**Capitulo 1 Principios SOLID**

**Clean Code Libro donde se ven con mayor detalle**

**A Handbook of Agile Software Craftsmanship**

Los **principios SOLID** son un conjunto de cinco principios de diseño orientado a objetos que ayudan a desarrollar software más mantenible, flexible y escalable. Fueron introducidos por **Robert C. Martin** y son fundamentales para crear código limpio y bien estructurado. Aquí tienes un resumen de cada principio:

1. **Single Responsibility Principle (SRP) - Principio de Responsabilidad Única:**
   * Cada clase debe tener una única responsabilidad o razón para cambiar. Esto significa que una clase debe encargarse de una sola tarea o función específica.
   * **Ejemplo:** Si tienes una clase que gestiona tanto la lógica de negocio como la visualización de datos, deberías dividirla en dos: una para la lógica y otra para la visualización.
2. **Open/Closed Principle (OCP) - Principio de Abierto/Cerrado:**
   * Las entidades de software (clases, módulos, funciones) deben estar **abiertas para la extensión, pero cerradas para la modificación**. Esto significa que deberías poder añadir nuevas funcionalidades sin cambiar el código existente.
   * **Ejemplo:** Si tienes que añadir una nueva funcionalidad, en lugar de modificar el código actual, puedes agregar nuevas clases o métodos que extiendan el comportamiento.
3. **Liskov Substitution Principle (LSP) - Principio de Sustitución de Liskov:**
   * Los objetos de una clase derivada deben poder sustituir a objetos de su clase base sin alterar el correcto funcionamiento del programa. En otras palabras, una subclase debe ser intercambiable con su superclase sin afectar la lógica.
   * **Ejemplo:** Si una clase Ave tiene un método volar(), una subclase como Pingüino, que no vuela, no debería heredar este comportamiento, porque no cumpliría con el principio.
4. **Interface Segregation Principle (ISP) - Principio de Segregación de Interfaces:**
   * Es mejor tener muchas interfaces específicas para cada cliente que una interfaz general y grande. Las interfaces deben ser pequeñas y especializadas para evitar obligar a las clases que las implementan a incluir métodos que no necesitan.
   * **Ejemplo:** En lugar de tener una interfaz Empleado con métodos como calcularSalario() y asistirAReuniones(), es mejor tener interfaces separadas como CalculadorDeSalario y AsistenteDeReuniones, que puedan ser implementadas según sea necesario.
5. **Dependency Inversion Principle (DIP) - Principio de Inversión de Dependencias:**
   * Las clases de alto nivel no deberían depender de clases de bajo nivel, ambas deben depender de **abstracciones**. Además, las abstracciones no deben depender de detalles, sino los detalles de las abstracciones.
   * **Ejemplo:** En lugar de que un objeto de una clase de alto nivel dependa directamente de una implementación específica de bajo nivel, debería depender de una interfaz o clase abstracta, permitiendo flexibilidad para cambiar la implementación.

**Single Responsability**

Lo que se entiende de una sola responsabilidad es que cada clase debería de tener una responsabilidad única.

Ejemplo sin Single Responsability