SOLID

O que a Universidade não me ensinou!

invillia
infinite.
digital.power

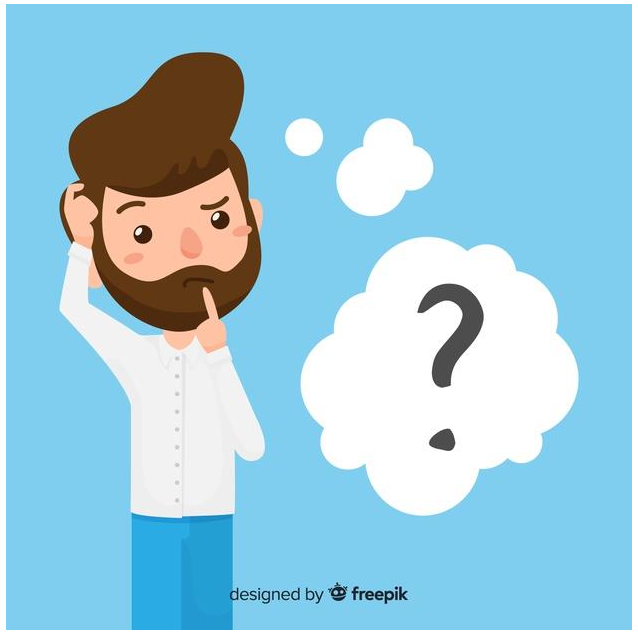
SOLID

- É um acrônimo criado por Michael Feathers (ano 2000), após observar que cinco princípios da orientação a objetos e design de código (criado por Robert C. Martin, também conhecido por Uncle Bob) e abordados em um artigo chamado "Principles of OOD", poderiam se encaixar nesta palavra.

Michael Feathers - https://michaelfeathers.typepad.com/
Uncle Bob - https://pt.wikipedia.org/wiki/Robert_Cecil_Martin
Artigo Principles of OOD - http://butunclebob.com/ArticleS.UncleBob.PrinciplesOfOod/

SOLID

- Seguindo esses princípios, nós, como desenvolvedores, conseguiremos gerar código limpo, fácil de dar manutenção e de evoluir.



SOLID

- **S — Single Responsiblity Principle** (Princípio da responsabilidade única)
- **O — Open-Closed Principle** (Princípio Aberto-Fechado)
- **L — Liskov Substitution Principle** (Princípio da substituição de Liskov)
- **I — Interface Segregation Principle** (Princípio da Segregação da Interface)
- **D — Dependency Inversion Principle** (Princípio da inversão da dependência)

SRP — Single Responsibility Principle

- Princípio da Responsabilidade Única — **Uma classe deve ter um, e somente um, motivo para mudar.**

- Parece algo fácil e até trivial, mas será que fazemos isso?
  - Quantas classes você conhece realizando diversas tarefas? Já criou alguma?

- O princípio da responsabilidade única não se limita somente a classes, ele também pode ser aplicado em métodos e funções, ou seja, tudo que é responsável por executar uma ação, deve ser responsável por apenas aquilo que se propõe a fazer.
- Princípio mais importante do SOLID, pois sem ele, os demais princípios sequer existiriam.
- Lida com a Coesão e Acoplamento das classes.

OCP — Open Closed Principle

- Princípio Aberto-Fechado — **Objetos ou Entidades devem estar abertos para extensão, mas fechados para modificação.**

- Esse princípio dificultou um pouco meu entendimento. O que exatamente é isso?

- Separe o comportamento extensível por trás de uma interface e inverta as dependências.

LSP — Liskov Substitution Principle

- Princípio da substituição de Liskov — **Uma classe derivada deve ser substituível por sua classe base.**
- introduzido por Barbara Liskov em sua conferência "Data abstraction" em 1987. A definição formal de Liskov diz que:
  - *se S é um subtipo de T, então os objetos do tipo T, em um programa, podem ser substituídos pelos objetos de tipo S sem que seja necessário alterar as propriedades deste programa.*

- *Para não quebrar este princípio devemos tomar cuidado para:*
  - *Não sobrescrever métodos e estes não fazerem nada.*
  - *Não lançar uma exceção inesperada.*
  - *Não alterar retorno de valores de tipos diferentes da classe base (Classe Pai)*

ISP — Interface Segregation Principle

- Princípio da Segregação da Interface — **Uma classe não deve ser forçada a implementar interfaces e métodos que não irão utilizar.**

- Esse princípio basicamente diz que é melhor criar interfaces mais específicas ao invés de termos uma única interface genérica.

DIP — Dependency Inversion Principle

- Princípio da Inversão de Dependência — **Dependa de abstrações e não de implementações.**
- Segundo Uncle Bob:
  - Módulos de alto nível não devem depender de módulos de baixo nível. Ambos devem depender de abstrações.
  - Abstrações não devem depender de detalhes. Detalhes devem depender de abstrações.

- **Importante:** *Inversão de Dependência* **não é igual** a *Injeção de Dependência,* **fique ciente disso**! A I*nversão de Dependência* é um princípio (Conceito) e a I*njeção de Dependência* é um padrão de projeto (D*esign Pattern*).

Conclusão

- Percorrer o caminho olhando esses princípios, nos ajudará sermos melhores desenvolvedores.
- Relembrar esses princípios pode nos abrir a mente para problemas cotidianos que estamos vivenciando.

- Espero que tenham gostado!
  - Obrigado.