Testes de Software - Prof. Eiji Adachi

Trabalho: Aplicação de Testes Funcionais e Testes Estruturais e Implementação de Testes Automatizados para Funcionalidade de Finalização de Compra em um E-commerce

---

Contexto do Trabalho

Você recebeu um projeto backend de uma aplicação de e-commerce estruturada como uma **API REST**, organizada em três camadas principais:

- **Controller**: Responsável por lidar com as requisições HTTP.
- **Service**: Contém a lógica de negócio.
- **Repository**: Responsável pela interação com o banco de dados.

Neste trabalho, o foco será implementar **testes de unidade automatizados** para a funcionalidade de **finalizar compra**, que faz parte de um processo de checkout em um e-commerce.

Este trabalho pode ser feito em grupos de 1 a 2 membros.

Descrição da Funcionalidade de Finalização de Compra

A funcionalidade de finalizar compra recebe como entradas um identificador para um **cliente** e um identificador para um **carrinho de compras**, que encapsula um conjunto de produtos selecionados pelo cliente. Ela realiza as seguintes operações:

1. **Consulta ao serviço de estoque**: Verifica se há quantidade suficiente de cada produto no estoque.
2. **Cálculo do preço total da compra**: Soma o valor total dos produtos no carrinho e aplica as regras de negócio descritas a seguir.
3. **Consulta ao serviço de pagamentos**: Verifica se o pagamento foi autorizado.
4. **Atualização do estoque**: Se o pagamento for autorizado, dá baixa no estoque, reduzindo a quantidade dos produtos.

Seu objetivo é implementar testes que cubram a funcionalidade de `calcularCustoTotal` de forma isolada.

Objetivos do Trabalho

Implementar a o cálculo do preço total da compra:

- Implementar o **método de cálculo do preço total da compra** na camada de serviço.

Testar a Camada de Serviço:

- Implementar testes automatizados compatíveis com JUnit 5 para o **método de cálculo do preço total da compra** na camada de serviço.
- Aplicar **critérios de testes de caixa preta**:
    - Identificar as partições do domínio;

    - Identificar os valores limites do domínio;
    - Elaborar tabela de decisão especificando as regras do domínio;
    - Criar uma classe de teste JUnit com métodos de teste cobrindo todas as partições identificadas;
    - Criar uma outra classe de teste JUnit com métodos de teste cobrindo todos os valores limites identificados;
    - Criar uma outra classe de teste JUnit com métodos de teste cobrindo todas as regras identificadas na tabela de decisão;
    - Documentar os casos de teste numa planilha ou numa tabela no README relacionando ID do teste, entrada, resultado esperado e critério coberto (partição, limite ou regra de decisão).

- Aplicar critérios de testes **caixa branca**:

    - **Cobertura obrigatória de arestas (branch coverage)**: atingir **100% de cobertura de arestas** no método de cálculo do custo total da compra.

    - **Cobertura MC/DC (Modified Condition/Decision Coverage)**: para decisões compostas, demonstrar que cada condição individual influencia o resultado da decisão e que foi devidamente coberta por um teste. O README deve conter uma tabela "decisão × condições × casos de teste", com marcação de T/F e o caso correspondente. Esse detalhamento deve ser feito apenas para a decisão composta mais complexa do seu código.

    - **Complexidade e independência de caminhos**: No README, incluir:

        - O **grafo de fluxo de controle (CFG)** do método;
        - A **complexidade ciclomática (V(G))** calculada;
        - O número mínimo de casos de teste independentes necessários (≥ V(G)).

- Além dos casos válidos, devem ser criados testes de validação e robustez, cobrindo entradas inválidas (ex.: quantidade ≤ 0, preços negativos, cliente nulo, etc.), com uso de `assertThrows`.

- Cada teste deve conter uma mensagem de falha descritiva (`assertThat(...).as("descrição do cenário")`) e nomes de método autoexplicativos -- ex.: `calcularCustoTotal_quandoPesoIgual10Kg_entaoAplicaFreteDe2PorKg` -- ou usar `@DisplayName`.

Boas Práticas Recomendadas

- Nomeie métodos de teste de forma descritiva, indicando condição e resultado esperado.
- Use `@BeforeEach` para inicialização de dados comuns.
- Prefira **AssertJ** para comparar `BigDecimal`: `assertThat(total).isEqualByComparingTo("414.00");`
- Evite valores mágicos: declare constantes (`DESCONTO_10 = new BigDecimal("0.10")`).
- Utilize **JUnit 5 Parameterized Tests** sempre que for adequado.
- Inclua testes de exceção com `assertThrows` e mensagens de erro verificadas.

Regras para cálculo do custo da compra

Definições

O **peso tributável** do item é definido como:

```
peso_tributável = max(peso_físico, peso_cúbico)
```

onde o **peso cúbico** é calculado como:

```
peso_cúbico = (C × L × A) / 6000
```

Os cálculos devem ser feitos com **duas casas decimais**, e o **arredondamento final** (half-up) ocorre **somente no valor total da compra**.

O **peso total** da compra é a soma dos pesos tributáveis de todos os itens, já considerando as quantidades.

O cliente possui um **nível de fidelidade**, que pode ser:

- **Ouro**
- **Prata**
- **Bronze**

O **CEP de entrega** pertence a uma **região** com um **multiplicador de frete**:

Região	Multiplicador
Sudeste	1,00
Sul	1,05
Nordeste	1,10
Centro-Oeste	1,20
Norte	1,30

Ordem de cálculo

1. **Subtotal dos itens**

   `Subtotal = soma(preço_unitário × quantidade)`

   (somente os produtos, sem considerar o frete)

2. **Desconto por múltiplos itens de mesmo tipo**

Se um carrinho de compras tiver diferentes `ItemCompra` com produtos de mesmo tipo (`TipoProduto`), concede-se desconto escalonado da seguinte forma:

- 3 a 4 itens do mesmo tipo → **5% de desconto**
- 5 a 7 itens do mesmo tipo → **10% de desconto**
- 8 ou mais itens do mesmo tipo → **15% de desconto**

O desconto é aplicado apenas ao subtotal dos itens dessa categoria.

3. **Desconto por valor de carrinho**
   - Se `Subtotal > R$ 1000,00` → **20% de desconto**

- Senão, se `Subtotal > R$ 500,00` → **10% de desconto**
- Caso contrário → **sem desconto**

Pode acumular com o **desconto por múltiplos itens de mesmo tipo**. Neste caso, o desconto por tipo é aplicado **antes** do desconto por valor de carrinho.

4. **Cálculo do frete base (por peso total)**
   - **Faixas de peso (kg):**

Faixa	Condição	Valor por kg
A	0,00 ≤ peso ≤ 5,00	**Isento (R$ 0,00)**
B	5,00 < peso ≤ 10,00	**R$ 2,00/kg**
C	10,00 < peso ≤ 50,00	**R$ 4,00/kg**
D	peso > 50,00	**R$ 7,00/kg**

   - Se a faixa não for isenta, **some uma taxa mínima de R$ 12,00** ao valor do frete.
   - Para cada item **marcado como frágil**, some **R$ 5,00 × quantidade** (taxa de manuseio especial).
   - Após somar todos os encargos, **multiplique pelo fator da região** definido anteriormente.
5. **Benefício de nível do cliente (só sobre o frete)**
   - **Ouro:** 100% de desconto no frete (frete final = R$ 0,00)
   - **Prata:** 50% de desconto sobre o frete calculado
   - **Bronze:** paga o frete integral

   > Mesmo para clientes Ouro, o frete deve ser calculado conforme as regras anteriores, mas o valor final é zerado após o cálculo.
6. **Total da compra**

   ```
   Total = (Subtotal com desconto) + (Frete após desconto de nível)
   ```

   O resultado final deve ser **arredondado para duas casas decimais**.

Critérios de Avaliação

1. **Correção dos Testes**: Verificar se os testes cobrem adequadamente os diferentes cenários e se validam corretamente o comportamento esperado da aplicação.
2. **Cobertura dos Critérios de Teste**: Avaliar se os critérios de testes funcionais e estruturais foram adequadamente aplicados e se atendem o enunciado.
3. **Organização e Boas Práticas**: Avaliar a organização do código dos testes, seguindo boas práticas de nomenclatura, estrutura de testes e clareza.

Entrega

Você deverá entregar esta atividade como um projeto Maven nomeado com os nomes dos membros seguindo o padrão nome1-nome2 (ex.: JoaoSilva-JoseSouza). Lembre-se de também alterar o nome do projeto, usando o mesmo padrão de nome, no atributo `artifactId` do arquivo `pom.xml`. Este projeto deverá ser compactado em formato .zip e entregue via SIGAA.

É obrigatório um arquivo do tipo README.md descrevendo como executar o projeto, como executar os testes e como verificar a cobertura dos testes. Também é obrigatório o projeto conter um documento (pode ser uma planilha a parte do README) com todo o projeto dos casos de testes (partições, limites, tabela de decisão, MC/DC, etc).