Objetos, clases y métodos. Parte I.

1 Objetos 2
2 Clases 2
2.1 Miembros de una clase 2
2.1.1 Atributos 2
2.1.2 Métodos 3
2.1.3 Ámbito de atributos y variables 3
2.2 Constructor y destructor 3
2.2.1 Constructor 4
2.2.2 Destructor 4
3 Objeto 5
3.1 Instancia 5
3.2 Estado y comportamiento 5
3.3 Mensaje 6

1 Objetos

Un objeto en POO representa alguna entidad de la vida real.

Cada objeto, al igual que la entidad de la vida real a la que representa, tiene un estado (es decir, unos atributos con unos valores concretos) y un comportamiento (tiene funcionalidades o sabe hacer unas acciones concretas).

Podemos decir que un objeto es cualquier elemento único del mundo real con el que se pueda interactuar.

2 Clases

Una clase es una plantilla a partir de la cual se crean los objetos. Dicho de otra forma, la clase es la abstracción de un objeto, la definición de un tipo de dato definido por el usuario.

Relación entre la clase y los objetos:

Un objeto, en vez de ser de un tipo básico (por ejemplo, int, char, float, etc.), es del tipo de una clase concreta definida por el programador.

En Java (y en muchos otros lenguajes), el código de una clase va en un único fichero cuyo nombre es el mismo que el de la clase. Es decir, el código de la clase *Persona* va en el fichero *Persona.java*. (Ver Ejemplo01)

2.1 Miembros de una clase

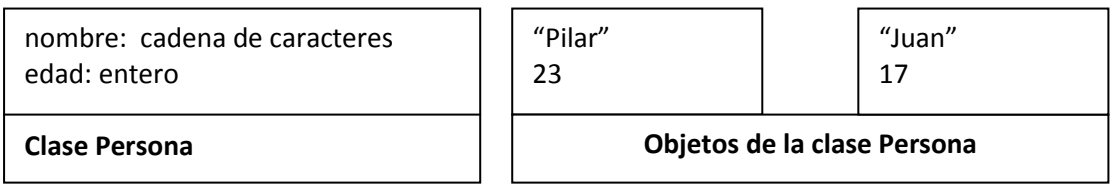
Como hemos visto, una clase está formada por unas variables y unas funciones. En términos de POO, a las variables se les llama **atributos** y a las funciones se les llama **métodos**. A su vez, al conjunto formado por los atributos y los métodos se le llama **miembros de una clase**. Así pues, tanto un atributo como un método son miembros de la clase.

2.1.1 Atributos

Los atributos, también llamados campos, son variables que codifican el estado de un objeto.

Si tenemos la clase *Persona* con los atributos *nombre* y *edad* –de tipo *cadena de caracteres* y *entero*, respectivamente–, cada objeto que se defina del tipo *Persona* tendrá estos dos atributos.

El estado de cada objeto *Persona* dependerá de los valores que se les asigne a estos dos atributos, tal como se ve en la figura siguiente.



Como podemos ver en la figura, cada objeto *Persona* tiene estados diferentes, es decir, valores distintos para cada atributo.

Aunque dos objetos compartan el mismo estado (es decir, tienen los mismos valores para todos sus atributos), estos dos objetos son diferentes. Solo hay que pensar que puede haber dos personas llamadas *Pilar* de edad *23* años en el mundo y, lógicamente, son personas (u objetos) diferentes.

Una clase **puede no tener** atributos.

Un atributo puede ser de tipo básico (entero, carácter, etc.) o de un tipo de clase concreta. Por ejemplo, de tipo *Persona*.

El tipo de los atributos se define como cualquier otra variable.

El **estado** de un objeto viene definido por los valores que toman en un instante determinado los atributos que definen al objeto.

2.1.2 Métodos

Los métodos implementan el comportamiento de un objeto o, mejor dicho, las funcionalidades que un objeto es capaz de realizar.

Un método, además de por el nombre, se caracteriza por los argumentos (también llamados parámetros) de entrada que recibe. La descripción de estos elementos se conoce como **firma del método o signatura del método**.

Sobrecarga de un método

Se produce cuando dos o más métodos tienen el mismo nombre pero diferente número y/o tipo de parámetros de entrada. El compilador decide qué método invocar comparando los argumentos de la llamada con los de la firma.

2.1.3 Ámbito de atributos y variables

Se llama "ámbito" al lugar donde puede ser utilizada una variable dentro de un programa.

Los **atributos de una clase**:

- Se pueden utilizar dentro de todos los métodos de dicha clase solo con poner el nombre.
- Se pueden utilizar fuera de la clase pero siempre a través del objeto y siempre que los modificadores de visibilidad den acceso a ellos. Como casi siempre son privados, fuera de la clase se accederá a ellos a través de los métodos públicos:

```
objeto.metodoDeAccesoAlAtributo();
```

Las **variables** que se declaran **dentro de un método** (variables **locales**) sólo pueden ser utilizadas dentro de dicho método. Cuando el método se cierra dicha variable desaparece.

Lo mismo ocurre con las variables que se declaran dentro de un **bloque**, que sólo pueden ser utilizadas dentro de dicho bloque. Un bloque es un grupo de cero o más sentencias encerradas entre llaves ({ y }). Se puede poner un bloque de sentencias en cualquier lugar en donde se pueda poner una sentencia individual.

2.2 Constructor y destructor

Las clases tienen dos tipos de métodos especiales llamados *constructor* y *destructor* que no se consideran miembros de una clase, como tales. No son miembros de la clase porque ni el constructor ni el destructor se heredan.

La mayoría de los lenguajes de programación orientados a objetos implementan el método constructor, incluso algunos obligan a codificar explícitamente uno. No ocurre lo mismo con el destructor, cuya codificación se puede obviar en muchos lenguajes, por ejemplo, en Java.

2.2.1 Constructor

El constructor se llama de forma automática cuando se crea un objeto para situarlo en memoria e inicializar los atributos declarados en la clase. En la mayoría de lenguajes, el constructor tiene las siguientes características:

1) El nombre del constructor es el mismo que el de la clase.
2) El constructor no tiene tipo de retorno, ni siquiera *void*.
3) Puede recibir parámetros (o argumentos) con el fin de inicializar los atributos de la clase para el objeto que se está creando en ese momento.
4) En general suele ser público, pero algunos lenguajes permiten que sea privado.

Hay lenguajes que permiten crear más de un constructor, por ejemplo C++, C# y Java, entre otros. Al constructor sin parámetros/argumentos se le suele llamar **constructor por defecto o predeterminado**, mientras que a aquellos que tienen parámetros se les llama **constructores con argumentos.** Esto es la **sobrecarga del constructor**.

En Java o en C++ y C#, si no se define ningún constructor para la clase, el propio compilador creará un constructor por defecto –es decir, sin argumentos– que no hará nada especial más allá de ubicar el objeto en memoria y asignar valores por defecto a los atributos.

Si en un método constructor no se establece un valor inicial para un atributo el propio compilador lo inicializará con su valor por defecto o valor predeterminado:

- El valor `0` para los tipos numéricos primitivos,
- El valor '\0 ' para el tipo char,
- El valor `false` para el tipo boolean, y
- El valor `null` para las referencias.

En el momento en que se define un constructor (aunque sea con argumentos), el compilador no añadirá automáticamente el constructor por defecto.

2.2.2 Destructor

El destructor es el método que sirve para eliminar un objeto concreto definitivamente de memoria, tanto la referencia al objeto como el espacio que ocupan sus datos. Hay que tener en cuenta que:

1) No todos los lenguajes necesitan implementar un método destructor, este es el caso de Java y C#.
2) Por norma general, una clase tiene un solo destructor.
3) En algunos lenguajes no tiene tipo de retorno, ni siquiera *void*. En otros, generalmente, tiene *void* como tipo de retorno.
4) No recibe parámetros.
5) En general suele ser público

La manera en la que se declara el método destructor varía entre lenguajes. Por ejemplo, en C++ y C#, el nombre del destructor es el mismo que el de la clase precedido por el símbolo ~, por ejemplo *~Persona()*.

En java no existen los destructores, esto es gracias al `Garbage Collector` (recolector de basura) de la máquina virtual de java. Este programa es un proceso de baja prioridad que se

ejecuta en la JVM. El ser de baja prioridad supone que no se está ejecutando constantemente, solo se le asigna su tarea cuando el procesador no tiene un trabajo con mayor prioridad en ejecución. Lo que hace es revisar la memoria y comprobar que todos los espacios que han sido reservados con un `new` tienen alguna referencia que apunta a ellos. Si no hay ninguna referencia que apunte a dicho espacio de memoria, lo libera para que vuelva a estar disponible.

Aunque Java maneja de manera automática el recolector de basura, el usuario también puede ordenar que se ejecute en un momento determinado con la instrucción:

```
System.gc()
```

Aunque la instrucción anterior poco o casi nunca se utiliza es importante saber que se puede llamar en cualquier momento.

Cuando el `Garbage Collector` libera un espacio de memoria, se ejecuta automáticamente el método `protected void finalize()`, siempre que dicho método esté implementado dentro de la clase a la que pertenece el espacio de memoria que se está liberando (no es obligatorio implementarlo).

3 Objeto

Vamos a ver en detalle en el concepto *objeto* y otros conceptos como **instancia** y **mensaje**.

3.1 Instancia

Como ya hemos comentado, los objetos son ejemplares de una clase. Así pues, a la hora de crear un objeto, debemos seguir los siguientes pasos:

1) Declarar la referencia al objeto.

```
Persona persona1;
```

2) Instanciar el objeto (crear un objeto a partir de una clase).

```
persona1 = new Persona("Pilar");
```

Debido a que la acción de crear un objeto a partir de una clase se le llama *instanciar*, muchas veces a los objetos se les llama *instancia*. Así pues, *persona1* es una instancia o un objeto de *Persona.*

3.2 Estado y comportamiento

Como hemos leído en la definición formal de **objeto**, todo objeto en la POO tiene un estado y un comportamiento. Esto es así porque los objetos (o entidades) de la vida real comparten estas dos características.

Debemos entender que el **estado** de un objeto viene definido por los valores que toman en un instante determinado los atributos que definen ese objeto.

Por su parte, el **comportamiento** del objeto se puede entender como las funcionalidades que ese objeto es capaz de realizar. Estas funcionalidades que definen el comportamiento de un objeto las define la clase a la que pertenece dicho objeto mediante los métodos de la clase. Por ejemplo:

- Un *coche* tiene un estado (velocidad actual, marcha actual, color, longitud, ancho, etc.) y un comportamiento (subir marcha, bajar marcha, encender intermitente, etc.).

- Una *factura* tiene un estado (cobrada o no, importe total, etc.) y un comportamiento (cambiar de no cobrada a cobrada y viceversa, modificar el valor del importe total, etc.).
- Un *televisor* tiene un estado (encendido o apagado, canal actual, volumen actual, etc.) y un comportamiento (encender, apagar, cambiar a un canal concreto, incrementar el número de canal, decrementar el número de canal, aumentar el volumen, disminuir el volumen, sintonizar, etc.).

Si un televisor concreto (el objeto) tiene el atributo canal actual igual a 5, ese televisor está en un estado diferente a si tuviera el canal actual igual al número 6.

Hay dos tipos de métodos:

1) Aquellos que hacen acciones que realiza la entidad real (por ejemplo, poner intermitente del coche, la acción de calcular el importe del IVA de la factura, etc.).
2) Aquellos que consisten en modificar o consultar el valor de los atributos del objeto (por ejemplo, modificar el número de teléfono de un contacto o el canal actual de un televisor) y, por consiguiente, cambian el estado del objeto. Estos métodos de modificación y consulta son los llamados ***setter*** y ***getter***, respectivamente.

3.3 Mensaje

Cuando los objetos quieren interactuar entre ellos utilizan **mensajes**. Un mensaje es la manera que existe de acceder a los atributos y métodos de un objeto.

La forma de un mensaje, en la mayoría de lenguajes, sigue la siguiente sintaxis:

```
variable_del_objeto.metodoPublico
```

Una vez definida una clase y **una vez instanciados los objetos** de dicha clase, **para interactuar con esos objetos y modificar su estado** (modificar los valores de sus atributos), **debemos provocar la ejecución de los métodos públicos que dichos objetos ofrecen**:

- Al hecho de **provocar la ejecución de un método de un objeto se le llama** "**realizar una petición**", "**solicitar un servicio**" o "**enviar un mensaje**" al objeto.

- Por su parte, desde el punto de vista del objeto, **recibir la solicitud para que ejecute uno de sus métodos recibe el nombre de** "**recibir una petición**", "**recibir una solicitud de servicio**" o "**recibir un mensaje**".