SOLID

Princípio da Responsabilidade Única (SRP)

Uma classe deve ter um, e apenas um, motivo para ser modificada.

Alguns benefícios do Single Responsibility Principle:

* Complexidade do código reduzida, mais explícita e direta;
* Facilitação da legibilidade;
* Redução de acoplamento;
* Código limpo e testável;
* Facilidade de evolução.

Princípio do Aberto/Fechado (OCP)

Entidades de software (classes, módulos, funções, etc) devem estar abertas para extensão, mas fechadas para modificação.

É uma das chaves da orientação a objetos, quando um novo comportamento ou funcionalidade precisar ser adicionado é esperado que as existentes sejam estendidas e e não alteradas, assim o código original permanece intacto e confiável enquanto as novas são implementadas através de extensibilidade.

Princípio da Substituição de Liskov (LSP)

Classes derivadas devem poder ser substituídas por suas classes base.

Preste atenção na sua hierarquia de classes! Faça bom uso do polimorfismo e não esqueça que uma relação “É UM” se refere a COMPORTAMENTO.

# Princípios SOLID: Princípio da Segregação de Interface (ISP)

O Princípio da Segregação de Interface trata da coesão de interfaces e diz que clientes não devem ser forçados a depender de métodos que não usam.

# Princípios SOLID: Princípio da Inversão de Dependência (DIP)

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.

Identificar abstrações e inverter as dependências garantem que o software seja mais flexível e robusto, estando melhor preparado para mudanças.