



# Tema 27

## Programación Orientada a Objetos

Clases · Objetos · Herencia · Polimorfismo · Interfaces

# 1.1 Introducción y motivación

La POO modela el mundo real combinando:

-  Estado → atributos
-  Comportamiento → métodos

 Ventajas:

- Escalabilidad
- Reutilización
- Mantenibilidad

## 1.2 Evolución histórica

Antes de la POO:

- C, Pascal → datos y lógica por separado

Origen:

- Smalltalk, C++, Objective-C

Consolidación:

- Java, C#, Kotlin
- Tipado fuerte, gestión automática de memoria

## 1.3 Ventajas clave

Concepto	Descripción
Abstracción	Representar entidades complejas con clases
Encapsulamiento	Ocultar datos internos, proteger estado
Herencia	Reutilizar y extender comportamientos
Polimorfismo	Usar distintas clases con la misma interfaz

## 1.4 Clases y objetos

Ejemplo:

```
Coche miTesla = new Coche();
```

- Clase: plantilla
- Objeto: instancia

Atributos:

- De instancia
- Estáticos
- Constantes ( `final` )

Métodos:

- De instancia

## 1.5 Visibilidad y encapsulamiento

Modificadores de acceso:

```
private int velocidad;
```

```
public void setVelocidad(int v) {  
    if (v >= 0) velocidad = v;  
}
```

- public: accesible desde cualquier parte
- private: solo desde la clase
- protected: accesible desde subclases

## 1.6 Ciclo de vida de un objeto

1. **Creación:** constructor
2. **Inicialización:** valores iniciales
3. **Uso:** invocación de métodos
4. **Finalización:** recolección de basura (Java), `__del__` en Python

## 1.7 Relaciones entre clases

- Asociación: Profesor ↔ Alumno
- Agregación: Universidad tiene Departamentos
- Composición: Coche tiene un Motor

💡 Analogía:

Composición es como un corazón dentro del cuerpo

Agregación es como libros dentro de una mochila



## 1.8 Herencia y polimorfismo

Ejemplo de sobrescritura:

```
class Animal {  
    void hacerSonido() {  
        System.out.println("...");  
    }  
}  
  
class Perro extends Animal {  
    void hacerSonido() {  
        System.out.println("Guau!");  
    }  
}
```

Uso de enlace dinámico:

## 1.9 Interfaces y abstracción en Java

```
interface Volador {  
    void volar();  
}
```

```
class Pajaro implements Volador {  
    public void volar() {  
        System.out.println("Estoy volando");  
    }  
}
```

Ventajas:

- Contrato de comportamiento
- Diseño desacoplado
- Herencia múltiple de comportamiento

## 1.10 Principios SOLID

Letra	Principio	Idea clave
S	Responsabilidad única	Una clase = una tarea
O	Abierto / Cerrado	Extensible sin modificar lo existente
L	Sustitución de Liskov	Subclases reemplazan a padres sin errores
I	Segregación de interfaces	Interfaces pequeñas y específicas
D	Inversión de dependencias	Depender de abstracciones, no implementaciones

## 1.11 Lenguajes y ejemplos

Lenguaje	Características
Java	OO puro, sin herencia múltiple, tipado fuerte
C++	Herencia múltiple, manejo manual de memoria
Python	Paradigma mixto, flexible, admite POO
C#	OO moderno, interfaces, propiedades, LINQ

## 1.12 Comparación con programación estructurada

Aspecto	Estructurada	Orientada a objetos
Unidad	Función	Clase / Objeto
Separación lógica	Datos y funciones separados	Estado + comportamiento unidos
Escalabilidad	Baja	Alta
Adecuado para	Algoritmos simples	Aplicaciones complejas

## Conclusión

- La POO mejora el diseño, claridad y escalabilidad
- Permite representar modelos del mundo real
- Herencia y polimorfismo favorecen la reutilización
- Java, Python o C++ son lenguajes clave en este paradigma

 ¡Dominar la POO te prepara para proyectos reales, grandes y complejos!