Principios SOLID con Ejemplos en Java

Nombre del Docente

2 de abril de 2024

1. Introducción

Los principios SOLID son un conjunto de cinco principios de diseño de software que promueven un diseño de código limpio, mantenible y escalable. En este documento, exploraremos cada uno de los principios SOLID junto con ejemplos en Java.

2. Principios SOLID

2.1. Principio de Responsabilidad Única (SRP)

Definición 1 *El Principio de Responsabilidad Única establece que una clase debe tener solo una razón para cambiar, es decir, debe tener una única responsabilidad.*

Ejemplo 1 *Supongamos que tenemos una clase llamada* `GestorArchivos` *que se encarga de leer y escribir archivos en el disco. Esta clase tiene dos responsabilidades: leer archivos y escribir archivos. Siguiendo el SRP, deberíamos dividir esta clase en dos: una clase para leer archivos (*`LectorArchivos`*) y otra clase para escribir archivos (*`EscritorArchivos`*).*

Ejemplo 2 *Otro ejemplo del SRP es una clase* `Empleado` *que tiene métodos para calcular el salario y métodos para manejar el registro de tiempo de trabajo. En este caso, sería mejor dividir la clase en dos: una clase* `CalculadoraSalario` *y otra clase* `RegistroTiempoTrabajo`*.*

2.2. Principio de Abierto/Cerrado (OCP)

Definición 2 *El Principio de Abierto/Cerrado establece que una clase debe estar abierta para extensión pero cerrada para modificación, es decir, se debe poder extender su comportamiento sin modificar su código fuente.*

Ejemplo 3 *Supongamos que tenemos una clase* `Forma` *con un método* `calcularArea()`*. Si queremos agregar una nueva forma como un triángulo, podemos extender la clase* `Forma` *creando una nueva clase* `Triangulo` *que implemente el método* `calcularArea()`*, sin necesidad de modificar la clase* `Forma`*.*

Ejemplo 4 *Otro ejemplo del OCP es un sistema de manejo de pagos que tiene una interfaz* `MetodoPago`*. Si queremos agregar un nuevo método de pago como PayPal, podemos crear una nueva clase* `PayPal` *que implemente la interfaz* `MetodoPago` *sin modificar las clases existentes.*

2.3. Principio de Sustitución de Liskov (LSP)

Definición 3 *El Principio de Sustitución de Liskov establece que los objetos de un tipo base deben poder ser reemplazados por objetos de un subtipo sin afectar la integridad del programa.*

Ejemplo 5 *Supongamos que tenemos una clase* `Cuadrado` *que hereda de una clase* `Rectangulo` *y ambas tienen un método* `setLado()` *para establecer el lado. Si tratamos un objeto* `Cuadrado` *como un* `Rectangulo` *y establecemos su ancho, violaría el principio de LSP porque cambiaría el comportamiento esperado.*

Ejemplo 6 *Otro ejemplo del LSP es una clase* `Ave` *que tiene un método* `volar()`*. Si creamos una clase* `Pingüino` *que hereda de* `Ave` *pero no puede volar, violaría el principio de LSP.*

2.4. Principio de Segregación de Interfaces (ISP)

Definición 4 *El Principio de Segregación de Interfaces establece que una clase no debe verse obligada a implementar interfaces que no utiliza. En su lugar, deben crearse interfaces específicas para cada cliente.*

Ejemplo 7 *Supongamos que tenemos una interfaz* `Impresora` *que tiene métodos* `imprimir()` *y* `escanear()`*. Si una clase solo necesita imprimir pero no escanear, estaría obligada a implementar el método* `escanear()` *aunque no lo utilice.*

Ejemplo 8 *Otro ejemplo del ISP es una interfaz* `GestorDocumentos` *que tiene métodos* `abrir()`*,* `guardar()`*,* `imprimir()` *y* `escanear()`*. Si una clase solo necesita abrir y guardar documentos, estaría obligada a implementar los métodos* `imprimir()` *y* `escanear()` *aunque no los utilice.*

2.5. Principio de Inversión de Dependencia (DIP)

Definición 5 *El Principio de Inversión de Dependencia establece que las clases de alto nivel no deben depender de las clases de bajo nivel. Ambos deben depender de abstracciones. Además, las abstracciones no deben depender de los detalles, sino que los detalles deben depender de las abstracciones.*

Ejemplo 9 *Supongamos que tenemos una clase* `CorreoElectronico` *que depende directamente de una clase* `ServicioCorreo` *concreta. Si queremos cambiar a otro servicio de correo, tendríamos que modificar la clase* `CorreoElectronico`*. En cambio, si creamos una interfaz* `InterfazServicioCorreo` *y hacemos que* `CorreoElectronico` *dependa de esta interfaz, podemos cambiar de servicio de correo sin modificar* `CorreoElectronico`*.*

Ejemplo 10 *Otro ejemplo del DIP es una clase* `Cliente` *que depende directamente de una clase* `Servicio` *concreta. Si queremos cambiar a otro servicio, tendríamos que modificar la clase* `Cliente`*. En cambio, si creamos una interfaz* `InterfazServicio` *y hacemos que* `Cliente` *dependa de esta interfaz, podemos cambiar de servicio sin modificar* `Cliente`*.*