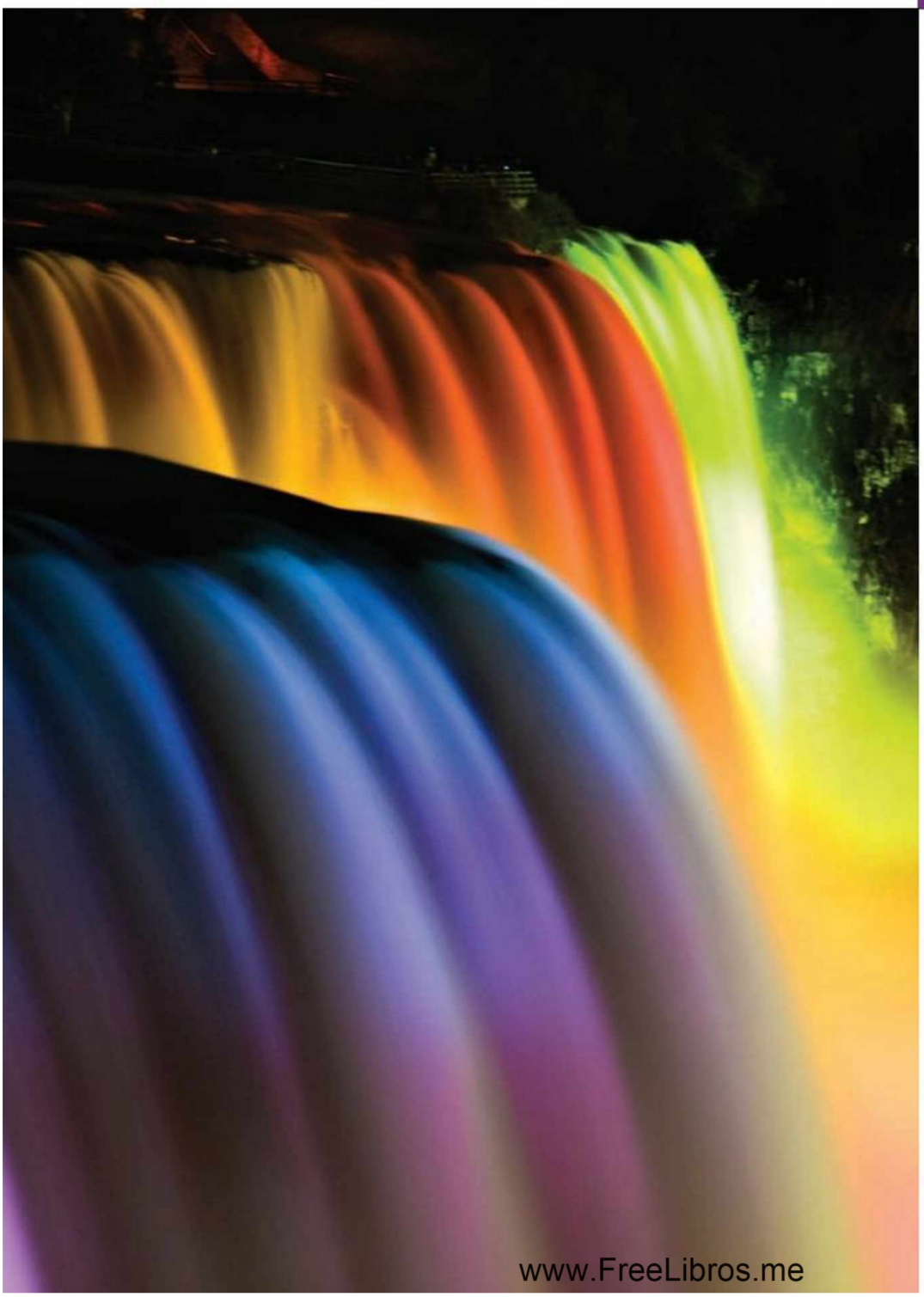
# Introducción a las clases, objetos, métodos y cadenas

# 3



*Nada puede tener valor sin ser un objeto de utilidad.*
—Karl Marx

*Sus servidores públicos le sirven bien.*
—Adlai E. Stevenson

*Usted verá algo nuevo. Dos cosas. Y las llamo Cosa Uno y Cosa Dos.*
—Dr. Theodor Seuss Geisel

## Objetivos

En este capítulo aprenderá a:

- Declarar una clase y utilizarla para crear un objeto.
- Implementar los comportamientos de una clase como métodos.
- Implementar los atributos de una clase como variables de instancia y propiedades.
- Llamar a los métodos de un objeto para hacer que realicen sus tareas.
- Conocer cuáles son las variables de instancia de una clase y las variables locales de un método.
- Utilizar un constructor para inicializar los datos de un objeto.
- Conocer las diferencias entre los tipos primitivos y los tipos por referencia.

www.FreeLibros.me

Plan general



## 3.1 Introducción

En la sección 1.6 le presentamos la terminología básica y los conceptos acerca de la programación orientada a objetos. En este capítulo le presentaremos un marco de trabajo simple para organizar aplicaciones orientadas a objetos en Java. Por lo general, las aplicaciones que desarrollará en este libro consistirán de dos o más clases. Si se vuelve parte de un equipo de desarrollo en la industria, podría trabajar en aplicaciones que contengan cientos, o incluso miles de clases.

Primero explicaremos el concepto de las clases con un ejemplo real. Después presentaremos cinco aplicaciones para demostrarle cómo crear y utilizar sus propias clases. Los primeros cuatro ejemplos empiezan nuestro caso de estudio acerca de cómo desarrollar una clase tipo libro de calificaciones, que los instructores pueden utilizar para mantener las calificaciones de las pruebas de sus estudiantes. Ampliaremos este ejemplo práctico en los capítulos 4, 5 y 7. El último ejemplo en este capítulo introduce los números de punto flotante (números que contienen puntos decimales) en una clase tipo cuenta bancaria, la cual mantiene el saldo de un cliente.

## 3.2 Declaración de una clase con un método e instanciamiento de un objeto de una clase

En las secciones 2.5 y 2.8 creó un objeto de la clase *existente* `Scanner`, y después lo utilizó para leer datos mediante el teclado. En esta sección creará una *nueva* clase y después la utilizará para crear un objeto. Comenzaremos por declarar las clases `LibroCalificaciones` (figura 3.1) y `PruebaLibroCalificaciones` (figura 3.2). La clase `LibroCalificaciones` (declarada en el archivo `LibroCalificaciones.java`) se utilizará para mostrar un mensaje en la pantalla (figura 3.2), para dar la bienvenida al instructor a la aplicación del libro de calificaciones. La clase `PruebaLibroCalificaciones` (declarada en el archivo `PruebaLibroCalificaciones.java`) es una clase de aplicación en la que el método `main` creará y utilizará un objeto de la clase `LibroCalificaciones`. *Cada declaración de clase que comienza con la palabra clave `public` debe almacenarse en un archivo que tenga el mismo nombre que la clase, y que termine con la extensión de archivo `.java`.* Por lo tanto, las clases `LibroCalificaciones` y `PruebaLibroCalificaciones` deben declararse en archivos separados, ya que cada clase se declara como `public`.

### *La clase `LibroCalificaciones`*

La declaración de la clase `LibroCalificaciones` (figura 3.1) contiene un método llamado `mostrarMensaje` (líneas 7-10), el cual muestra un mensaje en la pantalla. Necesitamos crear un objeto de esta clase y llamar a su método para hacer que se ejecute la línea 9 y que muestre su mensaje.

La *declaración de la clase* empieza en la línea 4. La palabra clave `public` es un **modificador de acceso**. Por ahora, simplemente declararemos cada clase como `public`. Toda declaración de clase contiene la

www.FreeLibros.me

```
1  // Fig. 3.1: LibroCalificaciones.java
2  // Declaración de una clase con un método.
3
4  public class LibroCalificaciones
5  {
6     // muestra un mensaje de bienvenida al usuario de LibroCalificaciones
7     public void mostrarMensaje()
8     {
9        System.out.println( "Bienvenido al Libro de calificaciones!" );
10    } // fin del método mostrarMensaje
11 } // fin de la clase LibroCalificaciones
```

**Fig. 3.1** | Declaración de una clase con un método.

palabra clave `class`, seguida de inmediato por el nombre de la clase. El cuerpo de toda clase se encierra entre un par de llaves izquierda y una derecha, como en las líneas 5 y 11 de la clase `LibroCalificaciones`.

En el capítulo 2, cada clase que declaramos tenía un método llamado `main`. La clase `LibroCalificaciones` también tiene un método: `mostrarMensaje` (líneas 7-10). Recuerde que `main` es un método especial, que *siempre* es llamado, automáticamente, por la Máquina Virtual de Java (JVM) a la hora de ejecutar una aplicación. La mayoría de los métodos no se llaman en forma automática. Como veremos en breve, es necesario llamar al método `mostrarMensaje` de manera explícita para indicarle que haga su trabajo.

La declaración del método comienza con la palabra clave `public` para indicar que el método está "disponible al público": los métodos de otras clases pueden llamarlo. A continuación está el **tipo de valor de retorno** del método, el cual especifica el tipo de datos que devuelve el método a quien lo llamó después de realizar su tarea. El tipo de valor de retorno `void` indica que este método realizará una tarea pero *no* devolverá (es decir, regresará) información al **método que lo llamó**. Ya hemos utilizado métodos que devuelven información; por ejemplo, en el capítulo 2 utilizó el método `nextInt` de `Scanner` para recibir un entero escrito por el usuario desde el teclado. Cuando `nextInt` recibe un valor de entrada, devuelve ese valor para utilizarlo en el programa.

El nombre del método, `mostrarMensaje`, va después del tipo de valor de retorno. Por convención, los nombres de los métodos comienzan con una letra minúscula, y el resto de las palabras en el nombre con mayúsculas. Los paréntesis después del nombre del método indican que éste es un método. Un conjunto vacío de paréntesis, como se muestra en la línea 7, indica que este método no requiere información adicional para realizar su tarea. La línea 7 se conoce comúnmente como el **encabezado del método**. El cuerpo de cada método se delimita mediante las llaves izquierda y derecha, como en las líneas 8 y 10.

El cuerpo de un método contiene una o varias instrucciones que realizan su trabajo. En este caso, el método contiene una instrucción (línea 9) que muestra el mensaje "`Bienvenido al Libro de calificaciones!`", seguido de una nueva línea (debido a `println`) en la ventana de comandos. Una vez que se ejecuta esta instrucción, el método ha completado su trabajo.

### *La clase* `PruebaLibroCalificaciones`

A continuación, nos gustaría utilizar la clase `LibroCalificaciones` en una aplicación. Como aprendió en el capítulo 2, el método `main` empieza la ejecución de *todas* las aplicaciones. Una clase que contiene el método `main` empieza la ejecución de una aplicación de Java. La clase `LibroCalificaciones` *no* es una aplicación, ya que *no* contiene a `main`. Por lo tanto, si trata de ejecutar `LibroCalificaciones` escribiendo `java LibroCalificaciones` en la ventana de comandos, se producirá un mensaje de error. Esto no fue un problema en el capítulo 2, ya que cada clase que declaramos tenía un método `main`. Para corregir este problema, debemos declarar una clase separada que contenga un método `main`, o colocar un método `main` en la clase `LibroCalificaciones`. Para ayudarlo a prepararse para los pro-

www.FreeLibros.me

gramas más extensos que encontrará más adelante en este libro y en la industria, utilizamos una clase separada (`PruebaLibroCalificaciones` en este ejemplo) que contiene el método `main` para probar cada nueva clase que vayamos a crear en este capítulo. Algunos programadores se refieren a este tipo de clases como una *clase controladora*.

La declaración de la clase `PruebaLibroCalificaciones` (figura 3.2) contiene el método `main` que controlará la ejecución de nuestra aplicación. La declaración de la clase `PruebaLibroCalificaciones` empieza en la línea 4 y termina en la línea 15. La clase *sólo* contiene un método `main`, algo común en muchas clases que empiezan la ejecución de una aplicación.

```
1  // Fig. 3.2: PruebaLibroCalificaciones.java
2  // Crea un objeto LibroCalificaciones y llama a su método mostrarMensaje.
3
4  public class PruebaLibroCalificaciones
5  {
6     // el método main empieza la ejecución del programa
7     public static void main( String[] args )
8     {
9        // crea un objeto LibroCalificaciones y lo asigna a miLibroCalificaciones
10       LibroCalificaciones miLibroCalificaciones = new LibroCalificaciones();
11
12       // llama al método mostrarMensaje de miLibroCalificaciones
13       miLibroCalificaciones.mostrarMensaje();
14    } // fin de main
15 } // fin de la clase PruebaLibroCalificaciones
```

```
Bienvenido al Libro de calificaciones!
```

**Fig. 3.2** | Cómo crear un objeto de la clase `LibroCalificaciones` y llamar a su método `mostrarMensaje`.

Las líneas 7 a la 14 declaran el método `main`. Una parte clave para permitir que la JVM localice y llame al método `main` para empezar la ejecución de la aplicación es la palabra clave `static` (línea 7), la cual indica que `main` es un método `static`. *Un método `static` es especial, ya que puede llamarse sin tener que crear primero un objeto de la clase en la cual se declara ese método.* En el capítulo 6, Métodos: un análisis más detallado, analizaremos los métodos `static`.

En esta aplicación nos gustaría llamar al método `mostrarMensaje` de la clase `LibroCalificaciones` para mostrar el mensaje de bienvenida en la ventana de comandos. Por lo general, no podemos llamar a un método que pertenece a otra clase sino hasta crear un objeto de esa clase, como se muestra en la línea 10. Empezaremos por declarar la variable `miLibroCalificaciones`. El tipo de la variable es `LibroCalificaciones`: la clase que declaramos en la figura 3.1. Cada nueva *clase* que creamos se convierte en un nuevo *tipo*, que puede usarse para declarar variables y crear objetos. Usted puede declarar nuevos tipos de clases según lo necesite; ésta es una razón por la cual Java se conoce como un **lenguaje extensible**.

La variable `miLibroCalificaciones` se inicializa (línea 10) con el resultado de la **expresión de creación de instancia de clase** `new LibroCalificaciones()`. La palabra clave **new** crea un nuevo objeto de la clase especificada a la derecha de la palabra clave (es decir, `LibroCalificaciones`). Los paréntesis a la derecha de `LibroCalificaciones` son obligatorios. Como veremos en la sección 3.6, esos paréntesis en combinación con el nombre de una clase representan una llamada a un **constructor**, que es similar a un método, pero se utiliza sólo cuando se crea un objeto para *inicializar* los datos de éste. En esa sección verá que los datos pueden colocarse entre paréntesis para especificar los *valores iniciales* para los datos del objeto. Por ahora, sólo dejaremos los paréntesis vacíos.

www.FreeLibros.me

Así como podemos usar el objeto `System.out` para llamar a sus métodos `print`, `printf` y `println`, también podemos usar el objeto `miLibroCalificaciones` para llamar a su método `mostrarMensaje`. La línea 13 llama al método `mostrarMensaje` (líneas 7-10 de la figura 3.1), mediante el uso de `miLibroCalificaciones` seguida de un **separador punto** (`.`), el nombre del método `mostrarMensaje` y un conjunto vacío de paréntesis. Esta llamada hace que el método `mostrarMensaje` realice su tarea. La llamada a este método difiere de las del capítulo 2 en las que se mostraba la información en una ventana de comandos; cada una de estas llamadas al método proporcionaba argumentos que especificaban los datos a mostrar. Al inicio de la línea 13, "`miLibroCalificaciones.`" indica que `main` debe utilizar el objeto `miLibroCalificaciones` que se creó en la línea 10. La línea 7 de la figura 3.1 indica que el método `mostrarMensaje` tiene una *lista de parámetros vacía*; es decir, `mostrarMensaje` *no* requiere información adicional para realizar su tarea. Por esta razón, la llamada al método (línea 13 de la figura 3.2) especifica un conjunto vacío de paréntesis después del nombre del método, para indicar que *no* se van a pasar *argumentos* al método `mostrarMensaje`. Cuando el método `mostrarMensaje` completa su tarea, el método `main` continúa su ejecución en la línea 14. Éste es el final del método `main`, por lo que el programa termina.

Cualquier clase puede contener un método `main`. La JVM lo invoca *sólo* en la clase que se utiliza para ejecutar la aplicación. Si una aplicación tiene varias clases que contengan `main`, el que se invoque será el de la clase nombrada en el comando `java`.

### *Compilación de una aplicación con varias clases*

Debe compilar las clases de las figuras 3.1 y 3.2 antes de poder ejecutar la aplicación. Primero, cambie al directorio que contiene los archivos de código fuente de la aplicación. Después, escriba el comando

```
javac LibroCalificaciones.java PruebaLibroCalificaciones.java
```

para compilar *ambas* clases a la vez. Si el directorio que contiene la aplicación sólo incluye los archivos de ésta, puede compilar *todas* las clases que haya en el directorio con el comando

```
javac *.java
```

El asterisco (`*`) en `*.java` indica que deben compilarse *todos* los archivos en el directorio actual que terminen con la extensión de nombre de archivo "`.java`".

### *Diagrama de clases de UML para la clase* `LibroCalificaciones`

La figura 3.3 presenta un **diagrama de clases de UML** para la clase `LibroCalificaciones` de la figura 3.1. En UML, cada clase se modela en un diagrama de clases en forma de un rectángulo con tres compartimientos. El compartimiento superior contiene el nombre de la clase, centrado en forma horizontal y en negrita. El compartimiento de en medio contiene los atributos de la clase, que en Java corresponden a las variables de instancia (las cuales analizaremos en la sección 3.4). En la figura 3.3, el compartimiento de en medio está vacío, ya que esta clase `LibroCalificaciones` *no* tiene atributos. El compartimiento inferior contiene las **operaciones** de la clase, que en Java corresponden a los métodos.

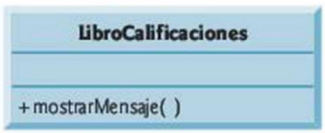



**Fig. 3.3** | Diagrama de clases de UML, el cual indica que la clase `LibroCalificaciones` tiene una operación `public` llamada `mostrarMensaje`.

www.FreeLibros.me

Para modelar las operaciones, UML lista el nombre de la operación precedido por un modificador de acceso (en este caso, +) y seguido de un conjunto de paréntesis. La clase `LibroCalificaciones` tiene un solo método llamado `mostrarMensaje`, por lo que el compartimiento inferior de la figura 3.3 lista una operación con este nombre. El método `mostrarMensaje` *no* requiere información adicional para realizar sus tareas, por lo que los paréntesis que van después del nombre del método en el diagrama de clases están *vacíos*, de igual forma que como aparecieron en la declaración del método, en la línea 7 de la figura 3.1. El signo más (+) que va antes del nombre de la operación indica que `mostrarMensaje` es una operación `pública` en UML (es decir, un método `public` en Java). Utilizaremos los diagramas de clases de UML con frecuencia para sintetizar los atributos y las operaciones de una clase.

## 3.3 Declaración de un método con un parámetro

En nuestra analogía del auto de la sección 1.6, hablamos sobre el hecho de que al pisar el pedal del acelerador se envía un mensaje al auto para que *realice la tarea*: que vaya más rápido. Pero, *¿qué tan rápido* debería acelerar el auto? Como sabe, cuanto más pisa el pedal, mayor será la aceleración del auto. Por lo tanto, el mensaje para el auto en realidad involucra tanto la *tarea a realizar* como *información adicional* que ayuda al auto a ejecutar su tarea. A la información adicional se le conoce como **parámetro**; el valor del parámetro ayuda al auto a determinar qué tan rápido debe acelerar. De manera similar, un método puede requerir uno o más parámetros que representan la información adicional que necesita para realizar su tarea. Los parámetros se definen en una **lista de parámetros** separada por comas, ubicada dentro de los paréntesis que van después del nombre del método. Cada parámetro debe especificar un tipo y un nombre de variable. La lista de parámetros puede contener cualquier número de éstos, o inclusive ninguno. Los paréntesis vacíos después del nombre del método (como en la línea 7 de la figura 3.1) indican que un método *no* requiere parámetros.

### *Argumentos para un método*

La llamada a un método proporciona valores (llamados *argumentos*) para cada uno de los parámetros de ese método. Por ejemplo, el método `System.out.println` requiere un argumento que especifica los datos a mostrar en una ventana de comandos. De manera similar, para realizar un depósito en una cuenta bancaria, un método llamado `deposito` especifica un parámetro que representa el monto a depositar. Cuando se hace una llamada al método `deposito`, se asigna al parámetro del método un valor como argumento, que representa el monto a depositar. Entonces, el método realiza un depósito por ese monto.

### *Declaración de una clase con un método que tiene un parámetro*

Ahora vamos a declarar la clase `LibroCalificaciones` (figura 3.4), con un método `mostrarMensaje` que muestra el nombre del curso como parte del mensaje de bienvenida (en la figura 3.5 podrá ver la ejecución de ejemplo). Este nuevo método requiere un parámetro que representa el nombre del curso a imprimir en pantalla.

Antes de hablar sobre las nuevas características de la clase `LibroCalificaciones`, veamos cómo se utiliza la nueva clase desde el método `main` de la clase `PruebaLibroCalificaciones` (figura 3.5). La línea 12 crea un objeto `Scanner` llamado `entrada`, para recibir el nombre del curso escrito por el usuario. La línea 15 crea el objeto `miLibroCalificaciones` de la clase `LibroCalificaciones`. La línea 18 pide al usuario que escriba el nombre de un curso. La línea 19 lee el nombre que introduce el usuario y lo asigna a la variable `nombreDelCurso`, mediante el uso del método `nextLine` de `Scanner` para realizar la operación de entrada. El usuario escribe el nombre del curso y oprime *Intro* para enviarlo al programa. Al oprimir *Intro* se inserta un carácter de nueva línea al final de los caracteres escritos por el usuario. El método `nextLine` los lee hasta encontrar la nueva línea, luego devuelve un objeto `String` que contiene los caracteres hasta la nueva línea, pero *sin* incluirla. El carácter de nueva línea se *descarta*.

www.FreeLibros.me

```
1  // Fig. 3.4: LibroCalificaciones.java
2  // Declaración de una clase con un método que tiene un parámetro.
3
4  public class LibroCalificaciones
5  {
6     // muestra un mensaje de bienvenida al usuario de LibroCalificaciones
7     public void mostrarMensaje( String nombreDelCurso )
8     {
9        System.out.printf( "Bienvenido al libro de calificaciones para\n%s!\n",
10          nombreDelCurso );
11    } // fin del método mostrarMensaje
12 } // fin de la clase LibroCalificaciones
```

**Fig. 3.4** | Declaración de una clase con un método que tiene un parámetro.

```
1  // Fig. 3.5: PruebaLibroCalificaciones.java
2  // Crea un objeto LibroCalificaciones y pasa un objeto String
3  // a su método mostrarMensaje.
4  import java.util.Scanner; // el programa usa la clase Scanner
5
6  public class PruebaLibroCalificaciones
7  {
8     // el método main empieza la ejecución del programa
9     public static void main( String[] args )
10    {
11       // crea un objeto Scanner para obtener la entrada de la ventana de comandos
12       Scanner entrada = new Scanner( System.in );
13
14       // crea un objeto LibroCalificaciones y lo asigna a miLibroCalificaciones
15       LibroCalificaciones miLibroCalificaciones = new LibroCalificaciones();
16
17       // pide y recibe el nombre del curso como entrada
18       System.out.println( "Escriba el nombre del curso:" );
19       String nombreDelCurso = entrada.nextLine(); // lee una línea de texto
20       System.out.println(); // imprime una línea en blanco
21
22       // llama al método mostrarMensaje de miLibroCalificaciones
23       // y pasa nombreDelCurso como argumento
24       miLibroCalificaciones.mostrarMensaje( nombreDelCurso );
25    } // fin de main
26 } // fin de la clase PruebaLibroCalificaciones
```

```
Escriba el nombre del curso:
CS101 Introduccion a la programacion en Java

Bienvenido al libro de calificaciones para
CS101 Introduccion a la programacion en Java!
```

**Fig. 3.5** | Cómo crear un objeto `LibroCalificaciones` y pasar un objeto `String` a su método `mostrarMensaje`.

La clase `Scanner` también cuenta con un método similar (`next`) para leer palabras individuales. Cuando el usuario oprime *Intro* después de escribir la entrada, el método `next` lee caracteres hasta encontrar un *carácter de espacio en blanco* (espacio, tabulador o nueva línea), y después devuelve un objeto

www.FreeLibros.me

`String` que contiene los caracteres hasta el carácter de espacio en blanco (que se descarta), pero *sin* incluirlo. No se pierde toda la información que va después del primer carácter de espacio en blanco; estará disponible para que la lean otras instrucciones que llamen a los métodos de `Scanner`, más adelante en el programa. La línea 20 imprime una línea en blanco.

La línea 24 llama al método `mostrarMensaje` de `miLibroCalificaciones`. La variable `nombreDelCurso` entre paréntesis es el *argumento* que se pasa al método `mostrarMensaje`, para que éste pueda realizar su tarea. El valor de la variable `nombreDelCurso` en `main` se convierte en el valor del *parámetro* `nombreDelCurso` del método `mostrarMensaje`, en la línea 7 de la figura 3.4. Al ejecutar esta aplicación, observe que el método `mostrarMensaje` imprime en pantalla el nombre que usted escribió como parte del mensaje de bienvenida (figura 3.5).

### *Más sobre los argumentos y los parámetros*

En la figura 3.4, la lista de parámetros de `mostrarMensaje` (línea 7) declara un parámetro que indica que el método requiere un objeto `String` para realizar su trabajo. En el instante en que se llama al método, el valor del argumento en la llamada se asigna al parámetro correspondiente (`nombreDelCurso`) en el encabezado del método. Después, el cuerpo del método utiliza el valor del parámetro `nombreDelCurso`. Las líneas 9 y 10 de la figura 3.4 muestran el valor del parámetro `nombreDelCurso`, mediante el uso del especificador de formato `%s` en la cadena de formato de `printf`. El nombre de la variable de parámetro (`nombreDelCurso` en la figura 3.4, línea 7) puede ser *igual o distinto* al nombre de la variable de argumento (`nombreDelCurso` en la figura 3.5, línea 24).

El número de argumentos en la llamada a un método *debe* coincidir con el de los parámetros en la lista de parámetros de la declaración del método. Además, los tipos de los argumentos en la llamada al método deben ser "consistentes con" los de los parámetros correspondientes en la declaración del método (como veremos en el capítulo 6, no siempre se requiere que el tipo de un argumento y el de su correspondiente parámetro sean *idénticos*). En nuestro ejemplo, la llamada al método pasa un argumento de tipo `String` (`nombreDelCurso` se declara como `String` en la línea 19 de la figura 3.5) y la declaración del método especifica un parámetro de tipo `String` (`nombreDelCurso` se declara como `String` en la línea 7 de la figura 3.4). Por lo tanto, en este ejemplo, el tipo del argumento en la llamada al método coincide exactamente con el tipo del parámetro en el encabezado del método.

### *Diagrama de clases de UML actualizado para la clase* `LibroCalificaciones`

El diagrama de clases de UML de la figura 3.6 modela la clase `LibroCalificaciones` de la figura 3.4. Al igual que la figura 3.1, esta clase `LibroCalificaciones` contiene la operación `public` llamada `mostrarMensaje`. Sin embargo, esta versión de `mostrarMensaje` tiene un parámetro. La forma en que UML modela un parámetro es un poco distinta a la de Java, ya que lista el nombre de éste, seguido de dos puntos y su tipo entre paréntesis, después del nombre de la operación. UML tiene sus propios tipos de datos, que son similares a los de Java (pero como veremos, no todos los tipos de datos de UML tienen los mismos nombres que los correspondientes en Java). El tipo `String` de UML corresponde al tipo `String` de Java. El método `mostrarMensaje` de `LibroCalificaciones` (figura 3.4) tiene un parámetro `String` llamado `nombreDelCurso`, por lo que en la figura 3.6 se lista a `nombreDelCurso : String` entre los paréntesis que van después de `mostrarMensaje`.

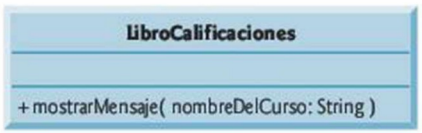



**Fig. 3.6** | Diagrama de clases de UML, que indica que la clase `LibroCalificaciones` tiene una operación llamada `mostrarMensaje`, con un parámetro llamado `nombreDelCurso` de tipo `String` de UML.

www.FreeLibros.me

### *Observaciones acerca del uso de las declaraciones* `import`

Observe la declaración `import` en la figura 3.5 (línea 4). Esto indica al compilador que el programa utiliza la clase `Scanner`. ¿Por qué necesitamos importar la clase `Scanner`, pero no las clases `System`, `String` o `LibroCalificaciones`? Las clases `System` y `String` están en el paquete `java.lang`, que se importa de manera implícita en *todo* programa de Java, por lo que todos los programas pueden usar las clases de ese paquete *sin tener que* importarlas de manera explícita. La mayoría de las otras clases que utilizará en los programas de Java deben importarse de manera explícita.

Hay una relación especial entre las clases que se compilan en el mismo directorio en el disco, como las clases `LibroCalificaciones` y `PruebaLibroCalificaciones`. De manera predeterminada, se considera que dichas clases se encuentran en el mismo paquete; a éste se le conoce como el **paquete predeterminado**. Las clases en el mismo paquete se *importan implícitamente* en los archivos de código fuente de las otras clases en el mismo paquete. Por ende, *no* se requiere una declaración `import` cuando la clase en un paquete utiliza a otra en el mismo paquete; como cuando `PruebaLibroCalificaciones` utiliza a la clase `LibroCalificaciones`.

La declaración `import` en la línea 4 *no* es obligatoria si siempre hacemos referencia a la clase `Scanner` como `java.util.Scanner`, que contiene el *nombre completo del paquete y de la clase*. Esto se conoce como el **nombre de clase completamente calificado**. Por ejemplo, la línea 12 podría escribirse como

```
java.util.Scanner entrada = new java.util.Scanner( System.in );
```

**Observación de ingeniería de software 3.1**

*El compilador de Java no requiere declaraciones `import` en un archivo de código fuente de Java, si se especifica el nombre de clase completamente calificado cada vez que se utilice el nombre de una clase en el código fuente. La mayoría de los programadores de Java prefieren usar declaraciones `import`.*

## 3.4 Variables de instancia, métodos *establecer* y métodos *obtener*

En el capítulo 2 declaramos todas las variables de una aplicación en el método `main`. Las variables que se declaran en el cuerpo de un método específico se conocen como **variables locales**, y sólo se pueden utilizar en ese método. Cuando termina ese método, se pierden los valores de sus variables locales. En la sección 1.6 vimos que un objeto tiene *atributos* que lleva consigo cuando se utiliza en un programa. Los cuales existen antes de que un objeto llame a un método, al momento y después de que éste se ejecuta.

Por lo general, una clase consiste en uno o más métodos que manipulan los atributos pertenecientes a un objeto específico de la clase. Los atributos se representan como variables en la declaración de la clase. Dichas variables se llaman **campos** y se declaran *dentro* de la declaración de una clase, pero *fuera* de los cuerpos de las declaraciones de los métodos de ésta. Cuando cada objeto de una clase mantiene su propia copia de un atributo, el campo que representa a ese atributo se conoce también como **variable de instancia**; cada objeto (instancia) de la clase tiene una instancia separada de la variable en memoria. El ejemplo en esta sección demuestra una clase `LibroCalificaciones`, que contiene una variable de instancia llamada `nombreDelCurso` para representar el nombre del curso de un objeto `LibroCalificaciones` específico.

### *La clase* `LibroCalificaciones` *con una variable de instancia, un método* establecer *y un método* obtener

En nuestra siguiente aplicación (figuras 3.7 y 3.8), la clase `LibroCalificaciones` (figura 3.7) mantiene el nombre del curso como una variable de instancia, para que pueda usarse o modificarse en cualquier momento, durante la ejecución de una aplicación. Esta clase contiene tres métodos: `establecerNombreDelCurso`, `obtenerNombreDelCurso` y `mostrarMensaje`. El método `establecerNombreDelCurso` almacena el nombre de un curso en un `LibroCalificaciones`. El método `obtenerNombreDelCurso` obtiene el nombre del curso de un `LibroCalificaciones`. El método `mostrarMensaje`, que en este caso no

www.FreeLibros.me

especifica parámetros, sigue mostrando un mensaje de bienvenida que incluye el nombre del curso; como veremos más adelante, el método ahora obtiene el nombre del curso mediante una llamada a otro método en la misma clase: `obtenerNombreDelCurso`.

```
1  // Fig. 3.7: LibroCalificaciones.java
2  // Clase LibroCalificaciones que contiene una variable de instancia nombreDelCurso
3  // y métodos para establecer y obtener su valor.
4
5  public class LibroCalificaciones
6  {
7     private String nombreDelCurso; // nombre del curso para este LibroCalificaciones
8
9     // método para establecer el nombre del curso
10    public void establecerNombreDelCurso( String nombre )
11    {
12       nombreDelCurso = nombre; // almacena el nombre del curso
13    } // fin del método establecerNombreDelCurso
14
15    // método para obtener el nombre del curso
16    public String obtenerNombreDelCurso()
17    {
18       return nombreDelCurso;
19    } // fin del método obtenerNombreDelCurso
20
21    // muestra un mensaje de bienvenida al usuario de LibroCalificaciones
22    public void mostrarMensaje()
23    {
24       // esta instrucción llama a obtenerNombreDelCurso para obtener el
25       // nombre del curso que representa este LibroCalificaciones
26       System.out.printf( "Bienvenido al libro de calificaciones para\n%s!\n",
27          obtenerNombreDelCurso() );
28    } // fin del método mostrarMensaje
29 } // fin de la clase LibroCalificaciones
```

**Fig. 3.7** | Cómo crear un objeto `LibroCalificaciones` y pasar un objeto `String` a su método `mostrarMensaje`.

Por lo general, un instructor enseña más de un curso, cada uno con su propio nombre. La línea 7 declara que `nombreDelCurso` es una variable de tipo `String`. Como la variable se declara *en* el cuerpo de la clase, pero *fuera* de los cuerpos de los métodos de la misma (líneas 10 a la 13, 16 a la 19 y 22 a la 28), la línea 7 es una declaración para una *variable de instancia*. Cada instancia (es decir, objeto) de la clase `LibroCalificaciones` contiene una copia de cada variable de instancia. Por ejemplo, si hay dos objetos `LibroCalificaciones`, cada objeto tiene su propia copia de `nombreDelCurso`. Un beneficio de hacer de `nombreDelCurso` una variable de instancia es que todos los métodos de la clase (en este caso, `LibroCalificaciones`) pueden manipular cualquier variable de instancia que aparezca en la clase (en este caso, `nombreDelCurso`).

### *Los modificadores de acceso* `public` *y* `private`

La mayoría de las declaraciones de variables de instancia van precedidas por la palabra clave `private` (como en la línea 7). Al igual que `public`, la palabra clave **`private`** es un *modificador de acceso*. *Las variables o los métodos declarados con el modificador de acceso* `private` *son accesibles sólo para los métodos de la clase en la que se declaran*. Así, la variable `nombreDelCurso` sólo puede utilizarse en los méto-

www.FreeLibros.me

dos `establecerNombreDelCurso`, `obtenerNombreDelCurso` y `mostrarMensaje` de (cada objeto de) la clase `LibroCalificaciones`.

El proceso de declarar variables de instancia con el modificador de acceso `private` se conoce como **ocultamiento de datos**, u ocultamiento de información. Cuando un programa crea (instancia) un objeto de la clase `LibroCalificaciones`, la variable `nombreDelCurso` se *encapsula* (oculta) en el objeto, y sólo está accesible para los métodos de la clase de ese objeto. Esto evita que una clase en otra parte del programa modifique a `nombreDelCurso` por accidente. En la clase `LibroCalificaciones`, los métodos `establecerNombreDelCurso` y `obtenerNombreDelCurso` manipulan a la variable de instancia `nombreDelCurso`.

### Observación de ingeniería de software 3.2

*Es necesario colocar un modificador de acceso antes de cada declaración de un campo y de un método. Por lo general, las variables de instancia deben declararse como `private` y los métodos como `public`. (Es apropiado declarar ciertos métodos como `private`, si sólo van a estar accesibles para otros métodos de la clase).*

### Buena práctica de programación 3.1

*Preferimos listar los campos de una clase primero, para que, a medida que usted lea el código, pueda ver los nombres y tipos de las variables antes de usarlas en los métodos de la clase. Es posible listar los campos de la clase en cualquier parte de la misma, fuera de las declaraciones de sus métodos, pero si se esparcen por todo el código, éste será más difícil de leer.*

### *Los métodos* `establecerNombreDelCurso` *y* `obtenerNombreDelCurso`

El método `establecerNombreDelCurso` (líneas 10 a la 13) no devuelve datos cuando completa su tarea, por lo que su tipo de valor de retorno es `void`. El método recibe un parámetro (`nombre`), el cual representa el nombre del curso que se pasará al método como un argumento. La línea 12 asigna `nombre` a la variable de instancia `nombreDelCurso`.

El método `obtenerNombreDelCurso` (líneas 16 a la 19) devuelve un `nombreDelCurso` de un objeto `LibroCalificaciones` específico. Tiene una lista de parámetros vacía, por lo que no requiere información adicional para realizar su tarea. Este método especifica que devuelve un objeto `String`; a éste se le conoce como el tipo de valor de retorno del método. Cuando se hace una llamada a un método que especifica un tipo de valor de retorno distinto de `void` y completa su tarea, devuelve un *resultado* al método que lo llamó. Por ejemplo, cuando usted va a un cajero automático (ATM) y solicita el saldo de su cuenta, espera que el ATM le devuelva un valor que representa su saldo. De manera similar, cuando una instrucción llama al método `obtenerNombreDelCurso` en un objeto `LibroCalificaciones`, la instrucción espera recibir el nombre del curso de `LibroCalificaciones` (en este caso, un objeto `String`, como se especifica en el tipo de valor de retorno de la declaración del método).

La instrucción `return` en la línea 18 pasa el valor de la variable de instancia `nombreDelCurso` de vuelta a la instrucción que llama al método `obtenerNombreDelCurso`. Ahora considere la línea 27 del método `mostrarMensaje`, que llama al método `obtenerNombreDelCurso`. Al devolver el valor, la instrucción en las líneas 26 y 27 usa ese valor para imprimir el nombre del curso. De manera similar, si tiene un método cuadrado que devuelve el cuadrado de su argumento, es de esperarse que la instrucción

```
int resultado = cuadrado( 2 );
```

devuelva 4 del método `cuadrado` y asigne 4 a la variable `resultado`. Si tiene un método `maximo` que devuelve el mayor de tres argumentos enteros, es de esperarse que la siguiente instrucción

```
int mayor = maximo( 27, 114, 51 );
```

devuelva 114 del método `maximo` y asigne 114 a la variable `mayor`.

www.FreeLibros.me

Las instrucciones en las líneas 12 y 18 utilizan `nombreDelCurso`, *aun cuando esta variable no se declaró en ninguno de los métodos*. Podemos utilizar `nombreDelCurso` en los métodos de la clase `LibroCalificaciones`, ya que `nombreDelCurso` es una variable de instancia de la clase.

### *El método* `mostrarMensaje`

El método `mostrarMensaje` (líneas 22 a la 28) *no* devuelve datos cuando completa su tarea, por lo que su tipo de valor de retorno es `void`. El método *no* recibe parámetros, por lo que la lista de parámetros está vacía. Las líneas 26 y 27 imprimen un mensaje de bienvenida, que incluye el valor de la variable de instancia `nombreDelCurso`, el cual se devuelve mediante la llamada al método `obtenerNombreDelCurso` en la línea 27. Observe que un método de una clase (`mostrarMensaje` en este caso) puede llamar a otro método de la *misma* clase con sólo usar su nombre (`obtenerNombreDelCurso` en este caso).

### *La clase* `PruebaLibroCalificaciones` *que demuestra a la clase* `LibroCalificaciones`

La clase `PruebaLibroCalificaciones` (figura 3.8) crea un objeto de la clase `LibroCalificaciones` y demuestra el uso de sus métodos. La línea 14 crea un objeto `LibroCalificaciones` y lo asigna a la variable local `miLibroCalificaciones`, de tipo `LibroCalificaciones`. Las líneas 17-18 muestran el nombre inicial del curso mediante una llamada al método `obtenerNombreDelCurso` del objeto. La primera línea de la salida muestra el nombre "`null`". *A diferencia de las variables locales, que no se inicializan de manera automática, cada campo tiene un* ***valor inicial predeterminado****: un valor que Java proporciona cuando el programador no especifica el valor inicial del campo*. Por ende, no se requiere que los campos se inicialicen de manera explícita antes de usarlos en un programa, a menos que deban hacerlo con valores *distintos* de los predeterminados. El valor predeterminado para un campo de tipo `String` (como `nombreDelCurso` en este ejemplo) es `null`, de lo cual hablaremos con más detalle en la sección 3.5.

La línea 21 pide al usuario que escriba el nombre para el curso. La variable `String` local `elNombre` (declarada en la línea 22) se inicializa con el nombre del curso que escribió el usuario, el cual se devuelve mediante la llamada al método `nextLine` del objeto `Scanner` llamado `entrada`. La línea 23 llama al método `establecerNombreDelCurso` del objeto `miLibroCalificaciones` y provee `elNombre` como argumento para el método. Cuando se hace la llamada al método, el valor del argumento se asigna al parámetro `nombre` (línea 10, figura 3.7) del método `establecerNombreDelCurso` (líneas 10 a la 13, figura 3.7). Después, el valor del parámetro se asigna a la variable de instancia `nombreDelCurso` (línea 12, figura 3.7). La línea 24 (figura 3.8) salta una línea en la salida, y después la línea 27 llama al método `mostrarMensaje` del objeto `miLibroCalificaciones` para mostrar en pantalla el mensaje de bienvenida, que contiene el nombre del curso.

```
1  // Fig. 3.8: PruebaLibroCalificaciones.java
2  // Crea y manipula un objeto LibroCalificaciones.
3  import java.util.Scanner; // el programa usa la clase Scanner
4
5  public class PruebaLibroCalificaciones
6  {
7     // el método main empieza la ejecución del programa
8     public static void main( String[] args )
9     {
10       // crea un objeto Scanner para obtener la entrada de la ventana de comandos
11       Scanner entrada = new Scanner( System.in );
12
13       // crea un objeto LibroCalificaciones y lo asigna a miLibroCalificaciones
14       LibroCalificaciones miLibroCalificaciones = new LibroCalificaciones();
15
```

**Fig. 3.8** | Creación y manipulación de un objeto `LibroCalificaciones` (parte 1 de 2).

www.FreeLibros.me

```
16      // muestra el valor inicial de nombreDelCurso
17      System.out.printf( "El nombre inicial del curso es: %s\n\n",
18        miLibroCalificaciones.obtenerNombreDelCurso() );
19
20      // pide y lee el nombre del curso
21      System.out.println( "Escriba el nombre del curso:" );
22      String elNombre = entrada.nextLine(); // lee una línea de texto
23      miLibroCalificaciones.establecerNombreDelCurso( elNombre ); // establece el nombre
        del curso
24      System.out.println(); // imprime una línea en blanco
25
26      // muestra el mensaje de bienvenida después de especificar el nombre del curso
27      miLibroCalificaciones.mostrarMensaje();
28    } // fin de main
29  } // fin de la clase PruebaLibroCalificaciones
```

```
El nombre inicial del curso es: null

Escriba el nombre del curso:
CS101 Introduccion a la programacion en Java

Bienvenido al libro de calificaciones para
CS101 Introduccion a la programacion en Java!
```

**Fig. 3.8** | Creación y manipulación de un objeto `LibroCalificaciones` (parte 2 de 2).

### *Los métodos* establecer *y* obtener

Los campos `private` de una clase pueden manipularse *sólo* mediante los métodos de esa clase. Por lo tanto, un **cliente de un objeto** (es decir, cualquier clase que llame a los métodos del objeto) llama a los métodos `public` de la clase para manipular los campos `private` de un objeto de esa clase. Esto explica por qué las instrucciones en el método `main` (figura 3.8) llaman a los métodos `establecerNombreDelCurso`, `obtenerNombreDelCurso` y `mostrarMensaje` en un objeto `LibroCalificaciones`. A menudo, las clases proporcionan métodos `public` para permitir a los clientes de la clase *establecer* (asignar valores a) u *obtener* (obtener los valores de) variables de instancia `private`. Los nombres de estos métodos no necesitan empezar con *establecer* u *obtener*, pero esta convención de nomenclatura es muy recomendada en Java, y es requerida para ciertos componentes de software especiales de Java, conocidos como JavaBeans, que pueden simplificar la programación en muchos entornos de desarrollo integrados (IDE). El método que *establece* la variable de instancia `nombreDelCurso` en este ejemplo se llama `establecerNombreDelCurso`, y el método que *obtiene* su valor se llama `obtenerNombreDelCurso`.

### *Diagrama de clases de UML para la clase* `LibroCalificaciones` *con una variable de instancia, y métodos* establecer *y* obtener

La figura 3.9 contiene un diagrama de clases de UML actualizado para la versión de la clase `LibroCalificaciones` de la figura 3.7. Este diagrama modela la variable de instancia `nombreDelCurso` de la clase `LibroCalificaciones` como un atributo en el compartimiento intermedio de la clase. UML representa a las variables de instancia como atributos, listando el nombre del atributo, seguido de dos puntos y del tipo del atributo. El tipo de UML del atributo `nombreDelCurso` es `String`. La variable de instancia `nombreDelCurso` es `private` en Java, por lo que el diagrama de clases lista un modificador de acceso de signo menos (–) en frente del nombre del atributo correspondiente. La clase `LibroCalificaciones` contiene tres métodos `public`, por lo que el diagrama de clases lista tres operaciones en el tercer compartimiento. Recuerde que el signo más (+) antes de cada nombre de operación indica que ésta es `public`. La operación `establecerNombreDelCurso` tiene un parámetro `String` llamado `nombre`. UML indica el tipo de valor de retorno de una operación colocando dos puntos y el tipo de valor de retorno después de los paréntesis que le siguen al nombre de la operación. El método `obtenerNombreDelCurso` de la clase `LibroCalifi-`

www.FreeLibros.me

caciones (figura 3.7) tiene un tipo de valor de retorno `String` en Java, por lo que el diagrama de clases muestra un tipo de valor de retorno `String` en UML. Las operaciones `establecerNombreDelCurso` y `mostrarMensaje` *no* devuelven valores (es decir, devuelven `void` en Java), por lo que el diagrama de clases de UML *no* especifica un tipo de valor de retorno después de los paréntesis de estas operaciones.

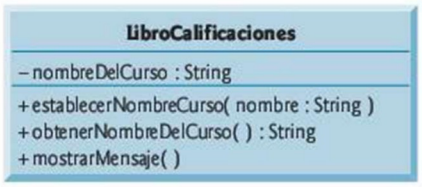



**Fig. 3.9** | Diagrama de clases de UML, en el que se indica que la clase `LibroCalificaciones` tiene un atributo privado `nombreDelCurso` de tipo `String` en UML, y tres operaciones públicas: `establecerNombreDelCurso` (con un parámetro nombre de tipo `String` de UML), `obtenerNombreDelCurso` (que devuelve el tipo `String` de UML) y `mostrarMensaje`.

## 3.5 Comparación entre tipos primitivos y tipos por referencia

Los tipos de datos en Java se dividen en dos categorías: tipos primitivos y **tipos por referencia**. Los tipos primitivos son `boolean`, `byte`, `char`, `short`, `int`, `long`, `float` y `double`. Todos los tipos no primitivos son tipos por referencia, por lo cual las clases, que especifican los tipos de objetos, son tipos por referencia.

Una variable de tipo primitivo puede almacenar sólo un *valor de su tipo declarado* a la vez. Por ejemplo, una variable `int` puede almacenar un número entero (como 7) a la vez. Cuando se le asigna otro valor, sustituye su valor inicial. Las variables de instancia de tipo primitivo se *inicializan de manera predeterminada*; las de los tipos `byte`, `char`, `short`, `int`, `long`, `float` y `double` se inicializan con 0, y las de tipo `boolean` se inicializan con `false`. Usted puede especificar su propio valor inicial para una variable de tipo primitivo al asignarle un valor en su declaración, como en

```
private int numeroDeEstudiantes = 10;
```

Recuerde que las variables locales *no* se inicializan de manera predeterminada.

**Tip para prevenir errores 3.1**
*Un intento de utilizar una variable local que no se haya inicializado produce un error de compilación.*

Los programas utilizan variables de tipo por referencia (que por lo general se llaman **referencias**) para almacenar las *ubicaciones* de los objetos en la memoria de la computadora. Se dice que dicha variable hace **referencia a un objeto** en el programa. Cada uno de los objetos a los que se hace referencia puede contener muchas variables de instancia. La línea 14 de la figura 3.8 crea un objeto de la clase `LibroCalificaciones`, y la variable `miLibroCalificaciones` contiene una referencia a ese objeto. *Las variables de instancia de tipo por referencia se inicializan de manera predeterminada con el valor* `null`: una palabra reservada que representa una "referencia a nada". Esto explica por qué la primera llamada a `obtenerNombreDelCurso` en la línea 18 de la figura 3.8 devolvía `null`; no se había establecido el valor de `nombreDelCurso`, por lo que se devolvía el valor inicial predeterminado `null`. En el apéndice C se muestra una lista completa de las palabras reservadas y las palabras clave.

Cuando usamos un objeto de otra clase, es obligatorio que una referencia a él **invoque** (es decir, llame) a sus métodos. En la aplicación de la figura 3.8, las instrucciones en el método `main` utilizan la

www.FreeLibros.me

variable miLibroCalificaciones para enviar mensajes al objeto LibroCalificaciones. Estos mensajes son llamadas a métodos (como establecerNombreDelCurso y obtenerNombreDelCurso) que permiten al programa interactuar con el objeto LibroCalificaciones. Por ejemplo, la instrucción en la línea 23 utiliza a miLibroCalificaciones para enviar el mensaje establecerNombreDelCurso al objeto LibroCalificaciones. El mensaje incluye el argumento que requiere establecerNombreDelCurso para realizar su tarea. El objeto LibroCalificaciones utiliza esta información para establecer la variable de instancia nombreDelCurso. Las variables de tipo primitivo no hacen referencias a objetos, por lo que dichas variables no pueden utilizarse para invocar métodos.

**Observación de ingeniería de software 3.3**

*El tipo declarado de una variable (por ejemplo, int, double o LibroCalificaciones) indica si la variable es de tipo primitivo o por referencia. Si el tipo de una variable no es uno de los ocho tipos primitivos, entonces es un tipo por referencia.*

## 3.6 Inicialización de objetos mediante constructores

Como mencionamos en la sección 3.4, cuando se crea un objeto de la clase LibroCalificaciones (figura 3.7), su variable de instancia nombreCurso se inicializa con null de manera predeterminada. ¿Qué pasa si desea proporcionar el nombre de un curso a la hora de crear un objeto LibroCalificaciones? Cada clase que usted declare puede proporcionar un método especial llamado constructor, el cual puede utilizarse para inicializar un objeto de una clase al momento de crearlo. De hecho, Java *requiere* una llamada al constructor para *cada* objeto que se crea. La palabra clave new solicita memoria del sistema para almacenar un objeto, y después llama al constructor de la clase correspondiente para inicializar el objeto. La llamada se indica mediante el nombre de la clase, seguido de paréntesis. Un constructor *debe* tener el *mismo nombre* que la clase. Por ejemplo, la línea 14 de la figura 3.8 primero utiliza new para crear un objeto LibroCalificaciones. Los paréntesis vacíos después de "new LibroCalificaciones" indican una llamada sin argumentos al constructor de la clase. De manera predeterminada, el compilador proporciona un **constructor predeterminado** *sin parámetros*, en cualquier clase que no incluya un constructor en forma explícita. Cuando una clase sólo tiene el constructor predeterminado, sus variables de instancia se inicializan con sus *valores predeterminados*.

Cuando usted declara una clase, puede proporcionar su propio constructor para especificar una inicialización personalizada para los objetos de su clase. Por ejemplo, tal vez quiera especificar el nombre de un curso para un objeto LibroCalificaciones al momento de crear este objeto, como en

```
LibroCalificaciones miLibroCalificaciones =
   new LibroCalificaciones( "CS101 Introduccion a la programacion en Java" );
```

En este caso, el argumento "CS101 Introduccion a la programacion en Java" se pasa al constructor del objeto LibroCalificaciones y se utiliza para inicializar el nombreDelCurso. La instrucción anterior requiere que la clase proporcione un constructor con un parámetro String. La figura 3.10 contiene una clase LibroCalificaciones modificada con dicho constructor.

```
1  // Fig. 3.10: LibroCalificaciones.java
2  // La clase LibroCalificaciones con un constructor para inicializar el nombre del curso.
3
4  public class LibroCalificaciones
5  {
6     private String nombreDelCurso; // nombre del curso para este LibroCalificaciones
7
```

**Fig. 3.10** | La clase LibroCalificaciones con un constructor para inicializar el nombre del curso (parte 1 de 2).

www.FreeLibros.me

```
8  // el constructor inicializa nombreDelCurso con un argumento String
9  public LibroCalificaciones( String nombre ) // el nombre del constructor es el nombre
                                               de la clase
10 {
11    nombreDelCurso = nombre; // inicializa nombreDelCurso
12 } // fin del constructor
13
14 // método para establecer el nombre del curso
15 public void establecerNombreDelCurso( String nombre )
16 {
17    nombreDelCurso = nombre; // almacena el nombre del curso
18 } // fin del método establecerNombreDelCurso
19
20 // método para obtener el nombre del curso
21 public String obtenerNombreDelCurso()
22 {
23    return nombreDelCurso;
24 } // fin del método obtenerNombreDelCurso
25
26 // muestra un mensaje de bienvenida al usuario de LibroCalificaciones
27 public void mostrarMensaje()
28 {
29    // esta instrucción llama a obtenerNombreDelCurso para obtener el
30    // nombre del curso que este LibroCalificaciones representa
31    System.out.printf( "Bienvenido al Libro de calificaciones para\n%s!\n",
32       obtenerNombreDelCurso() );
33 } // fin del método mostrarMensaje
34 } // fin de la clase LibroCalificaciones
```

**Fig. 3.10** | La clase `LibroCalificaciones` con un constructor para inicializar el nombre del curso (parte 2 de 2).

Las líneas 9 a la 12 declaran el constructor de `LibroCalificaciones`. Al igual que un método, un constructor especifica en su lista de parámetros los datos que requiere para realizar su tarea. Cuando usted crea un nuevo objeto (como haremos en la figura 3.11), estos datos se colocan en los *paréntesis que van después del nombre de la clase*. La línea 9 de la figura 3.10 indica que el constructor tiene un parámetro `String` llamado `nombre`. El `nombre` que se pasa al constructor se asigna a la variable de instancia `nombreDelCurso`en la línea 11.

La figura 3.11 inicializa los objetos `LibroCalificaciones` mediante el constructor. Las líneas 11 y 12 crean e inicializan el objeto `libroCalificaciones1` de `LibroCalificaciones`. El constructor de la clase `LibroCalificaciones` se llama con el argumento "`CS101 Introduccion a la programacion en Java`" para inicializar el nombre del curso. La expresión de creación de la instancia de la clase en las líneas 11 y 12 devuelve una referencia al nuevo objeto, el cual se asigna a la variable `libroCalificaciones1`. Las líneas 13 y 14 repiten este proceso, pero esta vez se pasa el argumento "`CS102 Estructuras de datos en Java`" para inicializar el nombre del curso para `libroCalificaciones2`. Las líneas 17 a la 20 utilizan el método `obtenerNombreDelCurso` de cada objeto para obtener los nombres de los cursos y mostrar que se inicializaron en el momento en el que se crearon los objetos. La salida confirma que cada objeto `LibroCalificaciones` mantiene su propia copia de la variable de instancia `nombreDelCurso`.

Una importante diferencia entre los constructores y los métodos es que los constructores no pueden devolver valores, por lo cual no pueden especificar un tipo de valor de retorno (ni siquiera `void`). Por lo general, los constructores se declaran como `public`. Si una clase no incluye un constructor, las variables de instancia de esa clase se inicializan con sus valores predeterminados. *Si un programador declara uno o más constructores para una clase, el compilador de Java no creará un constructor predeterminado para esa clase*. Por lo tanto, ya no podemos crear un objeto `LibroCalificaciones` con `new LibroCalificaciones()` como hicimos en los ejemplos anteriores.

www.FreeLibros.me

```
1  // Fig. 3.11: PruebaLibroCalificaciones.java
2  // El constructor de LibroCalificaciones se utiliza para especificar el
3  // nombre del curso cada vez que se crea cada objeto LibroCalificaciones.
4
5  public class PruebaLibroCalificaciones
6  {
7     // el método main empieza la ejecución del programa
8     public static void main( String[] args )
9     {
10       // crea objeto LibroCalificaciones
11       LibroCalificaciones libroCalificaciones1 = new LibroCalificaciones(
12          "CS101 Introduccion a la programacion en Java" );
13       LibroCalificaciones libroCalificaciones2 = new LibroCalificaciones(
14          "CS102 Estructuras de datos en Java" );
15
16       // muestra el valor inicial de nombreDelCurso para cada LibroCalificaciones
17       System.out.printf( "El nombre del curso de libroCalificaciones1 es: %s\n",
18          libroCalificaciones1.obtenerNombreDelCurso() );
19       System.out.printf( "El nombre del curso de libroCalificaciones2 es: %s\n",
20          libroCalificaciones2.obtenerNombreDelCurso() );
21    } // fin de main
22 } // fin de la clase PruebaLibroCalificaciones
```

```
El nombre del curso de libroCalificaciones1 es: CS101 Introduccion a la programacion en Java
El nombre del curso de libroCalificaciones2 es: CS102 Estructuras de datos en Java
```

**Fig. 3.11** | El constructor de `LibroCalificaciones` se utiliza para especificar el nombre del curso cada vez que se crea un objeto `LibroCalificaciones`.

**Observación de ingeniería de software 3.4**

*A menos que sea aceptable la inicialización predeterminada de las variables de instancia de su clase, deberá proporcionar un constructor para asegurarse que se inicialicen en forma apropiada con valores significativos a la hora de crear cada nuevo objeto.*

### *Agregar el constructor al diagrama de clases de UML de la clase* `LibroCalificaciones`

El diagrama de clases de UML de la figura 3.12 modela la clase `LibroCalificaciones` de la figura 3.10, la cual tiene un constructor con un parámetro llamado `nombre`, de tipo `String`. Al igual que las operaciones, en un diagrama de clases, UML modela a los constructores en el tercer compartimiento de una clase.

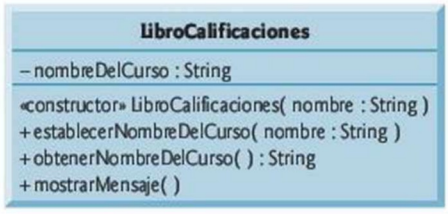



**Fig. 3.12** | Diagrama de clases de UML, en el cual se indica que la clase `LibroCalificaciones` tiene un constructor con un parámetro nombre del tipo `String` de UML.

www.FreeLibros.me

Para diferenciar a un constructor de las operaciones de una clase, UML requiere que se coloque la palabra "constructor" entre los signos « y » antes del nombre del constructor. Es *costumbre* listar los constructores *antes* de otras operaciones en el tercer compartimiento.

### *Constructores con varios parámetros*

Algunas veces es conveniente inicializar objetos con varios elementos de datos. En el ejercicio 3.11, le pedimos que almacene el nombre del curso y del instructor en un objeto `LibroCalificaciones`. En este caso, se modifica el constructor de `LibroCalificaciones` para que reciba dos parámetros `String`, como en

```
public LibroCalificaciones( String nombreDelCurso, String nombreDelInstructor )
```

y llamamos al constructor de `LibroCalificaciones` de la siguiente manera:

```
LibroCalificaciones libroCalificaciones = new LibroCalificaciones(
   "CS101 Introduccion a la programacion en Java", "Sue Green" );
```

## 3.7 Los números de punto flotante y el tipo double

Ahora dejaremos por un momento nuestro caso de estudio con la clase `LibroCalificaciones` para declarar una clase llamada `Cuenta`, la cual mantiene el saldo de una cuenta bancaria. La mayoría de los saldos de las cuentas no son números enteros (por ejemplo, 0, –22 y 1024). Por esta razón, la clase `Cuenta` representa el saldo de las cuentas como un **número de punto flotante** (es decir, un número con un punto decimal, como 7.33, 0.0975 o 1000.12345). Java cuenta con dos tipos primitivos para almacenar números de punto flotante en la memoria: `float` y `double`. La principal diferencia entre ellos es que las variables tipo `double` pueden almacenar números con mayor magnitud y detalle (más dígitos a la derecha del punto decimal; lo que también se conoce como **precisión** del número) que las variables `float`.

### *Precisión de los números de punto flotante y requerimientos de memoria*

Las variables de tipo `float` representan **números de punto flotante de precisión simple** y pueden representar hasta *siete dígitos significativos*. Las variables de tipo `double` representan **números de punto flotante de precisión doble**. Éstos requieren el doble de memoria que las variables `float` y proporcionan 15 *dígitos significativos*; aproximadamente el doble de precisión de las variables `float`. Para el rango de valores requeridos por la mayoría de los programas, debe bastar con las variables de tipo `float`, pero podemos utilizar variables tipo `double` para "ir a la segura". En algunas aplicaciones, incluso hasta las variables de tipo `double` serán inadecuadas. La mayoría de los programadores representan los números de punto flotante con el tipo `double`. De hecho, Java trata a todos los números de punto flotante que escribimos en el código fuente de un programa (como 7.33 y 0.0975) como valores `double` de manera predeterminada. Dichos valores en el código fuente se conocen como **literales de punto flotante**. En el apéndice D, Tipos primitivos, puede consultar los rangos de los valores para los tipos `float` y `double`.

Aunque los números de punto flotante no son siempre 100% precisos, tienen numerosas aplicaciones. Por ejemplo, cuando hablamos de una temperatura corporal "normal" de 36.8, no necesitamos una precisión con un número extenso de dígitos. Cuando leemos la temperatura en un termómetro como 36.8, en realidad podría ser 36.7999473210643. Si consideramos a este número simplemente como 36.8, está bien para la mayoría de las aplicaciones en las que se trabaja con las temperaturas corporales. Debido a la naturaleza imprecisa de los números de punto flotante, se prefiere el tipo `double` al tipo `float` ya que las variables `double` pueden representar números de punto flotante con más precisión. Por esta razón, utilizaremos el tipo `double` a lo largo de este libro. Para los números precisos de punto flotante, Java cuenta con la clase `BigDecimal` (paquete `java.math`).

Los números de punto flotante también surgen como resultado de la división. En la aritmética convencional, cuando dividimos 10 entre 3 el resultado es 3.3333333..., y la secuencia de números 3

www.FreeLibros.me

se repite en forma indefinida. La computadora asigna sólo una cantidad fija de espacio para almacenar un valor de este tipo, por lo que, sin duda, el valor de punto flotante almacenado sólo puede ser una aproximación.

### *La clase* Cuenta *con una variable de instancia de tipo* double

Nuestra siguiente aplicación (figuras 3.13 y 3.14) contiene una clase llamada Cuenta (figura 3.13), la cual mantiene el saldo de una cuenta bancaria. Un banco ordinario da servicio a muchas cuentas, cada una con su propio saldo, por lo que la línea 7 declara una variable de instancia, de tipo double, llamada saldo. La variable saldo es una variable de instancia, ya que está declarada en el cuerpo de la clase pero fuera de las declaraciones de los métodos de la misma (líneas 10 a la 16, 19 a la 22 y 25 a la 28). Cada instancia (objeto) de la clase Cuenta contiene su propia copia de saldo.

```
1  // Fig. 3.13: Cuenta.java
2  // La clase Cuenta con un constructor para validar e
3  // inicializar la variable de instancia saldo de tipo double.
4
5  public class Cuenta
6  {
7     private double saldo; // variable de instancia que almacena el saldo
8
9     // constructor
10    public Cuenta( double saldoInicial )
11    {
12       // valida que saldoInicial sea mayor que 0.0;
13       // si no lo es, saldo se inicializa con el valor predeterminado 0.0
14       if ( saldoInicial > 0.0 )
15          saldo = saldoInicial;
16    } // fin del constructor de Cuenta
17
18    // abona (suma) un monto a la cuenta
19    public void abonar( double monto )
20    {
21       saldo = saldo + monto; // suma el monto al saldo
22    } // fin del método abonar
23
24    // devuelve el saldo de la cuenta
25    public double obtenerSaldo()
26    {
27       return saldo; // proporciona el valor de saldo al método que hizo la llamada
28    } // fin del método obtenerSaldo
29 } // fin de la clase Cuenta
```

**Fig. 3.13** | La clase Cuenta con un constructor para validar e inicializar la variable de instancia saldo de tipo double.

La clase tiene un constructor y dos métodos. Debido a que es común que alguien abra una cuenta para depositar dinero de inmediato, el constructor (líneas 10 a la 16) recibe un parámetro llamado saldoInicial de tipo double, el cual representa el *saldo inicial* de la cuenta. Las líneas 14 y 15 aseguran que saldoInicial sea mayor que 0.0. De ser así, el valor de saldoInicial se asigna a la variable de instancia saldo. En caso contrario, saldo permanece en 0.0, su valor inicial predeterminado.

El método abonar (líneas 19 a la 22) *no* devuelve datos cuando completa su tarea, por lo que su tipo de valor de retorno es void. El método recibe un parámetro llamado monto: un valor double que

www.FreeLibros.me

se sumará al saldo. La línea 21 suma `monto` al valor actual de `saldo`, y después asigna el resultado a `saldo` (con lo cual se sustituye el monto del saldo anterior).

El método `obtenerSaldo` (líneas 25 a la 28) permite a los clientes de la clase (otras clases que utilicen esta clase) obtener el valor del `saldo` de un objeto `Cuenta` específico. El método especifica el tipo de valor de retorno `double` y una lista de parámetros vacía.

Observe una vez más que las instrucciones en las líneas 15, 21 y 27 utilizan la variable de instancia `saldo`, aún y cuando *no* se declaró en ninguno de los métodos. Podemos usar saldo en estos métodos, ya que es una variable de instancia de la clase.

### *La clase* **PruebaCuenta** *que utiliza a la clase* **Cuenta**

La clase `PruebaCuenta` (figura 3.14) crea dos objetos `Cuenta` (líneas 10 y 11) y los inicializa con `50.00` y `-7.53`, respectivamente. Las líneas 14 a la 17 imprimen el saldo en cada objeto `Cuenta` mediante una llamada al método `obtenerSaldo` de `Cuenta`. Cuando se hace una llamada al método `obtenerSaldo` para `cuenta1` en la línea 15, se devuelve el valor del saldo de `cuenta1` de la línea 27 en la figura 3.13, y se imprime en pantalla mediante la instrucción `System.out.printf` (figura 3.14, líneas 14 y 15). De manera similar, cuando se hace la llamada al método `obtenerSaldo` para `cuenta2` en la línea 17, se devuelve el valor del saldo de `cuenta2` de la línea 27 en la figura 3.13, y se imprime en pantalla mediante la instrucción `System.out.printf` (figura 3.14, líneas 16 y 17). El saldo de `cuenta2` es `0.00`, ya que el constructor se aseguró de que la cuenta *no* pudiera empezar con un saldo negativo. El valor se imprime en pantalla mediante `printf`, con el especificador de formato `%.2f`. El especificador de formato `%f` se utiliza para imprimir valores de tipo `float` o `double`. El `.2` entre `%` y `f` representa el número de lugares decimales (2) que deben imprimirse a la derecha del punto decimal en el número de punto flotante; a esto también se le conoce como la **precisión** del número. Cualquier valor de punto flotante que se imprima con `%.2f` se redondeará a la posición de las centenas; por ejemplo, 123.457 se redondearía a 123.46, 27.333 se redondearía a 27.33 y 123.455 se redondearía a 123.46.

```
1  // Fig. 3.14: PruebaCuenta.java
2  // Entrada y salida de números de punto flotante con objetos Cuenta.
3  import java.util.Scanner;
4
5  public class PruebaCuenta
6  {
7     // el método main empieza la ejecución de la aplicación de Java
8     public static void main( String[] args )
9     {
10       Cuenta cuenta1 = new Cuenta( 50.00 ); // crea objeto Cuenta
11       Cuenta cuenta2 = new Cuenta( -7.53 ); // crea objeto Cuenta
12
13       // muestra el saldo inicial de cada objeto
14       System.out.printf( "Saldo de cuenta1: $%.2f\n",
15          cuenta1.obtenerSaldo() );
16       System.out.printf( "Saldo de cuenta2: $%.2f\n\n",
17          cuenta2.obtenerSaldo() );
18
19    // crea objeto Scanner para obtener la entrada de la ventana de comandos
20    Scanner entrada = new Scanner( System.in );
21    double montoDeposito; // deposita el monto escrito por el usuario
```

**Fig. 3.14** | Entrada y salida de números de punto flotante con objetos `Cuenta` (parte 1 de 2).

www.FreeLibros.me

```
22
23       System.out.print( "Escriba el monto a depositar para cuenta1: " ); // indicador
24       montoDeposito = entrada.nextDouble(); // obtiene entrada del usuario
25       System.out.printf( "\nsumando %.2f al saldo de cuenta1\n\n",
26          montoDeposito );
27       cuenta1.abonar( montoDeposito ); // suma al saldo de cuenta1
28
29       // muestra los saldos
30       System.out.printf( "Saldo de cuenta1: $%.2f\n",
31          cuenta1.obtenerSaldo() );
32       System.out.printf( "Saldo de cuenta2: $%.2f\n\n",
33          cuenta2.obtenerSaldo() );
34
35       System.out.print( "Escriba el monto a depositar para cuenta2: " ); // indicador
36       montoDeposito = entrada.nextDouble(); // obtiene entrada del usuario
37       System.out.printf( "\nsumando %.2f al saldo de cuenta2\n\n",
38          montoDeposito );
39       cuenta2.abonar( montoDeposito ); // suma al saldo de cuenta2
40
41       // muestra los saldos
42       System.out.printf( "Saldo de cuenta1: $%.2f\n",
43          cuenta1.obtenerSaldo() );
44       System.out.printf( "Saldo de cuenta2: $%.2f\n",
45          cuenta2.obtenerSaldo() );
46    } // fin de main
47 } // fin de la clase PruebaCuenta
```

```
Saldo de cuenta1: $50.00
Saldo de cuenta2: $0.00

Escriba el monto a depositar para cuenta1: 25.53

sumando 25.53 al saldo de cuenta1

Saldo de cuenta1: $75.53
Saldo de cuenta2: $0.00

Escriba el monto a depositar para cuenta2: 123.45

sumando 123.45 al saldo de cuenta2

Saldo de cuenta1: $75.53
Saldo de cuenta2: $123.45
```

**Fig. 3.14** | Entrada y salida de números de punto flotante con objetos `Cuenta` (parte 2 de 2).

La línea 21 declara la variable local `montoDeposito` para almacenar cada monto de depósito introducido por el usuario. A diferencia de la variable de instancia `saldo` en la clase `Cuenta`, la variable local `montoDeposito` en `main` *no* se inicializa con 0.0 de manera predeterminada. Sin embargo, esta variable no necesita inicializarse aquí, ya que su valor se determinará con base a la entrada del usuario.

La línea 23 pide al usuario que escriba un monto a depositar para `cuenta1`. La línea 24 obtiene la entrada del usuario, llamando al método `nextDouble` del objeto `Scanner` llamado entrada, el cual devuelve un valor `double` introducido por el usuario. Las líneas 25 y 26 muestran el monto del depósito.

www.FreeLibros.me

La línea 27 llama al método abonar del objeto cuenta1 y le suministra montoDeposito como argumento. Cuando se hace la llamada al método, el valor del argumento se asigna al parámetro monto (línea 19 de la figura 3.13) del método abonar (líneas 19 a la 22 de la figura 3.13); después el método abonar suma ese valor al saldo (línea 21 de la figura 3.13). Las líneas 30 a la 33 (figura 3.14) imprimen en pantalla los saldos de ambos objetos Cuenta otra vez, para mostrar que sólo se modificó el saldo de cuenta1.

La línea 35 pide al usuario que escriba un monto a depositar para cuenta2. La línea 36 obtiene la entrada del usuario, para lo cual invoca al método nextDouble del objeto Scanner llamado entrada. Las líneas 37 y 38 muestran el monto del depósito. La línea 39 llama al método abonar del objeto cuenta2 y le suministra montoDeposito como argumento; después, el método abonar suma ese valor al saldo. Por último, las líneas 42 a la 45 imprimen en pantalla los saldos de ambos objetos Cuenta otra vez, para mostrar que sólo se modificó el saldo de cuenta2.

### *Diagrama de clases de UML para la clase* Cuenta

El diagrama de clases de UML en la figura 3.15 modela la clase Cuenta de la figura 3.13. El diagrama modela el atributo private llamado saldo con el tipo Double de UML, para que corresponda a la variable de instancia saldo de la clase, que tiene el tipo double de Java. Modela el constructor de la clase Cuenta con un parámetro saldoInicial del tipo Double de UML en el tercer compartimiento de la clase. Los dos métodos public de la clase se modelan como operaciones en el tercer compartimiento también. El diagrama también modela la operación abonar con un parámetro monto de tipo Double de UML (ya que el método correspondiente tiene un parámetro monto de tipo double en Java) y la operación obtenerSaldo con un tipo de valor de retorno Double (ya que el método correspondiente en Java devuelve un valor double).

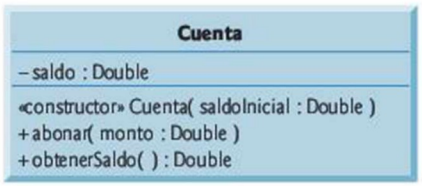



**Fig. 3.15** | Diagrama de clases de UML, el cual indica que la clase Cuenta tiene un atributo private llamado saldo, con el tipo Double de UML, un constructor (con un parámetro de tipo Double de UML) y dos operaciones public: abonar (con un parámetro monto de tipo Double de UML) y obtenerSaldo (devuelve el tipo Double de UML).

## 3.8 (Opcional) Caso de estudio de GUI y gráficos: uso de cuadros de diálogo

Este caso de estudio opcional está diseñado para aquellos que desean empezar a conocer las poderosas herramientas de Java para crear interfaces gráficas de usuario (GUI) y gráficos antes de los principales debates de estos temas en el capítulo 14 (en el sitio Web del libro), Componentes de la GUI: Parte 1, el capítulo 15 (también en el sitio Web), Gráficos y Java 2D, y el capítulo 25, Componentes de la GUI: Parte 2 (en inglés, en el sitio Web).

El caso de estudio de GUI y gráficos aparece en 10 secciones breves (vea la figura 3.16). Cada sección introduce unos cuantos conceptos básicos y proporciona ejemplos con capturas de pantalla que muestran interacciones de ejemplo y resultados. En las primeras secciones, creará sus primeras aplicaciones gráficas. En las secciones posteriores, utilizará los conceptos de programación orientada a obje-

www.FreeLibros.me

tos para crear una aplicación que dibuja una variedad de figuras. Cuando presentemos de manera formal las GUI en el capítulo 14, utilizaremos el ratón para elegir con exactitud qué figuras dibujar y en dónde. En el capítulo 15, agregaremos las herramientas de la API de gráficos en 2D de Java para dibujar las figuras con distintos grosores de línea y rellenos. Esperamos que este ejemplo práctico le sea informativo y divertido.

| Ubicación | Título – Ejercicio(s) |
|---|---|
| Sección 3.8 | Uso de cuadros de diálogo: entrada y salida básica con cuadros de diálogo |
| Sección 4.14 | Creación de dibujos simples: mostrar y dibujar líneas en la pantalla |
| Sección 5.10 | Dibujo de rectángulos y óvalos: uso de figuras para representar datos |
| Sección 6.13 | Colores y figuras rellenas: dibujar un tiro al blanco y gráficos aleatorios |
| Sección 7.15 | Dibujo de arcos: dibujar espirales con arcos |
| Sección 8.16 | Uso de objetos con gráficos: almacenar figuras como objetos |
| Sección 9.8 | Mostrar texto e imágenes usando etiquetas: proporcionar información de estado |
| Sección 10.8 | Realizar dibujos usando polimorfismo: identificar las similitudes entre figuras |
| Ejercicio 14.17 | Caso de estudio de GUI y gráficos: expansión de la interfaz |
| Ejercicio 15.31 | Caso de estudio de GUI y gráficos: Agregar Java 2D |

**Fig. 3.16** | Resumen del caso de estudio de GUI y gráficos en cada capítulo.

### *Cómo mostrar texto en un cuadro de diálogo*

Los programas que hemos presentado hasta ahora muestran su salida en la ventana de comandos. Muchas aplicaciones utilizan ventanas, o **cuadros de diálogo** (también llamados **diálogos**) para mostrar la salida. Por ejemplo, los navegadores Web como Firefox, Internet Explorer, Chrome y Safari muestran las páginas Web en sus propias ventanas. Los programas de correo electrónico le permiten escribir y leer mensajes en una ventana. Por lo general, los cuadros de diálogo son ventanas en las que los programas muestran mensajes importantes a los usuarios. La clase `JOptionPane` cuenta con cuadros de diálogo prefabricados, los cuales permiten a los programas mostrar ventanas que contengan mensajes; a dichas ventanas se les conoce como **diálogos de mensaje**. La figura 3.17 muestra el objeto String `"Bienvenido\na\nJava"` en un diálogo de mensaje.

```
1  // Fig. 3.17: Dialogo1.java
2  // Uso de JOptionPane para imprimir varias líneas en un cuadro de diálogo.
3  import javax.swing.JOptionPane; // importa la clase JOptionPane
4
5  public class Dialogo1
6  {
7     public static void main( String[] args )
8     {
9        // muestra un cuadro de diálogo con un mensaje
10       JOptionPane.showMessageDialog( null, "Bienvenido\na\nJava" );
11    } // fin de main
12 } // fin de la clase Dialogo1
```

**Fig. 3.17** | Uso de `JOptionPane` para mostrar varias líneas en un cuadro de diálogo (parte 1 de 2).

www.FreeLibros.me


**Fig. 3.17** | Uso de `JOptionPane` para mostrar varias líneas en un cuadro de diálogo (parte 2 de 2).

La línea 3 indica que el programa utiliza la clase `JOptionPane` del paquete `javax.swing`. El cual contiene muchas clases que le ayudan a crear **interfaces gráficas de usuario (GUI)**. Los **componentes de la GUI** facilitan la entrada de datos al usuario del programa, y la presentación de los datos de salida. La línea 10 llama al método `showMessageDialog` de `JOptionPane` para mostrar un cuadro de diálogo que contiene un mensaje. El método requiere dos argumentos. El primero ayuda a Java a determinar en dónde colocar el cuadro de diálogo. Por lo general, un diálogo se muestra desde una aplicación de GUI con su propia ventana. El primer argumento hace referencia a esa ventana (conocida como ventana padre) y hace que el diálogo aparezca centrado sobre la ventana de la aplicación. Si el primer argumento es `null`, el cuadro de diálogo aparece en el centro de la pantalla de la computadora. El segundo argumento es el objeto `String` a mostrar en el cuadro de diálogo.

### *Introducción de los métodos* `static`

El método `showMessageDialog` de la clase `JOptionPane` es lo que llamamos un **método `static`**. A menudo, dichos métodos definen las tareas que se utilizan con frecuencia. Por ejemplo, muchos programas muestran cuadros de diálogo, y el código para hacer esto es el mismo siempre. En vez de que usted tenga que "reinventar la rueda" y crear código para realizar esta tarea, los diseñadores de la clase `JOptionPane` declararon un método `static` que realiza esta tarea por usted. La llamada a un método `static` se realiza mediante el uso del nombre de su clase, seguido de un punto (`.`) y del nombre del método, como en

```
NombreClase.nombreMétodo( argumentos )
```

Observe que *no* tiene que crear un objeto de la clase `JOptionPane` para usar su método `static` llamado `showMessageDialog`. En el capítulo 6 analizaremos los métodos `static` con más detalle.

### *Introducir texto en un cuadro de diálogo*

La aplicación de la figura 3.18 utiliza otro cuadro de diálogo `JOptionPane` predefinido, conocido como **diálogo de entrada**, el cual permite al usuario introducir datos en un programa. Éste pide el nombre del usuario, y responde con un diálogo de mensaje que contiene un saludo y el nombre introducido por el usuario.

Las líneas 10 y 11 utilizan el método **`showInputDialog`** de `JOptionPane` para mostrar un diálogo de entrada que contiene un indicador y un campo (conocido como **campo de texto**), en donde el usuario puede escribir texto. El argumento del método `showInputDialog` es el indicador que muestra lo que el usuario debe escribir. El usuario escribe caracteres en el campo de texto, y después hace clic en el botón **Aceptar** u oprime la tecla *Intro* para devolver el objeto `String` al programa. El método `showInputDialog` (línea 11) devuelve un objeto `String` que contiene los caracteres escritos por el usuario. Almacenamos el objeto `String` en la variable `nombre` (línea 10). [*Nota:* si oprime el botón **Cancelar** en el cuadro de diálogo u oprime *Esc*, el método devuelve `null` y el programa muestra la palabra clave "null" como el nombre].

Las líneas 14 y 15 utilizan el método `static String` llamado **`format`** para devolver un objeto `String` que contiene un saludo con el nombre del usuario. El método `format` es similar al método `System.out.printf`, excepto que `format` devuelve el objeto `String` con formato, en vez de mostrarlo en una ventana de comandos. La línea 18 muestra el saludo en un cuadro de diálogo de mensaje, como hicimos en la figura 3.17.

www.FreeLibros.me

```
1  // Fig. 3.18: DialogoNombre.java
2  // Entrada básica con un cuadro de diálogo.
3  import javax.swing.JOptionPane;
4
5  public class DialogoNombre
6  {
7     public static void main( String[] args )
8     {
9        // pide al usuario que escriba su nombre
10       String nombre =
11          JOptionPane.showInputDialog( "Cual es su nombre?" );
12
13       // crea el mensaje
14       String mensaje =
15          String.format( "Bienvenido, %s, a la programacion en Java!", nombre );
16
17       // muestra el mensaje para dar la bienvenida al usuario por su nombre
18       JOptionPane.showMessageDialog( null, mensaje );
19    } // fin de main
20 } // fin de la clase DialogoNombre
```

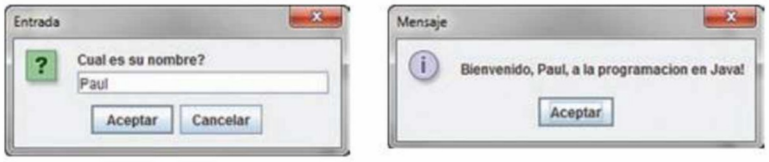



**Fig. 3.18** | Cómo obtener la entrada del usuario mediante un cuadro de diálogo.

### *Ejercicio del ejemplo práctico de GUI y gráficos*

**3.1** Modifique el programa de suma en la figura 2.7 para usar la entrada y salida con base en el cuadro de diálogo con los métodos de la clase `JOptionPane`. Como el método `showInputDialog` devuelve un objeto `String`, debe convertir el objeto `String` que introduce el usuario a un `int` para usarlo en los cálculos. El método `static parseInt` de la clase `Integer` recibe un argumento `String` que representa un entero (es decir, el resultado de `JOptionPane.showInputDialog`) y devuelve el valor completo como un número `int`. El método `parseInt` es un método `static` de la clase `Integer` (del paquete `java.lang`). Si el objeto `String` no contiene un entero válido, el programa terminará con un error.

## 3.9 Conclusión

En este capítulo aprendió a declarar variables de instancia de una clase para mantener los datos de cada objeto, y cómo declarar métodos que operen sobre esos datos. Aprendió cómo llamar a un método para decirle que realice su tarea y cómo pasar información a los métodos en forma de argumentos. Vio la diferencia entre una variable local de un método y una variable de instancia de una clase, y que sólo las variables de instancia se inicializan en forma automática. También aprendió a utilizar el constructor de una clase para especificar los valores iniciales para las variables de instancia de un objeto. A lo largo del capítulo, vio cómo puede usarse UML para crear diagramas de clases que modelen los constructores, métodos y atributos de las clases. Por último, aprendió acerca de los números de punto flotante: cómo almacenarlos con variables del tipo primitivo `double`, cómo recibirlos en forma de datos de entrada

www.FreeLibros.me

mediante un objeto `Scanner` y cómo darles formato con `printf` y el especificador de formato `%f` para fines de visualización. En el siguiente capítulo empezaremos nuestra introducción a las instrucciones de control, las cuales especifican el orden en el que se realizan las acciones de un programa. Utilizará estas instrucciones en sus métodos para especificar cómo deben realizar sus tareas.

## Resumen

### *Sección 3.2 Declaración de una clase con un método e instanciamiento de un objeto de una clase*

- Cada declaración de clase que empieza con el modificador de acceso `public` (pág. 72) debe almacenarse en un archivo que tenga exactamente el mismo nombre que la clase, y que termine con la extensión de nombre de archivo `.java`.
- Cada declaración de clase contiene la palabra clave `class`, seguida inmediatamente por el nombre de la clase.
- La declaración de un método que empieza con la palabra clave `public` indica que a ese método lo pueden llamar otras clases declaradas fuera de la declaración de esa clase.
- La palabra clave `void` indica que un método realizará una tarea, pero no devolverá información cuando la termine.
- Por convención, los nombres de los métodos empiezan con la primera letra en minúscula, y todas las palabras subsiguientes en el nombre empiezan con la primera letra en mayúscula.
- Los paréntesis vacíos después del nombre de un método indican que éste no requiere parámetros para realizar su tarea.
- El cuerpo de todos los métodos está delimitado por llaves izquierda y derecha (`{` y `}`).
- El cuerpo de un método contiene instrucciones que realizan la tarea de éste. Una vez que se ejecutan las instrucciones, el método ha terminado su tarea.
- Cuando intentamos ejecutar una clase, Java busca el método `main` de la clase para empezar la ejecución.
- Por lo general, no podemos llamar a un método que pertenece a otra clase, sino hasta crear un objeto de esa clase.
- Una expresión de creación de instancia de clase (pág. 74) empieza con la palabra clave `new` y crea un nuevo objeto.
- Para llamar a un método de un objeto, se pone después del nombre de la variable un separador punto (`.`; pág. 75), el nombre del método y un conjunto de paréntesis que contienen los argumentos del método.
- En UML, cada clase se modela en un diagrama de clases en forma de rectángulo con tres compartimientos. El compartimiento superior contiene el nombre de la clase, centrado horizontalmente y en negrita. El compartimiento intermedio contiene los atributos de la clase, que corresponden a los campos en Java. El compartimiento inferior contiene las operaciones de la clase (pág. 76), que corresponden a los métodos y constructores en Java.
- Para modelar las operaciones, UML enumera el nombre de la operación, seguido de un conjunto de paréntesis. Un signo más (+) enfrente del nombre de la operación indica que ésta es una operación `public` en UML (es decir, un método `public` en Java).

### *Sección 3.3 Declaración de un método con un parámetro*

- A menudo, los métodos requieren parámetros (pág. 76) para realizar sus tareas. Dicha información adicional se proporciona mediante argumentos en las llamadas a los métodos.
- El método `nextLine` de `Scanner` (pág. 76) lee caracteres hasta encontrar una nueva línea y después devuelve los caracteres que leyó en forma de un objeto `String`.
- El método `next` de `Scanner` (pág. 77) lee caracteres hasta encontrar cualquier carácter de espacio en blanco, y después devuelve los caracteres que leyó en forma de un objeto `String`.
- Un método que requiere datos para realizar su tarea debe especificar esto en su declaración, para lo cual coloca información adicional en la lista de parámetros del método (pág. 76).
- Cada parámetro debe especificar tanto un tipo como un nombre de variable.

www.FreeLibros.me

- Cuando se hace la llamada a un método, sus argumentos se asignan a sus parámetros. Entonces, el cuerpo del método utiliza las variables de los parámetros para acceder a los valores de los argumentos.
- Un método especifica varios parámetros en una lista separada por comas.
- El número de argumentos en la llamada a un método debe coincidir con el de los parámetros en la lista de parámetros de la declaración del método. Además, los tipos de los argumentos en la llamada al método deben ser consistentes con los de los parámetros correspondientes en la declaración de éste.
- La clase `String` está en el paquete `java.lang`, que por lo general se importa de manera implícita en todos los archivos de código fuente.
- De manera predeterminada, las clases que se compilan en el mismo directorio están en el mismo paquete. Las clases en el mismo paquete se importan implícitamente en los archivos de código fuente de las otras clases que están en el mismo paquete.
- Las declaraciones `import` no son obligatorias si usamos siempre nombres de clases completamente calificados (pág. 79).
- Para modelar un parámetro de una operación, UML lista el nombre del parámetro, seguido de dos puntos y el tipo del parámetro entre los paréntesis que van después del nombre de la operación.
- UML tiene sus propios tipos de datos, similares a los de Java. No todos los tipos de datos de UML tienen los mismos nombres que los tipos correspondientes en Java.
- El tipo `String` de UML corresponde al tipo `String` de Java.

### *Sección 3.4 Variables de instancia, métodos* establecer *y métodos* obtener

- Las variables que se declaran en el cuerpo de un método son variables locales, y pueden utilizarse sólo en ese método.
- Por lo general, una clase consiste en uno o más métodos que manipulan los atributos (datos) pertenecientes a un objeto específico de esa clase. Dichas variables se llaman campos y se declaran dentro de la declaración de una clase, pero fuera de los cuerpos de las declaraciones de los métodos de esa clase.
- Cuando cada objeto de una clase mantiene su propia copia de un atributo, al campo correspondiente se le conoce como variable de instancia.
- Las variables o métodos declarados con el modificador de acceso `private` sólo están accesibles para los métodos de la clase en la que están declarados.
- Al proceso de declarar variables de instancia con el modificador de acceso `private` (pág. 80) se le conoce como ocultamiento de datos.
- Un beneficio de los campos es que todos los métodos de la clase pueden usarlos. Otra diferencia entre un campo y una variable local es que un campo tiene un valor inicial predeterminado (pág. 82), que Java proporciona cuando el programador no especifica el valor inicial del campo, pero una variable local no hace esto.
- El valor predeterminado para un campo de tipo `String` (o cualquier otro tipo por referencia) es `null`.
- Cuando se llama a un método que especifica un tipo de valor de retorno (pág. 73) y completa su tarea, devuelve un resultado al método que lo llamó (pág. 73).
- A menudo, las clases proporcionan métodos `public` para permitir que los clientes de la clase *establezcan* u *obtengan* variables de instancia `private` (pág. 83). Los nombres de estos métodos no necesitan comenzar con *establecer* u *obtener*, pero esta convención de nomenclatura es muy recomendada en Java, y requerida para ciertos componentes de software de Java especiales, conocidos como JavaBeans.
- UML representa a las variables de instancia como un nombre de atributo, seguido de dos puntos y el tipo del atributo.
- En UML, los atributos privados van precedidos por un signo menos (-).
- Para indicar el tipo de valor de retorno de una operación, UML coloca dos puntos y el tipo de valor de retorno después de los paréntesis que siguen del nombre de la operación.
- Los diagramas de clases de UML (pág. 75) no especifican tipos de valores de retorno para las operaciones que no devuelven valores.

### *Sección 3.5 Comparación entre tipos primitivos y tipos por referencia*

- En Java, los tipos se dividen en dos categorías: tipos primitivos y tipos por referencia. Los tipos primitivos son `boolean`, `byte`, `char`, `short`, `int`, `long`, `float` y `double`. Todos los demás tipos son por referencia, por lo cual, las clases que especifican los tipos de los objetos, son tipos por referencia.

www.FreeLibros.me

- Una variable de tipo primitivo puede almacenar exactamente un valor de su tipo declarado, en un momento dado.
- Las variables de instancia de tipos primitivos se inicializan de manera predeterminada. Las variables de los tipos `byte`, `char`, `short`, `int`, `long`, `float` y `double` se inicializan con `0`. Las variables de tipo `boolean` se inicializan con `false`.
- Las variables de tipos por referencia (llamadas referencias; pág. 84) almacenan la ubicación de un objeto en la memoria de la computadora. Dichas variables hacen referencia a los objetos en el programa. El objeto al que se hace referencia puede contener muchas variables de instancia y métodos.
- Los campos de tipo por referencia se inicializan de manera predeterminada con el valor `null`.
- Para invocar a los métodos de instancia de un objeto, se requiere una referencia a éste (pág. 84). Una variable de tipo primitivo no hace referencia a un objeto, por lo cual no puede usarse para invocar a un método.

### *Sección 3.6 Inicialización de objetos mediante constructores*

- La palabra clave `new` solicita memoria del sistema para almacenar un objeto, y después llama al constructor de la clase correspondiente (pág. 74) para inicializar el objeto.
- Un constructor puede usarse para inicializar un objeto de una clase, a la hora de crearlo.
- Los constructores pueden especificar parámetros, pero no tipos de valores de retorno.
- Si una clase no define constructores, el compilador proporciona uno predeterminado (pág. 85) sin parámetros, y las variables de instancia de la clase se inicializan con sus valores predeterminados.
- UML modela a los constructores en el tercer compartimiento de un diagrama de clases. Para diferenciar a un constructor con base en las operaciones de una clase, UML coloca la palabra "constructor" entre los signos « y » (pág. 88) antes del nombre de éste.

### *Sección 3.7 Los números de punto flotante y el tipo `double`*

- Un número de punto flotante (pág. 88) es un número con un punto decimal. Java proporciona dos tipos primitivos para almacenar números de punto flotante (pág. 88) en la memoria: `float` y `double`. La principal diferencia entre estos tipos es que las variables `double` pueden almacenar números con mayor magnitud y detalle (a esto se le conoce como la precisión del número; pág. 88) que las variables `float`.
- Las variables de tipo `float` representan números de punto flotante de precisión simple, y tienen siete dígitos significativos. Las variables de tipo `double` representan números de punto flotante de precisión doble. Éstos requieren el doble de memoria que las variables `float` y proporcionan 15 dígitos significativos; tienen aproximadamente el doble de precisión de las variables `float`.
- Las literales de punto flotante (pág. 88) son de tipo `double` de manera predeterminada.
- El método `nextDouble` de `Scanner` (pág. 91) devuelve un valor `double`.
- El especificador de formato `%f` (pág. 90) se utiliza para mostrar valores de tipo `float` o `double`. El especificador de formato `%.2f` especifica que se deben mostrar dos dígitos de precisión (pág. 90) a la derecha del punto decimal, en el número de punto flotante.
- El valor predeterminado para un campo de tipo `double` es `0.0`, y el valor predeterminado para un campo de tipo `int` es `0`.

## Ejercicios de autoevaluación

**3.2** Complete las siguientes oraciones:

a) Cada declaración de clase que empieza con la palabra clave _____ debe almacenarse en un archivo que tenga exactamente el mismo nombre de la clase, y que termine con la extensión de nombre de archivo `.java`.
b) En la declaración de una clase, la palabra clave _____ va seguida inmediatamente por el nombre de la clase.
c) La palabra clave _____ solicita memoria del sistema para almacenar un objeto, y después llama al constructor de la clase correspondiente para inicializarlo.
d) Cada parámetro debe especificar un(a) _____ y un(a) _____ .
e) De manera predeterminada, se considera que las clases que se compilan en el mismo directorio están en el mismo paquete, conocido como _____ .

www.FreeLibros.me

f) Cuando cada objeto de una clase mantiene su propia copia de un atributo, el campo que representa a este atributo se conoce también como _____ .
g) Java proporciona dos tipos primitivos para almacenar números de punto flotante en la memoria: _____ y _____ .
h) Las variables de tipo `double` representan a los números de punto flotante _____ .
i) El método _____ de la clase `Scanner` devuelve un valor `double`.
j) La palabra clave `public` es un _____ de acceso.
k) El tipo de valor de retorno _____ indica que un método no devolverá un valor.
l) El método _____ de `Scanner` lee caracteres hasta encontrar una nueva línea y después devuelve esos caracteres como un objeto `String`.
m) La clase `String` está en el paquete _____ .
n) No se requiere un(a) _____ si siempre hacemos referencia a una clase con su nombre completamente calificado.
o) Un(a) _____ es un número con un punto decimal, como 7.33, 0.0975 o 1000.12345.
p) Las variables de tipo `float` representan números de punto flotante _____ .
q) El especificador de formato _____ se utiliza para mostrar valores de tipo `float` o `double`.
r) Los tipos en Java se dividen en dos categorías: tipos _____ y tipos _____ .

**3.3** Conteste con *verdadero* o *falso* a cada una de las siguientes proposiciones; en caso de ser *falso*, explique por qué.
a) Por convención, los nombres de los métodos empiezan con la primera letra en mayúscula, en el nombre todas las palabras subsiguientes comienzan con la primera letra en mayúscula.
b) Una declaración `import` no es obligatoria cuando una clase en un paquete utiliza a otra en el mismo paquete.
c) Los paréntesis vacíos que van después del nombre de un método en su declaración indican que no requiere parámetros para realizar su tarea.
d) Las variables o los métodos declarados con el modificador de acceso `private` son accesibles sólo para los métodos de la clase en la que se declaran.
e) Una variable de tipo primitivo puede usarse para invocar un método.
f) Las variables que se declaran en el cuerpo de un método específico se conocen como variables de instancia, y pueden utilizarse en todos los métodos de la clase.
g) El cuerpo de cada método está delimitado por llaves izquierda y derecha (`{` y `}`).
h) Las variables locales de tipo primitivo se inicializan de manera predeterminada.
i) Las variables de instancia de tipo por referencia se inicializan de manera predeterminada con el valor `null`.
j) Cualquier clase que contenga `public static void main( String[] args )` puede usarse para ejecutar una aplicación.
k) El número de argumentos en la llamada a un método debe coincidir con el de parámetros en la lista de parámetros de la declaración del método.
l) Los valores de punto flotante que aparecen en código fuente se conocen como literales de punto flotante, y son de tipo `float` de manera predeterminada.

**3.4** ¿Cuál es la diferencia entre una variable local y un campo?

**3.5** Explique el propósito de un parámetro de un método. ¿Cuál es la diferencia entre un parámetro y un argumento?

## Respuestas a los ejercicios de autoevaluación

**3.1** a) `public`. b) `class`. c) `new`. d) tipo, nombre. e) paquete predeterminado. f) variable de instancia. g) `float`, `double`. h) de precisión doble. i) `nextDouble`. j) modificador. k) `void`. l) `nextLine`. m) `java.lang`. n) declaracion `import`. o) número de punto flotante. p) de precisión simple. q) `%f`. r) primitivo, por referencia.

**3.2** a) Falso. Por convención, los nombres de los métodos empiezan con una primera letra en minúscula y todas las palabras subsiguientes con una letra en mayúscula. b) Verdadero. c) Verdadero. d) Verdadero. e) Falso. Una variable de tipo

www.FreeLibros.me

primitivo no puede usarse para invocar a un método; se requiere una referencia a un objeto para invocar a sus métodos. f) Falso. Dichas variables se llaman variables locales, y sólo se pueden utilizar en el método en el que están declaradas. g) Verdadero. h) Falso. Las variables de instancia de tipo primitivo se inicializan de manera predeterminada. A cada variable local se le debe asignar un valor de manera explícita. i) Verdadero. j) Verdadero. k) Verdadero. l) Falso. Dichas literales son de tipo double de manera predeterminada.

**3.3** Una variable local se declara en el cuerpo de un método, y sólo puede utilizarse desde el punto en el que se declaró, hasta el final de la declaración del método. Un campo se declara en una clase, pero no en el cuerpo de alguno de los métodos de ella. Además, los campos están accesibles para todos los métodos de la clase. (En el capítulo 8, Clases y objetos: un análisis más detallado, veremos una excepción a esto).

**3.4** Un parámetro representa la información adicional que requiere un método para realizar su tarea. Cada parámetro requerido por un método está especificado en la declaración del método. Un argumento es el valor actual para un parámetro del método. Cuando se llama a un método, los valores de los argumentos se pasan a sus parámetros correspondientes para que éste pueda realizar su tarea.

## Ejercicios

**3.5** ***(Palabra clave new)*** ¿Cuál es el propósito de la palabra clave `new`? Explique lo que ocurre cuando se utiliza en una aplicación.

**3.6** ***(Constructores predeterminados)*** ¿Qué es un constructor predeterminado? ¿Cómo se inicializan las variables de instancia de un objeto, si una clase sólo tiene un constructor predeterminado?

**3.7** ***(Variables de instancia)*** Explique el propósito de una variable de instancia.

**3.8** ***(Usar clases sin importarlas)*** La mayoría de las clases necesitan importarse antes de poder utilizarlas en una aplicación ¿Por qué cualquier aplicación puede utilizar las clases `System` y `String` sin tener que importarlas primero?

**3.9** ***(Usar una clase sin importarla)*** Explique cómo podría un programa utilizar la clase `Scanner` sin importarla.

**3.10** ***(Métodos establecer y obtener)*** Explique por qué una clase podría proporcionar un método establecer y un método obtener para una variable de instancia.

**3.11** ***(Clase `LibroCalificaciones` modificada)*** Modifique la clase `LibroCalificaciones` (figura 3.10) de la siguiente manera:

a) Incluya una segunda variable de instancia `String`, que represente el nombre del instructor del curso.
b) Proporcione un método *establecer* para modificar el nombre del instructor, y un método *obtener* para conseguir el nombre.
c) Modifique el constructor para especificar dos parámetros: uno para el nombre del curso y otro para el del instructor.
d) Modifique el método `mostrarMensaje`, de tal forma que primero imprima el mensaje de bienvenida y el nombre del curso, seguidos de `"Este curso es presentado por: "` y el nombre del instructor.

Use su clase modificada en una aplicación de prueba que demuestre las nuevas capacidades que tiene.

**3.12** ***(Clase `Cuenta` modificada)*** Modifique la clase `Cuenta` (figura 3.13) para proporcionar un método llamado `cargar`, que retire dinero de un objeto `Cuenta`. Asegure que el monto a cargar no exceda el saldo de `Cuenta`. Si lo hace, el saldo debe permanecer sin cambio y el método debe imprimir un mensaje que indique `"El monto a cargar excede el saldo de la cuenta."` Modifique la clase `PruebaCuenta` (figura 3.14) para probar el método `cargar`.

**3.13** ***(La clase `Factura`)*** Cree una clase llamada `Factura`, que una ferretería podría utilizar para representar una factura para un artículo vendido en la tienda. Una `Factura` debe incluir cuatro piezas de información como variables de instancia: un número de pieza (tipo `String`), la descripción de la pieza (tipo `String`), la cantidad de artículos de ese tipo que se van a comprar (tipo `int`) y el precio por artículo (`double`). Su clase debe tener un constructor que inicialice las cuatro variables de instancia. Proporcione un método *establecer* y uno *obtener* para cada variable de instancia. Además, proporcione un método llamado `obtenerMontoFactura`, que calcule el monto de la factura (es decir, que multiplique la cantidad por el precio por artículo) y después lo devuelva como un valor `double`. Si la cantidad no es positiva, debe estable-

www.FreeLibros.me

cerse en 0. Si el precio por artículo no es positivo, debe establecerse en 0.0. Escriba una aplicación de prueba llamada PruebaFactura, que demuestre las capacidades de la clase Factura.

**3.14** ***(La clase Empleado)*** Cree una clase llamada Empleado, que incluya tres variables de instancia: un primer nombre (tipo String), un apellido paterno (tipo String) y un salario mensual (double). Su clase debe tener un constructor que inicialice las tres variables de instancia. Proporcione un método *establecer* y un método *obtener* para cada variable de instancia. Si el salario mensual no es positivo, no establezca su valor. Escriba una aplicación de prueba llamada PruebaEmpleado, que demuestre las capacidades de la clase Empleado. Cree dos objetos Empleado y muestre el salario anual de cada objeto. Después, proporcione a cada Empleado un aumento del 10% y muestre el salario anual de cada Empleado otra vez.

**3.15** ***(La clase Fecha)*** Cree una clase llamada Fecha, que incluya tres variables de instancia: un mes (tipo int), un día (tipo int) y un año (tipo int). Su clase debe tener un constructor que inicialice las tres variables de instancia, y debe asumir que los valores que se proporcionan son correctos. Proporcione un método *establecer* y un método *obtener* para cada variable de instancia. Proporcione un método mostrarFecha, que muestre el mes, día y año, separados por barras diagonales (/). Escriba una aplicación de prueba llamada PruebaFecha, que demuestre las capacidades de la clase Fecha.

## Marcar la diferencia

**3.16** ***(Calculadora de la frecuencia cardiaca esperada)*** Mientras se ejercita, puede usar un monitor de frecuencia cardiaca para ver que su corazón permanezca dentro de un rango seguro sugerido por sus entrenadores y doctores. De acuerdo con la Asociación Estadounidense del Corazón (AHA) (www.americanheart.org/), la fórmula para calcular su frecuencia cardiaca máxima en pulsos por minuto es de 220 menos su edad en años. Su frecuencia cardiaca esperada es un rango que está entre el 50 y el 85% de su frecuencia cardiaca máxima. [*Nota:* estas fórmulas son estimaciones proporcionadas por la AHA. Las frecuencias cardiacas máxima y esperada pueden variar de acuerdo con la salud, condición física y sexo del individuo. Siempre debe consultar un médico o a un profesional de la salud antes de empezar o modificar un programa de ejercicios.] Cree una clase llamada FrecuenciasCardiacas. Los atributos de la clase deben incluir el primer nombre de la persona, su apellido y fecha de nacimiento (la cual debe consistir de atributos separados para el mes, día y año de nacimiento). Su clase debe tener un constructor que reciba estos datos como parámetros. Para cada atributo debe proveer métodos *establecer* y *obtener*. La clase también debe incluir un método que calcule y devuelva la edad de la persona (en años), uno que calcule y devuelva la frecuencia cardiaca máxima de esa persona, y otro que calcule y devuelva la frecuencia cardiaca esperada de la persona. Escriba una aplicación de Java que pida la información de la persona, cree una instancia de un objeto de la clase FrecuenciasCardiacas e imprima la información a partir de ese objeto (incluya el primer nombre de la persona, su apellido y fecha de nacimiento), y que después calcule e imprima la edad de la persona en (años), frecuencia cardiaca máxima y rango de frecuencia cardiaca esperada.

**3.17** ***(Computarización de los registros médicos)*** Un problema relacionado con la salud que ha estado últimamente en las noticias es la computarización de los registros médicos. Esta posibilidad se está tratando con mucho cuidado, debido a las delicadas cuestiones de privacidad y seguridad, entre otras cosas. [Trataremos esas cuestiones en ejercicios posteriores.] La computarización de los registros médicos puede facilitar a los pacientes el proceso de compartir sus perfiles e historiales médicos con los diversos profesionales de la salud que consulten. Esto podría mejorar la calidad del servicio médico, ayudar a evitar conflictos de fármacos y prescripciones erróneas, reducir los costos y, en emergencias, ayudar a salvar vidas. En este ejercicio usted diseñará una clase inicial llamada PerfilMedico para una persona. Los atributos de la clase deben llevar el primer nombre de la persona, su apellido, sexo, fecha de nacimiento (que debe consistir de atributos separados para el día, mes y año de nacimiento), altura (en centímetros) y peso (en kilogramos). Su clase debe tener un constructor que reciba estos datos. Para cada atributo, debe proveer los métodos *establecer* y *obtener*. La clase también debe tener métodos que calculen y devuelvan la edad del usuario en años, la frecuencia cardiaca máxima y el rango de frecuencia cardiaca esperada (vea el ejercicio 3.16), además del índice de masa corporal (BMI; vea el ejercicio 2.33). Escriba una aplicación de Java que pida la información de la persona, cree una instancia de un objeto de la clase PerfilMedico para esa persona e imprima la información de ese objeto (debe contener el primer nombre de la persona, apellido, sexo, fecha de nacimiento, altura y peso), y que después calcule e imprima la edad de esa persona en años, junto con el BMI, la frecuencia cardiaca máxima y el rango de frecuencia cardiaca esperada. También debe mostrar la tabla de valores del BMI del ejercicio 2.33.

www.FreeLibros.me