Clean Code

O código Limpito

Que brille el código

El "Clean Code" o "Código Limpio" se refiere a un conjunto de prácticas y principios en la programación de software que apuntan a escribir código que sea fácil de entender y mantener.

La idea detrás del código limpio es que el software debe ser escrito de tal manera que sea legible por humanos, no solo por máquinas.

Esto significa que el código debería ser claro, simple, y directo, facilitando así su lectura, comprensión y modificación por parte de otros desarrolladores, incluido uno mismo en el futuro.

Algunos aspectos clave del código limpio incluyen...

Claridad: El código debería ser tan claro como sea posible, evitando complejidades innecesarias.

Simplicidad: El código no debería contener más funcionalidades de las necesarias para cumplir con sus requerimientos actuales, siguiendo el principio KISS (Keep It Simple, Stupid).

Uso de nombres significativos: Las variables, funciones, clases y otros elementos del código deben tener nombres que reflejen claramente su propósito y uso.

Código bien organizado: La estructura del código debería facilitar su lectura, con una organización lógica que haga evidente su flujo y funcionalidad.

Comentarios y documentación: Aunque un buen código debería ser autoexplicativo, los comentarios y la documentación son importantes para explicar el "por qué" detrás de decisiones complejas o no obvias.

Pruebas: El código limpio es código que ha sido bien probado. Las pruebas aseguran que el código hace lo que debe hacer y facilitan su mantenimiento y refactorización.

El tío Bob says...

La promoción del código limpio es una parte fundamental de las buenas prácticas en el desarrollo de software y es crucial para el trabajo en equipo, donde múltiples personas deben entender y contribuir al mismo código base.

Autores como Robert C. Martin (también conocido como Uncle Bob) han escrito extensamente sobre este tema, ofreciendo guías y principios para escribir código de alta calidad.

Los principios SOLID van muy de la mano del CLEAN CODE. Este nos ayuda a escribir mejor código y SOLID a diseñarlo.

S - Single Responsibility Principle (Principio de Responsabilidad Única)

O - Open-Closed Principle (Principio de Abierto-Cerrado)

L - Liskov Substitution Principle (Principio de Sustitución de Liskov)

I - Interface Segregation Principle (Principio de Segregación de Interfaz)

D - Dependency Inversion Principle (Principio de Inversión de Dependencia)

1. **Principio del Boy Scout**

Este principio se basa en dejar el código en un estado mejor de lo que estaba antes de tocarlo. Al igual que los **Boy Scouts** dejan un campamento mejor de lo que lo encontraron, los desarrolladores deben mejorar el código cada vez que lo modifican, aunque sea en pequeñas mejoras.

2. **Ley de la Navaja de Ockham**

Esta ley establece que entre varias explicaciones o soluciones para un problema, la más simple suele ser la correcta. En términos de Clean Code, esto se traduce en buscar la solución más simple y directa para implementar una funcionalidad.

3. **Principio DRY (Don't Repeat Yourself)**

Este principio promueve la reutilización del código para evitar la repetición. La idea es escribir cada pieza de información una vez y no duplicar lógica o datos innecesariamente.

4. **Principio YAGNI (You Aren't Gonna Need It)**

Este principio sugiere no agregar funcionalidades o código que no se necesitan actualmente. En lugar de anticipar futuros requisitos, se debe escribir código que satisfaga los requisitos actuales.

5. **La regla de los tres pases**

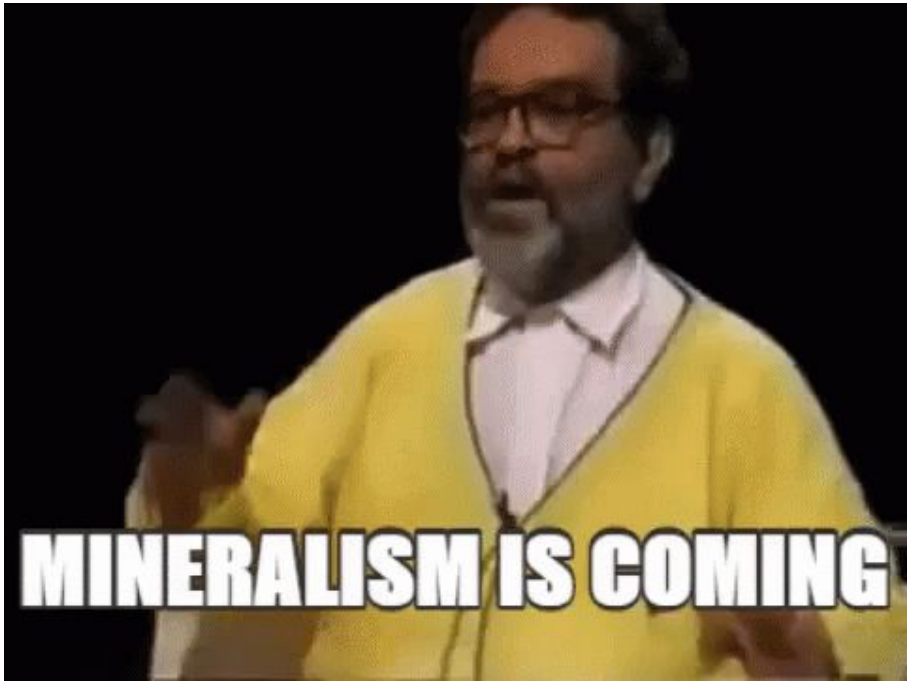
Esta regla establece que el código debe escribirse en tres etapas:

primero hacerlo funcionar,
luego hacerlo bien (refactorizar),
y **finalmente** hacerlo rápido (optimizar).

Esto ayuda a mantener un enfoque pragmático en el desarrollo de software.

6. **Principio de la minimalidad**

La minimalidad en el código se refiere a escribir la menor cantidad de código necesaria para lograr el resultado deseado. Esto incluye evitar la redundancia, mantener las funciones pequeñas y eliminar el código muerto.



7. **Principio del pitón herido**

Este principio se refiere al instinto humano de evitar lo que parece peligroso o dañado. En términos de código, sugiere que si un fragmento de código parece complicado o confuso, es mejor refactorizarlo o reescribirlo para que sea más comprensible y seguro.

8. **Principio del cinturón negro**

Este principio compara el código con las artes marciales, donde un cinturón negro es un maestro que ha dominado las técnicas básicas y puede aplicarlas con precisión. En términos de código, implica dominar los fundamentos de la programación y aplicarlos con habilidad y precisión para escribir un código limpio y eficiente.

9. **Principio de la responsabilidad única**

Este principio establece que cada módulo o componente de software debe tener una única responsabilidad. Si un módulo tiene más de una responsabilidad, es más difícil de entender, probar y mantener.

Es la **S** de los principios **SOLID** de programación (SRP)

10. **Principio de mínima sorpresa**

El código debería comportarse de manera predecible y no debería sorprender a quien lo lee o lo usa. Esto significa que las funciones y clases deben hacer lo que su nombre sugiere, y los efectos secundarios inesperados deben evitarse.

Naming y testing

BONUS. **DATA añade 3 más**

Just do it, but THINK before: 80% pensar y planificar. 20% Actuar: Actuar sin pensar hará que dupliques el tiempo de pensar y seguir actuando con ensayo error en un futuro.

Divide y vencerás: Un problema es un problema en conjunto, en partes son solo inconvenientes.

Parálisis por análisis: Si algo no funciona, buscar solución, sino buscar otra alternativa. Ser autosuficiente y resolutivo.