**PRINCIPIOS SOLID**

Los principios SOLID son un conjunto de cinco principios de diseño de software que fueron introducidos por Robert C. Martin y son considerados fundamentales para lograr un diseño de software robusto, mantenible y escalable. Estos principios están destinados a ayudar a los desarrolladores a crear sistemas flexibles y fáciles de mantener. Aquí están los cinco principios SOLID:

**Principio de Responsabilidad Única (Single Responsibility Principle - SRP):**

Este principio establece que una clase debe tener una única razón para cambiar. En otras palabras, una clase debe tener una sola responsabilidad o función. Esto facilita el mantenimiento y la comprensión del código, ya que cada clase tiene un propósito claro y específico.

**Principio de Abierto/Cerrado (Open/Closed Principle - OCP):**

Este principio sugiere que una clase debe estar abierta para la extensión, pero cerrada para la modificación. En lugar de modificar el código existente, se debe extender mediante la creación de nuevas clases o módulos que agreguen funcionalidades. Esto promueve un diseño más flexible y evita cambios frecuentes en el código existente.

**Principio de Sustitución de Liskov (Liskov Substitution Principle - LSP):**

Según el principio de Liskov, los objetos de una clase derivada deben poder sustituirse por objetos de la clase base sin afectar la corrección del programa. Esto significa que las clases derivadas deben ser compatibles con sus clases base en términos de comportamiento.

**Principio de Segregación de Interfaces (Interface Segregation Principle - ISP):**

Este principio sostiene que una clase no debe verse obligada a implementar interfaces que no utiliza. En lugar de tener interfaces grandes y monolíticas, es preferible tener interfaces más pequeñas y específicas. De esta manera, las clases solo implementan las interfaces que son relevantes para ellas.

**Principio de Inversión de Dependencia (Dependency Inversion Principle - DIP):**

El principio de inversión de dependencia establece que los módulos de alto nivel no deben depender de módulos de bajo nivel, sino que ambos deben depender de abstracciones. Además, las abstracciones no deben depender de los detalles, sino al revés. Este principio busca reducir las dependencias directas entre las clases y fomentar el uso de interfaces o clases abstractas para la interacción entre componentes.

La aplicación de estos principios SOLID puede conducir a un diseño de software más flexible, mantenible y resistente a cambios. Estos principios son guías valiosas para los desarrolladores que buscan escribir código limpio y eficiente.

**EJEMPLOS**

**Principio de Responsabilidad Única (SRP):**

Ejercicio: Imagina que estás trabajando en un sistema de gestión de empleados. Divide la clase Empleado en dos clases separadas: una para la gestión de la información personal del empleado y otra para la gestión de las responsabilidades laborales. Asegúrate de que cada clase tenga una única responsabilidad.

**Principio de Abierto/Cerrado (OCP):**

Ejercicio: Tienes una clase Forma con métodos para calcular el área. Agrega la capacidad de calcular el perímetro sin modificar la clase Forma. Crea una nueva clase llamada Calculadora Perímetro que extienda la funcionalidad sin modificar la clase original.

**Principio de Sustitución de Liskov (LSP):**

Ejercicio: Define una interfaz Ave que tenga un método volar. Luego, crea las clases pájaro y Pingüino que implementen esta interfaz. Asegúrate de que ambas clases puedan sustituirse en cualquier contexto donde se espere un objeto de tipo Ave.

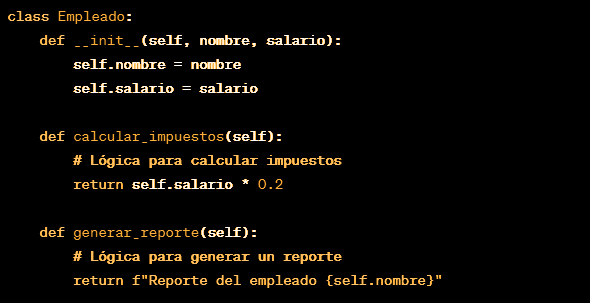
**Principio de Segregación de Interfaces (ISP):**

Ejercicio: Supongamos que tienes una interfaz llamada Trabajador con métodos como trabajar, descansar y reunirse. Divide esta interfaz en interfaces más pequeñas y específicas, como Trabajador Productivo y Trabajador Social. Implementa estas interfaces en las clases apropiadas.

**Principio de Inversión de Dependencia (DIP):**

Ejercicio: En un sistema de comercio electrónico, refactoriza una clase Carrito De Compras para que dependa de una interfaz proveedor de Inventario en lugar de una clase concreta. Luego, implementa diferentes proveedores de inventario que cumplan con esa interfaz.

Principio de Responsabilidad Única (SRP):



Principio de Inversión de Dependencia (DIP):

