REPASO TEÓRICO Progra II

Conceptos Básicos de OOP (Programación Orientada a Objetos)

OOP / Programación basada en objetos:

Paradigma de programación que organiza el código en **objetos** que tienen **atributos (datos)** y **métodos (funciones)** para manipular esos datos. Es como modelar el mundo real en código.

Clases y objetos:

- **Clase**: Plantilla o molde que define atributos y métodos.
- **Objeto**: Instancia de una clase, es decir, un ejemplar creado a partir de esa plantilla.

Introducción:

La POO ayuda a organizar mejor el código, hacerlo reutilizable, escalable y fácil de mantener. Está basada en los principios de **encapsulamiento, herencia, polimorfismo y abstracción.**

Visibilidad y Encapsulamiento

Vista pública de las clases:

Todo lo que se define como `public` en una clase es accesible desde otras clases. Ejemplo: métodos públicos para interactuar con objetos.

Vista pública de los objetos:

Los objetos pueden exponer ciertas funcionalidades al exterior mediante métodos públicos, pero no todos sus datos directamente (buena práctica).

Vista privada de las clases / Ejemplo: `Calculadora.java`:

Atributos privados (`private`) no pueden ser accedidos directamente desde fuera de la clase. Se usan métodos `get` y `set` para acceder/controlarlos.

j

```
public class Calculadora {
   private int resultado;

   public void sumar(int x, int y) {
      resultado = x + y;
   }
```

```
    public int getResultado() {
        return resultado;
    }
}
```

Vista privada de los objetos:

Los objetos ocultan su implementación interna, exponiendo solo lo necesario (encapsulamiento). Esto permite evitar errores y proteger los datos.

Miembros de Clase y Colaboración

Miembros de clase:

- **Atributos**: variables dentro de la clase.
- **Métodos**: funciones que pueden usar o modificar los atributos.
- También hay miembros `static` que pertenecen a la clase en general, no a una instancia específica.

Colaboración entre objetos: HOOD (Hierarchical Object-Oriented Design):

Metodología para diseñar sistemas usando objetos que colaboran entre sí. En HOOD:

- Se identifican responsabilidades.
- Se distribuyen entre objetos.
- Se definen relaciones entre ellos.

Aplicaciones con HOOD

Tic Tac Toe:

Juego simple con tres clases principales: **Tablero, Jugador, Juego**.

- Usa lógica de turnos, validaciones de posición y verificación de ganadores.

Klondike:

Versión del Solitario. Usa clases como **Carta, Mazo, Tablero**.

- Implementa lógica de movimientos legales y orden de cartas.

Blackjack:

Juego de cartas. Clases: **Jugador, Dealer, Baraja, Carta**.

- Lógica de sumar puntos, repartir cartas y determinar ganadores.

Herencia y Polimorfismo en OOP

Relación de Herencia:

Una clase puede heredar atributos y métodos de otra clase. La clase base se llama **superclase**, la derivada **subclase**.

Herencia por extensión:

Se usa la palabra clave `extends` en Java.

r

```
class Animal {
   void hablar() {
      System.out.println("Hace un sonido");
   }
}

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

Clases abstractas:

No se pueden instanciar. Sirven como base para otras clases. Pueden tener métodos abstractos (sin cuerpo) y métodos normales.

```
abstract class Figura {
   abstract void dibujar();
}
```

Herencia por implementación:

Se refiere a cuando una clase implementa una interfaz. Se usa la palabra `implements`.

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

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

Limitaciones de la Herencia:

- Java no permite **herencia múltiple** directa (una clase no puede extender dos clases).
- Puede crear un acoplamiento fuerte entre clases.

Beneficios de la Herencia:

- Reutilización de código.
- Organización jerárquica.
- Facilita la extensión y mantenimiento.

Polimorfismo

🔹 **¿Qué es?**

Permite que un mismo método tenga diferentes comportamientos dependiendo del objeto que lo usa.

```
Animal a = new Perro();
a.hablar(); // Guau
```

Conversión de Tipos:

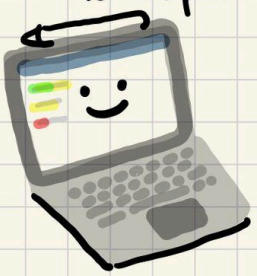
Puedes tratar un objeto como si fuera de su clase base, lo que permite el polimorfismo. Ejemplo: `Perro` como `Animal`.

Herencia y Enumerados:

Los **enum** en Java son tipos especiales que pueden tener comportamientos (métodos) y también pueden implementar interfaces, pero **no pueden extender clases**.

Sistema

Conjunto de componentes interactuando o interdependientes formando un todo integrado.

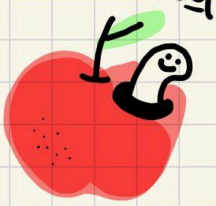


Eficacia

alto cumplimiento de objetivos frente a incumplimiento por errores....

Eficiencia: bajo consumo de recursos (tiempo, energía, espacio...).

60% fracaso al cumplir la efectividad requerida.



Sistemas Complejos

- Características

- Estructura jerárquica
- Elementos primitivos relativos
- Separación de asuntos
- Patrones comunes

- Formas intermedias estables

Capacidades cualitativas

Abstracción : proceso de extracción de las características esenciales de algo. Ignorando lo superfluos.

Encapsulación : proceso por el que se ocultan los detalles del soporte de las características esenciales de una abstracción.

Modularización: proceso de descomposición de un sistema en un conjunto de piezas poco acopladas y cohesivas.

Cohesión : Sentido con el que está acoplado un modelo.

Acoplamiento : Cantidad de elemento con los que está asociado un sistema.

Jerarquización : Proceso de estructurización por el que se produce una org. de un conjunto de elementos en grados o niveles de responsabilidad