Relatório de Refatoração de Programa Java

Introdução

O projeto de software inicialmente desenvolvido seguiu uma estrutura orientada a objetos, com o objetivo de cadastrar clientes, produtos, realizar vendas e calcular impostos, descontos e fretes, conforme definido pelos requisitos do enunciado. No entanto, após a conclusão do desenvolvimento, foi necessário refatorar o código para melhorar sua legibilidade, manutenção e qualidade geral. Este relatório explora o processo de refatoração, relacionando-o com os princípios de bom projeto de código e os maus-cheiros identificados por Martin Fowler.

Princípios de Bom Projeto de Código

1. **Simplicidade**
    - **Definição**: Um código simples é fácil de entender, modificar e manter. Evita-se complexidade desnecessária e foco na clareza.
    - **Maus-cheiros** relacionados: *Complexidade desnecessária* e *Código difícil de entender*.
    - **Análise no Projeto**: Durante a refatoração, foram removidos métodos desnecessariamente complexos, como funções que combinavam múltiplas responsabilidades (por exemplo, calcular desconto e frete ao mesmo tempo). As funções foram divididas para atender ao princípio de responsabilidade única.
2. **Elegância**
    - **Definição**: Código elegante é aquele que resolve problemas de forma eficiente e limpa, sem redundâncias ou excessos.
    - **Maus-cheiros** relacionados: *Código duplicado* e *Soluções improvisadas*.
    - **Análise no Projeto**: Foi identificado código duplicado em cálculos de impostos para diferentes tipos de clientes. Isso foi refatorado para uma abordagem mais genérica, reutilizando funções comuns e eliminando duplicações.
3. **Modularidade**
    - **Definição**: Modularidade é a capacidade do código ser dividido em partes independentes que podem ser desenvolvidas, testadas e mantidas separadamente.
    - **Maus-cheiros** relacionados: *Classes grandes* e *Métodos longos*.

- **Análise no Projeto**: O projeto original continha classes de vendas que eram responsáveis por múltiplas funcionalidades. A refatoração envolveu a divisão dessas classes em módulos menores e coesos, separando a lógica de cálculo de impostos, gerenciamento de clientes e processamento de pagamentos.

4. **Boas Interfaces**
    - **Definição**: Interfaces bem definidas garantem que os módulos se comuniquem de maneira clara e estável.
    - **Maus-cheiros** relacionados: *Interfaces desnecessárias* ou *Métodos que não são intuitivos*.
    - **Análise no Projeto**: Algumas interfaces públicas possuíam métodos confusos ou mal documentados. Foi realizada uma revisão para garantir que todos os métodos fossem intuitivos, bem nomeados e tivessem uma responsabilidade clara.
5. **Extensibilidade**
    - **Definição**: Extensibilidade refere-se à facilidade de adicionar novas funcionalidades ao código existente sem alterar significativamente o código já implementado.
    - **Maus-cheiros** relacionados: *Código rígido* e *Falta de flexibilidade*.
    - **Análise no Projeto**: Inicialmente, o projeto possuía uma estrutura rígida, onde a adição de novos métodos de pagamento ou tipos de cliente exigia alterações em várias partes do código. A refatoração focou na aplicação do padrão de projeto *Strategy*, permitindo que novos comportamentos fossem adicionados sem grandes modificações.
6. **Evitar Duplicação**
    - **Definição**: Evitar duplicação significa manter o código DRY (*Don't Repeat Yourself*), garantindo que funcionalidades similares sejam implementadas de maneira única e reutilizável.
    - **Maus-cheiros** relacionados: *Código duplicado*.
    - **Análise no Projeto**: Durante a revisão, foram encontrados trechos de código duplicados, como a lógica de cálculo de impostos para clientes de diferentes estados. Isso foi consolidado em uma função reutilizável.
7. **Portabilidade**

- **Definição**: Código portável pode ser facilmente adaptado para diferentes ambientes e plataformas.
- **Maus-cheiros** relacionados: *Código dependente de plataforma*.
- **Análise no Projeto**: O código foi revisado para garantir que nenhuma dependência de plataforma específica estivesse presente, como caminhos de arquivos ou configurações específicas de um sistema operacional.

8. **Código Idiomático e Bem Documentado**
    - **Definição**: Código idiomático segue as convenções da linguagem de programação usada, e um código bem documentado facilita sua compreensão por outros desenvolvedores.
    - **Maus-cheiros** relacionados: *Comentários obsoletos* e *Código de difícil leitura*.
    - **Análise no Projeto**: Algumas partes do código não seguiam as convenções de nomenclatura do Java, e a documentação era escassa. Durante a refatoração, o código foi alinhado às boas práticas da linguagem, e a documentação foi expandida para descrever as principais funcionalidades e fluxos de execução.

Maus-Cheiros Persistentes e Possíveis Refatorações

Apesar da refatoração, alguns maus-cheiros ainda persistem no projeto. Abaixo estão os maus-cheiros identificados e as operações de refatoração que podem ser aplicadas para eliminá-los:

1. **Métodos Longos**
    - **Princípio violado**: Simplicidade, Elegância.
    - **Operação de refatoração**: Extrair Método - Dividir métodos longos em métodos menores e mais específicos.
2. **Classes Grandes**
    - **Princípio violado**: Modularidade.
    - **Operação de refatoração**: Extrair Classe - Reorganizar a lógica em classes menores com responsabilidades bem definidas.
3. **Código Duplicado**
    - **Princípio violado**: Evitar Duplicação.

- **Operação de refatoração**: Consolidar Funções Duplicadas - Identificar padrões repetidos e movê-los para funções reutilizáveis.

4. **Dependência Externa Excessiva**
    - **Princípio violado**: Extensibilidade.
    - **Operação de refatoração**: Introduzir Abstração - Reduzir acoplamento com dependências externas usando interfaces ou fábricas para criar instâncias de objetos.

Conclusão

O processo de refatoração melhorou significativamente a qualidade do código, seguindo os princípios de bom projeto de software. No entanto, o trabalho de refatoração é contínuo, e alguns maus-cheiros ainda precisam ser abordados em futuras iterações. O código agora é mais modular, fácil de entender, e preparado para futuras extensões, o que facilita sua manutenção a longo prazo.