M.3611.056.003 - Curso de Introducción a la programación con JAVA (Presencia Virtual Online - Cádiz)

POO1: Clases y Objetos

Eduardo Corral Muñoz
eoi_eduardo@corral.es

Programación Orientada a Objetos 1

Índice

— 01

Introducción

Programación Orientada a Objetos
Object-Oriented Programming (OOP)

- Estamos familiarizados con objetos de la vida real que tienen sus características y comportamientos.
- El paradigma de la OOP es llevar los objetos reales al mundo virtual, a la programación.
- Los objetos son entidades que tienen estado (atributos) y comportamiento (métodos).
- Los objetos se crean como copia de un “molde” → una Clase

Programación Orienta a Objetos / OOP

- La **programación tradicional** (procedimental) trata de escribir procedimientos o métodos para realizar operaciones con los datos.
- La **programación orientada a objetos** trata de crear objetos que contienen tanto métodos como datos.

OOP - Ventajas

- Es más rápida y fácil de ejecutar.
- Dota a los programas de una estructura clara.
- Ayuda a mantener el código Java libre de repeticiones (**DRY**), haciéndolo más fácil de mantener, modificar y depurar (refactorizar).
- Hace posible la creación de aplicaciones completamente reutilizables con menos código y menor tiempo de desarrollo.

OOP - DRY

- DRY = Don’t Repeat Yourself = No te repitas
- Es un principio de programación para reducir la repetición de código.
- Se extraen del código de la aplicación aquellos bloques que son comunes o repetidos y se ponen en funciones o métodos que son llamados donde sea preciso en lugar de repetir el bloque de código.

— 02

Clases y objetos

Clases y Objetos

- Son la base de la programación orientada a objetos.
- Una **clase** es un modelo o prototipo a partir del cual se crean los objetos. Las clases básicamente modelan cómo debe verse y qué debe hacer un objeto (estado y comportamiento), con atributos y métodos. (Coche)
- Un **objeto** es una abstracción utilizada en el desarrollo de software para representar un objeto de la vida real, que tiene estado y comportamiento. Los objetos de software se usan a menudo para modelar objetos del mundo real que se encuentran en la vida cotidiana. (Toyota Yaris Cross)

Clases y Objetos

Clase	Objetos
Fruta	Manzana Naranja Platano …
Coche	Volvo BMW Mazda …
Perro	Setter Galgo Caniche …

Clases y Objetos

- Una clase es una plantilla:

  Tiene definidos unos atributos (variables - representan el estado).

  Tiene definidos unos métodos (representan el comportamiento).

  Una clase es como un objeto constructor, los “planos” para crear objetos.

- Un objeto es una instancia (una replica) de una clase:

  Al crearlo (instanciarlo) hereda todos los atributos y métodos que tiene definidos la clase.

Clases y Objetos

- Crear una clase:

Clase Coche	
Atributos	marca modelo color cilindrada …
Métodos	acelerar frenar …

```
public class Coche {
        String marca;
        String modelo;
        String color;
        double cilindrada;
        double kilometros;
        public void acelerar(){
                …
        }
        public void frenar(){
                …
        }
}
```

Clases y Objetos

- Crear (instanciar) un objeto de tipo Coche:

```
public class MiClase {
    public static void main(String[] args){
        Coche miCocheNuevo = new Coche();
        miCocheNuevo.marca = “Toyota”;
        miCocheNuevo.modelo = “Yaris Cross”;
        …
        System.out.println(miCocheNuevo.marca);
    }
}
```

Clases y Objetos

- Los **atributos** (características) de una clase son las variables declaradas dentro de ella.
- Se las conoce también como campos (fields).
- El modificador **final** en la declaración de un atributo lo convierte en constante (como pi) y no es posible cambiarlo durante la ejecución del programa.
- En un objeto creado (instanciado) desde una clase, podemos acceder a los atributos leer/cambiar su valor con:

miObjeto.atributo = nuevo valor;

Clases y Objetos

- Los **Métodos** de una clase son los métodos declaradas dentro de ella (comportamiento).
- Se suelen declarar con modificadores **static** o **public.**
- **static** implica que el método declarado puede ser llamado sin tener que crear un objeto de la clase.

```
static void myStaticMethod() {
    System.out.println(“Método static – Se le puede llamar sin crear objetos");
}
```

- **public** se emplea para declarar un método que solo puede ser llamado por objetos.

```
public void myStaticMethod() {
    System.out.println(“Método public– Para llamarlo hay que crear objetos");
}
```

Clases y Objetos

- Cuando en un **método** de una clase nos referimos a un atributo o método de la misma clase, para referenciarlo empleamos la palabra clave **this**.

```
public void detallesCoche(){
    System.out.println("Marca: " + this.marca);
    System.out.println("Modelo: " + this.modelo);
    System.out.println("Color: " + this.color);
    System.out.println("Cilindrada: " + this.cilindrada);
    System.out.println("Kilómetros: " + this.kilometros);
}
```

Clases y Objetos

- Dentro de una clase podemos **sobrecargar** métodos para que trabajen con diferentes tipos o número de parámetros.

```
public float consumo(int distancia, float porcentajeDeposito){
    return distancia/(capacidadDeposito*porcentajeDeposito);
}

public float consumo(int distancia, int porcentajeDeposito){
    return distancia/(capacidadDeposito*(porcentajeDeposito/100));
}
```

- Desde le punto de vista del objeto solo existe un método al que se puede llamas con diferentes tipos de parámetro, están **encapsulados** dentro de la clase.

Clases y Objetos

- Podemos declarar atributos y/o métodos estáticos, que pueden ser modificados o llamados desde otra clase con el nombre de la clase seguido del nombre del atributo o método.

  NombreClase.atributo = nuevo valor;

  NombreClase.metodo(params);

- Un método estático, no puede hacer referencia a atributos de la clase que no sean estáticos.
- Si cambiamos el valor de un atributo declarado como estático en la clase, ese cambio afecta a todas las instancias de esa clase que se hayan creado, ya se haga sobre la clase o sobre cualquiera de los objetos de esa clase.

— 03

Encapsulado

Encapsulado

- Es una buena práctica que los atributos de una clase sean privados o protegidos y que solo se pueda acceder a ellos mediate métodos que permitan leer o escribir su contenido.
- Asegurarse de que los datos “sensibles” están ocultos para los usuarios.
- Declarar los atributos (variables) de la clase como private**:**

private tipo atributo;

- Los atributos con estos modificadores dejan de ser accesibles directamente en el objeto instanciado y para poder acceder a ellos debemos añadir los métodos **set** y **get** (nombres estandarizados) para todos los atributos.

Encapsulado

- Solo se puede acceder a los atributos privados (**private**) desde el interior de la propia clase a la que pertenecen. Fuera de esa clase no están disponibles.
- Solo se podrá acceder a ellos con el objeto instanciado usando métodos **get** y **set**, que han de ser públicos (**public**).
- Por cada atributo privado crearemos los métodos **getNombre(),** para obtener su valor y **setNombre**(), para modificarlo o actualizarlo.

```
public String getMarca(){
      return this.marca;
}
…
```

```
public void setMarca(String marca){
      this.marca = marca;
}
…
```

Encapsulado

- No es necesario teclear para crear los getters (métodos get) y setters (métodos set). Nuestro IDE lo hace automáticamente, en el menú contextual:

Generate… → Getter and Setter → seleccionar atributos → OK

Encapsulado

- Mejora el control de los atributos y métodos de clase.
- Podemos hacer un atributo de solo-lectura creando solo el método get asociado pero no el método set.
- Podemos hacer un atributo de solo-escritura creando solo el método set asociado pero no el método get.
- Es flexible, el programador puede cambiar parte del código sin afectar al resto de las partes.
- Incrementa la seguridad de los datos.

04

Constructor

Constructor

- Método que nos permite crear un objeto a partir de una clase.
- Por defecto, si no definimos un constructor, podemos crear el objeto con **new NombreClase()**, construyendo un objeto “vacio”, sin valores iniciales en los atributos.
- Para crear un objeto personalizado hemos de definir un método **constructor** al que podamos “pasar” algunos parámetros que inicialicen los atributos del objeto.
- El nombre del método ha de ser el mismo que la clase y no retorna ningún valor, ni void.

Constructor

- Podemos crear el método con o sin parámetros:

```
// sin parámetros
public NombreClase(){
    setAttr1 = “valor1”;
    …
}

// con parámetros
public NombreClase(tipo attr1, tipo attr2){
    this.attr1 = attr1;
    this.attr2 = attr2;
    …
}
```

Constructor

- Podemos crear varios constructores con distinto número de parámetros, sobrecargar el método, desde ningún parámetro hasta todos los parámetros de la clase.
- Diferentes formas de crear el objeto → Polimorfismo
- Los métodos constructores se pueden llamar entre sí, añadiendo parámetros al método anterior.
- No es necesario teclear para crear un método constructor. Nuestro IDE lo hace automáticamente, en el menú contextual:

  Generate… → Constructor → seleccionar atributos → OK