2. Conceptos

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

La metodología de programación orientada a objetos no es exclusivo del ámbito informático, ya que si pensamos en cualquier aspecto de una cadena de montaje o proceso de producción nos encontramos con una forma muy similar de ejecutar las cosas. Suponer el ejemplo de un coche, donde cada elemento está compuesto por multitud de piezas que todas juntas hacen que el coche funcione correctamente. Es imposible tener un plano o diseño donde aparezcan todas y cada una de las piezas, sino que cada parte del coche tiene su propio plano donde a su vez se divide en muy pequeñas funcionalidades: motor, chasis, cableado, etc...

Una vez se han comprendido la funcionalidad de la orientación a objetos y los elementos que la hacen posible en java, es importante tener en cuenta los principales conceptos que hacen de la orientación a objetos un mecanismo muy potente a la hora de realizar un programa informático: encapsulación, herencia y polimorfismo

Encapsulación

Encapsulación

Consiste en ocultar aquellos elementos que no se quieren mostrar cuando un objeto es utilizado, pero son necesarios para el funcionamiento de este. Si volvemos al ejemplo de la fabricación de un coche, un conductor no necesita saber como funciona la caja de cambio o el motor, simplemente lo utiliza sin necesidad de conocer su funcionamiento al detalle. En informática pasa exactamente lo mismo; en un programa informático se utilizarán objetos que no es necesario que quién lo esté utilizando sepa con exactitud su funcionamiento.

Como se ha visto en el punto anterior, una clase cuenta con propiedades y métodos. Precisamente son las propiedades las que sirven como ejemplo perfecto para el tema de la encapsulación, ya que en la mayoría de los casos (excepto casos de uso de static) las propiedades o variables deben ir en privado

```
public class Coche { private String marca, modelo; private
int cv, cc; }
```

Con esto se consigue que nadie sea capaz de acceder a las propiedades del objeto, ni siquiera instanciandolo.

```
public class Main { public static void main(String[] args) {
Coche coche = new Coche(); System.out.println(coche.marca);
// esto daría error al ser un atributo encapsulado } }
```

Para poder acceder a las propiedades del objeto sería necesario hacerlo a través de los métodos getter y setter, que en este caso si son públicos

```
public class Coche { private String marca, modelo; private
int cv, cc; public String getMarca() { return marca; } public
void setMarca(String marca) { this.marca = marca; } public
String getModelo() { return modelo; } public void
setModelo(String modelo) { this.modelo = modelo; } public int
getCv() { return cv; } public void setCv(int cv) { this.cv =
cv; } public int getCc() { return cc; } public void setCc(int
cc) { this.cc = cc; } }
```

Y en el caso de querer hacerlo se podría acceder a las propiedades mediante estos métodos

```
public class Main { public static void main(String[] args) {
Coche coche = new Coche();
System.out.println(coche.getMarca()); } }
```

Herencia

En el lenguaje de programación Java se permite la creación de clases a través de clases ya existentes, cogiendo todos sus métodos - variables y haciéndolas propias de la clase. Este concepto es muy importante ya que de esta forma se puede realizar clases con muy poco código, siempre especializando clases superiores. Además esta característica del lenguaje permite utilizar el concepto de *polimorfismo*. Para poder utilizar la herencia se usa la palabra reservada extends

```
// archivo Coche.java public class Coche { String bastidor; i
nt caballos, velocidad; public void acelerar(int v){ this.vel
ocidad = this.velocidad + v; } public void deceledad(int v){
this.velocidad = this.velocidad - v; } public void reprograma
r(int cv) { this.caballos = cv; } }
```

```
// archivo Deportivo.java public class Deportivo extends Coch
e{ int par; int cilindros; public void calcularParMotor(){ pa
r = (velocidad * cilindros)/2; } }
```

Por defecto la clase Deportivo no tiene ninguna variable llamada velocidad, pero al haber extendido de Coche que si lo tiene, está disponible para la clase.
Se suele decir que las clases que extienden de otras son especializaciones. En Java, todas las clases son subclases de la superase Object

Polimorfismo

Este concepto va de la mano del visto en el punto anterior. Gracias a esta característica una clase puede ser utilizada con un tipo diferente

esta característica una clase puede ser utilizada con un tipo diferente dependiendo de las necesidades del programa. En el ejemplo anterior si se quería instancia un objeto de los tipos coche y deportivo se hacía de la siguiente forma

```
Coche c = new Coche() Deportivo d = new Deportivo()
```

Gracias a que la clase Deportivo ha extendido de coche, también podría valer la siguiente instancia

```
Coche d = new Deportivo
```

En el siguiente tema trataremos este concepto de forma profunda