Programación Orientada a Objetos en Java

José Ángel Olmedo Guevara

Introducción

La Programación Orientada a Objetos (POO) es un paradigma de programación que organiza el código en unidades llamadas "objetos", que contienen atributos (variables de instancia), identidad (nombre), comportamientos(métodos).

1 Abstracción

La abstracción consiste en representar entidades del mundo real con clases, mostrando solo la información esencial e ignorando los detalles innecesarios.

```
abstract class Animal {
    private String nombre;
    private int edad;

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

    public String getNombre() {
        return nombre;
    }

    public int getEdad() {
        return edad;
    }

    public abstract void hacerSonido();
}
```

Ejemplo 1: Ejemplo de abstracción

2 Encapsulamiento

El encapsulamiento protege los datos del objeto, ocultando los atributos y exponiéndolos mediante métodos públicos (getters y setters).

```
public class CuentaBancaria {
    private double saldo;

    public CuentaBancaria(double saldoInicial) {
        saldo = saldoInicial;
    }

    public void depositar(double cantidad) {
        if (cantidad > 0) saldo += cantidad;
    }

    public void retirar(double cantidad) {
        if (cantidad > 0 && cantidad <= saldo) saldo -= cantidad;
    }

    public double getSaldo() {
        return saldo;
    }
}
```

Ejemplo 2: Encapsulamiento con atributos privados

3 Herencia

La herencia permite que una clase herede atributos y métodos de otra clase, promoviendo la reutilización del código.

```
class Perro extends Animal {
    public Perro(String nombre, int edad) {
        super(nombre, edad);
    }

    @Override
    public void hacerSonido() {
        System.out.println(getNombre() + " dice:  Guau  guau!");
    }
}

class Gato extends Animal {
    public Gato(String nombre, int edad) {
        super(nombre, edad);
    }

    @Override
    public void hacerSonido() {
        System.out.println(getNombre() + " dice:  Miau  !");
    }
}
```

Ejemplo 3: Ejemplo de herencia

4 Polimorfismo

El polimorfismo permite que una misma interfaz o clase base se utilice para representar distintos comportamientos en clases derivadas.

```
public class Main {
    public static void main(String[] args) {
        Animal miPerro = new Perro("Fido", 5);
        Animal miGato = new Gato("Michi", 3);

        miPerro.hacerSonido(); // Fido dice:  Guau  guau!
        miGato.hacerSonido();  // Michi dice:  Miau  !
    }
}
```

Ejemplo 4: Ejemplo de polimorfismo