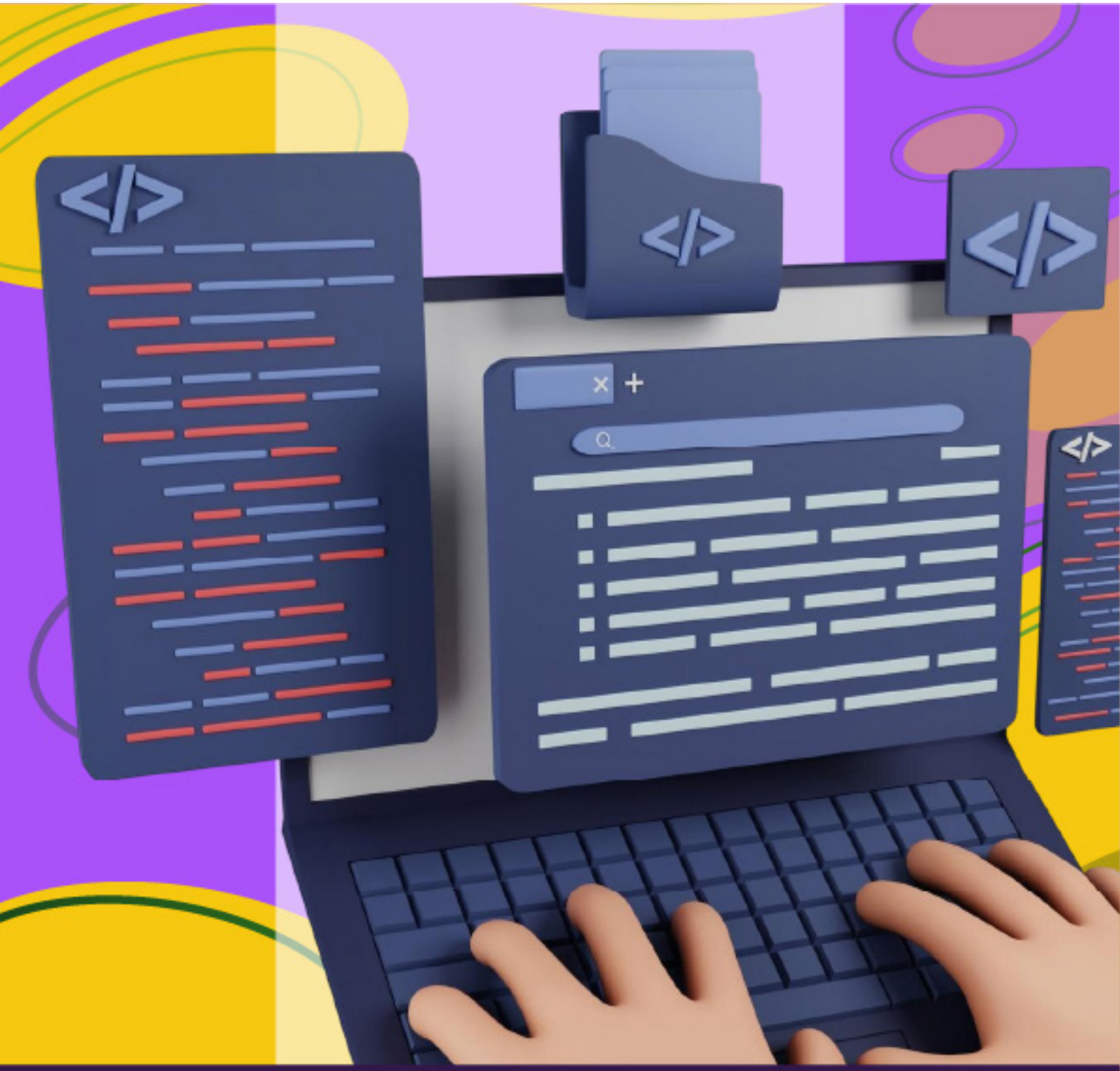
TECNOLÓGICA DEL ORIENTE
INSTITUCIÓN DE EDUCACIÓN SUPERIOR



ANALISIS Y DISEÑO DE SOFTWARE

PRINCIPIO DE INVERSIÓN DE DEPENDENCIAS (DIP)

PRINCIPIO DE INVERSIÓN DE DEPENDENCIAS (DIP)

El Principio de Inversión de Dependencias (Dependency Inversion Principle - DIP) establece que:

- Los módulos de alto nivel no deben depender de módulos de bajo nivel. Ambos deben depender de abstracciones.
- Las abstracciones no deben depender de los detalles. Los detalles deben depender de las abstracciones.

Este principio busca reducir el acoplamiento entre componentes del sistema, promoviendo que las dependencias se establezcan hacia interfaces o clases abstractas, en lugar de implementaciones concretas (Leiva, 2021).

Problema habitual: dependencia directa de clases concretas

Cuando una clase de alto nivel (por ejemplo, un controlador o servicio principal), depende directamente de clases concretas (por ejemplo, una base de datos específica o una API externa), se vuelve difícil de modificar, escalar o testear.

Ejemplo de violación del DIP:

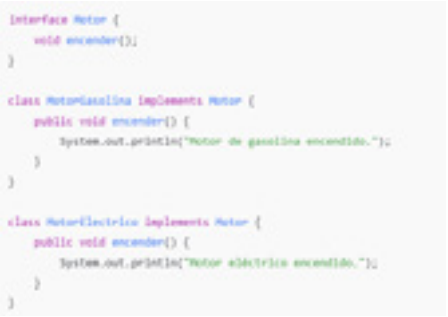
```
class MotorGasolina {
    public void encender() {
        System.out.println("Motor de gasolina encendido.");
    }
}

class Automovil {
    private MotorGasolina motor;

    public Automovil() {
        motor = new MotorGasolina(); // Dependencia directa
    }

    public void arrancar() {
        motor.encender();
    }
}
```

En este caso, Automovil depende directamente de MotorGasolina. Si se deseara cambiar a un MotorElectrico, se requerirían cambios internos en la clase Automovil, lo que indica una violación del DIP.

Solución: invertir la dependencia usando una abstracción

Para cumplir con el DIP, se introduce una abstracción (una interfaz o clase abstracta), que representa el comportamiento esperado del motor. Las clases concretas implementan dicha interfaz, y el Automóvil depende de la abstracción, no de la implementación específica (García Carmona, 2012).

Ejemplo de cumplimiento del DIP:



Ahora, Automóvil puede funcionar con cualquier tipo de Motor que implemente la interfaz, sin acoplarse a una clase específica. Esta estructura permite flexibilidad, reutilización y facilidad de pruebas.

Implementación práctica con inyección de dependencias

El cumplimiento del DIP suele complementarse con patrones como la Inversión de Control (IoC) o la Inyección de Dependencias (DI). Esto permite que las dependencias sean proporcionadas desde el exterior (por ejemplo, por un framework o contenedor), en lugar de ser creadas internamente por las clases.

Ejemplo en un framework como Spring (Java):

Aquí, el framework se encarga de inyectar una instancia concreta de Motor al Automóvil, cumpliendo con DIP de forma automática y estructurada.

Beneficios del DIP

- Reducción del acoplamiento entre componentes.
- Mayor flexibilidad al intercambiar implementaciones sin afectar al cliente.
- Facilidad de prueba, al poder inyectar mocks.
- Mejor mantenimiento y escalabilidad del sistema.