Capitulo 3.

"Principio SOLID"

SOLID es uno de los acrónimos más famosos en el mundo de la Programación. El cual fue introducido por Robert C. Martin, quién además de crear la estructura de trabajo Clean. Diseñó SOLID que consta de 5 Principios que debe Complir el código para que sea Factible.

-Los Principios SOLID:

Los 5 Principios SOLID de diseño de aplicaciones de software son:

- S - Single Responsibility Principle (SRP)
- O - Open/Closed Principle (OCP)
- L - Liskov Substitution Principle (LSP)
- I - Interface Segregation Principle (ISP)
- D - Dependency Inversion Principle (DIP)

Entre los objetivos de tener en cuenta estos 5 Principios a la hora de escribir código encontramos:

- Crear un software eficaz, que cumpla con su cometido y que sea robusto y estable.
- Escribir un código limpio y flexible ante los cambios necesarios en un futuro, que sea reutilizable y mantenible.
- Permitir escalabilidad, que acepte ser ampliado con nuevas funcionalidades de manera ágil.

*Definiciones de Principios.

- Single responsability Principle. (Principio de responsabilidad única). Este establece que una clase, componente o microservicio debe ser responsable de una cosa.
- Open/Closed Principle (Principio de abierto/Cerrado). Establece que las entidades de software (clases, módulos y funciones) deberían estar abiertos para su extensión, pero cerrados para su modificación. Hablando de las herencias de clase.

- **Liskov substitution Principle** (Principio de Sustitución de Liskov). Se refiere a que una subclase debe ser sustituible por su superclase, y si al momento de hacer esto el programa falla, estamos violando este principio. Para cumplir este principio se confirmará que nuestro programa tiene una jerarquía de clases fácil de entender y contar con un código reutilizable.

- **Interface segregation Principle** (Principio de segregación de la interfaz). Este establece que los usuarios no deberían verse forzados a usar o depender de interfaces que no usan. Una forma de evitar es declarar diversas interfaces que se adapten a las necesidades del usuario.

- **Dependency Inversion Principle** (Principio de inversión de dependencia). Establece que se depende de abstracciones, no de clases concretas. Es decir:
  - Los módulos de alto nivel no deberían depender de módulos de bajo nivel.
  - Las abstracciones no deberían depender de los detalles.