Comportamiento

Comportamiento

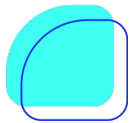
Hasta ahora, nos hemos centrado en el estado del objeto, los atributos y el concepto de encapsulamiento.

Otra parte importante de los objetos es **su comportamiento.**

En POO, el comportamiento se refiere a las **acciones que un objeto puede realizar**.

Función

- Actualiza el **estado interno del objeto** asegurándose que se mantenga su integridad.
- Coordina la **interacción** entre distintos objetos para realizar una tarea.
- Se define mediante **métodos** en la definición de la clase.



Métodos

Definición y uso

Los métodos son **funciones definidas dentro de una clase u objeto.**

Representan **acciones** que el objeto puede realizar:

- Permiten a los objetos **interactuar con su estado interno** y con otros objetos.
- Definen la **funcionalidad** específica del objeto.

Los métodos se dividen en dos categorías:

	Métodos de instancia	Métodos estáticos
Descripción	Estos métodos **operan en instancias individuales de una clase.** Tienen acceso al estado del objeto (atributos) y pueden modificarlo o utilizarlo para realizar acciones específicas.	Estos métodos **pertenecen a la clase en sí, en lugar de a cualquier instancia particular** de la clase. No pueden acceder ni modificar los atributos de instancia, pero **pueden trabajar con atributos estáticos** (compartidos por todas las instancias de la clase) **o realizar operaciones generales** relacionadas con la clase.
Ejemplo	En una clase **`Coche`**, un método de instancia podría ser **`acelerar`**, que aumenta la velocidad del coche.	En la clase **`Matemáticas`**, un método estático podría ser **`sumar`**, que toma dos números como parámetros y devuelve su suma.

Ejemplo de una clase *Cuenta*

Una **`Cuenta`** es como una caja donde podemos guardar dinero y desde donde podemos sacarlo cuando lo necesitamos.

- **Operaciones básicas:**
  - **Depositar:** Añadimos dinero a la cuenta.

    Ejemplo: **`cuentaDeJuan.depositar(500.0);`** aumenta el saldo en $500.

  - **Retirar:** Sacamos dinero de la cuenta, si hay suficiente.

    Ejemplo: **`cuentaDeJuan.retirar(200.0);`** reduce el saldo en $200, si hay suficiente dinero.

```
public class Cuenta {
    private String titular;
    private double saldo;

   ...
    public void depositar(double cantidad) {
        if (cantidad <= 0) {
            throw new Error("Error: La cantidad a depositar debe ser positiva.");
        }
        saldo += cantidad;
    }

    public void retirar(double cantidad) {
        if (cantidad <= 0) {
            throw new Error("Error: La cantidad a retirar debe ser positiva.");
        }
        if (cantidad > saldo) {
            throw new Error("Error: La cantidad a retirar excede el saldo disponible.");
        }
        saldo -= cantidad;
    }
```

Comportamiento, *getters* y *setters*

Los ***getters*** (métodos de acceso) y ***setters*** (métodos de modificación) también permiten **leer y escribir los valores de los atributos privados de un objeto desde fuera de la clase.**

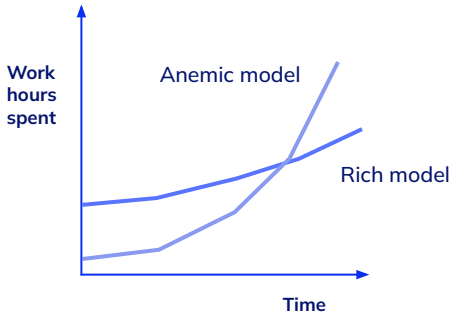
No obstante, podemos argumentar que **no son técnicamente parte del comportamiento** de una clase, en el sentido estricto del término, dentro del paradigma de la Programación Orientada a Objetos (POO). Aunque son esenciales para el **encapsulamiento** y la buena práctica de diseño, no representan de forma directa acciones o comportamientos que modifiquen el estado conceptual del objeto en términos de su función principal.

Separación de responsabilidades

Mantener los *getters* y *setters* separados del comportamiento principal de la clase promueve una **mejor organización del código** y facilita la comprensión y mantenimiento a largo plazo.

Modelo anémico

Es aquel en el que la definición de las clases y, por consiguiente, **los objetos generados**, contienen principalmente datos (atributos, *getters* y *setters)* pero **carecen de comportamiento significativo.**



Los modelos anémicos pueden ser útiles en contextos simples pero **limitan la capacidad de los objetos para modelar comportamientos complejos de negocio.**

Resulta fundamental:

- **Equilibrar el diseño, entre datos y comportamiento,** para maximizar la cohesión y el encapsulamiento en el diseño de *software*.
- **Definir el comportamiento de nuestros objetos es una tarea central** en el diseño orientado a objetos.