**Grupo:** 24
**Alunos:** Marcos Deus, Gabriel Costa
**Data:** 15/08/2024
**Disciplina:** Técnicas de programação para plataformas emergentes

# Trabalho 3

**Questão 1:**

## 1. Simplicidade

Simplicidade no código significa que ele deve ser fácil de entender e de modificar. Um código simples é claro e direto, com a menor complexidade possível.

Relação com Maus-Cheiros de Código:

**Long Method:** Métodos longos geralmente são complexos e difíceis de entender. Simplificar métodos longos em partes menores e mais compreensíveis melhora a simplicidade.
**Large Class:** Classes que fazem muitas coisas tendem a ser complexas. Dividir essas classes em classes menores e com responsabilidades bem definidas ajuda a manter a simplicidade.
**Primitive Obsession:** Usar primitivas em vez de abstrações pode tornar o código complexo e difícil de entender. Criar classes para encapsular dados complexos melhora a simplicidade.

## 2. Elegância

A elegância no código refere-se a soluções que são não apenas corretas, mas também refinadas e bem projetadas. O código elegante é eficiente e usa boas práticas de programação para atingir seus objetivos de forma clara e eficaz.

Relação com Maus-Cheiros de Código:

**Feature Envy:** Métodos que mostram inveja de outras classes são um sinal de que o design pode não estar elegante. Reorganizar o código para que métodos e dados relacionados estejam juntos melhora a elegância.
**Inappropriate Intimacy:** Quando as classes têm um relacionamento excessivamente íntimo, o design pode parecer desordenado e pouco elegante. Usar a refatoração para reduzir a dependência e melhorar a separação entre classes torna o código mais elegante.

## 3. Modularidade

Modularidade refere-se à prática de dividir o código em módulos (ou componentes) independentes que podem ser desenvolvidos, testados e mantidos separadamente.

Relação com Maus-Cheiros de Código:

**Duplicated Code:** O código duplicado indica uma falta de modularidade. Refatorar o código duplicado em métodos ou classes reutilizáveis para melhorar a modularidade.
**Long Parameter List:** Passar muitos parâmetros pode sinalizar que o código não está bem modularizado. Usar objetos para agrupar parâmetros relacionados pode ajudar a modularizar melhor o código.
**Data Clumps:** Dados agrupados indicam que talvez haja uma falta de modularidade. Criar uma classe para encapsular dados relacionados para melhorar a modularidade.

## 4. Boas Interfaces

Boas interfaces são interfaces de classe que são claras, intuitivas e fornecem apenas os métodos necessários para o cliente interagir com a classe.

Relação com Maus-Cheiros de Código:

**Switch Statements:** O uso excessivo de switch statements pode indicar interfaces mal projetadas. Polimorfismo pode substituir switch statements e criar interfaces mais limpas e intuitivas.
**Alternative Classes with Different Interfaces:** Quando classes têm métodos semelhantes com assinaturas diferentes, isso pode indicar uma interface ruim. Refinar interfaces para que sejam consistentes melhora a clareza e a usabilidade.

## 5. Extensibilidade

Extensibilidade é a capacidade do código ser facilmente modificado para adicionar novas funcionalidades sem alterar o código existente de forma significativa.

Relação com Maus-Cheiros de Código:

**Refused Bequest:** Subclasses que não utilizam o comportamento herdado podem indicar problemas de extensibilidade. Refinar a hierarquia de classes e usar delegação em vez de herança pode melhorar a extensibilidade.
**Speculative Generality:** Implementações excessivamente genéricas podem dificultar a extensão do código. Remover código especulativo e focar em necessidades reais ajuda a manter o código extensível.

## 6. Evitar Duplicação

Evitar duplicação significa garantir que o código não tenha seções redundantes e que o mesmo comportamento não esteja implementado em múltiplos lugares.

Relação com Maus-Cheiros de Código:

**Duplicated Code:** O principal cheiro de código associado à duplicação. Refatorar o código duplicado em métodos ou classes comuns ajuda a eliminar esse problema e promove a reutilização de código.

## 7. Portabilidade

Portabilidade refere-se à capacidade do código de ser executado em diferentes ambientes ou plataformas com o mínimo de alterações.

Relação com Maus-Cheiros de Código:

**Incomplete Library Class:** Quando uma classe da biblioteca não atende às necessidades, pode ser necessário adicionar comportamentos específicos. Usar extensões e métodos estrangeiros pode ajudar a tornar a classe mais portátil.

## 8. Código deve ser Idiomático e Bem Documentado

O código deve seguir as convenções e padrões da linguagem de programação utilizada e deve ser documentado de maneira clara para facilitar a compreensão e manutenção.

Relação com Maus-Cheiros de Código:

**Comments:** Comentários são frequentemente usados para explicar código que não é idiomático ou que é difícil de entender. Refatorar o código para torná-lo mais claro e idiomático pode reduzir a necessidade de comentários.
**Long Method, Large Class:** Métodos e classes grandes geralmente são menos idiomáticos e podem se beneficiar de refatoração para torná-los mais claros e melhor documentados.
Ao aplicar os princípios de um bom projeto de código e resolver os maus cheiros descritos, você pode melhorar significativamente a qualidade e a manutenção do seu código.

**Questão 2**: Analisando os Mau-Cheiros no trabalho prático 2

## 1. Long Method (Método Longo)

**Onde ocorre:**

Método execute() na classe CalcularValores.
Método calcularValores() na classe Venda.
Método realizarVenda() na classe SistemaVarejo.

**Princípio Violado:** Simplicidade

**Refatoração Aplicável:**

Extrair Método: Dividir esses métodos em partes menores, cada uma com uma única responsabilidade, pode melhorar a clareza e a manutenção.

## 2. Large Class (Classe Grande)

**Onde ocorre:**

Classe Venda ainda possui diversas responsabilidades.
Classe SistemaVarejo acumula várias responsabilidades relacionadas ao gerenciamento de clientes, produtos e vendas.

**Princípio Violado:** Modularidade

**Refatoração Aplicável:**

Extrair Classe: Criar classes separadas para lidar com cashback e impostos na classe Venda. Dividir SistemaVarejo em módulos mais específicos como GerenciadorDeClientes, GerenciadorDeProdutos e GerenciadorDeVendas

## 3. Feature Envy (Inveja de Função)

**Onde ocorre:**

Na classe CalcularValores, métodos como calcularICMS() e calcularImpostoMunicipal() utilizam muitos detalhes da classe Cliente.
Na classe Venda, métodos que dependem muito da estrutura interna de Cliente para cálculos.

**Princípio Violado:** Elegância

**Refatoração Aplicável:**

Mover Método: Mover esses métodos para a classe Cliente ou criar uma interface específica para esses cálculos na classe Cliente.

## 4. Duplicated Code (Código Duplicado)

**Onde ocorre:**

Lógica de desconto, ICMS, e imposto municipal em Venda e CalcularValores.
Verificação do tipo de cliente em vários lugares (Cliente e Venda).

**Princípio Violado:** Evitar Duplicação

**Refatoração Aplicável:**

Extrair Método ou Consolidar Condicionais: Consolidar a lógica de cálculo em um único método ou classe, evitando duplicação e promovendo a consistência.

## 5. Comments (Comentários Excessivos)

**Onde ocorre:**

Seções de código que possuem muitos comentários explicativos.

**Princípio Violado:** Código deve ser Idiomático e Bem Documentado

**Refatoração Aplicável:**

Refatorar para Melhor Legibilidade: Escrever código mais claro e autoexplicativo, reduzindo a necessidade de comentários extensivos.

## 6. Long Parameter List (Lista Longa de Parâmetros)

**Onde ocorre:**

Construtor de CalcularValores e vários métodos dentro de SistemaVarejo e Venda.

**Princípio Violado:** Boas Interfaces

**Refatoração Aplicável:**

Introduzir Objeto Parâmetro: Criar um objeto que encapsule todos os parâmetros relacionados e o passe como um único parâmetro para simplificar a interface.

## Referências

Martin Fowler. Refactoring: Improving the design of Existing Code. Addison-Wesley Professional, 1999.

Pete Goodliffe. Code Craft: The practice of Writing Excellent Code. No Starch Press, 2006