20-3-23

Clean code

Si el código no es bueno, con el tiempo la productividad comienza a decaer.

Objetivos → código mantenible
- código claro
- que sea un código legible
- código flexible
- código simple

Variables

Nombres significativos y pronunciables, es preferible nombres claros antes que comentarios. Que los nombres revelen su intención. no dejar valores fijos en el código, usar constantes.

Métodos

Elegir una palabra por concepto. No mezclar palabras que hacen lo mismo, usar siempre la misma. Funciones cortas (no más de 8 líneas por función)
Asignar una sola responsabilidad por método. Evitar los switch, porque rompe el hecho de realizar solamente una cosa.
No está bueno que los métodos tengan muchos argumentos, probablemente quedó sin modelar una entidad. Evitar los booleanos como argumento, es preferible utilizar polymorfismo.
No tener funciones que repitan lógica.
Los métodos deben hacer lo que dice su nombre.
Revisar código, e ir haciendo mini refactors para mejorar la calidad.

Comentarios

Menos documentacion, mejor. Hacer métodos con nombres lo suf claros, de tal forma que no se requieran comentarios

Buenos comentarios → legales
→ informativos
→ explicación de una intención (algo que no queda claro porque se hizo así)

Comentario TODO → Bueno solo para uso temporal
→ no deben quedar ni siquiera compilados

Malos comentarios → Redundancia (si no se actualizan, pueden traer errores)
→ Comentarios obligatorios
→ Ruido
→ Código comentado (no se debe commitear cód comentado)

Formateo

Los linters nos permite formatear a estandares generados.

Test unitarios → clave hacerlo en el momento justo.

Principios de diseño

- Un buen diseño para ucear con el cambio y la complejidad.
- Síntomas de un mal diseño → Fragilidad (ante un cambio ⇒ bug)
  - Rigidez (cambio x y yes romper y)
  - Inmobilidad (cuando quiero reutilizar algo en otro módulo y no sirve)
  - Viscosidad (tener que hacer otra cosa que la planeada por mayor facilidad)

hay síntomas cuando hay un bajo acoplamiento y una alta cohesión

- Para lograr un buen diseño → SOLID
  - Principio de única responsabilidad (Single Responsability) → que cada método tenga una única responsabilidad. Que haya un único motivo de cambio en lo que se está trabajando. Es mejor para llevar acabo los test (unitarios)
  - Open - closed principle → abierto para la extensión, cerrado para la modificación. Solo permite algunos cambios. Si no lo tengo que modificar, significa que está correcta. Extensión a nuevos casos. Buena solución al mantener las cosas simples.
  - Principio de sustitución (Liskov Substitution principle) → ¡Herencia! A pesar de que muchas funcionalidades se compartan puede que no sea una clase derivada de una clase base. Se tiene que cumplir el "ES UN".
  - Segregación de interfaces. (Interface Segregation principle) → Dar a conocer lo menor posible. Solo darle lo que necesitas de mi objeto y no su totalidad. Segregar interfaces en las funcionalidades. Es menos propenso a cambios.

- Inversion de dependencias (Dependency Inversion principle) → Un obj necesita de conocer a otro para "funcionar".
La problemática es que la dependencia es directa
El diseño es inmóvil, se debe invertir la dependencia, para lograr la movilidad.
Imp → la idea es pq el diseño sea lo más abstracto posible.

Material en el campus para reforzar conceptos. Los patrones cumplen estos ppios.

Hay un ejercicio para entregar individual para el jueves 30, está en el campus.