# Testes de Software – Prof. Eiji Adachi

## Trabalho da 3ª Unidade – Testes de Mutação e Testes com Dublês (Fakes e Mocks)

### 1. Contexto

Este trabalho é uma continuação direta do enunciado da 2ª unidade, no qual você testou o método `calcularCustoTotal()` de uma aplicação de e-commerce.

Nesta etapa, você deverá testar uma versão simplificada do cálculo de custo total e, adicionalmente, criar testes para o método `finalizarCompra()`, utilizando fakes e mocks para simular dependências.

O trabalho pode ser realizado individualmente ou em dupla.

---

# 2. Método Simplificado `calcularCustoTotal()`

Para esta unidade, você deverá implementar novamente o método de calcular custo total, mas desta vez uma versão reduzida que segue apenas as regras abaixo.

### 2.1 Regra de Desconto

Apenas uma regra: **Desconto por valor total do carrinho**

- Se valor total >= R$ 1000,00, então aplique 20% de desconto.
- Se valor total >= R$ 500,00 && < R$ 1000,00, então aplique 10% de desconto.
- Outros valores: sem desconto.

### 2.2 Regra de Frete

Com o peso total calculado com base apenas no peso físico, aplica-se a seguinte tabela:

| Faixa | Peso total | Valor do frete |
|---|---|---|
| A | 0–5 kg | frete isento (R$ 0) |
| B | > 5 kg e ≤ 10 kg | R$ 2,00 por kg |
| C | > 10 kg e ≤ 50 kg | R$ 4,00 por kg |
| D | > 50 kg | R$ 7,00 por kg |

**Observações importantes:**

- **Produtos frágeis:** Para cada item marcado como “frágil”, são cobrados **R$ 5,00 adicionais por unidade**.
- Não existe adicional por região nem desconto por perfil de fidelidade (cliente Ouro, Prata ou Bronze).

### 2.3 Ordem de Cálculo

1. Subtotal = soma do preço unitário multiplicado pela quantidade de cada item.
2. Aplicar o desconto conforme a regra acima.
3. Calcular o frete conforme a regra acima.
4. Total = subtotalComDesconto + frete.
5. O valor final deve ser arredondado para duas casas decimais.

---

## 3. Testes Obrigatórios para `calcularCustoTotal()`

### 3.1 Cobertura Estrutural

Os testes devem atingir obrigatoriamente:

- 100% de cobertura de arestas (*branch coverage*).

O relatório do JaCoCo deve demonstrar claramente que todas as arestas do método foram cobertas.

### 3.2 Mutação

É obrigatório:

- Utilizar PITEST para análise de mutação.
- Atingir 100% de mutantes mortos no método.
- Documentar no README:
    - Linha de comando usada.
    - Como verificar que não restaram mutantes sobreviventes.
    - Estratégias usadas para matar mutantes sobreviventes.

---

## 4. Testes do Método `finalizarCompra()`

Além dos testes estruturais do cálculo, você deverá criar testes para o método `finalizarCompra()`, garantindo a simulação de todo o comportamento usando dublês adequados.

### 4.1 Objetivo do Teste

Criar testes automatizados cobrindo o fluxo completo:

1. Verificação de estoque via serviço externo.
2. Cálculo do custo total.
3. Autorização de pagamento.
4. Atualização do estoque e finalização da compra.

Os testes devem atingir 100% de cobertura de decisão do método `finalizarCompra()`. Devem verificar também:

- Se os métodos esperados foram invocados.

- Se o resultado retornado corresponde ao comportamento especificado.

## 4.2 Cenários de Teste

Você deve implementar dois cenários de testes distintos, conforme especificação a seguir. Cada cenário deve ser implementado em um arquivo de teste (arquivo .java) distinto.

### Cenário 1

No cenário 1, você deve criar fakes para simular serviços externos definidos pelas interafaces `IEstoqueExternal` e `IPagamentoExternal`, garantindo que se atinja a meta de cobertura de teste. As outras dependências devem ser implementadas usando mock objects implementados com Mockito.

### Cenário 2

No cenário 2, você deve criar mock objects para simular serviços externos definidos pelas interafaces `IEstoqueExternal` e `IPagamentoExternal`, garantindo que se atinja a meta de cobertura de teste.

Você deve usar fakes para implementar as dependências com a camada `repository`.

---

# 5. Entrega

---

A entrega deve conter:

1. Projeto Maven compactado em formato ZIP.
2. Nome do projeto e `artifactId` do `pom.xml` no formato nome1-nome2.
3. Arquivo README.md contendo:
    - Nome dos autores
    - Instruções de execução
    - Como rodar os testes
    - Como visualizar os relatórios de cobertura
    - Como gerar e interpretar o relatório de mutação