MEDAC
Instituto Oficial de Formación Profesional

PROGRAMACIÓN

Clases, atributos y métodos

9

ÍNDICE

OBJETIVOS

Diferenciar entre atributos convencionales y estáticos.

Conocer los tipos de constructores que existen.

Conocer la funcionalidad de los métodos get y set.

Conocer la funcionalidad del método toString.

Documentar una clase con Doxygen.

Documentar los métodos de una clase con Doxygen.

/ 1. Introducción y contextualización práctica

En este tema tratará de los componentes de las clases, su funcionalidad y su sintaxis.

Vamos a hablar sobre los constructores, sus tipos y la importancia de su uso, así como sobre cómo hay crearlos de forma correcta.

También vamos a estudiar los métodos get y set de una clase, el porqué de su uso, y su importancia, así como también nos adentraremos en el método toString.

Por último, vamos a ver cómo podemos documentar correctamente una clase y todos sus métodos, y la importancia que conlleva hacerlo correctamente para un futuro.

Escucha el siguiente audio en el que planteamos el caso práctico que iremos resolviendo a lo largo de esta unidad.

Fig. 1. Programando.

Audio Intro. La programación dirigida a objetos

https://bit.ly/38RPEex

/ 2. Atributos

En programación dirigida a objetos, los atributos de las clases son las propiedades individuales que diferencian unos objetos de otros. Estos también son conocidos como '**campos' o 'variables'** de la clase.

Los atributos pueden ser tanto variables de tipo primitivo, como objetos de otra clase, por ejemplo, podemos tener un **dato de tipo String**[1] como variable de una clase.

Como vimos en lecciones anteriores, los atributos de una clase tienen un tipo de visibilidad, que normalmente, siempre va a ser privada, para así cumplir con los principios de encapsulación y ocultación de la información. Hay algunos casos en los que los atributos podrán ser públicos, pero por general deberán ser privados.

Lo primero que tendremos que definir después de crear una clase serán sus **atributos**, los cuales podremos cambiar cuando necesitemos, ya que, en un primer momento, probablemente no sepamos cuáles van a ser todos los atributos que necesitemos.

Procedamos a ver un ejemplo de creación de atributos de una clase. Vamos a crear una clase Persona, que tendrá como atributos nombre, apellidos, edad y dirección.

```
// Clase Persona
public class Persona {
    // Atributos de la clase Persona
    private String nombre, apellidos, direccion;
    private int edad;
}
```

Fig. 2. Clase Persona con sus atributos.

> **¡! Recuerda que...**
>
> Recuerda que los datos de tipo String instancian la clase String, por lo que son objetos, no variables primitivas.

2.1. Atributos estáticos

Los atributos estáticos, o atributos de clase, son unos atributos especiales que no dependen de los objetos, sino de las clases. Esto quiere decir que para acceder a ellos no necesitaremos crear un objeto de la clase, sino que desde la misma clase podremos llamarlos.

Estos atributos, al ser de clase, son compartidos por todos los objetos, es decir, que si se modifican, ese cambio será visto por todos los objetos que haya creados de la clase correspondiente o se creen en un futuro.

Estos atributos por norma serán públicos, he aquí una de las excepciones a que los atributos sean privados.

Para declarar un atributo como estático o de clase, utilizaremos la palabra reservada static, la cual pondremos delante del tipo del atributo que queramos crear estático.

Un ejemplo de atributo estático puede ser el siguiente: Imaginemos que creamos la clase Coche, con los siguientes atributos:

- **Marca, de tipo String**, que almacenará la marca del coche.
- **Modelo, de tipo String**, que almacenará el modelo del coche.

- **Matricula, de tipo String**, que almacenará la matrícula del coche.
- **Precio, de tipo double,** que almacenará el precio del coche.
- **Descuento, de tipo double**, que almacenará el descuento que se hará a los coches. Este descuento será común a todos los coches, por lo que el atributo será estático.

```
// Clase Coche
class Coche {
    // Atributos de la clase
    private String marca, modelo, matricula;
    private double precio;
    // Variable estática común para todos los objetos de coche
    static double descuento = 1500;
}

public class EjemploTema5 {

    public static void main(String[] args) {
        // Creamos un objeto de tipo coche
        Coche hyundaii30 = new Coche();
        System.out.println("El descuento es: " + Coche.descuento);
    }
}
```

Fig. 3. Ejemplo de atributo estático.

/ 3. Caso práctico 1: "¿Existe más de una forma de declarar las variables?"

Planteamiento. Pilar y José están realizando una clase que necesita de varios atributos, estos concretamente son los siguientes: 3 atributos de tipo int, 2 atributos de tipo String, 2 atributos de tipo double y 1 atributo de tipo int de clase. En su afán de perfeccionismo, Pilar los ha declarado en una única línea por cada tipo, mientras que José ha usado una línea por atributo.

Nudo. ¿Qué piensas sobre ello? ¿Cómo crees que deberían escribirse los atributos, mediante una declaración conjunta por tipo, o uno por cada línea? ¿Tendrá alguna repercusión en el comportamiento final de los objetos creados?

Desenlace. Como ya hemos visto en más de una ocasión, en programación se pueden hacer las cosas de diferentes formas y llegar al mismo resultado, obteniendo por tanto códigos que no se parecen en nada unos a los otros, pero que resuelven el mismo problema con los mismos resultados. En este caso, la declaración de variables puedes hacerse tanto de una como de otra forma, el resultado será el mismo, y la única diferencia será que en una de las formas habrá más líneas de código, pero nada más. Con el tiempo, normalmente los programadores suelen terminar utilizando la forma de declaración conjunta en una línea, ya que les ahorrará tiempo y alguna que otra línea de código.

Fig. 4. Diferentes caminos para ir obtener el mismo resultado.

/ 4. Métodos

Los métodos, junto a los atributos, son la otra parte que componen a las clases. Son un elemento fundamental y, al igual que no tenía sentido una clase sin atributos, no tiene sentido una clase sin métodos.

Tenemos diferentes tipos de métodos en las clases en Java:

- **Constructores**. Estos son los encargados de inicializar los objetos. Pueden ser por defecto o con parámetros.
- **Observadores**. Su misión es la de consultar el valor de los atributos de las clases.
- **Modificadores**. Con estos se modifica el valor de los atributos de las clases.
- **Métodos "personalizados"**. Son los métodos que tendremos que definir según las funcionalidades que necesitemos que ejecute nuestra clase.

Los métodos se aplican siempre a un objeto de la clase usando el operador 'punto'. Los métodos pueden llevar argumentos o parámetros, y devolver un dato de cualquiera de los tipos que hemos visto. Los argumentos deberán ir entre paréntesis. Solo podrá devolverse un único valor, mediante la instrucción return.

Los parámetros se podrán pasar por valor o por referencia, siendo la diferencia entre ellos que por referencia podrán ser modificados, mientras que por valor no, ya que serán una copia del valor original aportado. En Java, las variables primitivas (int, long, float, double, char y bool) se pasan por valor, mientras que los objetos se pasan siempre por referencia...

Cuando un método tenga parámetros, estos podrán ser utilizados como una variable normal y corriente dentro del método.

Dentro de los métodos podremos declarar variables, las cuales serán locales, es decir, que fuera de dicho método es como si no existieran, solo podrán ser usadas en el método donde se han declarado.

Fig. 5. Estructura de un objeto.

4.1. Constructores por defecto

Como hemos visto en el punto anterior, los **constructores** son los encargados de **inicializar los objetos de las clases**, no teniendo sentido una clase sin estos métodos, ya que los objetos no se podrían crear de forma correcta.

Los constructores, sean del tipo que sean, se han de llamar exactamente igual que la clase, y como peculiaridad, **no devolverán ningún tipo de dato**, ya que esta será la forma de reconocerlos para el compilador.

La forma de llamar a los constructores es mediante el operador new, al que llamamos cuando vamos a instanciar un objeto.

Si por despiste no creamos ningún constructor en una clase, el compilador creará uno por defecto, pero no inicializará los atributos de forma adecuada, solo servirá para que el proceso de compilación no falle.

Para evitar posibles fallos en un futuro, tomaremos la regla de que el constructor por defecto en una clase será de uso obligatorio.

La forma de inicializar los datos en un constructor por defecto será el ponerles un valor determinado, bien porque se nos indique en una especificación de algún desarrollo o ejercicio, o bien porque sepamos que han de tener dicho valor en su defecto. Si no sabemos qué poner podremos seguir las siguientes reglas:

- **Valor por defecto para enteros y reales**: 0.
- **Valor por defecto para String**: cadena vacía.
- **Valor por defecto para boolean**: verdadero.
- **Valor por defecto para otros objetos**: llamar al constructor por defecto de ese objeto.

Siguiendo el ejemplo de la clase Coche,

```
// Clase Coche
class Coche {
    // Atributos de la clase
    private String marca, modelo, matricula;
    private double precio;
    // Variable estática común para todos los objetos de coche
    static double descuento = 1500;

    public Coche()
    {
        marca = "";
        modelo = "";
        matricula = "";
        precio = 0;
    }
}
```

Fig. 6. Ejemplo de constructor por defecto.

4.2. Constructores con parámetros

Los constructores con parámetros son otro tipo de constructores, pero a diferencia que los constructores por defecto, estos **pueden llevar** tantos **parámetros** como necesitemos.

Podremos hacer tantos constructores con parámetros como **combinaciones diferentes** con los atributos de la clase podamos tener.

A la hora de crear los parámetros sería conveniente seguir la regla de ponerles el mismo nombre de la variable a la que dará valor, pero precedido por una 'n' (de nuevo), así, por ejemplo, si nuestra clase "Coche" tiene un atributo llamado "matricula", el nombre del parámetro será "matricula".

Puede darse el caso de que pongamos el mismo nombre al parámetro que a la variable de clase, en ese caso, tendremos que indicarle al constructor quién en quién. Para indicar quién es la variable de clase utilizaremos el operador this, que sirve para justamente esto, hacer referencia a las variables o métodos de la clase. El operador this puede usarse igualmente en cualquier parte de la clase, e independientemente de que las variables no tengan el mismo nombre.

Importante: En el caso de que no pasemos todos los valores de la clase por parámetros, tendremos que iniciar a su valor por defecto a aquellos que nos falten.

Siguiendo el ejemplo de la clase Coche su constructor será el que viene indicado en la siguiente figura.

```
public Coche(String nmarca)
{
    marca = nmarca;
    modelo = "";
    matricula = "";
    precio = 0;
}

public Coche(String nmarca, String nmodelo, String nmatricula, int precio)
{
    marca = nmarca;
    modelo = nmodelo;
    this.matricula = nmatricula;
    this.precio = precio;
}
```

Fig. 7. Ejemplo de constructores con parámetros.

4.3. Getters

Hasta el momento tenemos un problema con nuestras clases, sobre el que igual no hemos hecho hincapié. Y es que, los **atributos los tenemos que declarar privados**, y por ello, desde los objetos no podemos acceder a ellos... ¿y si queremos saber qué valores tienen?, ¿acaso no podemos?

Ya hemos comentado que para cumplir con los principios de encapsulación y ocultación de información los atributos de las clases han de ser privados, pero como hemos visto, si queremos obtener el valor de alguna de las variables no podremos. Para solucionar esto tenemos los métodos getters u observadores.

Los **métodos get** van a permitir obtener el valor de los atributos que queramos y cumplirán con los principios de encapsulación y ocultación de la información.

Normalmente necesitaremos generar **un método get** para cada **uno de los atributos** de nuestra clase. Estos métodos no recibirán ningún parámetro y devolverán el dato que queramos observar.

Los métodos get se nombrarán siempre de la misma forma:

[valor a devolver] get[NombreVariable] ()

Donde 'valor a devolver' será el mismo tipo de la variable que devolverá.

Estos métodos consistirán en una única línea con la instrucción return y el nombre de la variable a devolver.

Siguiendo el ejemplo de la clase Coche sus métodos get serán los que vienen indicados en la siguiente figura.

```
public String getMarca() {
    return marca;
}

public String getModelo() {
    return modelo;
}

public String getMatricula() {
    return matricula;
}

public double getPrecio() {
    return precio;
}
```

Fig. 8. Ejemplo de métodos get.

4.4. Setters

Ya hemos visto que disponemos de los métodos get para poder observar u obtener el valor de una variable de clase, pero ¿y si queremos modificar el valor de dicha variable?

Hasta ahora el hecho de modificar el valor de una variable, más allá de los constructores, no era posible, por el hecho de ser estas privadas.

Para resolver este problema tenemos los métodos setters, o modificadores, que serán los encargados de modificar el valor de una variable de nuestra clase, cumpliendo con los principios de encapsulación y ocultación de la información.

Normalmente necesitaremos crear un método set para cada uno de los atributos de nuestra clase. Estos métodos recibirán el nuevo valor a asignar en el atributo, como parámetro, y no devolverán ningún dato.

Los **métodos set** se nombrarán siempre de la misma forma:

void set[NombreVariable] ([tipo variable] [nombre parámetro])

Con la palabra reservada void estamos indicando que este método no devolverá ningún valor.

Estos métodos consistirán en una única línea con la igualación de la variable a su nuevo valor. Además, solo tendrán un único parámetro, ya que vamos a modificar el valor de una única variable de nuestra clase.

Una vez que llamemos a un método set y cambiemos el valor de un atributo, se sobreescribirá el valor anterior y no se podrá volver a recuperar.

```
public void setNombre(String nnombre)
{
    this.nombre = nnombre;
}
```

Fig. 9. Ejemplo de método set.

Video 1. "babytalk"
www.biti/75sfr.com

4.5. Método toString

El **método toString** es un método del que disponen las clases, y que **muestra un "resumen" del contenido** de las variables. Como es lógico, si recordamos, cada objeto es independiente de los demás, por tanto, el resultado del método toString deberá ser diferente en distintos objetos (a no ser que tengan los mismos valores en las variables).

Este método, se llamará y ejecutará automáticamente cuando mostremos por consola el valor de un objeto.

El modo de formatear la información dentro del método toString se puede hacer libremente, no habiendo un estándar definido para ello, pudiéndose mostrar, además, todas las variables de la clase o solamente las que se deseen. Incluso se podrá llamar a algún método.

Este método **no recibirá ningún parámetro** y **devolverá siempre un dato de tipo String**, donde estará la información con los valores de nuestro objeto formateados.

La cabecera de este método es:

```
String toString()
```

En una clase, podrá haber únicamente un método **toString**, no pudiendo ser duplicado.

Siguiendo el ejemplo de la clase Coche, su método toString será el que viene indicado en la siguiente figura:

```
@Override
public String toString() {
    String mensaje = "Datos del coche:\n";
    mensaje += "\tMarca: " + marca;
    mensaje += "\n\tModelo: " + modelo;
    mensaje += "\n\tMatrícula: " + matricula;
    mensaje += "\n\tPrecio: " + precio + "€";
    return mensaje;
}
```

Fig. 10. Ejemplo de método toString.

Pudiéndolos llamar de la siguiente forma:

```
public static void main(String[] args) {
    // Creamos un objeto de tipo coche
    Coche hyundaii30 = new Coche("Hyundai", "i30", "12345KLM", 16000);
    System.out.println(hyundaii30);
}
```

Fig. 11. Llamando al método toString.

4.6. Métodos personalizados

Vamos a llamar métodos personalizados a todos aquellos métodos que no sean constructores, get, set o el método toString.

Los métodos personalizados van a ser los métodos que hagan las operaciones necesarias, para las acciones que pueda necesitar nuestra clase.

Vamos a poder definir tantos métodos como necesitemos y éstos podrán recibir o no parámetros, y devolver o no un valor calculado a partir de las variables de la clase.

La cabecera de los métodos que devuelvan un valor será:

[valor devuelto] nombreMétodo([parámetros])

La cabecera de los métodos que no devuelvan un valor será:

void nombreMétodo([parámetros])

A estos métodos, podremos pasarles tantos parámetros como queramos, siendo su sintaxis:

tipo nombreparámetro

En caso de pasar más de un parámetro, los separaremos mediante comas.

Puede ocurrir que un método no tenga parámetros, en ese caso, no pondremos ningún parámetro y solo pondremos los paréntesis vacíos, es decir:

[valor devuelto / void] nombreMétodo ()

Los métodos que devuelven un valor se conocen como **funciones**, mientras que los que no devuelven un valor se conocen como **procedimientos**.

```
public double obtenerPrecioIVA()
{
    return this.precio * 1.21;
}
```

Fig. 12. Ejemplo de método que calcula el precio con IVA.

Audio 1. "El lenguaje del educador"
www.biti/75sfr.com

/ 5. Caso práctico 2: "¿Dar acceso completo o no?"

Planteamiento: Pilar y José siguen realizando la misma clase del ejercicio anterior, y se encuentran ahora con que deben permitir tanto el acceso como la observación a las variables de la clase, pero tienen la duda de si deben permitir el acceso a todas las variables o solo a algunas, además, tienen una variable estática o de clase.

Nudo: ¿Qué piensas al respecto? ¿Qué crees que es lo más correcto, dar acceso a todas las variables o solo a algunas? ¿Qué ocurre con la variable estática?

Desenlace: Cuando creamos una clase, lo más lógico es crear un método observador, o get, y un método modificador, o set, para todos y cada uno de los atributos de la clase. Pero en programación las generalizaciones están prohibidas, así que puede darse el caso en el que tengamos alguna variable de clase que no nos interese permitir que se vea su valor y/o que se modifique. Puede ocurrir, por ejemplo, porque sea una variable auxiliar, en la que nos sustentamos para realizar algún cálculo intermedio, necesario en alguno de los métodos de nuestra clase. En cuanto a los atributos estáticos, o de clase, no tiene sentido hacerles un método observador o modificador, ya que no son atributos de objeto, sino de clase.

Fig. 13. Mostrar todo el interior o no.

/ 6. Documentación de una clase

La documentación del código es una parte fundamental en cualquier programa, independientemente del lenguaje de programación que se utilice.

Cuando **documentamos código**, estamos **explicando cuál es la funcionalidad** del mismo, para así evitar tener que leer y entender el código que hay escrito.

Una de las formas más fáciles de documentar un programa es mediante **Doxygen.**

Doxygen es un generador de código que está disponible para infinidad de lenguajes de programación, Java entre ellos.

Doxygen significa generador de documentación para código fuente, y viene de DOX (documento) y GEN (generador).

Una de las cosas que podremos documentar con Doxygen serán las clases, para ello vamos a utilizar las siguientes etiquetas:

- **@version**. Esta etiqueta indica la versión de la clase.
- **@author**. Esta etiqueta indicará quién es el autor de la clase.

Todo el código de documentación Doxygen deberá ir entre /** y */.

Siguiendo el ejemplo de la clase Coche podríamos documentarla de la siguiente forma:

```
/**
 * Esta clase representa un coche
 * @version 1.0
 * @author Francisco Jesús Delgado Almirón
 */
public class Coche {
    // Atributos de la clase
    private String matricula, marca, modelo;
    private double precio;
```

Fig. 14. Documentación de la clase Coche.

/ 7. Documentación de los métodos

Además de las clases, otra cosa que debemos documentar son los métodos de las mismas.

Todos y cada uno de los métodos que desarrollemos en las clases se deberán documentar, ya sean funciones o procedimientos.

Los métodos "especiales", por llamarlos así, como son los constructores por defecto y con parámetros, métodos get, métodos set y el método toString también deberán documentarse.

La forma de documentarlos será de igual forma mediante el generador de documentación Doxygen, concretamente entre /** y */.

Las etiquetas de las que disponemos para documentar los métodos son las siguientes:

- **@param**. Esta etiqueta indicará qué representa un parámetro del método. Solo se utilizará en caso de que el método tenga parámetros, y se tendrá que poner etiqueta param por cada parámetro.
- **@return**. Esta etiqueta indicará qué es lo que devuelve el método. Solo se utilizará en caso de que el método devuelva algo, es decir, sea una función.
- **@deprecated**. Utilizada sólo cuando queramos indicar que el método está obsoleto.
- **@see**. Necesaria en caso de que en nuestro método llame a otro método, con la que indicaremos el nombre del método al que llamamos. Habrá que poner una por cada método que llamemos.
- **@throws**. Esta etiqueta indicará una excepción que lance nuestro método, mediante la que explicaremos el porqué de la excepción.

Habrá que poner una por cada excepción que se lance.

Video 1. "babytalk"
www.biti/75sfr.com

/ 8. Resumen y resolución del caso práctico de la unidad

A lo largo de este tema, hemos visto una de las partes fundamentales de una clase, los métodos.

Los métodos pueden ser de diferentes tipos: constructores, métodos observadores, métodos modificadores, el método toString y los métodos que hemos llamado "personalizados".

Hemos comprobado como mediante el uso de los constructores podremos inicializar los objetos de forma correcta, pudiendo éstos ser por defecto, o con tantos parámetros como necesitemos.

Hemos estudiado también los métodos get y set, que nos permitían poder obtener y modificar el valor de las variables de la clase cumpliendo con los principios de encapsulación y ocultación de la información. El método toString nos permitía tener un resumen de las características de nuestro objeto.

Por último, hemos visto cómo documentar con Doxygen una clase y todos sus métodos.

Resolución del caso práctico de la unidad

Cuando estamos utilizando la programación dirigida a objetos nos damos cuenta que vamos a tener muchos ficheros en nuestro proyecto, concretamente uno por cada clase que creemos, y eso puede ser un descontrol.

Al tener tantos ficheros debemos de poder organizarlos de alguna forma, y para ello ya hemos visto que tenemos los paquetes, pudiendo crear tantos como necesitemos para organizar las clases por funcionalidad y tenerlas más a nuestra disposición.

Pero, además, las clases pueden tener tantos métodos como necesitemos, y debemos de poder saber qué hace cada método de forma rápida para poder identificar qué clase necesitamos. Por ello, debemos documentar todas las clases junto a sus métodos, para así, saber cuál es su comportamiento.

Fig. 15. Hay que evitar perderse entre tanto código.

/ 9. Bibliografía

Sanz, M. A. (2019, enero 23). Curso de Java. Clases, Atributos, Modificadores, Objetos y Métodos. Recuperado 18 de mayo de 2020, de https://codesitio.com/recursos-utiles-para-tu-web-o-blog/cursos/curso-de-java-clases-atributos-modificadores-objetos-y-metodos/

Método toString para clases en Java | Disco Duro de Roer. (2013, noviembre 25). Recuperado 18 de mayo de 2020, de https://www.discoduroderoer.es/funcion-tostring-para-clases-en-java/

Doxygen. (2020, abril 18). Recuperado 18 de mayo de 2020, de https://es.wikipedia.org/wiki/Doxygen

Hossein, P. B. C. A. (2015, febrero 19). Doxygen una herramienta para documentar código |. Recuperado 18 de mayo de 2020, de https://booleanbite.com/web/doxygen-una-herramienta-para-documentar-codigo/