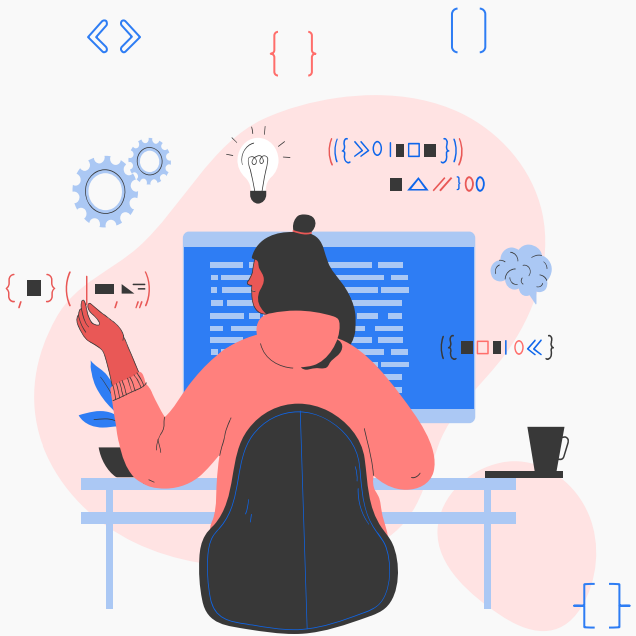
Programación

Tema 4 - Programación orientada a objetos

Marina Hurtado Rosales
marina.hurtado@escuelaartegranada.com



Índice

Programación orientada a objetos (POO)

Por qué surge la POO

Los primeros lenguajes de programación usaban el paradigma de la programación estructurada.

- **Las secuencias** -> instrucciones simples
- **Las estructuras condicionales**
- **Las estructuras repetitivas o bucles**

Con el paso de los años, los programas eran mayores en cuanto a código y complejidad, haciendo difícil entenderlo y mantenerlo.

Esto originó la creación de un nuevo paradigma, la **programación orientada a objetos**.

¿Qué es la POO?

La **Programación Orientada a Objetos (POO)** es un **paradigma de diseño** cuyo objetivo es “traducir” los **elementos de la vida real** a una estructura utilizable dentro de un **lenguaje de programación**.

Esta “traducción” se realiza mediante un proceso llamado **Abstracción**. Este proceso trata de obtener los atributos y las acciones básicas que representan la entidad que se quiere abstraer.

¿Qué es la POO?

Los **Atributos** obtenidos mediante abstracción, al traducirlos a código, forman un conjunto de variables que otorgan las **características a la clase**.

Por otro lado, las **acciones** estarán representadas por los **métodos**; estructuras de código a las que se pueden referenciar en cualquier parte del código.

Ejemplos de abstracción

Entidad real: Persona

Características	¿Se incluye en la clase?
Nombre	✓
DNI	✓
Fecha de nacimiento	✓
Género	Depende
Altura	Depende
ADN, etc…	✕

¿Qué puede hacer una persona?

- saludar()
- cumplirAños()
- Etc….

Abstraer NO es copiar la realidad, sino escoger lo relevante para resolver el problema

Ejemplos de abstracción

Entidad real: Coche

Características	¿Se incluye en la clase?
Color	✓
Matrícula	✓
Velocidad	✓
Presión de neumáticos	Depende
Marca	Depende
Composición química del combustible, etc…	✕

¿Qué puede hacer un coche?

- arrancar()
- acelerar()
- frenar()

Principios de la POO

- **Abstracción:** capacidad de representar objetos del mundo real como clases y objetos en un programa informático
- **Encapsulamiento:** cohesión de los datos en un objeto, estos datos están encapsulados en el objeto, aunque no se vean
- **Modularización:** se divide un problema en problemas más pequeños.
- **Herencia:** relaciones entre clases, una clase puede heredar comportamientos de otras.
- **Polimorfismo:** capacidad de un objeto de tener diferentes comportamientos.
- **Ocultación de información:** los detalles internos de un objeto (como sus atributos y la implementación de sus métodos) deben estar ocultos o protegidos del acceso directo desde fuera del objeto

Ventajas e inconvenientes

Ventajas e inconvenientes

Ventajas:

- Reutilización del código: podemos usarlo en los proyectos que queramos
- Mantenimiento: más fácil leer, entender y mantener el código.
- Modificabilidad: son más fáciles de modificar
- Fiabilidad: al descomponer los problemas en problemas pequeños, es más fácil encontrar posibles errores.

Inconvenientes:

- Hay un cambio en la forma de pensar que posee cierta dificultad.
- La ejecución de los programas es más lenta

Clases y objetos

Clases y objetos

Clase:

Una clase es la **plantilla** o molde que utilizamos para construir los objetos. Indica los atributos que deben poseer los objetos y las acciones que pueden realizar, pero éstos **no tienen valores concretos**. Tampoco puede ejecutar los métodos.

Objeto:

Es un elemento **creado a partir de la plantilla** (clase) a la que pertenece. Los atributos que componen el objeto tienen valores específicos y puede llamar a los métodos. Tiene **valores propios** y **puede ejecutar métodos**.

Un ejemplo muy claro de esto es una persona, que en respuesta a las preguntas tenemos:

- **¿Esto existe en el mundo real?**: Sí, las personas existen el mundo real.
- **¿Tiene propiedades?**: Sí, una persona tiene un DNI, un nombre, unos apellidos, una edad...
- **¿Puede realizar acciones?**: Sí, una persona puede saludar, puede andar...

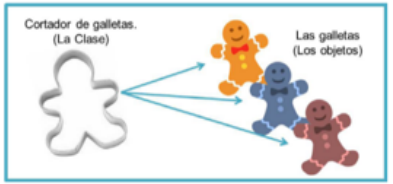


Fig. 6. La clase es el modelo a partir del cual podemos crear los objetos.

Conceptos básicos

Un **objeto** es una variable de la clase que hemos creado, que está compuesto por todos los atributos que tiene la clase y todos los métodos (funciones) que realiza.

Ejemplo:

Clase	Atributos	Métodos
Persona	nombre, edad, dni	comer(), dormir(), estudiar()

Objeto	Atributos	Métodos
persona1	Fernando, 33, 71828391R	comer()
persona2	Alonso, 34, 35890512T	comer(), dormir()

Protección de datos

En la programación orientada a objetos es muy importante proteger los datos, esto hace alusión a la **encapsulación** y a la **ocultación de información**.

Los datos que componen un objeto **no serán visibles desde fuera**, teniendo que usar métodos concretos para poder visualizarlos y modificarlos.

Esto nos proporciona un nivel de protección de datos bastante alto, ya que desde fuera no se podrá acceder a la forma en la que se ha diseñado el objeto, evitando así posibles problemas futuros.

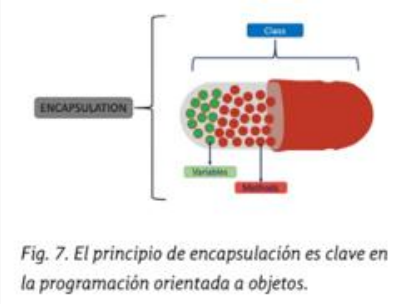


Fig. 7. El principio de encapsulación es clave en la programación orientada a objetos.

Visibilidad de los miembros de una clase

- **Estándar (sin modificador)**: Es el estado predeterminado.
- **Público (public)**: Este método permite que los miembros de la clase puedan ser accedidos desde el exterior de la propia clase y desde cualquier parte del programa.
- **Protegido (protected)**: Los miembros de la clase solo serán accesibles desde la clase y las clases que heredan (a cualquier nivel).
- **Privado (private)**: Los miembros de la clase solo serán accesibles desde la propia clase, no desde fuera.

Visibilidad de los miembros de una clase

Modificador	Public	Protected	Private	Defecto
Acceso desde la propia clase	SI	SI	SI	SI
Acceso desde otras clases del mismo paquete	SI	SI	NO	SI
Acceso desde una subclase en el mismo paquete	SI	SI	NO	SI
Acceso desde subclases en otros paquetes	SI	SI	NO	NO
Acceso desde otras clases en otros paquetes	SI	NO	NO	NO

Tabla 1. Resumen visibilidad.

Declarar una clase

Creación de una clase

Para declarar una clase, crearemos un fichero java nuevo.
OJO. Para cada clase se utilizará un fichero ÚNICO para esa clase.

La cabecera se construye de la siguiente forma:

```
public class Persona
{
    String DNI, nombre, apellidos;
    int edad;

    public void saludar()
    {
        System.out.println("Hola soy " + nombre);
    }
}
```

Fig. 8. Clase Persona.

- Los nombres de las clases **siempre** empiezan con la primera letra en mayúscula.
- **Atributos:** propiedades de una clase
- **Métodos:** funcionalidad de una clase

Creación de una clase

La estructura que vamos a seguir para construir una clase es la misma siempre.

```
public class Nombre
{
    //Bloque de atributos
    //Bloque de métodos
}
```

Declaración de atributos. Visibilidad

Los **atributos** son las propiedades de una clase, las que diferencian un objeto de otro cuando se crean.

Los atributos pueden ser cualquier tipo de dato: int, String, array…

Su declaración suele ser lo primero que se define al crear una clase, y se hace de la siguiente manera:

```
public int nombre;
private String nombre2;
```

La **visibilidad de un atributo** establece el acceso que tendrá el dato concreto, hay tres tipos distintos.

- **public**. Lo ve todo el mundo.
- **private**. Sólo lo ve la propia clase.
- **protected**. Visibilidad pública sólo entre los elementos del mismo paquete.

Declaración de métodos

Los métodos indican la funcionalidad de la clase. Su declaración, al igual que el resto de elementos, sigue una estructura similar a la cabecera de la clase..

```
public int nombreMetodo() {
    // Bloque de código del método
}
```

A la hora de declarar los métodos tenemos que tener en cuenta qué valor es el que va a devolver. Para devolver el valor del resultado se utiliza la palabra reservada **return** seguida de la **variable que almacena el valor resultado. Solo se indica un return por método.**

En los métodos se pueden declarar variables, pero éstas serán de ámbito local, por lo que no existirán ni se podrán utilizar fuera del método.

Si el método no devuelve nada, al igual que con las funciones, se utiliza como tipo de dato **void.**

Declaración de métodos

A veces, los métodos requieren de datos que se están utilizando en otra parte del código.
Un bloque de código no tiene acceso a las variables que se están utilizando en otro, por lo que se utiliza el **paso de mensajes** para solucionarlo.

Esto no es más que indicarle al método correspondiente la **lista de variables** que se le tienen que proporcionar para poder funcionar.

Para indicarle esta lista de **parámetros** a los métodos, solo hay que introducir la lista entre los paréntesis de la cabecera, separados por comas.

```
public int nombre(tipo var1, tipo var2, ... ) {
    // Bloque de código del método
}
```

Declarar un objeto

Uso de clases y objetos

Hasta ahora hemos aprendido a:

- Definir clases
- Declarar atributos
- Crear métodos
- Proteger los datos mediante encapsulación

El siguiente paso es aprender a **utilizar esas clases desde un programa principal**, normalmente desde el método **main**.

Un programa Java real suele estar formado por:

- Una clase principal que contiene el método **main**
- Otras clases que representan **entidades** del problema

Declaración de un objeto

Antes de poder utilizar un objeto, es necesario **declararlo**.

```
Persona persona1;
```

La declaración de un objeto:

- Crea una variable de tipo Persona
- NO crea todavía el objeto
- NO reserva memoria para él

En este punto, el objeto **no existe**. Recordemos que un objeto tiene que tener valores concretos de los atributos de la clase.

Instanciación de un objeto

Para crear realmente el objeto y reservar memoria, es necesario **instanciarlo** utilizando la palabra reservada new.

```
persona1 = new Persona();
```

La instanciación:

- Reserva memoria en el **heap**.
- Crea el objeto
- Permite utilizar sus métodos

En el caso de los objetos, la variable no guarda el objeto directamente, sino una **referencia** a la zona de memoria donde se encuentra.

Declaración e instanciación en una sola instrucción

Lo más habitual es declarar e instanciar el objeto en una sola línea:

```
Persona persona1 = new Persona();
```

A partir de este momento:

- El objeto existe
- Puede ejecutar métodos
- Tiene atributos con valores por defecto

Valores iniciales de los atributos

Cuando se crea un objeto con:

```
Persona persona1 = new Persona();
```

Los atributos del objeto reciben **valores iniciales** automáticamente.

Si no se indica lo contrario, Java asigna los siguientes valores de forma automática:

Tipos de datos	Valor por defecto asignado
Tipos numéricos (int, double…)	0
boolean	false
char	‘\u0000’
String y objetos	null

Problema de los valores por defecto

En muchos casos, crear objetos con valores por defecto:

- No tiene sentido lógico
- Puede provocar errores
- Obliga a modificar el objeto justo después de crearlo

Ejemplo

- Persona con nombre null
- Edad 0

Necesitamos una forma de **inicializar correctamente** los objetos.

Constructores

¿Qué es un constructor?

Un **constructor** es un método especial que:

- Se ejecuta **automáticamente** al crear un objeto
- **Inicializa** los **atributos**
- Tiene el **mismo nombre** que la **clase**
- No devuelve ningún valor

Los constructores se ejecutan al usar **new**.

Si no se define ningún constructor, Java crea uno automáticamente y asigna los valores por defecto tal y como vimos en diapositivas anteriores.

Constructor por defecto

Este constructor se trata de un **método público** que **no recibe** ningún **parámetro**.

Todos los objetos creados con este constructor tendrán **los valores iniciales definidos en él.** Estos valores pueden estar personalizados.

```
public class Perro {
    // Declaración de atributos
    private String raza;

    // Constructor por defecto
    public Perro() {
        this.raza = "Yorkshire";
    }
```

La palabra reservada **this** se utiliza para hacer referencia:

- Al objeto actual
- A los atributos de la clase

Se utiliza especialmente cuando el **nombre del parámetro** coincide con el **nombre del atributo**

Constructor por parámetros

Un **constructor con parámetros** requiere que se le pasen una serie de parámetros para instanciar el objeto.

De esta manera, **permite inicializar** objetos con distintos valores para sus atributos.

```
public class Perro {
    // Declaración de atributos
    private String raza;
    private String nombre;

    // Constructor por parámetros
    public Perro(String razaParametro, String nombreParametro){
        this.raza = razaParametro;
        this.nombre = nombreParametro;
    }
```

```
Perro perrete = new Perro("Chihuahua", "Dexter");
```

Constructor por parámetros

Una clase puede tener **varios constructores**

- Con **distinto número** de parámetros
- Con **distinto tipo** de parámetros

```
public Persona() {

}

public Persona(String nombre) {
    this.nombre = nombre;
}

public Persona(String nombre, int edad) {
    this.nombre = nombre;
    this.edad = edad;
}
```

Constructor de copia

```
Persona p1 = new Persona("Ana", 20);
Persona p2 = p1;
```

Esto **NO** crea un nuevo objeto. Ambas variables apuntan al mismo objeto, por lo que si se modifica una de ellas la otra también se ve afectada.

Para solucionar este problema y poder tener dos objetos diferentes, cada uno alojado en su propia dirección de memoria, debemos usar un **constructor de copia.**

Un **constructor de copia** nos permite crear un objeto nuevo a partir de otro objeto del mismo tipo copiando los valores de sus atributos.

```
public Persona(Persona otra) {
    this.nombre = otra.nombre;
    this.edad = otra.edad;
}
```

Acceso a elementos de un objeto

Uso de métodos de un objeto

Para acceder a los métodos de un objeto se utiliza el **operador punto (.)**

```
persona1.saludar();
```

Desde una clase solo se pueden utilizar:

- Métodos public
- Métodos visibles según las reglas de visibilidad

Acceso a los atributos de un objeto

En Programación Orientada a Objetos **los atributos suelen declararse como private.**

No deben accederse directamente desde fuera de la clase.

Acceder directamente a un atributo provoca un **error de compilación**.

```
persona1.nombre = "Ana"; // Incorrecto
```

La encapsulación protege los datos, pero plantea una cuestión importante:

¿Cómo podemos consultar o modificar los valores de los atributos?

La solución es proporcionar **métodos públicos controlados**.

Getters y Setters

Métodos de acceso: getters y setters

Los **getters** y **setters** son métodos que permiten:

- Acceder a los atributos
- Modificarlos de forma controlada
- Mantener la encapsulación

Estos métodos forman parte del **diseño correcto de una clase**.

Getters

Un **getter** es un método que nos permite obtener el valor del atributo que necesitamos. Se trata de un método que:

- Devuelve el valor de un atributo
- No recibe parámetros
- Es público

En una clase debemos tener un **getter por cada atributo** que tenga la clase.

Para nombrar un getter se utiliza la nomenclatura **get + nombre del atributo** a obtener (con camelCase)

```
public class Persona {
    String nombre;

    public String getNombre(){
        return this.nombre;
    }
```

Setters

Un **setter** es un método que nos permite modificar el valor de un atributo de la clase. Se trata de un método que:

- Modifica el valor de un atributo
- Recibe un parámetro
- Es público
- No devuelve ningún valor (es de tipo **void**)

En una clase debemos tener un **setter por cada atributo** que tenga la clase.

Para nombrar un setter se utiliza la nomenclatura **set + nombre del atributo** a obtener (con camelCase)

```
public class Persona {
    String nombre;

    public void setNombre(String nombre){
        this.nombre = nombre;
    }
```

Ciclo de vida de un objeto

Ciclo de vida de un objeto

Un objeto no existe indefinidamente. Desde que se crea hasta que deja de existir pasa por distintas etapas.

Este proceso se denomina **ciclo de vida de un objeto**.

Estas etapas son:

- **Creación:** un objeto se crea cuando se utiliza la palabra reservada **new** en el main. En ese momento se **reserva la memoria**, se **ejecuta el constructor** correspondiente y el objeto pasa a **existir**.

- **Utilización o uso:** se usan los métodos y atributos del objeto. Mientras exista una referencia al objeto se pueden **usar métodos**, **modificar sus atributos** o pasarse como **parámetro** a otros métodos.

- **Destrucción**: un objeto deja de ser accesible cuando no existe ninguna referencia que apunte a él. Java cuenta con un sistema de gestión automática de memoria a través del recolector de basura (**Garbage Collector**) que elimina automáticamente los objetos

Paquetes

¿Qué es un paquete?

Un paquete es un contenedor de clases que funciona de forma similar a una **“carpeta”** dentro de un proyecto. Los paquetes nos permiten:

- Organizar las clases de un proyecto.
- Agrupar clases relacionadas entre sí con características comunes.
- Reutilizar código.
- Tener una mayor seguridad al poder crear niveles de acceso a las clases.

Cada clase **pertenece a un paquete**. Este paquete se **declara** al **principio del archivo**, antes de la definición de la clase:

```
package nombrePaquete;
```

Si no se indica ningún paquete, la clase pertenece al **paquete por defecto**.

Cuando necesitamos utilizar una clase que esté en un paquete diferente **debemos importarla** haciendo uso de la instrucción **import**. Podremos importar tantas clases que estén en paquetes diferentes como necesitemos.

```
import java.util.Scanner;
```

Paso por valor y por referencia

Paso de parámetros a métodos

Cuando se llama a un método y se le pasan parámetros:

- El método recibe una **copia de la información**
- El **comportamiento** depende del tipo de dato del parámetro

En Java se diferencian dos casos:

- **Tipos primitivos** -> paso por valor
- **Objetos** -> paso por referencia

Paso por valor

Este comportamiento se aplica cuando pasamos como argumento a un método un **tipo primitivo**. En el **paso por valor**:

- Se copia el valor de la variable
- El método trabaja con esa copia
- El valor original no se modifica

```
public void cambiarNumero(int n) {
    n = 10;
}
```

Paso por referencia

Este comportamiento se aplica cuando pasamos como argumento a un método un **objeto**. En el **paso por referencia**:

- Se copia la referencia al objeto
- El método trabaja sobre el mismo objeto

```
public void cambiarEdad(Persona p) {
    p.setEdad(30);
}
```

Método toString

Método toString

Se trata de un método definido dentro de la clase **Object** de Java que nos permite mostrar información relevante de un objeto.

Es un método que usamos para mostrar un “resumen” del contenido de los atributos de la clase en forma de cadena de caracteres.

Gracias a este método, en el main podemos hacer un “sout” que nos mostrará el contenido

```
@Override
public String toString() {
    return "Nombre: " + nombre + ", Edad: " + edad;
}
```

Estructura final de una clase

Estructura final de una clase

```
public class NombreClase {
    // Declaración de atributos

    // Constructor por defecto

    // Constructor por parámetros

    // Constructor de copia

    // Declaración de getters (1 por atributo)

    // Declaración de setters (1 por atributo)

    // Declaración de métodos personalizados

    // Declaración toString() (con @Override)

}
```