Programación Orientada a Objetos (POO)

Son como las “reglas base” que definen cómo deben estructurarse y comportarse las clases y los objetos. En total, son cuatro pilares principales

🧱 Pilares

Abstracción

Qué es?
La abstracción es la capacidad de sacar los aspectos mas importantes o lo que le caracteriza a un objeto, e ignorar las cosas irrelevantes

Ejemplo
En una clase Auto, se puede abstraer atributos como ‘marca’, ‘modelo’ y ‘velocidad’, y no preocuparse por cómo está hecho el motor internamente.

Encapsulamiento

Qué es?
El encapsulamiento consiste en ocultar los datos internos de un objeto y exponer solo lo necesario mediante métodos públicos.

Ejemplo
Usando el ejemplo de los carros, se puede usar el método Velocidad() para acceder a la velocidad de un auto, pero no se permite modificarla directamente desde fuera del objeto si está bien encapsulada.

Herencia

Qué es?
La herencia permite que una clase (hija) herede atributos y métodos de otra clase (padre), facilitando la reutilización de código.

Ejemplo
Una clase ‘Perro’ puede heredar de la clase ‘Animal’, obteniendo comportamientos como comer() o dormir() sin necesidad de redefinirlos.

Polimorfismo

Qué es?
El polimorfismo permite que un mismo método tenga comportamientos diferentes dependiendo del objeto que lo utilice.

Ejemplo
El método hacerSonido() se comporta de manera distinta si lo usa un objeto ‘Perro’ o un objeto ‘Gato’.

✅ Ventajas de la POO

En pocas palabras, POO tiene la ventaja de organizar el código en clases reutilizables, lo que facilita su mantenimiento y crecimiento.

Además, permite que varios programadores trabajen al mismo tiempo sin conflictos, mejorando la colaboración en equipo.

Representación del mundo real
La POO permite crear objetos que representan cosas reales, lo que hace que el programa sea más fácil de entender.

Ejemplo: En un sistema de biblioteca, se pueden tener objetos como ‘Libro’, ‘Usuario’ o ‘Préstamo’, que cualquier persona puede reconocer.

Se puede usar el mismo código varias veces
Gracias a las clases y la herencia, no es necesario escribir todo desde cero. Se puede usar el mismo código para hacer cosas parecidas.

Ejemplo: Una clase Vehículo puede servir para crear ‘Auto’, ‘Moto’ o ‘Camión’, usando parte del mismo código.

Es más fácil de arreglar o cambiar
Si algo del programa necesita cambiar, se puede hacer sin afectar todo lo demás, especialmente si el código está bien organizado.

Ejemplo: Si se cambia cómo se calcula el precio en un ‘Producto’, no hace falta cambiar otras partes del sistema.

Permite agregar cosas nuevas fácilmente
Si se necesita agregar nuevas funciones, se puede hacer sin borrar lo que ya está hecho.

Ejemplo: Si se quiere agregar una nueva opción a un sistema de reservas, se puede crear una nueva clase sin tocar las otras.

Ayuda al trabajo en equipo
Al dividir el programa en partes (clases), varias personas pueden trabajar al mismo tiempo sin molestarse unas a otras.

Ejemplo: Una persona puede hacer la clase ‘Cliente’ y otra la clase ‘Factura’, al mismo tiempo.

Nombre: Julio Mera