Guía de Estudio: Programación Orientada a Objetos (POO)

1. Introducción a la Programación Orientada a Objetos (POO)

• Paradigmas de programación (estructurada vs. orientada a objetos)

La programación estructurada se basa en una secuencia lógica de instrucciones y el uso de estructuras como condicionales y bucles. En cambio, la programación orientada a objetos (POO) organiza el código en objetos que contienen datos (atributos) y comportamientos (métodos). Ejemplo de lenguaje estructurado: C. Ejemplos de lenguajes orientados a objetos: Java, C++, Python.

• Historia y evolución de la POO

La POO surgió en los años 60 con Simula (1967), el primer lenguaje orientado a objetos. Fue popularizada por Smalltalk en los años 80. En los 90, lenguajes como C++ y Java consolidaron la POO como el estándar dominante en el desarrollo de software.

• Ventajas del enfoque orientado a objetos

Entre las ventajas están: modularidad (código organizado en clases), reutilización (herencia de código), mantenibilidad (código fácil de entender y actualizar) y escalabilidad (ideal para proyectos grandes).

• Lenguajes orientados a objetos

Ejemplos: Java (completamente orientado a objetos), C++ (mezcla de estructurado y objetos), Python (multiparadigma), además de C#, Ruby, Kotlin, Swift, entre otros.

2. Fundamentos de la POO

• Clases y objetos

Una clase es una plantilla que define atributos y métodos. Un objeto es una instancia concreta de una clase.

• Atributos (propiedades) y métodos (funciones miembro)

Los atributos son características del objeto (como nombre, edad), y los métodos son acciones que el objeto puede realizar (como hablar(), caminar()).

• Instanciación de objetos

Es el proceso de crear un objeto a partir de una clase. Ejemplo en Java: Persona p = new Persona();

• Encapsulamiento

Es el principio que protege los datos de acceso externo y no autorizado, usando modificadores de acceso como private, public y protected.

3. Relaciones entre Clases

• Asociación, agregación y composición

Asociación: relación general entre clases. Agregación: una clase contiene otra pero pueden existir por separado. Composición: una clase contiene otra y dependen entre sí, si una se elimina, la otra también.

• Relaciones "has-a" y "uses-a"

Has-a: una clase posee otra. Uses-a: una clase utiliza otra temporalmente para cumplir una función.

• Diagramas de clases UML básicos

Los diagramas UML muestran gráficamente clases, atributos, métodos y relaciones. Son útiles para planificar el diseño del sistema.

4. Principios de la POO

• Abstracción

Oculta los detalles complejos y muestra solo lo esencial. Por ejemplo, usar imprimir() sin saber su implementación.

• Encapsulamiento

Protege los datos internos de un objeto y permite el acceso controlado a ellos.

• Herencia

Permite a una clase heredar atributos y métodos de otra. Facilita la reutilización del código.

• Polimorfismo

Permite que el mismo método tenga distintos comportamientos dependiendo del objeto. Se puede lograr mediante sobrecarga o sobreescritura.+

5. Herencia

• Clases base y clases derivadas

Una clase base (padre) proporciona atributos/métodos, y una clase derivada (hija) los hereda y puede extenderlos.

• Constructores y destructores en la herencia

El constructor de la clase base se invoca desde la clase hija usando super. Algunos lenguajes permiten destructores (como C++).

• Sobreescritura (override) de métodos

Una subclase redefine un método heredado para cambiar su comportamiento. Se usa el modificador override.

• Uso del modificador super o equivalente

super permite acceder a métodos o atributos de la clase padre desde la clase hija.