Programación Orientada a Objetos (POO) en Java

Fundamentos teóricos de la POO

Introducción

- La Programación Orientada a Objetos (POO) es un paradigma que organiza el software en torno a **objetos** en lugar de funciones y datos.
- Permite estructurar el código de manera modular, reutilizable y escalable.
- Se basa en cuatro pilares fundamentales: **Encapsulamiento, Herencia, Polimorfismo y Abstracción**.

Encapsulamiento

- **Definición:**
  - Proceso de ocultar los detalles internos de un objeto y exponer solo lo necesario.
  - Protege los datos y mejora la modularidad.
- **Beneficios:**
  ✅ **Seguridad: Restringe el acceso a los datos.**
  ✅ **Modularidad: Facilita el mantenimiento del código.**
  ✅ **Control: Permite definir reglas de acceso mediante métodos (getters y setters).**
- **Analogía:** Una cápsula de medicamento protege su contenido y solo se accede a él de forma controlad

Ejemplo en Java:

```java
class Persona {
    private String nombre; // Atributo privado

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

    // Método getter para acceder al nombre
    public String getNombre() {
        return nombre;
    }

    // Método setter para modificar el nombre
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
}
```

Herencia

Definición:

- Mecanismo que permite que una clase (hija) herede atributos y métodos de otra (padre).
- Promueve la reutilización de código y reduce la redundancia.

Tipos:

- **Herencia simple** (una clase base).
- **Herencia múltiple (no permitida en Java directamente)**.

Ejemplo real: Un automóvil eléctrico hereda características de la clase general “Automóvil”.

Ejemplo en Java:

```java
// Clase Padre
class Animal {
    void hacerSonido() {
        System.out.println("El animal hace un sonido.");
    }
}

// Clase Hija
class Perro extends Animal {
    void hacerSonido() {
        System.out.println("El perro ladra.");
    }
}

// Uso
public class Main {
    public static void main(String[] args) {
        Perro miPerro = new Perro();
        miPerro.hacerSonido(); // "El perro ladra."
    }
}
```

Polimorfismo

- **Definición:**
  - Capacidad de un objeto para comportarse de diferentes maneras según el contexto.
- **Tipos:**
  - **Polimorfismo en tiempo de compilación** (sobrecarga de métodos).
  - **Polimorfismo en tiempo de ejecución** (sobrescritura de métodos).
- **Beneficios:**
  - ✅ **Flexibilidad en el código.**
  - ✅ **Permite la extensibilidad de programas sin modificar código existente.**
- **Ejemplo real:** Un actor puede desempeñar distintos roles en diferentes películas.

Ejemplo en Java (Polimorfismo en tiempo de ejecución - Sobrescritura de métodos):

```java
class Vehiculo {
    void arrancar() {
        System.out.println("El vehículo arranca.");
    }
}

class Coche extends Vehiculo {
    void arrancar() {
        System.out.println("El coche arranca con llave.");
    }
}

class Moto extends Vehiculo {
    void arrancar() {
        System.out.println("La moto arranca con botón.");
    }
}
```

```java
public class Main {
    public static void main(String[] args) {
        Vehiculo miVehiculo = new Coche();
        miVehiculo.arrancar(); // "El coche arranca con llave."

        miVehiculo = new Moto();
        miVehiculo.arrancar(); // "La moto arranca con botón."
    }
}
```

Ejemplo en Java (Polimorfismo en tiempo de compilación - Sobrecarga de métodos):

```java
class Calculadora {
    int sumar(int a, int b) {
        return a + b;
    }

    double sumar(double a, double b) {
        return a + b;
    }
}

public class Main {
    public static void main(String[] args) {
        Calculadora calc = new Calculadora();
        System.out.println(calc.sumar(5, 3));    // 8
        System.out.println(calc.sumar(2.5, 3.7)); // 6.2
    }
}
```

Abstracción

- **Definición:**
  - Oculta detalles de implementación y muestra solo lo esencial.
  - Se logra mediante **clases abstractas** e **interfaces**.
- **Beneficios:**
  ✅ **Reduce la complejidad del código.**
  ✅ **Permite definir comportamientos generales para múltiples clases.**
- **Analogía:** Un control remoto oculta su funcionamiento interno y solo expone botones para la interacción.

Ejemplo en Java con clases abstractas

```java
abstract class Figura {
    abstract double calcularArea(); // Método abstracto
}

class Circulo extends Figura {
    private double radio;

    public Circulo(double radio) {
        this.radio = radio;
    }

    @Override
    double calcularArea() {
        return Math.PI * radio * radio;
    }
}

public class Main {
    public static void main(String[] args) {
        Figura miFigura = new Circulo(5);
        System.out.println("Área del círculo: " + miFigura.calcularArea());
    }
}
```

Comparación de los pilares

Pilar	Propósito	Beneficio clave
Encapsulación	Ocultar datos	Seguridad y modularidad
Herencia	Reutilizar código	Reducción de redundancia
Polimorfismo	Adaptabilidad	Flexibilidad y escalabilidad
Abstracción	Ocultar detalles de implementación	Simplificación del código

Conclusión

- La POO es un enfoque poderoso que mejora la calidad del software.
- Aplicar estos principios facilita la escalabilidad, mantenibilidad y reutilización del código.
- Java es un lenguaje fuertemente orientado a objetos, y comprender estos pilares es esencial para su dominio.