1.Testes TDD

1.1Estrutura do Projeto

O projeto consiste em diversas classes que modelam o comportamento de usuários, compras, produtos e endereços. As principais classes e suas responsabilidades são:

- **Address:** Representa um endereço e contém métodos para identificar a região e se a cidade é capital.
- **Card**: Representa um cartão de crédito.
- **Buy:** Representa uma compra realizada por um usuário.
- **Product:** Representa um produto da loja.
- **User:** Representa um usuário da loja, podendo ser do tipo standard, prime ou special.
- **Store:** Gerencia as compras realizadas na loja.
- **TaxCalculator:** Calcula os impostos ICMS e municipais.
- **FreightCalculator:** Calcula o valor do frete com base na localização e tipo de usuário.

1.2 Testes Desenvolvidos

- **TestAdress:** A classe Address possui métodos para obter a região do endereço e verificar se a cidade é uma capital.
- **TestBuy:** A classe Buy representa uma compra realizada por um usuário. Os testes para esta classe foram desenvolvidos para validar o cálculo do total da compra, incluindo descontos, impostos e frete.
- **TestCard:** A classe Card representa um cartão de crédito. Os testes para esta classe verificam se o cartão foi criado corretamente com o número de cartão fornecido.
- **TestProductTax:** A classe Product possui métodos para calcular impostos (ICMS e municipais) sobre o produto. Os testes para esta classe verificam se esses cálculos estão corretos.
- **TestStore:** A classe Store gerencia as compras realizadas na loja. Os testes para esta classe verificam se as compras são adicionadas e removidas corretamente da lista de compras.
- **TestUser:** A classe User representa um usuário da loja. Os testes para esta classe verificam se os diferentes tipos de usuários (standard, prime e special) são criados corretamente e se suas propriedades são configuradas corretamente.

2. Princípios de Bom Projeto de Código e Maus-Cheiros de Código

2.1 Princípio de Coesão

- **Definição**: Coesão se refere ao grau em que os elementos de um módulo (ou classe) estão relacionados e trabalham juntos para cumprir uma única responsabilidade. Uma classe altamente coesa tem um único propósito bem definido.

- **Maus-cheiros Relacionados**:
    - *Large Class (Classe Grande)*: Quando uma classe tem muitas responsabilidades, violando o princípio da coesão.
    - *Long Method (Método Longo)*: Um método que faz muitas coisas, o que pode indicar uma falta de coesão.

2.2 Princípio de Acoplamento

- **Definição**: Acoplamento é o grau de dependência entre módulos ou classes. Um bom projeto de código busca um baixo acoplamento para garantir que mudanças em uma classe/módulo tenham um impacto mínimo em outras partes do sistema.
- **Maus-cheiros Relacionados**:
    - *Feature Envy (Inveja de Função)*: Quando um método em uma classe está muito interessado nos detalhes de outra classe, resultando em um acoplamento excessivo.
    - *Inappropriate Intimacy (Intimidade Inapropriada)*: Quando duas classes estão excessivamente acopladas, conhecendo detalhes internos uma da outra.

2.3 Princípio de Encapsulamento

- **Definição**: Encapsulamento refere-se à prática de esconder os detalhes internos de uma classe e expor apenas o que é necessário para o uso externo. Isso ajuda a proteger a integridade dos dados e facilita a manutenção.
- **Maus-cheiros Relacionados**:
    - *Data Clumps (Agrupamento de Dados)*: Quando certos grupos de variáveis aparecem repetidamente juntos, violando o princípio de encapsulamento.
    - *Data Class (Classe de Dados)*: Uma classe que só contém campos e métodos de acesso sem comportamentos próprios.

3. Identificação de Maus-Cheiros no Trabalho Prático 2

3.1 Maus-Cheiros Persistentes

- *Large Class*: Se o código que implementa as diferentes regras para clientes (padrão, especiais e prime) está concentrado em uma única classe, isso sugere uma classe com muitas responsabilidades, violando o princípio de coesão.
- *Feature Envy*: Se há métodos que dependem excessivamente dos detalhes internos de outras classes, como o cálculo de descontos e impostos em métodos que deveriam ser independentes, isso indica um acoplamento elevado.

3.2 Princípios Violados e Operações de Refatoração

- **Princípio Violado**: *Coesão*
    - **Operações de Refatoração**: Aplicar *Extract Class* para separar responsabilidades distintas em classes menores e mais coesas. Por exemplo, uma classe para calcular descontos, outra para impostos, e outra para frete.
- **Princípio Violado**: *Acoplamento*

    - **Operações de Refatoração**: Usar *Move Method* para mover métodos que estão muito acoplados a outras classes para as classes apropriadas. Por exemplo, o cálculo do cashback pode ser movido para uma classe específica de cashback.
- **Princípio Violado**: *Encapsulamento*
    - **Operações de Refatoração**: Aplicar *Encapsulate Field* para proteger os dados e *Extract Method* para simplificar métodos longos que estão expondo muitos detalhes internos.

Com essas análises, o grupo pode focar em melhorar a estrutura do código, seguindo os princípios de bom design e considerando as operações de refatoração para remover os maus-cheiros identificados.