1.8 Iniciación a la Programación Orientada a Objetos

1. Introducción

La Programación Orientada a Objetos (***POO***) es una técnica de programar aplicaciones basada en una serie de objetos independientes que se comunican entre sí.

A Java se le considera un lenguaje orientado a objetos ya que siempre que se crea un programa en Java, por simple que sea, se necesita declarar una clase, y el concepto de clase pertenece a la programación orientada a objetos.

Un ***objeto*** es un elemento del programa que integra sus propios datos y su propio funcionamiento. Es decir, un objeto está formado por datos (atributos o propiedades) y por las funciones que es capaz de realizar el objeto (métodos). Esta forma de programar se asemeja más al pensamiento humano. La cuestión es detectar adecuadamente los objetos necesarios para una aplicación. De hecho hay que detectar las distintas clases de objetos.

Una ***clase*** es lo que define a un tipo de objeto. Al definir una clase lo que se hace es indicar como funciona un determinado tipo de objeto. Luego, a partir de la clase, podremos crear objetos de esa clase, es decir, la clase es como un molde a partir del cual se crean los objetos que pertenecen a ella. Realmente la programación orientada a objetos es una programación orientada a clases. Es decir, lo que necesitamos programar es como funcionan las clases de objetos.

Por ejemplo, una clase podría ser la clase *Coche*. Cuando se defina esta clase, indicaremos los atributos o propiedades (como el color, modelo, marca, velocidad máxima,...) y los métodos (arrancar, parar, repostar, acelerar, frenar...). Todos los coches, es decir, todos los objetos de la clase *Coche*, tendrán esas propiedades y esos métodos. Para explicar la diferencia entre clase y objeto:

- la clase *Coche* representa a todos los coches.

- un coche concreto es un objeto, es decir, un ejemplar de una clase es un objeto. También se le llama a los objetos *instancias* de la clase. Este término procede del inglés, *instance*, que realmente significa ejemplar.

Por ejemplo, si quisiéramos crear el juego del parchís en Java, una clase sería la casilla, otra las fichas, otra el dado, etc. En el caso de la casilla, se definiría la clase para indicar su funcionamiento y sus propiedades, y luego se crearían tantos objetos casilla como casillas tenga el juego. Lo mismo ocurriría con las fichas, la clase ficha definiría las propiedades de la ficha (color y posición por ejemplo) y su funcionamiento mediante sus métodos (por ejemplo un método sería mover, otro llegar a la meta, etc), luego se crearían tantos objetos ficha como fichas tenga el juego.

2. Creación de objetos

Una vez definida la clase, ya se pueden crear objetos de la misma. Para crear un objeto, hay que declarar una variable cuyo tipo será la propia clase.

Si por ejemplo definiéramos una clase llamada *Vehicle* para modelar vehículos, para crear un objeto tendríamos que declarar una variable de tipo *Vehicle*:

```
Vehicle car; //car es una variable de tipo Vehicle
```

Una vez definida la variable, se le crea el objeto llamando a un método que se llama constructor. Un constructor es un método que se invoca cuando se crea un objeto y que sirve para inicializar los atributos del objeto y para realizar las acciones pertinentes que requiera el mismo para ser creado. El constructor tiene el mismo nombre que la clase y para invocarlo se utiliza el operador **new**.

```
car = new Vehicle(); //Vehicle() es un método constructor
```

También se puede hacer todo en la misma línea:

```
Vehicle car = new Vehicle();
```

3. Acceso a los atributos y métodos del objeto

Una vez creado el objeto, se puede acceder a sus **atributos** de la siguiente manera: `objeto.atributo`.

```
car.wheelCount = 4;//Se le asigna 4 al atributo número de ruedas de la
variable car
```

Los **métodos** se utilizan de la misma forma que los atributos, a excepción de que los métodos poseen siempre paréntesis ya que son funciones que pertenecen a un objeto: `objeto.método(argumentos)`.

```
car.accelerate(30);
/*El coche incrementa su velocidad en 30. Es decir, si iba a 90km/h, después
de ejecutar el método el coche va a 120km/h */
```

4. Sobrecarga de métodos

Java admite sobrecargar los métodos, es decir, crear distintas variantes del mismo método con el mismo nombre pero que se diferencien en el orden, tipo o número de los parámetros.

Por ejemplo, tenemos el método para sumar `add(int x,double y)`:

- Sí podríamos definir el método `add(double x,int y)` porque varía el orden de los parámetros.

Otro ejemplo donde tenemos el método `add(int x,int y)`:

- No podríamos definir otro método `add(int a,int b)` porque no varía el tipo ni el número de parámetros.
- Sí podríamos definir `add(int a)` y `add(int a,int b,int c)` porque el número de parámetros varía.
- También podríamos definir `add(int x,double y)` porque aunque no varíe el número de parámetros, sí varía uno de los tipos.

5. Métodos estáticos y dinámicos

A los métodos asociados a los objetos se les conoce como métodos dinámicos. Pero puede ocurrir que tengamos métodos que no estén asociados a ningún objeto, por ejemplo, métodos de utilidad general. Dichos métodos se les conoce como métodos estáticos y se definen con la palabra **static**. Al no estar asociados a ningún objeto, se utilizan con el nombre de la clase: `Clase.metodoEstático(argumentos)`. Ejemplos de llamadas a métodos dinámicos y estáticos:

- Llamada a método dinámico: `car.accelerate(30);`
- Llamada a método estático: `Math.pow(2,3);`

6. Api de Java

La **API de Java** es una interfaz de programación de aplicaciones (**API**, por sus siglas del inglés: Application Programming Interface) provista por los creadores del lenguaje de programación **Java**, que da a los programadores los medios para desarrollar aplicaciones **Java**.

Al instalar Java (el paquete JDK) en nuestro ordenador, además del compilador y la máquina virtual de Java se instalan bastantes más elementos. Entre ellos, una cantidad muy importante de clases que ofrece la multinacional desarrolladora de Java y que están a disposición de todos los programadores listas para ser usadas. Estas clases junto a otros elementos forman lo que se denomina API de Java.

Los paquetes donde se encuentran dichas clases los podemos encontrar en https://docs.oracle.com → *Java* → *Java SE Documentation* → *JDK* de la versión deseada → *Specifications* → *API Documentation* → *Module: java.base*.

Otra manera de acceder rápido es poniendo en un buscador de Internet `Api Java Version Clase`, como por ejemplo: *Api Java 12 Math*.

7. La clase Math

La clase Math contiene los métodos para realizar operaciones matemáticas básicas, como potencias, logaritmos, raices cuadradas y funciones trigonométricas.

Si observamos esta clase en la API, todos los métodos tienen al principio la palabra *static,* ya que son métodos estáticos porque son funciones de utilidad que no se utilizan asociadas a un objeto. Después de la palabra static nos encontramos con el tipo del resultado que devuelve el método. Y a continuación, nos encontramos con el nombre del método y sus parámetros. Ejemplo:

`static double abs(double a)` Esto se conoce como la firma del método (**signature** en inglés).

```
package tema1_8_IniciacionPOO;

public class MathClass {

    public static void main(String[] args) {

        System.out.println(Math.abs(-3.2));
        System.out.println(Math.pow(2, 3));
        System.out.println(Math.sqrt(16));
        System.out.println(Math.min(20, 5));

    }

}
```

Obsérvese en la API la sobrecarga de los métodos *abs* y *min*.

8. La clase String

El texto es uno de los tipos de datos más importantes y por ello Java lo trata de manera especial. Para Java, las cadenas de texto son objetos especiales. Los textos deben manejarse creando objetos de tipo **String**.

Las cadenas se pueden inicializar de dos maneras:

- Usando el operador asignación: `String s="hola";`
- Usando el constructor: `String s=new String("hola");`

Los **literales** cadena se escriben entre comillas dobles: `"Esto es un literal cadena"`.

En java existe también la **cadena vacía o nula**, es decir, una cadena sin ningún carácter. Ejemplo: `String s="";` A la variable *s* se le está asignando la cadena vacía o nula.

Como vimos en el tema 1.5 Operadores, el operador concatenación `+` es un operador binario que devuelve una cadena resultado de concatenar las dos cadenas que actúan como operandos. Si sólo uno de los operandos es de tipo cadena, el otro operando se convierte implícitamente en tipo cadena.

Obsérvese en la API el método **valueOf**: es estático y está sobrecargado. Sirve para obtener la representación String de un valor u objeto.

```
package tema1_8_IniciacionPOO;

public class StringClass1 {

    public static void main(String[] args) {

        int i = 100;
        String string1, string2, string3, string4;

        string1 = "Esto es un literal cadena"; //Se le da un valor inicial a
la cadena con el operador de asignación =
        System.out.println(string1);
        System.out.println(string1 + " al cual le hemos concatenado este
literal cadena"); //Se concatena otra cadena con el operador +

        string2 = "hola";
        string3 = " que tal";
```

```
        string4 = string2 + string3;
        System.out.println(string4);

        System.out.println(i + 100); //Suma de enteros
        System.out.println(String.valueOf(i) + 100); //Concatenación de
cadenas
    }

}
```

Otros métodos de las cadenas muy útiles son:

- **charAt**: devuelve el carácter de la cadena del especificado índice. Dicho índice empieza en cero, es decir, con el cero se obtiene el primer carácter de la cadena.
- **length**: devuelve la longitud de la cadena.
- **equals**: compara si dos cadenas son iguales.

```
package tema1_8_IniciacionPOO;

public class StringClass2 {

    public static void main(String[] args) {

        String string = "hola";
        System.out.println(string.charAt(0));//h
        System.out.println(string.charAt(1));//o
        System.out.println(string.charAt(2));//l
        System.out.println(string.charAt(3));//a
        System.out.println(string.length());//4
        System.out.println(string.equals("hola"));//true
        System.out.println(string.equals("adiós"));//false

        //También se le pueden aplicar métodos a un literal cadena:
        System.out.println("hola".equals("hola")); //true
        System.out.println("adios".equals("hola"));//false

    }

}
```

9. Wrappers

En ocasiones es muy conveniente poder tratar los datos primitivos (int, boolean, etc.) como objetos. Pero los datos primitivos no son objetos. Para resolver esta situación, la API de Java incorpora las clases envoltorio (**wrapper class**), que no son más que dotar a los datos primitivos con un envoltorio que permita tratarlos como objetos. Las clases envoltorio proporcionan métodos de utilidad para la manipulación de datos primitivos (conversiones de / hacia datos primitivos, conversiones a String, etc).

La siguiente tabla muestra los tipos primitivos y sus wrappers asociados:

Tipo primitivo	Wrapper asociado
byte	Byte
short	Short
int	Integer

Tipo primitivo	Wrapper asociado
long	Long
float	Float
double	Double
char	Character
boolean	Boolean

Todos estos wrappers los encontraremos en la API de Java. Por ejemplo, si observamos en la API la clase Integer, podemos ver la siguiente firma de método: `static int parseInt(String s)` que convierte la cadena pasada por parámetro a entero.

```
package tema1_8_IniciacionPOO;

public class IntegerClass {

    public static void main(String[] args) {

        Integer integer1, integer2;
        int i;
        integer1 = 5;
        System.out.println(integer1);
        i = Integer.parseInt("7");//Convierte la cadena a int. Método estático
por lo que se utiliza con Integer
        System.out.println(i);
        integer2 = Integer.valueOf(i);//Convierte el int a Integer. También es
estático
        System.out.println(integer2);
        i = integer1.intValue();//Convierte el Integer a int. Método dinámico
por lo que se utiliza con el objeto
        System.out.println(i);

    }

}
```

10. Encadenamiento de llamadas a métodos

Se emplea cuando invocamos a un método de un objeto que nos devuelve como resultado otro objeto al que podemos volver a invocar otro método y así encadenar varias operaciones.

```
package tema1_8_IniciacionPOO;

public class CallsToMethods {

    public void showCallsToMethods() {

        Boolean b;
        String string;

        string = "EntornosDeDesarrollo";
        System.out.println(string.substring(10).toUpperCase()); //DESARROLLO

        b = Boolean.TRUE;
        System.out.println(b.toString().charAt(2)); //u
```

```
    }

    public static void main(String[] args) {

        new CallsToMethods().showCallsToMethods();

    }

}
```

En este ejemplo, el método `substring(10)` está devolviendo una subcadena de la cadena string a partir del carácter 10 empezando en 0, es decir, "Desarrollo", al que se le invoca luego el método `toUpperCase` devolviendo como resultado la cadena "DESARROLLO".

Y el método `toString()` de la variable b de tipo Boolean está devolviendo la cadena "true" a la que se le encadena el método `charAt(2)` devolviendo el carácter 'u'.