CAPITULO 2. COMPRENDER LAS DEFINICIONES DE CLASE

2.3 Campos, constructores y métodos

El código de la mayoría de las clases puede descomponerse en dos partes principales: una envoltura exterior pequeña que simplemente da nombre a la clase y una parte interna mucho más grande que hace todo el trabajo.

public class <NombreClase>
{
<parte interna>
}

La envoltura exterior de las diferentes clases es muy parecida, su principal finalidad es proporcionar un nombre a la clase.

La parte interna de la clase es el lugar en el que definimos los campos, los constructores y los métodos que dan a los objetos de la clase sus características particulares y su comportamiento. Podemos resumir las características esenciales de estos tres componentes de una clase como sigue:

Los campos almacenan datos para que cada objeto los use.
Los constructores permiten que cada objeto se prepare adecuadamente cuando es creado.
Los métodos implementan el comportamiento de los objetos.

En Java existen muy pocas reglas sobre el orden que se puede elegir para definir los campos, los constructores y los métodos dentro de una clase. Es importante elegir un estilo y luego usarlo de manera consistente, porque de este modo las clases serán más fáciles de leer y de comprender.

2.3.1 Campos

Los **campos** almacenan datos para que un objeto los use. Los campos también son conocidos como **variables de instancia.** Puesto que los campos pueden almacenar valores que pueden variar a lo largo del tiempo, suponen un tipo de *variables.*

Los campos son *pequeñas cantidades de espacio dentro de un objeto que pueden usarse para almacenar valores.* Todo objeto, una vez creado, dispondrá de un espacio para cada campo declarado en su clase.

|private/public| <tipo> <nombreCampo> |= <valorInicial>|¡, <nombreCampo> |= <valorInicial>|!;

*Los **comentarios** se insertan en el código de una clase para proporcionar explicaciones a los lectores humanos. No tienen ningún efecto sobre la funcionalidad de la clase.*

Se introduce una sola línea de comentario mediante los dos caracteres «//».

Los comentarios más detallados, que frecuentemente ocupan varias líneas, se escriben generalmente en la forma de comentarios multilínea: comienzan con el par de caracteres «/*» y terminan con el par «*/».

Comentarios de Documentación:
/**
* (descripción principal) objetivo general de la clase o método
* (sección de etiqueta)
*/

Sección etiqueta:
Puede usarse de dos maneras:
*En bloques de etiquetas----------> la más importante.
**Etiquetas de sólo una linea*.....*(ver Javadoc, Tools an Utilities)*

Hay alrrededor de 20, pero las más importantes:

Comentarios de clase e interfaz:
@author
@version
@see

Comentarios de metodos y constructores:
@param
@throws
@see
@return (solo para métodos)

@see referencia cruzada:
Adopta varias formas:
**@see "The Java Language Specification" (encierra un texto en forma de cadena sin hipervículo)*
**@see <a href=http://www.bluej.org/> (hipervinculo)*
**@see #estaVivo (vinculo a la documentación del método estaVivo de la misma clase)*
**@see java.util.ArrayList#add (vincula la documentación del método add cn la clase java.util.ArrayList)*

2.3.2 Constructores

Los **constructores** permiten que cada objeto sea preparado adecuadamente cuando es creado. Esta operación se denomina *inicialización.* El constructor inicializa el objeto en un estado razonable.

Uno de los rasgos distintivos de los constructores es que tienen el mismo nombre que la clase en la que son definidos.

Los campos del objeto se inicializan en el constructor, bien con valores fijos, o bien con parámetros del propio constructor.

Nota: en Java, todos los **campos** son inicializados automáticamente con un valor por defecto, si es que no están inicializados explícitamente. El valor por defecto para los **campos enteros** es 0. Sin embargo, es preferible escribir explícitamente las asignaciones. No hay ninguna desventaja en hacer esto y sirve para documentar lo que está ocurriendo realmente.

EJEMPLO:

```
//Ejemplo de clase con dos constructores y un método
public class Persona
{
// campos
 private String nombre;
 private int edad;
```

```
//CONSTRUCTOR 1
public Persona (String nombrePersona)
{
nombre = nombrePersona;
edad = 0;
 }
```

```
 //CONSTRUCTOR2
 public Persona ()
{
nombre = "";
edad = 0;
}
```

```
//METODO

public String getNombre ()
{
return nombre;
}
```

}//Cierre de la clase

2.4 Pasar datos mediante parámetros

La manera en que los constructores y los métodos reciben valores es mediante sus *Parámetros.*
Los parámetros se definen en el encabezado de un constructor o un método:

|public/private| |<tipoDevuelto>| <nombreMetodo> (|<tipoParam> <nombreParam>¡,<tipoParam> <nombreParam> !|)

*Distinguimos entre nombres de los parámetros **dentro** de un constructor o un método, y valores de los parámetros **fuera** de un constructor o un método: hacemos referencia a los nombres como **parámetros formales** y a los valores como **parámetros actuales.***
(argumento formal (C+-)= parametro formal (Java)) ; (argumento real (C+-) = parametro actual (Java))

Puesto que permiten almacenar valores, **los parámetros formales** constituyen otra clase de variables.

El **ALCANCE** de una variable define la sección de código en la que la variable puede ser accedida. Un parámetro formal está disponible para un objeto sólo dentro del cuerpo del constructor o del método que lo declara.

Decimos que el *alcance* de un parámetro está restringido al cuerpo del constructor o del método en el que es declarado.
En cambio, el alcance de un campo es toda la clase y puede ser accedido desde cualquier lugar en la misma clase.

(variable global (C++) = campo(Java)) ; (variable local (C++) = variable(Java)

Un concepto relacionado con el alcance de una variable es el *tiempo de vida* de la variable.

El **tiempo de vida** de una variable describe cuánto tiempo continuará existiendo la variable antes de ser destruida.
El tiempo de vida de un parámetro se limita a una sola llamada de un constructor o método. Una vez que completó su tarea, los parámetros formales desaparecen y se pierden los valores que contienen.
Por el contrario, el tiempo de vida de un campo es el mismo tiempo de vida que el del objeto al que pertenece.

2.5 Asignación

Las **sentencias de asignación** almacenan el valor representado por el lado derecho de la sentencia en una variable nombrada a la izquierda.
<variable> = <expresión>;

Una **regla** sobre las sentencias de asignación es que el tipo de una expresión debe coincidir con el tipo de la variable a la que es asignada. La misma regla se aplica también entre los parámetros formales y los parámetros actuales: el tipo de una expresión de un parámetro actual debe coincidir con el tipo de una variable parámetro formal.

2.6 Métodos de Acceso

Los **métodos** se componen de dos partes: un encabezado y un cuerpo.

|<comentarioDescriptivo>|
|public/private| <tipoDevuelto> <nombreMetodo>(¡<parámetro>!) -> Signatura[1]
{<cuerpo>}

Es importante distinguir entre signatura del método y declaración de campos porque son muy parecidos.
Podemos decir que algo es un método y no un campo porque está seguido de un par de paréntesis: «(» y «)». Observese también

1 Esta definición difiere ligeramente de la definición más formal de la especificación del lenguaje Java donde la signatura no incluye al modificador de acceso ni al tipo de retorno.

que no hay un punto y coma al final de la signatura.

El cuerpo del método es la parte restante del método, que aparece a continuación del encabezado. Está siempre encerrado entre llaves: «{« y »}». Los cuerpos de los métodos contienen las *declaraciones* y las *sentencias* que definen qué ocurre dentro de un objeto cuando es invocado ese método.

Cualquier conjunto de declaraciones y sentencias, ubicado entre un par de llaves, es conocido como un ***bloque***. Por lo que el cuerpo de una clase y los cuerpos de todos los métodos de las clases son bloques.

Existen, dos diferencias significativas entre las signaturas de los constructores de una clase y de los demás métodos:
por un lado los constructores tienen el mismo nombre que la clase en la que están definidos, y por otro l**os métodos siempre tienes un *tipo de retorno* (si es un void hablaríamos de metodo modificador,)** mientras que el constructor no tiene tipo de retorno; el tipo de retorno se escribe exactamente antes del nombre del método; es una regla de Java que el constructor no puede tener ningún tipo de retorno.

Por otro lado, tanto los constructores como los métodos pueden tener cualquier número de parámetros formales, inclusive pueden no tener ninguno.

Los métodos pueden tener una *sentencia return* y es la responsable de devolver un valor que coincida con el tipo de retorno de la signatura del método. Cuando un método contiene una sentencia return, siempre es la última sentencia que se ejecuta del mismo porque una vez que se ejecutó esta sentencia no se ejecutarán más sentencias en el método.

Los **métodos de acceso** devuelven información sobre el estado del objeto. Un método de acceso contiene generalmente una sentencia return para devolver información de un valor en particular.

2.7 Métodos de modificación

A los métodos que modifican el estado de su objeto los llamamos *métodos de modificación* (o sólo *modificadores)*. Los **métodos de modificación** cambian el estado de un objeto.

La signatura de un método de modificación normalmente tiene tipo de retorno void y un solo parámetro formal, el nuevo valor del campo a modificar. Un tipo de retorno void significa que el método no devuelve ningún valor cuando es llamado; es significativamente diferente de todos los otros tipos de retorno. En el cuerpo de un método void, esta diferencia se refleja en el hecho de que no hay ninguna sentencia return.[2]

Los métodos de modificación siempre tienen a menos una sentencia de asignación. El sumar (o restar) una cantidad al valor de una variable es algo tan común que existe un operador de asignación compuesto, especial para hacerlo: «+=».
<variablle> += <expresion>;
<variablle> -= <expresion>;

Nota: *convenciones Java sobre métodos de acceso y de modificación. En Java, los nombres de los* ***métodos de acceso*** *suelen comenzar con la palabra «get» y los nombres de los* ***métodos de modificación*** *con la palabra «set».*

2.8 Imprimir desde Métodos

El método **System.out.println(<parametro>)** imprime su parámetro en la terminal de texto. Una sentencia como System.out.println("# Línea BlueJ"); imprime literalmente la cadena que aparece entre el par de comillas dobles. Todas estas sentencias de impresión son invocaciones al método println del objeto System.out que está construido dentro del lenguaje Java.

Cuando se usa el símbolo «+» entre una cadena y cualquier otra cosa, este símbolo es un operador de concatenación de cadenas (es decir, concatena o reúne cadenas para crear una nueva cadena) en lugar de ser el operador aritmético de suma.

El método println se puede llamar sin contener ningún parámetro de tipo cadena. Esto está permitido y el resultado de la llamada será dejar una línea en blanco entre esta salida y cualquier otra que le siga.

2 En realidad, Java permite que los métodos void contengan una forma especial de sentencia de retorno en la que no se devuelve ningún valor. Esta sentencia toma la forma return; y simplemente hace que el método finalice sin ejecutar ninguna línea más de código.

2.11 Hacer Elecciones. La sentencia condicional

Una **sentencia condicional** realiza una de dos acciones posibles basándose en el resultado de una prueba; también son conocidas como *sentencias if*. Se evalúa el resultado de una verificación o prueba: si el resultado es verdadero entonces hacemos una cosa, de lo contrario hacemos algo diferente. Una sentencia condicional tiene la forma general descrita en el siguiente *pseudo-código:*

```
if (se lleva a cabo alguna prueba que da un resultado verdadero o falso) {
    Si la prueba dio resultado verdadero, ejecutar estas sentencias
}
else {
    Si el resultado dio falso. ejecutar estas sentencias
}
```

La prueba que se usa en una sentencia condicional es un ejemplo de una *expresión booleana.* Las **expresiones booleanas** tienen sólo dos valores posibles: verdadero o falso (true/false). Se las encuentra comúnmente controlando la elección entre los dos caminos posibles de una sentencia condicional.

2.13 Variables locales

Una **variable local** es una variable que se declara y se usa dentro de un solo método. Su alcance y *tiempo de vida* se limitan a los del método. Es muy común inicializar variables locales cuando se las declara; las declaraciones de las variables locales son muy similares a las declaraciones de los campos pero las palabras private o public nunca forman parte de ellas. Tal como ocurre con los parámetros formales, las variables locales tienen un alcance que está limitado a las sentencias del método al que pertenecen. Su tiempo de vida es el tiempo de la ejecución del método: se crean cuando se invoca un método y se destruyen cuando el método termina.

Los constructores también pueden tener variables locales. Las variables locales se usan frecuentemente como lugares de almacenamiento temporal para ayudar a un método a completar su tarea.

> **Cuidado**, una variable local del mismo nombre que un campo evitará que el campo sea accedido dentro de un método.
> Esto se soluciona con la palabra clave **this**:
> Al escribir this. nos estamos refiriendo al campo de la clase. (ver Capitulo 3)

2.14 Campos, parámetros y variables locales

Las tres clases de variables pueden almacenar un valor acorde a su definición de tipo de dato.

> **Los campos** se definen fuera de los constructores y de los métodos.
> Los campos se usan para almacenar datos que persisten durante la vida del objeto, de esta manera mantienen el estado actual de un objeto. Tienen un tiempo de vida que finaliza cuando termina el objeto.
> El alcance de los campos es la clase: la accesibilidad de los campos se extiende a toda la clase y por este motivo pueden usarse dentro de cualquier constructor o método de clase en la que estén definidos.
> Como son definidos como privados (private), los campos no pueden ser accedidos desde el exterior de la clase.

> **Los parámetros formales y las variables locales** persisten solamente en el lapso durante el cual se ejecuta un constructor o un método. Su tiempo de vida es tan largo como una llamada, por lo que sus valores se pierden entre llamadas. Por este motivo, actúan como lugares de almacenamiento temporales antes que permanentes.
>
> Los parámetros formales **se definen** en el encabezado de un constructor o de un método.
> Reciben sus valores desde el exterior, se inicializan con los valores de los parámetros actuales que forman parte de la llamada al constructor o al método.
>
> Los parámetros formales tienen un alcance limitado a su definición de constructor o de método.

Las variables locales se declaran dentro del cuerpo de un constructor o de un método.
Pueden ser inicializadas y usadas solamente dentro del cuerpo de las definiciones de constructores o métodos.
Las variables locales deben ser inicializadas antes de ser usadas en una expresión, no tienen un valor por defecto.
Las variables locales tienen un alcance limitado al bloque en el que son declaradas. No son accesibles desde ningún lugar fuera de ese bloque.

Nota. La clase String tiene, entre otros, los métodos de acceso substring y length, con las siguientes signaturas en Java:

```
/**
* Return a new string containing the characters from
* beginIndex to (endIndex-1) from this string.
*/
public String substring(int beginIndex, int endIndex)
```

El valor cero del índice representa el primer carácter de una cadena. Si endIndex es superior a la longitud de la cadena -1 Java genera un error.

```
/**
* Return the number of characters in this string.
*/
public int length( )
```

Es decir que el método de acceso length de la clase String de Java devuelve la cantidad de caracteres de una cadena.