Clean Code Avanzado

Índice

Code

Repositorio en GitHub:

- M21_CCA: https://github.com/LabsAdemy/m2i_cca

Requerimientos ejecución local

- NODE: https://nodejs.org/es/download/
- GIT : https://git-scm.com/download/wintps

Instrucciones:

```
git clone https://github.com/LabsAdemy/m2i_cca.git
cd m2i_cca
npm i
```

Mostrar la intención y ocultar los detalles

- Evitar los *code smells* escribiendo un código fácil de leer y modificar.
- Muchas estructuras y funciones pequeñas y bien nombradas.

I - Clean Code – Code smells

Lack of Style

Código homogéneo, consistente y sujeto a estándares.

Bad name

Nombres para datos y verbos para funciones.

Magic numbers

Explicación y configuración

Long method

Los métodos pequeños siempre son mejores (nombres que documentan, una comprensión más fácil, menos código duplicado).

Deeply nested

Estructuras repetitivas y condicionales unas dentro de otras.

Long parameter list

Propio del estilo procedimental en lugar de orientado a objetos. También puede indicar que el método haga demasiadas cosas.

Ungrouped data groups

Un conjunto de variables que siempre aparecen juntas... pero no se organizan.

Large class

Si tiene demasiadas variables, propiedades o métodos seguro que viola el principio de "responsabilidad única".

II – Principles of change: the Good ones

Y.A.G.N.I. You aren't going to need it.

Implementa cosas cuando las necesites, no cuando preveas que las necesitas.

K.I.S.S. Keep It Simple, Stupid!

La simplicidad es un objetivo para evitar la complejidad innecesaria.

D.R.Y. Don't repeat yourself.

Cada regla o atributo debe tener una representación única e inequívoca dentro de un sistema.

P.O.L.A. Principle Of Least Astonishment.

No me sorprendas; no me hagas pensar.

H.P. Hollywood principle.

No nos llames, ya te llamaremos. Inversión del control y Push en lugar de Pull.

T.D.A. Tell don’t ask.

Decirle a los objetos lo que quieres que hagan con sus propios datos, no consultarles para actuar con ellos después.

C.Q.S. Command–Query Segregation.

Cada método debe ser o un comando que realice una acción o una consulta que devuelva datos; pero no ambos.

CoC Convention over configuration.

Establecer y cumplir convenios que minimicen la cantidad de decisiones necesarias.

LoD Low of Demeter.

Un objeto debe asumir lo menos posible acerca de cualquier otro. Antiglobalización!.

Obstáculos del cambio

Rigidez Un cambio afecta a muchas partes. Al cambiar un objeto hay que cambiar otros muchos.

Fragilidad Las errores saltan en lugares inesperados. Los cambios en un objeto tienen efectos en otros muchos.

Inmovilidad No se puede reutilizar el código fuera de su entorno. Los cambios en un objeto dependen de otros muchos.

Divergent change.
Una clase que se cambia de diferentes maneras o por diferentes razones.

Shotgun surgery.
Un cambio que requiere cambios en muchas clases. Difícil encontrarlos, fácil olvidarse.

Cyclomatic complexity.
Número de rutas únicas que aumenta con el anidamiento, los switches y las condiciones complejas.

Feature envy and Inappropriate Intimacy.
Método que parece más interesado en una clase distinta de la suya y clases que se conocen demasiado.

Middle Man
Si una clase delega todo su trabajo a otra clase, ¿por qué existe?.

Primitive obsession
Uso de valores primitivos en lugar de una clase, o estructura que aporte cohesión y validación.

Encapsular lo que varía y depender de interfaces

Single responsibility principle

Un objeto solo debería tener una única responsabilidad, o razón para cambiar.

Open/closed principle

Las entidades deben estar abiertas para su extensión, pero cerradas para su modificación.

Liskov substitution principle

Los objetos deberían ser reemplazables por subtipos sin alterar el funcionamiento del programa.

Interface segregation principle

Muchas interfaces específicas son mejores que una interfaz de propósito general.

Dependency inversion principle

Depender de abstracciones, no de implementaciones. Resolver usando la Inyección de Dependencias.

IV - Patrones de diseño : Creacionales

¿Cómo instanciar un objeto o grupo de objetos relacionados?

Proporcionan mecanismos de creación de objetos que aumentan la flexibilidad y la reutilización del código existente.

Abstract Factory: una factoría de factorías sin exponer nada concreto.

Builder: crea subtipos de objetos y facilita la construcción de otros complejos.

Factory Method: delega la lógica de creación de instancias a clases secundarias.

Simple Factory: genera una instancia sin exponer lógica de creación.

Singleton: asegura una instancia única de una clase.

IV - Patrones de diseño : Estructurales

¿Cómo montar un componente a partir de objetos?

Explican cómo ensamblar objetos y clases en estructuras más grandes, manteniendo las estructuras flexibles y eficientes.

Adapter: envuelve un objeto en un adaptador para que sea compatible con otro.

Bridge: se trata de preferir la composición sobre la herencia.

Composite: permite tratar con objetos individuales de manera uniforme.

Decorator: permite agregar comportamiento envolviendo un objeto en otro.

Façade: proporciona un acceso simple a un sistema de objetos complejo.

Flyweight: reduce el consumo de memoria o CPU compartiendo recursos.

Proxy: una clase actúa como representante de otra.

IV - Patrones de diseño : De Comportamiento

¿Cómo ejecutar una funcionalidad entre varios objetos?

Cuidan la comunicación efectiva y la asignación de responsabilidades entre objetos.

Chain of Responsibility: encadena llamadas entre objetos o métodos.

Command: encapsula acciones en objetos.

Iterator accede a los elementos de un conjunto sin revelar cómo.

Mediator: desacopla dos objetos comunicándose con ambos.

Memento: guarda el estado actual para un uso futuro.

Observer: notifica cambios a suscriptores interesados.

Strategy: cambia el algoritmo según las circunstancias.

Layered

Separación de responsabilidad por tecnología
Presentación -> Lógica -> Infraestructura

CQRS

Separación de responsabilidad por uso de los datos
Es CQS llevado al modelo, al despliegue y al almacenamiento
Puede requerir **Event Driven Architectures**

Port – Adapters

Un puerto es una interfaz
Los adaptadores son sus implementaciones
La lógica (el dominio) es independiente de lo demás (los detalles)
Variantes: **Hexagonal, Onion, Clean**


Formación