Prática 08 - Prática de Dublês de Teste

Breno Farias da Silva

2025-06-20

Introdução

Este relatório descreve a aplicação de técnicas de *dublês de teste* (*Test Doubles*) no contexto da disciplina **PPGCC12 - Teste de Software**. A atividade consiste na implementação e estudo dos exemplos apresentados nas seções 6.3.2 e 6.3.3 do livro *Effective Software Testing*, de Maurício Aniche.

O objetivo principal é compreender na prática o uso de *mocks*, *stubs* e *wrappers* para dependências, avaliando quais classes devem ou não ser *mockadas*, bem como entender como criar abstrações para dependências como data e hora.

Descrição dos Casos Implementados

Caso da Seção 6.3.2 — *BookStore*

O sistema simula uma livraria (*BookStore*) capaz de calcular o preço total de um pedido, considerando a disponibilidade dos livros em estoque. Foram implementadas as seguintes classes:

- *BookStore*: classe principal que processa o carrinho de compras.
- *BookRepository*: interface responsável por buscar livros. Esta interface foi *mockada* nos testes.
- *BuyBookProcess*: interface que representa o processo de compra, também *mockada* nos testes.
- *Book*: entidade que representa um livro.
- *Overview*: objeto que acumula o preço total e os livros não disponíveis.

Caso da Seção 6.3.3 — *ChristmasDiscount*

O sistema aplica um desconto de 15% no Natal (25 de dezembro). Para isso, foi criada uma abstração chamada *Clock*, que encapsula chamadas ao sistema relacionadas a data e hora. Essa abordagem permite que o comportamento dependente de tempo seja controlado durante os testes.

Estratégia de Teste e Uso de Dublês

Foram utilizados *mocks* para as classes *BookRepository*, *BuyBookProcess* e *Clock*. As entidades simples, como *Book* e *Overview*, não foram *mockadas*, pois são objetos de domínio simples e fáceis de instanciar.

A utilização do *mock* para *Clock* permitiu testar cenários específicos de datas, como Natal e outros dias. Nos testes do *BookStore*, os *mocks* foram essenciais para simular o estoque dos livros e o processo de compra, além de permitir verificar as interações entre os componentes.

Implementação dos Testes

Os testes foram escritos utilizando *JUnit 5*, *Mockito* e *AssertJ*.

Teste para o *BookStore*

- *Mock* de *BookRepository* para simular o estoque dos livros.
- *Mock* de *BuyBookProcess* para verificar se o processo de compra é corretamente acionado.
- Verificação do valor total calculado.
- Verificação dos livros não disponíveis.
- Verificação das chamadas do método `buyBook` com os parâmetros esperados.

Teste para o *ChristmasDiscount*

- *Mock* de *Clock* para controlar a data simulada.
- Teste do cenário em que a data é 25 de dezembro, aplicando-se o desconto.
- Teste do cenário em que a data não é Natal, não aplicando o desconto.

Execução dos Testes e Geração dos Relatórios

Os testes foram executados utilizando o comando padrão do Maven: `mvn clean test`.

Esse comando compila o projeto, executa os testes e gera os relatórios.

- O **relatório de execução dos testes** foi gerado pelo plugin *Surefire* na pasta:
  `target/surefire-reports/`
  Nos arquivos `*.txt` foi possível verificar que foram executados 3 testes no total, todos com sucesso e sem falhas ou erros.
- O **relatório de cobertura de código** foi gerado pelo plugin *Jacoco* na pasta:
  `target/site/jacoco/`
  O arquivo principal é `index.html`, que ao ser aberto em navegador exibe a cobertura detalhada.
  A partir desse relatório, verificou-se que 93% das linhas e 80% dos branches do código foram cobertos pelos testes.

Resultados dos Testes

Teste	Status
BookStoreTest — cálculo correto	Passou
BookStoreTest — controle de estoque	Passou
ChristmasDiscountTest — Natal	Passou
ChristmasDiscountTest — Não Natal	Passou

Foram executados 3 testes no total, todos com sucesso, demonstrando que os *mocks* foram utilizados corretamente para isolar as dependências e verificar tanto o comportamento quanto as interações esperadas.

Relatórios Gerados

- Cobertura de código (*Jacoco*): 93% de linhas e 80% de *branches* cobertos, conforme relatório gerado em `target/site/jacoco/index.html`.
- Relatório de execução de testes (*Surefire*): todos os testes executados com sucesso, conforme arquivos em `target/surefire-reports/`.

Conclusão

A prática permitiu compreender de forma clara quais dependências devem ser *mockadas*, como interfaces e dependências externas, e quais não devem ser, como entidades simples.

A utilização de abstrações como *Clock* mostrou-se fundamental para tornar os testes determinísticos e robustos, especialmente em situações que envolvem dependências com o sistema operacional, como data e hora.

A prática reforça a importância dos *dublês de teste* no desenvolvimento de testes unitários eficazes, contribuindo para a melhoria da manutenibilidade e da qualidade do software.

Referências

- Livro: *Effective Software Testing* — Maurício Aniche, 2022.
- Documentação *Mockito*: https://site.mockito.org/
- Documentação *JUnit*: https://junit.org/junit5/
- Documentação *AssertJ*: https://assertj.github.io/doc/