SOLID é o acrônimo usado para representar os 05 princípios da orientação a objetos:

1. S – Single Responsability Principle – Princípio da Responsabilidade Única.

Uma classe deve ter um, e somente um, motivo para mudar.

2. O – Open Closed Principle – Princípio Aberto-Fechado.

Objetos ou entidades devem estar abertos para extensão, mas fechados para modificação.

3. L – Liskov Substitution Principle – Princípio da Substituição de Liskov.

Uma classe derivada deve ser substituível por sua classe base.

4. I – Interface Segregation Principle – Princípio da Segregação da Interface.

Uma classe não deve ser forçada a implementar interfaces e métodos que não irão utilizar.

5. D – Dependency Inversion Principle – Princípio da Inversão da Dependência.

Dependa de abstrações e não de implementações.

Esses princípios ajudam o programador a escrever códigos mais limpos, separando responsabilidades, diminuindo acoplamentos, facilitando na refatoração e estimulando o reaproveitamento do código.

No exercício de hoje, iremos trabalhar o SRP.

1. SRP – Single Responsability Principle.

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.

Quando estamos aprendendo programação orientada a objetos, sem sabermos, damos a uma classe mais de uma responsabilidade e acabamos criando classes que fazem de tudo – *God Class*. Num primeiro momento isso pode parecer eficiente, mas como as responsabilidades acabam se misturando, quando há necessidade de realizar alterações nessa classe, será difícil modificar uma dessas responsabilidades sem comprometer as outras. Toda alteração acaba sendo introduzida com um certo nível de incerteza em nosso sistema – principalmente se não existirem testes automatizados.

A violação do SRP pode gerar alguns problemas, sendo eles:

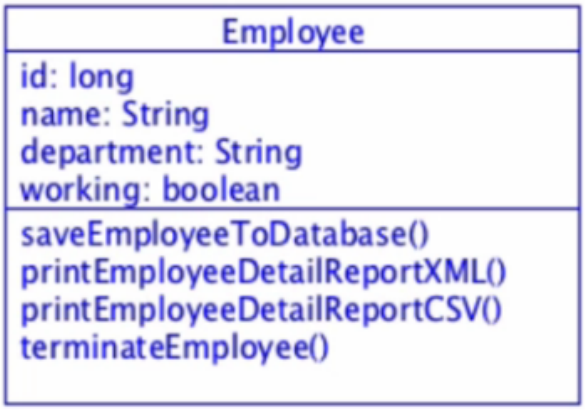
- Falta de coesão – uma classe não deve assumir responsabilidades que não são suas;
- Alto acoplamento – mais responsabilidades geram um maior nível de dependências, deixando o sistema engessado e frágil para alterações;
- Dificuldades na implementação de testes automatizados – é difícil de mockar esse tipo de classe;
- Dificuldades para reaproveitar o código;

Grupo:

1.
2.
3.
4.

Exercício:

A classe abaixo representa Empregado.



Exercício:

1. Discuta em grupo e liste abaixo as violações ao princípio SRP.
2. Proponha um novo projeto de classes, utilizando diagrama de classes UML.
3. Discuta na classe a proposta do novo projeto de classes.
4. Implemente um main(), projeto e implemente uma simulação das classes e publique o código no repl.it, encaminhando o link via entrega de tarefa no classroom.