Introducción a la programación orientada a objetos

Programación

Introducción

- La programación orientada a objetos es uno de los paradigmas de programación más utilizados en la actualidad.
- La programación orientada a objetos es programación imperativa, es decir, cada sentencia especifica qué y cómo se va a ejecutar el programa.
- Este paradigma supuso una revolución en su día al incluir novedosos conceptos que permitían generar código de mayor calidad, facilitar el desarrollo y reutilizar el código de una forma más sencilla
- Algunos lenguajes de programación orientada a objetos son Java, C#, C++, Go, Kotlin...

Clases

- Las clases son el elemento básico de este paradigma.
- Una clase describe un conjunto de datos relacionados y un conjunto de operaciones que se pueden realizar sobre ellos.
- Podemos ver una clase como una plantilla que nos permite representar objetos que conceptualmente representan algo parecido.
- Las clases se componen de:
  - Campos: son variables que definen el conjunto de valores que conforman la clase
  - Métodos: cada método define una operación que los objetos de
  - Constructores: son métodos que sirven para crear objetos de la clase

Ejemplo de clase persona

```java
public class Persona {

    String nombre;
    String apellidos;
    String curso;


    public Persona() {
    }

    public void saluda() {
        System.out.println("Hola soy " + nombre + " " + apellidos + " del curso " + curso);
    }
}
```

Objetos

- Cuando creamos una instancia de una clase tenemos un objeto.
- Un objeto es una combinación de valores concretos de los tipos definidos en su clase
- Por ejemplo, si tenemos una clase Alumno que define los siguientes campos:
  - Nombre
  - Apellidos
  - Curso
- Un objeto de esa clase podría ser un alumno llamado Fulanito López Sánchez de 1º de DAM y otro objeto podría ser Menganita Martínez Álvarez.

Ejemplo de objetos

```java
public static void main(String[] args) {
    Persona fulatino = new Persona();

    fulatino.nombre = "Fulanito";
    fulatino.apellidos = "López Sánchez";
    fulatino.curso = "1º DAM";

    Persona menganita = new Persona();

    menganita.nombre = "Menganita";
    menganita.apellidos = "Martínez Álvarez";
    menganita.curso = "2º DAM";

    fulatino.saluda();
    menganita.saluda();
}
```

Campos

- Es un elemento que se comporta como una variable propia de cada instancia de una clase y que define un atributo de un objeto.
- En ellos se almacenan datos, sus valores pueden ser inicializados, modificados y leidos desde la propia instancia.
- Un campo puede ser de cualquier tipo: tipos primitivos o clases.
- Se pueden inicializar en desde el constructor o directamente en la línea que están declarados

Ejemplo de clase persona

```java
public class Persona {

    String nombre;
    String apellidos;
    String curso;


    public Persona() {
    }

    public void saluda() {
        System.out.println("Hola soy " + nombre + " " + apellidos + " del curso " + curso);
    }
}
```

Métodos de objeto

- Hasta ahora hemos visto métodos estáticos, también conocidos como métodos de clase, esos métodos se podían invocar sin necesidad de instanciar un objeto de la clase en la que están definidos.
- Los métodos de objeto (no estáticos) solo se pueden invocar cuando existe un objeto de la clase en la que están definidos. Por ejemplo en la clase String tenemos el método split, para poder ejecutar ese método necesitamos una instancia de String.
- Estos métodos pueden acceder a los valores almacenados en los campos del objeto.
- La definición de métodos de objeto sigue las mismas reglas que los métodos de clase.

Ejemplo de método de objeto

```java
public class Persona {

    String nombre;
    String apellidos;
    String curso;


    public Persona() {
    }

    public void saluda() {
        System.out.println("Hola soy " + nombre + " " + apellidos + " del curso " + curso);
    }
}
```

Ejemplo de objetos

```java
public static void main(String[] args) {
    Persona fulatino = new Persona();

    fulatino.nombre = "Fulanito";
    fulatino.apellidos = "López Sánchez";
    fulatino.curso = "1º DAM";

    Persona menganita = new Persona();

    menganita.nombre = "Menganita";
    menganita.apellidos = "Martínez Álvarez";
    menganita.curso = "2º DAM";

    fulatino.saluda();
    menganita.saluda();
}
```

Constructores

- Son métodos que sirven para instanciar objetos de la clase en la que están definidos.
- Tienen el mismo nombre que la clase
- Normalmente reciben como parámetro valores que se van a asignar en los campos del objeto
- Se pueden definir varios constructores en una misma clase, siempre y cuando difieran en los parámetros que definen.
- Si no se define ningún constructor el compilador crea un método constructor por defecto que inicializará los campos a los valores por defecto que les corresponda

Ejemplo de constructor

```java
public class Persona {

    String nombre;
    String apellidos;
    String curso;


    public Persona() {
    }

    public void saluda() {
        System.out.println("Hola soy " + nombre + " " + apellidos + " del curso " + curso);
    }
}
```

Ejercicio

- Crea una clase para guardar información de personas
- De cada persona queremos guardar:
  - Nombre
  - Apellidos
  - DNI / NIF
  - Edad
- Define los siguientes métodos vinculados a cada persona:
  - Saludo: imprimirá en pantalla el mensaje "Hola soy <nombre> <apellidos> y mi DNI/NIF es <DNI>"
  - Despedida: imprimirá en pantalla el mensaje "Hasta la próxima! Firmado <nombre>"
- Instancia un objeto Persona que almacene tus datos personales, una vez creado invoca el método de saludo y después el de despedida.

Principios de la POO

- Encapsulamiento: cada objeto contiene un estado concreto (los valores de los campos de ese objeto), el encapsulamiento consiste en mantener ese estado aislado del exterior, de forma que solo se pueda interaccionar con él a través de los métodos que se proporcionen para ello.
- Abstracción: La abstracción consiste en que cada objeto expone únicamente la funcionalidad que ofrece a alto nivel, sin dar detalles de la implementación de la misma.
- Herencia: Es el mecanismo que permite establecer jerarquías de clases de forma que las clases hijas heredan los campos y métodos de la clase padre
- Polimorfismo: Es la propiedad de los objetos que les permite adoptar diferentes formas.

Paquetes

- Los paquetes nos sirven para estructurar el código en una especie de carpetas que luego podemos importar en otros lugares.
- Al crear una clase podemos especificar el paquete de la misma
- Los paquetes son una serie de palabras separadas por puntos

```java
package org.iesfm.tienda.telefono;

public class Telefono {
    private String modelo;
    private int stock;
    private double price;


    public Telefono(String modelo, int stock, double price) {
        this.modelo = modelo;
        this.stock = stock;
        this.price = price;
    }

    public String getModelo() {
        return modelo;
    }

    public void setModelo(String modelo) {
        this.modelo = modelo;
    }

    public int getStock() {
```

Paquetes

Para utilizar una clase definida en otro paquete hay que importarla desde la clase donde se quiere usar

```java
package org.iesfm.tienda;

import org.iesfm.tienda.telefono.Telefono;

public class Tienda {

    public static void main(String[] args) {
        Telefono iPhone = new Telefono( modelo: "iphone",  stock: 100,  price: 1000d);

        System.out.println(iPhone.getModelo());
    }
}
```

Modificadores de acceso

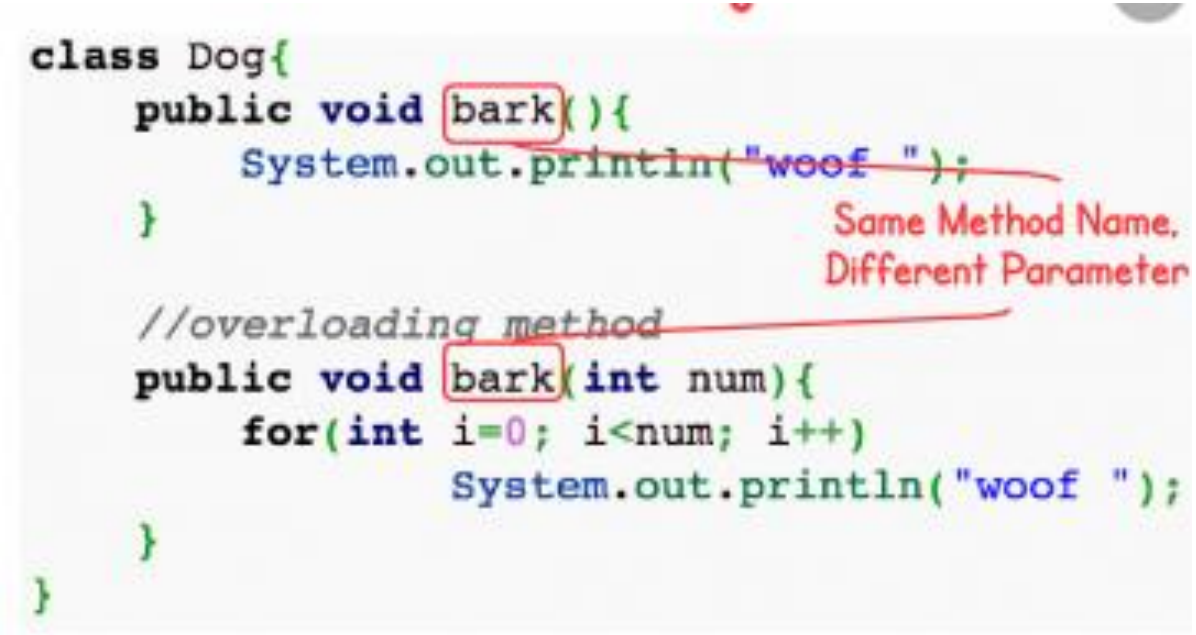
Los modificadores de acceso definen desde dónde es accesible un campo, método o clase.

Access Levels

Modifier	Class	Package	Subclass	World
`public`	Y	Y	Y	Y
`protected`	Y	Y	Y	N
no modifier	Y	Y	N	N
`private`	Y	N	N	N

Sobrecarga de métodos

- Es posible definir varios métodos con el mismo nombre en una clase, siempre y cuando cada uno de ellos reciba parámetros con tipos distintos
- A esto se le llama sobrecarga de métodos (method overloading)



Comparando objetos

- Para poder comprobar si dos objetos son equivalentes es necesario implementar el método *equals* en la clase.
- Si comparamos dos objetos utilizando el operador == devolverá true únicamente cuando a izquierda y derecha pongamos la misma instancia. No funcionará cuando comparemos dos instancias con los mismos valores.
- La implementación del método *equals* es tediosa y es muy fácil introducir bugs, por ello es recomendable que el método lo genere el IDE.
- La implementación de *equals* se debe actualizar cada vez que se introduzca un cambio en los campos de la clase.