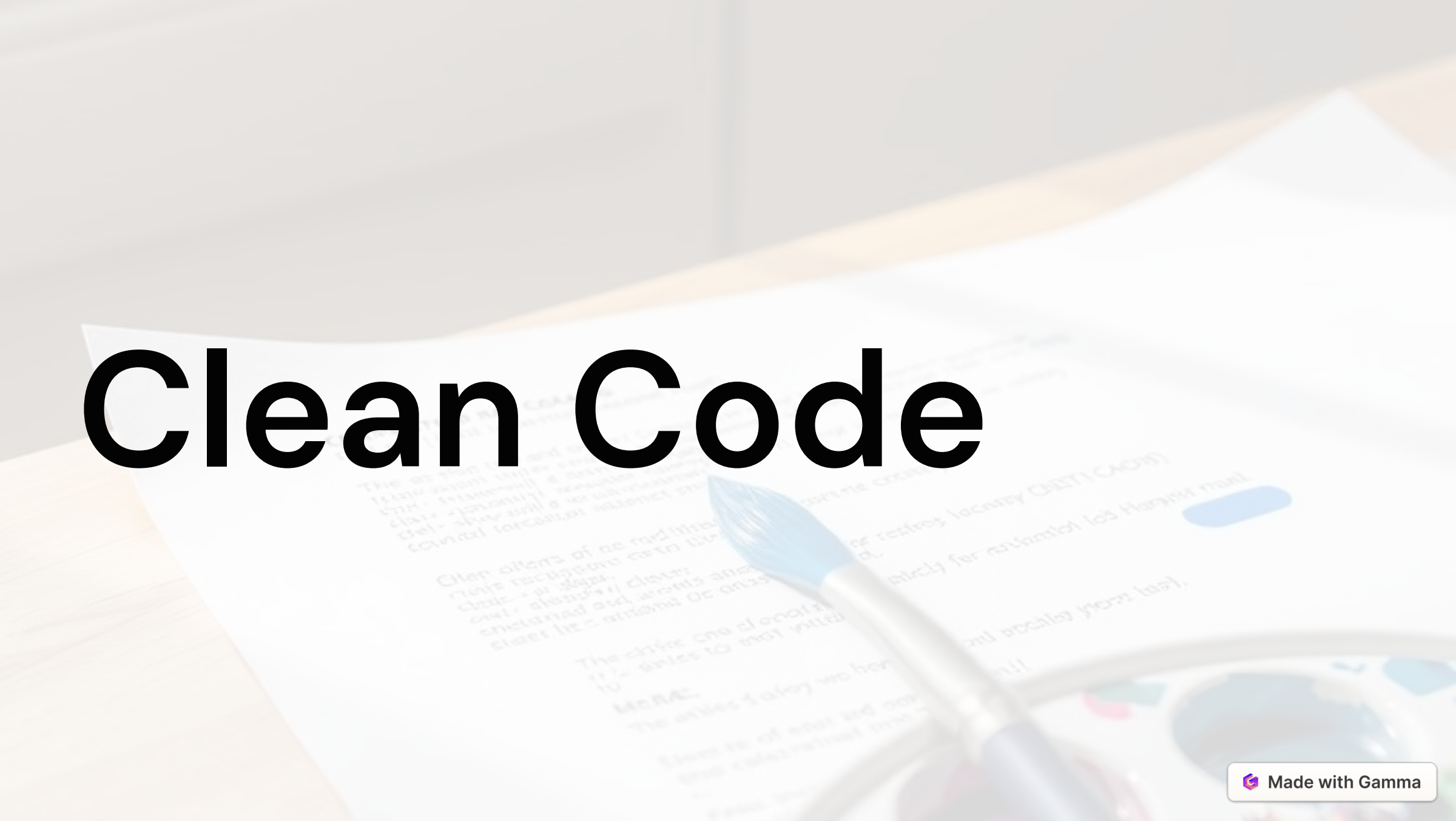
Clean Code



Código limpio

- Código fácil de entender y analizar
- Código fácil de mantener
- Código fácil de actualizar
- Código fácil de escalar

Cómo logramos código limpio

- Mantener bajo acoplamiento
- Utilizar sintaxis simple y actual
- Evitar incorporar muchas librerías de terceros
- Distribución de responsabilidades
- Creación de componentes pequeños

Principios SOLID: El fundamento de un buen diseño de software

Los principios SOLID son un conjunto de directrices para el diseño de software orientado a objetos que promueven la creación de código flexible, mantenible y extensible. Estos principios son fundamentales para la construcción de sistemas robustos y adaptables.

by Ariel Romero

Principio de Responsabilidad Única (SRP)

El SRP establece que cada clase o módulo debe tener una única responsabilidad. Esto ayuda a mantener el código limpio, modular y fácil de entender.

Beneficios del SRP

Facilita el mantenimiento y la actualización del código.

Reduce la probabilidad de errores al limitar el alcance de los cambios.

Mejora la reutilización de código al crear clases más especializadas.

Ejemplo de violación del SRP

Una clase que maneja la lógica de negocio y la persistencia de datos simultáneamente.

Principio de Abierto/Cerrado (OCP)

El OCP establece que las entidades de software deben estar abiertas para la extensión, pero cerradas para la modificación. Esto permite agregar nuevas funcionalidades sin modificar el código existente.

1 **Extensibilidad**

Agregar nuevas funcionalidades sin alterar el código base.

2 **Mantenimiento**

Reducir los riesgos de introducir errores al modificar código existente.

3 **Reutilización**

Facilitar la reutilización de código al mantener las clases separadas.

Principio de Sustitución de Liskov (LSP)

El LSP establece que las subclases deben ser sustituibles por sus clases base sin afectar el comportamiento del programa.

1

Clase base

Define un comportamiento general.

2

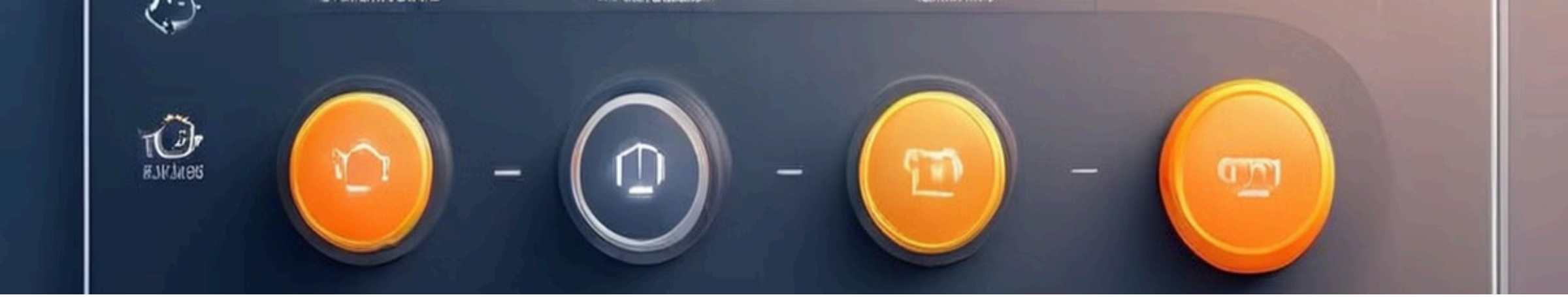
Subclase

Implementa el comportamiento específico.

3

Substitución

La subclase se puede utilizar en lugar de la clase base.



Principio de Segregación de Interfaces (ISP)

El ISP establece que las interfaces deben ser pequeñas y específicas. Las interfaces grandes pueden ser difíciles de implementar y dificultar la reutilización del código.

Ventajas	Desventajas
Interfaces más específicas	Mayor cantidad de interfaces
Mejor reutilización de código	Mayor complejidad en la gestión de interfaces
Menos dependencias entre clases	Dificultad para encontrar la interfaz correcta

Principio de Inversión de Dependencias (DIP)

El DIP establece que las clases de alto nivel no deben depender de clases de bajo nivel. En cambio, ambas deben depender de abstracciones.

Abstracciones

Interfaces o clases abstractas que definen el comportamiento general.

Dependencias

Relaciones entre clases que definen cómo se interactúan.

Implementaciones

Clases concretas que implementan las abstracciones.

Aplicación de los principios SOLID

Los principios SOLID se pueden aplicar durante todo el ciclo de vida del desarrollo de software, desde el diseño inicial hasta la implementación y el mantenimiento.

1

Diseño

Identificar las responsabilidades y las dependencias en el sistema.

2

Implementación

Utilizar interfaces y abstracciones para desacoplar las clases.

3

Pruebas

Verificar que las subclases cumplen con el LSP.

4

Mantenimiento

Facilitar la adición de nuevas funcionalidades sin afectar el código existente.

Conclusión: Beneficios de SOLID

Al seguir los principios SOLID, se crea código de alta calidad que es más fácil de mantener, actualizar y ampliar.

Flexibilidad

Facilita la adaptación del código a nuevos requisitos.

Reutilización

Permite reutilizar componentes de código de forma eficiente.

Mantenimiento

Reduce el tiempo y el esfuerzo necesarios para corregir errores.