

ENCAPSULACION EN JAVA

La **encapsulación** es un pilar de la Programación Orientada a Objetos que consiste en ocultar el estado interno de un objeto y obligar la interacción a través de un conjunto de métodos públicos.

Piensa en el tablero de un coche. No puedes cambiar la velocidad del motor directamente; usas el pedal del acelerador. El pedal es una **interfaz pública** que controla de forma segura el **estado interno privado** del motor. La encapsulación protege los datos de manipulaciones incorrectas.

Modificadores de Acceso: Las Reglas del Juego

Para encapsular, usamos palabras clave que definen la visibilidad de los atributos y métodos. Las principales son:

- **public**: Accesible desde cualquier parte del programa. Son los "pedales y botones" que exponemos al exterior.
- **private**: Accesible **únicamente** desde dentro de la misma clase. Son los "componentes internos del motor" que queremos proteger.
- **protected** : Accesible únicamente desde dentro de la misma clase, desde otras clases del **mismo paquete**, y desde cualquier **clase hija (subclase)**, incluso si esa clase hija está en un paquete diferente.

Para permitir un acceso controlado a los datos **private**, usamos métodos públicos **get** y **set**.

- **Getters (get)**: Métodos para **obtener** (leer) el valor de un atributo. Siguen la convención `getNombreDelAtributo()`.
- **Setters (set)**: Métodos para **establecer** (modificar) el valor de un atributo. Siguen la convención `setNombreDelAtributo()`. Su mayor poder es que permiten añadir **reglas de validación**.

Beneficios de la Encapsulación

- **Protección y Control**: Evitas que los datos adquieran valores no válidos (ej. un saldo negativo). Tú controlas cómo se modifican los datos.
 - **Ocultamiento de la Complejidad**: El usuario de la clase no necesita saber cómo funciona por dentro, solo qué métodos usar.
 - **Flexibilidad**: Puedes cambiar la lógica interna de la clase sin afectar a los otros componentes del programa que la usan.
-

Ejemplo Práctico: Cuenta Bancaria Segura

Vamos a transformar el ejercicio de la cuenta bancaria para que use encapsulación de forma correcta.

"Antes": La Clase Insegura

Aquí, el saldo es público y cualquiera puede asignarle un valor incorrecto.

```
// Versión vulnerable con atributos públicos
class CuentaBancariaInsegura {
    public String numeroDeCuenta;
    public double saldo; // ¡Cualquiera puede modificarlo!

    // ... constructor ...
}

// En el main:
// CuentaBancariaInsegura miCuenta = new CuentaBancariaInsegura("123",
// 100.0);
// miCuenta.saldo = -1000.0; // ¡Desastre! Esto es posible.
```

"Después": La Clase Encapsulada

Ahora, protegemos los atributos y controlamos el acceso con métodos get y set.

```
// Archivo: CuentaBancaria.java
public class CuentaBancaria {
    // 1. Atributos privados para protegerlos
    private String numeroDeCuenta;
    private double saldo;

    public CuentaBancaria(String numero, double saldoInicial) {
        this.numeroDeCuenta = numero;
        // Se puede usar el setter en el constructor para validar
        if (saldoInicial >= 0) {
            this.saldo = saldoInicial;
        } else {
            this.saldo = 0;
        }
    }

    // --- GETTERS: Para leer los datos ---

    public String getNumeroDeCuenta() {
```

```

        return this.numeroDeCuenta;
    }

    public double getSaldo() {
        return this.saldo;
    }

    // --- SETTERS: Para modificar los datos de forma controlada ---

    // Un setter tradicional para el número de cuenta
    public void setNumeroDeCuenta(String numeroDeCuenta) {
        this.numeroDeCuenta = numeroDeCuenta;
    }

    // Aunque podríamos tener un setSaldo(), es más claro e intuitivo
    // usar métodos con nombres de acciones reales como depositar y
    retirar.
    // Estos actúan como setters controlados.

    public void depositar(double cantidad) {
        if (cantidad > 0) {
            this.saldo += cantidad;
            System.out.println("Depósito exitoso. Nuevo saldo: " +
this.saldo);
        } else {
            System.out.println("Error: La cantidad a depositar debe ser
positiva.");
        }
    }

    public void retirar(double cantidad) {
        if (cantidad > 0 && cantidad <= this.saldo) {
            this.saldo -= cantidad;
            System.out.println("Retiro exitoso. Nuevo saldo: " +
this.saldo);
        } else {
            System.out.println("Error: Cantidad inválida o saldo
insuficiente.");
        }
    }
}

```

Usando la clase segura en main:

```

// Archivo: Banco.java
public class Banco {
    public static void main(String[] args) {
        CuentaBancaria miCuenta = new CuentaBancaria("12345", 100.0);

        // ERROR: La siguiente línea no compilará porque 'saldo' es
privado.
        // miCuenta.saldo = -1000.0;

        // Forma correcta de interactuar:
        System.out.println("Saldo inicial: " + miCuenta.getSaldo()); // Leer con GET

        miCuenta.depositar(50.0); // Modificar con método controlado
        miCuenta.retirar(20.0); // Modificar con método controlado

        System.out.println("Saldo final: " + miCuenta.getSaldo());
    }
}

```

5 Ejercicios Propuestos para Practicar

1. Tienda (Producto y Usuario):

- **Clase Producto:** Haz nombre y precio privados. Crea getNombre() y getPrecio(). Crea setPrecio(double nuevoPrecio) que solo acepte valores mayores que cero.
- **Clase Usuario:** Haz la lista carrito privada. Crea un método getCarrito() que devuelva la lista. Asegúrate de que solo se puedan añadir productos a través del método agregarProductoAlCarrito.

2. Biblioteca (Libro):

- **Clase Libro:** Haz titulo y autor privados. Crea getTitulo() y getAutor(). Crea setTitulo(String titulo) y setAutor(String autor). En el setter del título, no permitas que se asigne un valor vacío o nulo.

3. Gestión de Mascotas (Mascota):

- **Clase Mascota:** Haz nombre y edad privados. Crea getNombre() y getEdad(). Crea un setEdad(int edad) que solo acepte edades entre 0 y 30.

4. Inscripción a Cursos (Estudiante):

- **Clase Estudiante:** Haz nombre y idEstudiante privados. Crea getters para ambos. El idEstudiante no debería tener un setter, para que no se

pueda cambiar una vez creado el estudiante. El nombre sí puede tener un `setNombre()`.

5. **Pizzería (Pizza):**

- **Clase Pizza:** Haz sabor y precio privados. Crea `getSabor()` y `getPrecio()`. Crea un `setPrecio(double precio)` que impida asignar un precio de cero o menor. Crea un `setSabor(String sabor)` que no permita un sabor vacío.