modulo03_exercicios_semana05

M3S03Ex01

Clean Code é uma prática de desenvolvimento de código que ressalta que um bom código deve ser simples e direto, de fácil legibilidade, evitar ao máximo que outros desenvolvedores, ao lerem o código, fiquem confusos e revelando sua real intenção. Ele foi criado pelo Roberto C. Martin, visando auxiliar equipes de desenvolvimento ágil.

Os fundamentos englobam:

- Identação de código: (formatação e recuo) facilita a organização do código, contribui para a legibilidade e ajuda a entender a ordem do que está escrito no código.
- Convenções de funções nomeadas: devem consistir de títulos autoexplicativos para melhorar o entendimento assim que alguém lê esse nome, também aplicado para variáveis e classes.
- Evitar nomes genéricos: sempre quando possível, colocar nomes que forneçam contexto, sem ser muito curtos ou muitos longos e cuidar com abreviações e formatação.
- Comentários limpos e concisos: quando necessário, é importante evitar textos muito longos nos comentários. Devem ter como objetivo auxiliar o suporte e a legibilidade do código.
- Reusabilidade e escabilidade: a reutilização é uma meta essencial. Isso pode reduzir custos e recursos de desenvolvimento. Isso também evita repetições, abordado na prática DRY - Don’t Repeat Yourself. Também é importante ter em mente a escabilidade do sistema, já que nos features são desenvolvidas constantemente.
- Ter uma boa prática de testes: isso garante que o código não e quebre e gere mais problemas para outros desenvolvedores que entrarem em contato com o código

M3S03Ex02

```
public void run() {
  while(ballsPresent()){
    if(frontIsClear()){
      move();
    }
  }
}
```

M3S03Ex03

```
public class Produto {
  private String nome;
  private int quantidade;

  public produto() {
    //Código do construtor
  }

  public void mostraProduto(){
    //Código do método que apresenta o construtor para o usuário
  }
}
```

M3S03Ex04

```
public class Computador{
  private String texto_docs;

  public computador() {
    //Código do construtor
  }

  public void gravaNaMemoria(itemParaGravar){
      //Código da função
  }
}
```

M3S03Ex05

Os testes que eu faria seriam os unitários, primeiro para verificar se a classe Computador existe e se o método está sendo executado corretamente, se ele grava apenas textos (e não outros tipos de dados).

M3S03Ex06

- Single Responsibility Principle - princípio da responsabilidade única
- Open-Closed Principle - princípio aberto-fechado
- Liskov Substituition Principle -princípio da substituição de Liskov
- Interface Segregation Principle - princípio da segregação da interface
- Dependency Inversion Principle - princípio da inversão da dependência

M3S03Ex07

A principal vantagem do POO é que ajuda na legibilidade do código, pois um dos seus objetivos é aproximá-lo do mundo real, além de usar a abstração. Também auxilia na reutilização de partes do código, já que a organização deste é realizada em classes. Então em suma, ajuda a desenvolver uma aplicação mais robusta, permite que novas features sejam desenvolvidas com menos risco envolvido, facilita a escrita de testes e aumenta a organização do código de uma maneira geral

M3S03Ex08

O código exemplificado está com nomes genéricos para seus métodos e atributos (método1, atributo1), não fornecendo contexto sobre o que o código faz e dificulta o entendimento rápido. Além disso, ele possui muitos parâmetros, o que configura um ‘code smell’ e possui uma mensagem, como se fosse um log, fazendo “hard coding”, dificultando o rastreamento do código.

M3S03Ex09

Esse código também está com nomes genéricos, como ‘metodo1’, ‘valor’, além do fato do método não ter nenhuma funcionalidade, fazendo com que seja um algoritmo ineficiente.