SOLID

Na programação de computadores, SOLID (responsabilidade única, abrir-fechar, substituição Liskov, segregação interface e inversão de dependência) é uma sigla introduzida por Michael Feathers para os "primeiros cinco princípios" nomeado por Robert C. Martin em início de 2000, que representa cinco princípios básicos da programação orientada ao objeto e projeto . A intenção é que estes princípios, quando aplicados em conjunto, vai torná-lo mais provável que um programador irá criar um sistema que seja fácil de manter e estender ao longo do tempo. Os princípios da SOLID são diretrizes que podem ser aplicadas ao trabalhar em software para remover código com mal cheiro , fornecendo um quadro através do qual o programador pode refatorar o software até que ele é ao mesmo tempo fique

legível e extensível. É parte de uma estratégia global de ágil e desenvolvimento de software adaptado.

Os aperfeiçoamentos são muitas vezes fruto da refatoração - processo de alteração de um código de modo que o comportamento externo não se modifique, mas que sua estrutura interna seja melhorada.

Algumas técnicas de refatoração são:

- Extrair Método (Extract Method) □ É quando se têm partes similares do código que podem ser agrupadas em um único método com um nome que incentive a assimilação da função. □ A principal motivação para esta refatoração é eliminar do código métodos muito extensos que realizam mais de uma tarefa e geralmente precisam estar cheios de comentários para que seja compreendido.

O código Java abaixo demonstra a aplicação da refatoração para extrair método (Extract Method).

- **Antes**

```
/** Salva o produto no banco de dados. */
```

```
public void save() {
    // Verifica propriedades
    if (this.getName() == null) {
        throw new Exception("Falta nome");
    }
    else if (this.getDescription() == null) {
        throw new Exception("Falta a descrição");
    }

    this.getDatabase().save(this);
}
```

- **Depois**

```
/** Salva o produto no banco de dados. */
public void save() {
    this.checkProperties();
    this.getDatabase().save(this);
}

/** Verifica as propriedades do produto. */
private void checkProperties() {
    if (this.getName() == null) {
            throw new Exception("Falta nome do
produto.");
    }
    else if (this.getDescription() == null) {
          throw new Exception("Falta a descrição do
produto.");
```

```
        }
    }
```

- Mover Método (Move Method)☐ Quando você possui um método que será utilizado muitas vezes por outra classe diferente da classe na qual ele foi definido, crie um método similar nesta classe que o está sendo mais utilizando, então substitua as chamadas para utilizarem o método local. Pode-se também avaliar se o método original deve ser preservado ou se pode ser eliminado.

- Mover Atributo (Move Field) ☐Similar à técnica Mover Método, é aplicada quando outra classe diferente da classe onde o atributo foi definido, o utiliza muitas vezes. Nesse caso devemos apenas copiar o atributo de uma classe para outra e alterar as chamadas para que passem a utilizar o atributo local. Mais uma vez cabe avaliar se o atributo original deve ou não ser eliminado.

- Extrair Classe (Extract Class)

Quando você possui uma classe que está executando a tarefa de duas, então crie uma nova classe, depois mova todos os atributos relevantes para esta nova classe. □ A motivação desta técnica é bastante óbvia, cada classe deve ser coesa e ter objetivos bastante claros. Quando um desenvolvedor adiciona uma nova funcionalidade em certa classe, talvez não lhe parecesse tão errada, mas com o tempo esta funcionalidade irá crescer, e trazer mais responsabilidades para a classe. □ Ao extrair estas responsabilidades da classe e a colocando em outra, há um ganho em coesão, legibilidade e reutilização do código.

- Encapsular Atributo (Encapsulate Field)

Técnica utilizada para distinguir atributos semelhantes os agrupando em objetos ou métodos de acesso distintos. □Isto evita o conflito ou confusão ao manuseá-los.

- Renomear Método (Rename Method) □
Quando o nome de um método não revela qual a sua intenção, deve-se altera-lo imediatamente, de modo que o novo nome deve descrever claramente o que ele se propõe a fazer.

O Princípio da Inversão de Dependência (DIP) também é primordial para um bom design orientado a objetos, ao passo que se trata de uma maneira específica para desacoplar as dependências entre os objetos, modificando a maneira tradicional como as estabelecemos.

Sua definição formal é: □ 1. Componentes de mais alto nível não devem depender de componentes de níveis mais baixos, mas ambos devem depender de abstrações. □ 2. Abstrações não devem depender de implementações, mas as implementações devem depender de abstrações.

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

- SOFA

A aplicação da Refatoração acaba remetendo a uma outra pratica conhecida como "**SOFA"** Que seria uma uma forma resumida dos requisitos **Solid** seguindo os mesmos princípios.

S (short) - O Método deve ser curto.

O (one thing) - Deve fazer uma única tarefa.

F (few) - Deve possuir poucos argumentos.

A (abstraction) - Manter um nível consistente de abstração.

Pontos positivos :

Fácil manutenção, entendimento e organização;

Arquitetura aberta a receber atualizações, melhorias e novos recursos sem danos colaterais;

Aplicação de testes de forma fácil e de simples entendimento;

Fácil reaproveitamento de código;

Fácil adaptação a mudanças no escopo do projeto.

Pontos negativos :

Repetição de Código, ou seja, uma simples alteração deve ser replicada em vários pontos diferentes da sua aplicação;

Código sem estrutura coesa ou padronizada;

Rigidez e fragilidade, ou seja, qualquer alteração provoca uma cascata de operações ou falhas em várias partes do sistema;

Dificuldade na execução e criação de testes;

Não reaproveitamento, ou seja, nenhuma ou quase nenhuma funcionalidade pode ser reaproveitada para outros sistemas.

Referencias :

https://en.wikipedia.org/wiki/SOLID_(object-oriented_design)

https://pt.wikipedia.org/wiki/Refatora%C3%A7%C3%A3o