SOLID com Java

**S.O.L.I.D: Os 5 princípios da POO**

**1 – Princípio da responsabilidade única**

**2 – Princípio Aberto-Fechado**

**3 – Princípio da substituição de Liskov**

**4 – Princípio da Segregação da Interface**

**5 – Princípio da inversão da dependência**

**1 – Single Responsibility Principe:**

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

Esse princípio declara que uma classe deve ser especializada em um único assunto e possuir apenas uma responsabilidade dentro do software, ou seja, a classe deve ter uma única tarefa ou ação para executar.

*\*****God Class* — Classe Deus:**Na programação orientada a objetos, é uma classe que sabe demais ou faz demais.

**2 - OCP — Open-Closed Principle:**

Princípio Aberto-Fechado — **Objetos ou entidades devem estar abertos para extensão, mas fechados para modificação**, ou seja, quando novos comportamentos e recursos precisam ser adicionados no software, devemos estender e não alterar o código fonte original.

**3 - LSP— Liskov Substitution Principle:**

Princípio da substituição de Liskov — **Uma** **classe derivada deve ser substituível por sua classe base**.

**Exemplos de violação do LSP:**

* Sobrescrever/implementar um método que não faz nada;
* Lançar uma exceção inesperada;
* Retornar valores de tipos diferentes da classe base;

Para não violar o Liskov Substitution Principle, além de estruturar muito bem as suas abstrações, em alguns casos, você precisara usar a injeção de dependência e também usar outros princípios do SOLID, como por exemplo, o Open-Closed Principle e o Interface Segregation Principle

Seguir o **LSP** nos permite usar o polimorfismo com mais confiança. Podemos chamar nossas classes derivadas referindo-se à sua classe base sem preocupações com resultados inesperados.

**4 -  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.

**5 - DIP — Dependency Inversion Principle**

Princípio da Inversão de Dependência — **Dependa de abstrações e não de implementações.**

No contexto da programação orientada a objetos, é comum que as pessoas confundam a Inversão de Dependência com a [Injeção de Dependência](https://pt.wikipedia.org/wiki/Inje%C3%A7%C3%A3o_de_depend%C3%AAncia), porém são coisas distintas, mas que relacionam entre si com um proposito em comum, deixar o código desacoplado.

**\***A I*nversão de Dependência* é um princípio (Conceito) e a I*njeção de Dependência* é um padrão de projeto (D*esign Pattern*)

**\*** *“Programe para uma interface e não para uma implementação.”*

*Fonte:*

*https://medium.com/desenvolvendo-com-paixao/o-que-%C3%A9-solid-o-guia-completo-para-voc%C3%AA-entender-os-5-princ%C3%ADpios-da-poo-2b937b3fc530*