







OBJECT-ORIENTED PROGRAMMING

1. Introducción

 Date	Empty
 Status	Not started
 Type	Empty
Units	 5. Orientación a objetos

Introducción a la programación orientada a objetos

La programación orientada se trata de un paradigma que permite el desarrollo de programas informáticos de manteniendo un código ordenado y manejable, de forma que sea mucho más sencillo el desarrollo. Este tipo de programación acerca mucho más los conceptos de la vida real a la forma en la que un programador lo debe transmitir al ordenador la información. Hasta este punto toda la programación que se ha hecho ha sido una programación estructurada, donde las líneas de código se ejecutaban una tras otra tan solo saltando entre ellas cuando algún bloque de ejecución lo decía (if, while, switch). Con este tipo de programación nos podemos encontrar muchas limitaciones, como por ejemplo intentar ejecutar códigos completos para diferentes objetos primitivos. Para poder entender el concepto de programación orientada a objetos se puede pensar en un videojuego, donde cada uno de los componentes que forman la pantalla es un objeto que tiene unas características diferentes y son capaces de interactuar entre ellos.

Para poder entender mejor el concepto, antes es necesario conocer los elementos que forma la programación orientada a objetos:

- **Paquetes**

Los paquetes son los contenedores que permiten organizar cada una de las clases que se crean en un programa. Normalmente se suelen utilizar para hacer agrupaciones lógicas, donde cada una de las clases que pertenecen al mismo paquete tendrán una similitud en funcionamiento. Además la pertenencia o no al mismo paquete podrá marcar el acceso a la clase, dependiendo de la visibilidad que sea configurada

- **Clases**

Se trata de archivos .java que representa una funcionalidad completa, un modelo para poder crear un tipo de datos completo. Hasta ahora se han utilizado datos primitivos (int, double, boolean) y algunos complejos (String, Object) pero estos son elementos que ya están creados en el lenguaje. En el caso de que nosotros necesitemos un tipo de dato concreto, se crea una clase que representará dicho tipo (en concreto representará un objeto). Pensad en la necesidad de tener un coche en código de programación, por lo que se necesitará crear un tipo *Coche* el cual podrá ser creado

```
// archivo coche.java public class Coche { }
```

- **Propiedades**

Las propiedades son todas aquellas características que los tipos que un programador crea (las clases). En el ejemplo anterior, el tipo *Coche* tendrá propiedades como *velocidad*, *bastidos*, *modelo*, etc...

```
public class Coche { int velocidad; String bastidor, modelo, marca; }
```

- **Métodos**

Los métodos son todas las funcionalidades que los tipos que un programador crea (las clases) tienen. Estos métodos actúan sobre las propiedades de los mismos. En el ejemplo anterior, imaginad que el tipo *Coche* creado en el punto anterior necesita tener una acción que

sea acelerar. Para ello se creará un método con el nombre *acelerarCoche* donde se define el comportamiento del *Coche* cuando se acelera

```
public class Coche { int velocidad; String bastidor, modelo,
marca; public void acelerar(int velocidadAcelerar){ velocidad
= velocidad +velocidadAcelerar; } public void frenar(int velo
cidadFrenar){ velocidad = velocidad -velocidadFrenar; } }
```

- **Visibilidad**

La visibilidad de una clase, métodos y propiedades el el acceso que cada uno de los elementos tiene, desde donde podrán ser vistos. Más adelante se verá con detalle pero cada uno de los elementos podrán tener una visibilidad *public*, *protected*, *private*

- **Objetos**

Cuando se desarrolla una clase se dice que en realidad lo que se está haciendo es un modelo que puede ser utilizado en cualquier parte del código de programación. Para poder utilizar estas clases (con toda la funcionalidad que se define en su interior) es necesario implementar o crear un objeto de la propia clase. Digamos que la clase representa el molde y el objeto representa la clase llevada a la realizada (el concepto es instancia). Para poder crear un objeto (ya hemos utilizado simples y complejos) se utiliza la palabra ***new***

```
// crea un objeto de tipo teclado (por lo tanto existe una cl
ase de tipo teclado) Scanner teclado = new Scanner(System.in)
// crea un objeto de tipo String array String[] arrayPalabras
= new String[]
```

Hasta este punto hemos utilizado objetos de tipo primitivo como los *int*, *float*, etc... que no necesitan palabra *new* al ser datos muy simples, pero también en algunos casos se ha utilizado la palabra *new* para poder crear aquellos tipos complejos que nos permiten una funcionalidad un poco más avanzada (uso de sus métodos).

```
Scanner teclado = new Scanner(System.in); teclado.nextInt();
```

Todos los datos complejos que hemos utilizado son tipos que ya están creados en el sistema, pero gracias al uso de las clases que se han definido al principio, nosotros podemos crear nuestros propios tipos para crear objetos. Imaginad que en el programa se necesita utilizar un tipo especial que represente un coche:

```
// archivo coche.java public class Coche { String bastidor; int
caballos, velocidad; public void acelerar(int v){ this.velocidad = this.velocidad + v; } public void decelerar(int v){ this.velocidad = this.velocidad - v; } public void reprogramar(int cv) { this.caballos = cv; } }
```

Mediante esta clase ya se tiene la posibilidad de crear un objeto de tipo Coche desde partes del programa con el acceso a todas sus funcionalidades. Para poder utilizarlo hay que utilizar la palabra reservada new

```
public class Entrada { public static void main(String[] args) { Coche cocheUno = new Coche(); cocheUno.acelerar(100); Coche cocheDos = new Coche(); cocheDos.acelerar(50); } }
```

Lo interesante de esto es que una vez se ha creado la clase (el molde para poder generar objetos), se podrán crear tantos objetos como se quiera teniendo la posibilidad de personalizarlos