Introducción a las clases, los objetos, los métodos y las cadenas

Variables de instancia, métodos establecer y métodos obtener

En esta sección creará dos clases: **`Cuenta`** (figura 1) y **`PruebaCuenta`** (figura 2). La clase **`PruebaCuenta`** es una clase de aplicación en la que el método **`main`** creará y usará un objeto **`Cuenta`** para demostrar las capacidades de la clase **`Cuenta`**.

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

Por lo general las distintas cuentas tienen nombres diferentes. Por esta razón, la clase **`Cuenta`** (figura 3.1) contiene una *variable de instancia* llamada **`nombre`**. Las variables de instancia de una clase mantienen los datos para cada objeto (es decir, cada instancia) de la clase. Más adelante agregaremos una variable de instancia llamada **`saldo`** para poder llevar el registro de cuánto dinero hay en la cuenta. La clase **`Cuenta`** contiene dos métodos: el método **`establecerNombre`** almacena un nombre en un objeto **`Cuenta`** y el método **`obtenerNombre`** obtiene el nombre de un objeto **`Cuenta`**.

```
 1  // Fig. 1: Cuenta.java
 2  // Clase Cuenta que contiene una variable de instancia llamada nombre
 3  // y métodos para establecer y obtener su valor.
 4
 5  public class Cuenta
 6  {
 7      private String nombre; // variable de instancia
 8
 9      // método para establecer el nombre en el objeto
10      public void establecerNombre(String nombre)
11      {
12          this.nombre = nombre; // almacenar el nombre
13      }
14
15      // método para obtener el nombre del objeto
16      public String obtenerNombre()
17      {
18          return nombre; // devuelve el valor de nombre a quien lo invocó
19      }
20  } // fin de la clase Cuenta
```

Fig. 1 *La clase **`Cuenta`** que contiene una variable de instancia llamada **`nombre`** y métodos para establecer y obtener su valor*

Declaración de la clase

La *declaración de la clase* empieza en la línea 5:

`public class Cuenta`

La palabra clave `public` es un modificador de acceso. Por ahora, simplemente declararemos todas las clases como `public`. Cada declaración de clase `public` debe almacenarse en un archivo de texto que tenga el mismo nombre que la clase y termine con la extensión de nombre de archivo **`.java`**; de lo contrario, ocurrirá un error de compilación. Por ende, las clases `Cuenta` y `PruebaCuenta` (figura 2) *deben* declararse en los archivos independientes `Cuenta.java` y `PruebaCuenta.java`, respectivamente.

Cada declaración de clase contiene la palabra clave `class`, seguida inmediatamente por el nombre de la clase; en este caso, `Cuenta`. El cuerpo de toda clase se encierra entre un par de llaves, una izquierda y una derecha, como en las líneas 6 y 20 de figura 1.

Los identificadores y la nomenclatura de lomo de camello (CamelCase)

Los nombres de las clases, los métodos y las variables son todos *identificadores* y, por convención, usan el mismo esquema de nomenclatura **lomo de camello**. Además, por convención los nombres de las clases comienzan con una letra *mayúscula*, y los nombres de los métodos y variables comienzan con una letra en *minúscula*.

La variable de instancia nombre

Vimos que un objeto tiene atributos, los cuales se implementan como variables de instancia y los lleva consigo durante su vida útil. Las variables de instancia existen antes de invocar los métodos en un objeto, durante su ejecución y después de que éstos terminan su ejecución. Cada objeto (instancia) de la clase tiene su *propia* copia de las variables de instancia de la clase. Por lo general una clase contiene uno o más métodos que manipulan a las variables de instancia que pertenecen a objetos específicos de la clase.

Las variables de instancia se declaran *dentro* de la declaración de una clase pero *fuera* de los cuerpos de los métodos de la misma. La línea 7

```
private String nombre; // variable de instancia
```

declara la variable de instancia de tipo **`String`** *fuera* de los cuerpos de los métodos **`establecerNombre`** (líneas 10 a 13) y **`obtenerNombre`** (líneas 16 a 19). Las variables `String` pueden contener valores de cadenas de caracteres tales como "`Jane Green`". Si hay muchos objetos `Cuenta`, cada uno tiene su propio nombre. Puesto que nombre es una variable de instancia, cada uno de los métodos de la clase puede manipularla.

> **Buena práctica de programación 1**
> *Preferimos listar primero las variables de instancia de una clase en el cuerpo de la misma, para que usted pueda ver los nombres y tipos de las variables antes de usarlas en los métodos de la clase. Es posible listar las variables de instancia 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 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. Por lo tanto, la variable nombre sólo puede utilizarse en los métodos de cada objeto `Cuenta` (`establecerNombre` y `obtenerNombre` en este caso).

El método establecerNombre de la clase Cuenta

Vamos a recorrer el código de la declaración del método establecerNombre (líneas 10 a 13):

```
public void establecerNombre(String nombre)     <----- Esta línea es el encabezado del método
{
    this.nombre = nombre; // almacenar el nombre
}
```

Nos referimos a la primera línea de la declaración de cada método (línea 10 en este caso) como el encabezado del método. El **tipo de valor de retorno** del método (que aparece antes del nombre del método) especifica el tipo de datos que el método devuelve a *quien lo llamó* después de realizar su tarea. El tipo de valor de retorno void (línea 10) indica que `establecerNombre` realizará una tarea pero *no* regresará (es decir, no devolverá) ninguna información a quien lo invocó. Anteriormente usó métodos que devuelven información; por ejemplo, usó el método `Scanner nextInt` para recibir como entrada un entero escrito por el usuario mediante el teclado. Cuando `nextInt` lee un valor del usuario, devuelve ese valor para usarlo en el programa. Como pronto veremos, el método `obtenerNombre` de `Cuenta` devuelve un valor.

El método `establecerNombre` recibe el parámetro nombre de tipo `String`, el cual representa el nombre que se pasará al método como un argumento. Cuando hablemos sobre la llamada al método en la línea 21 de la figura 2, podrá ver cómo trabajan juntos los parámetros y los argumentos.

Los parámetros se declaran en una lista de parámetros, la cual se encuentra dentro de los paréntesis que van después del nombre del método en el encabezado del mismo. Cuando hay varios parámetros, cada uno va separado del siguiente mediante una coma. Cada parámetro debee especificar un tipo (en este caso, `String`) seguido de un nombre de variable (en este caso, `nombre`).

Los parámetros son variables locales

Anteriormente declaramos todas las variables de una aplicación en el método main. Las variables declaradas en el cuerpo de un método específico (como main) son variables locales, las cuales pueden usarse *sólo* en ese método. Cada método puede acceder sólo a sus propias variables locales y no a las de los otros métodos. Cuando un método termina, se *pierden* los valores de sus variables locales. Los parámetros de un método también son variables locales del mismo.

El cuerpo del método `establecerNombre`

Cada cuerpo de método está delimitado por un par de llaves (como en las líneas 11 y 13 de la figura 1), las cuales contienen una o más instrucciones que realizan las tareas del método. En este caso, el cuerpo del método contiene una sola instrucción (línea 12) que asigna el valor del parámetro `nombre` (un objeto `String`) a la *variable de instancia* nombre de la clase, con lo cual almacena el nombre de la cuenta en el objeto.

Si un método contiene una variable local con el *mismo* nombre que una variable de instancia (como en las líneas 10 y 7, respectivamente), el cuerpo de ese método se referirá a la variable local en vez de a la variable de instancia. En este caso, se dice que la variable local *oculta* a la variable de instancia en el cuerpo del método, el cual puede usar la palabra clave this para referirse de manera explícita a la variable de instancia oculta, como se muestra en el lado izquierdo de la asignación en la línea 12.

Buena práctica de programación 2
Podríamos haber evitado el uso de la palabra clave this aquí si eligiéramos un nombre diferente para el parámetro en la línea 10, pero usar la palabra clave `this` *como se muestra en la línea 12 es una práctica aceptada ampliamente para minimizar la proliferación de nombres de identificadores.*

Una vez que se ejecuta la línea 12, el método completa su tarea por lo que regresa a *quien lo llamó*. Como pronto veremos, la instrucción en la línea 21 de `main` (figura 2) llama al método `establecerNombre`.

El método `obtenerNombre` de la clase `Cuenta`

El método `obtenerNombre` (líneas 16 a 19)

```
public String obtenerNombre()
{
    return nombre; // devuelve el valor de nombre a quien lo invocó
}
```

La palabra clave `return` devuelve el nombre de `String` a quien invocó al método.

devuelve el nombre de un objeto **`Cuenta`** específico a quien lo llamó. El método tiene una lista de parámetros *vacía*, por lo que *no* requiere información adicional para realizar su tarea. El método devuelve un objeto **`String`**. Cuando un método que especifica un tipo de valor de retorno *distinto* de **`void`** se invoca y completa su tarea, debe devolver un resultado a quien lo llamó. Una instrucción que llama al método **`obtenerNombre`** en un objeto **`Cuenta`** (como los de las líneas 16 y 26 de la figura 2) espera recibir el nombre de ese objeto **`Cuenta`**, es decir, 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 de la figura 1 regresa el valor **`String`** de la variable de instancia nombre a quien hizo la llamada. Por ejemplo, cuando el valor se devuelve a la instrucción en las líneas 25 y 26 de la figura 2, la instrucción usa ese valor para mostrar el nombre en pantalla.

La clase `PruebaCuenta` que crea y usa un objeto de la clase `Cuenta`

A continuación, nos gustaría utilizar la clase **`Cuenta`** en una aplicación y *llamar* a cada uno de sus métodos. Una clase que contiene el método main empieza la ejecución de una aplicación de Java. La clase **`Cuenta`** *no* se puede ejecutar a sí misma ya que no contiene un método **`main`**. Si escribe **`java Cuenta`** en el símbolo del sistema, obtendrá el siguiente error: "**`Main method not found in class Cuenta`**". Para corregir este problema, debe declarar una clase separada que contenga un método **`main`** o colocar un método **`main`** en la clase **`Cuenta`**.

La clase controladora `PruebaCuenta`

Para ayudarlo a prepararse para los programas extensos que encontrará más adelante en este libro y en la industria, utilizamos una clase separada **`PruebaCuenta`** (figura 2) que contiene el método **`main`** para probar la clase **`Cuenta`**. Una vez que **`main`** comienza a ejecutarse, puede llamar a otros métodos en ésta y otras clases; a su vez, estos métodos pueden llamar a otros métodos, y así en lo sucesivo. El método **`main`** de la clase **`PruebaCuenta`** crea un objeto **`Cuenta`** y llama a sus métodos **`obtenerNombre`** y **`establecerNombre`**. A este tipo de clases se le conoce algunas veces como *clase controladora*. Así como un objeto **`Persona`** conduce un objeto **`Auto`** diciéndole lo que debe hacer (ir más rápido o más lento, girar a la izquierda o a la derecha, etc.), la clase **`PruebaCuenta`** conduce un objeto **`Cuenta`** y llama a sus métodos para decirle lo que debe hacer.

```
 1  // Fig. 2: PruebaCuenta.java
 2  // Crear y manipular un objeto Cuenta.
 3  import java.util.Scanner;
 4
 5  public class PruebaCuenta
 6  {
 7      public static void main(String[] args)
 8      {
 9          // crea un objeto Scanner para obtener la entrada desde el símbolo del sistema
10          Scanner entrada = new Scanner(System.in);
11
12          // crea un objeto Cuenta y lo asigna a miCuenta
13          Cuenta miCuenta = new Cuenta();
14
15          // muestra el valor inicial del nombre (null)
16          System.out.printf("El nombre inicial es: %s%n%n", miCuenta.obtenerNombre());
17
18          // pide y lee el nombre
19          System.out.println("Introduzca el nombre:");
20          String elNombre = entrada.nextLine(); // lee una línea de texto
21          miCuenta.establecerNombre(elNombre); // coloca elNombre en miCuenta
22          System.out.println(); // imprime una línea en blanco
23
24          // muestra el nombre almacenado en el objeto miCuenta
25          System.out.printf("El nombre en el objeto miCuenta es:%n%s%n",
26              miCuenta.obtenerNombre());
27      }
28  } // fin de la clase PruebaCuenta
```

Fig. 2 *Creación y manipulación de un objeto* ***`Cuenta`***

```
El nombre inicial es: null

Introduzca el nombre:
Alfonso Romero

El nombre en el objeto miCuenta es:
Alfonso Romero
```

Objeto *Scanner* para recibir entrada del usuario

La línea 10 crea un objeto **`Scanner`** llamado **`input`** para recibir como entrada el nombre del usuario. La línea 19 pide al usuario que introduzca un nombre. La línea 20 usa el método **`nextLine`** del objeto **`Scanner`** para leer el nombre del usuario y asignarlo a la variable `local` **`elNombre`**. Usted escribe el nombre y oprime **Intro** para enviarlo al programa. Al oprimir **Intro** se inserta un carácter de nueva línea después de los caracteres que escribió. El método **`nextLine`** lee caracteres (incluyendo caracteres de espacio en blanco, como el espacio en "*Alfonso Romero*") hasta encontrar la nueva línea, la cual descarta.

La clase **`Scanner`** proporciona varios métodos de entrada más, como veremos después. Hay un método similar a **`nextLine`** (llamado **`next`**) que lee la siguiente palabra. Cuando oprime **Intro** después de escribir algo de texto, el método `next` lee caracteres hasta encontrar un carácter de espacio en blanco (como un espacio, un tabulador o una nueva línea) y luego devuelve un objeto **`String`** que contiene los caracteres hasta (sin incluir) el carácter de espacio en blanco, el cual se descarta. La información que está después del primer carácter de espacio en blanco no se pierde: puede leerse en otras instrucciones que llamen a los métodos de **`Scanner`** posteriormente en el programa.

Instanciación de un objeto: la palabra clave `new` y los constructores

La línea 13 crea un objeto **`Cuenta`** y lo asigna a la variable **`miCuenta`** de tipo `Cuenta`. La variable **`miCuenta`** se inicializa con el resultado de la **expresión de creación de instancia de clase `new Cuenta()`**. La palabra clave ***new*** crea un nuevo objeto de la clase especificada; en este caso, **`Cuenta`**. Los paréntesis a la derecha de **`Cuenta`** son *obligatorios*. Como veremos postetiormente, esos paréntesis en combinación con el nombre de una clase representan una llamada a un **constructor**, que es similar a un método pero es invocado de manera implícita por el operador **`new`** para *inicializar* las variables de instancia de un objeto al momento de *crearlo*. Posteriormente veremos cómo colocar un *argumento* en los paréntesis para especificar un *valor inicial* para la variable de instancia nombre de un objeto Cuenta; para poder hacer esto usted mejorará la clase **`Cuenta`**. Por ahora sólo dejaremos los paréntesis *vacíos*. La línea 10 contiene una expresión de creación de instancia de clase para un objeto **`Scanner`**. La expresión inicializa el objeto **`Scanner`** con **`System.in`**, que indica a **`Scanner`** de dónde debe leer la entrada (por ejemplo, del teclado).

Llamada al método obtenerNombre de la clase `Cuenta`

La línea 16 muestra el nombre *inicial*,que se obtiene mediante una llamada al método **`obtenerNombre`** del objeto. Así como podemos usar el objeto **`System.out`** para llamar a sus métodos **`print`**, **`printf`** y **`println`**, también podemos usar el objeto **`miCuenta`** para llamar a sus métodos **`obtenerNombre`** y **`establecerNombre`**. La línea 16 llama al método **`obtenerNombre`** usando el objeto **`miCuenta`** creado en la línea 13, seguido tanto de un separador punto (.), como del nombre del método **`obtenerNombre`** y de un conjunto vacío de paréntesis ya que no se pasan argumentos. Cuando se hace la llamada a **`obtenerNombre`**:

1. La aplicación transfiere la ejecución del programa de la llamada (línea 16 en **`main`**) a la declaración del método **`obtenerNombre`** (líneas 16 a 19 de la figura 1). Como la llamada a **`obtenerNombre`** fue a través del objeto **`miCuenta`**, **`obtenerNombre`** "sabe" qué variable de instancia del objeto manipular.

2. A continuación, el método **`obtenerNombre`** realiza su tarea; es decir, devuelve el nombre (línea 18 de la figura 1). Cuando se ejecuta la instrucción **`return`**, la ejecución del programa continúa a partir de donde se hizo la llamada a **`obtenerNombre`** (línea 16 de la figura 2).

3. **`System.out.printf`** muestra el objeto **`String`** devuelto por el método **`obtenerNombre`** y luego el programa continúa su ejecución en la línea 19 de **`main`**.

Tip para prevenir errores

*Nunca use como control de formato una cadena que el usuario haya introducido. Cuando el método **`System.out.printf`** evalúa la cadena de control de formato en su primer argumento, el método realiza las tareas con base en los especificadores de conversión de esa cadena. Si la cadena de control de formato se obtuviera del usuario, un usuario malicioso podría proveer especificadores de conversión que **`System.out.printf`** ejecutara, lo que probablemente generará una infracción de seguridad.*

`null`: el valor inicial predeterminado de las variables `String`

La primera línea de la salida muestra el nombre **`null`**. A diferencia de las variables locales, que no se inicializan de manera automática, cada variable de instancia tiene un valor inicial predeterminado, que es un valor que Java proporciona cuando el programador no especifica el valor inicial de la variable de instancia. Por ende, no se requiere que las *variables de instancia* se inicialicen de manera explícita antes de usarlas en un programa, a menos que deban inicializarse con valores *distintos de* los predeterminados. El valor predeterminado para una variable de instancia de tipo **`String`** (como nombre en este ejemplo) es **`null`**, de lo cual hablaremos con más detalle cuando consideremos los *tipos por referencia*.

*Llamada al método establecerNombre de la clase **`Cuenta`***

La línea 21 llama al método **`establecerNombre`** de la clase **`miCuenta`**. La llamada a un método puede proveer argumentos cuyos valores se asignen a los parámetros correspondientes del método. En este caso, el valor de la variable local entre paréntesis **`elNombre`** de **`main`** es el argumento que se pasa a **`establecerNombre`** para que el método pueda realizar su tarea. Cuando se hace la llamada a **`establecerNombre`**:

1. La aplicación transfiere la ejecución del programa de la línea 21 en **`main`** a la declaración del método **`establecerNombre`** (líneas 10 a 13 de la figura 1) y el valor del argumento en los paréntesis de la llamada (**`elNombre`**) se asigna al parámetro correspondiente (**`nombre`**) en el encabezado del método (línea 10 de la figura 1). Puesto que **`establecerNombre`** se llamó a través del objeto **`miCuenta`**, establecerNombre "sabe" qué variable de instancia del objeto manipular.
2. A continuación, el método **`establecerNombre`** realiza su tarea; es decir, asigna el valor del parámetro nombre a la variable de instancia nombre (línea 12 de la figura 1).
3. Cuando la ejecución del programa llega a la llave derecha de cierre de **`estable- cerNombre`**, regresa hasta donde se hizo la llamada a **`establecerNombre`** (línea 21 de la figura 2) y continúa en la línea 22 de la figura 2.

El número de *argumentos* en la llamada a un método debe *coincidir* con el número de *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 tipos de los parámetros correspondientes en la declaración del método. En nuestro ejemplo, la llamada al método pasa un argumento de tipo **`String`** (**`elNombre`**) y la declaración del método especifica un parámetro de tipo **`String`** (nombre, declarado en la línea 10 de la figura 1). Por lo tanto, en este ejemplo el tipo del argumento en la llamada al método coincide exactamentee con el tipo del parámetro en el encabezado del método.

Obtención del nombre que el usuario introdujo

La línea 22 de la figura 2 muestra una línea en blanco. Cuando se ejecuta la segunda llamada al método obtenerNombre (línea 26), se muestra el nombre introducido por el usuario en la línea 20. Cuando la instrucción en las líneas 25 y 26 termina de ejecutarse, se llega al final del método **`main`**, por lo que el programa termina.

Compilación y ejecución de una aplicación con varias clases

Debe compilar las clases de las figuras 3.1 y 3.2 antes de poder ejecutarr la aplicación. Ésta es la primera vez que crea una aplicación con varias clases. La clase PruebaCuenta tiene un método main; la clase Cuenta no. Para compilar esta aplicación, primero cambie al directorio que contiene los archivos de código fuente de la aplicación. Después, escriba el comando

```
javac Cuenta.java PruebaCuenta.java
```

para compilar *ambas* clases a la vez. Si el directorio que contiene la aplicación sólo incluye los archivos de esta aplicación, puede compilar ambass clases con el comando

```
javac *.java
```

El asterisco (*) en **`*.java`** indica que deben compilarse todoss los archivos del directorio actuall que terminen con la extensión de nombre de archivo "**`.java`**". Si ambas clases se compilan en forma correcta (es decir, que no aparezcan errores de compilación), podrá entonces ejecutar la aplicación con el comando

```
java PruebaCuenta
```