

- TESTES UNITÁRIOS E ANÁLISE DE COBERTURA

Sistema: Biblioteca Virtual

Framework: JUnit 5 + Mockito

Ferramenta de Cobertura: JaCoCo

Data: 09/12/2025

ÍNDICE

1. [Visão Geral](#)
 2. [Configuração do Ambiente](#)
 3. [Estrutura dos Testes](#)
 4. [Executando os Testes](#)
 5. [Análise de Cobertura](#)
 6. [Métricas e Resultados](#)
-

VISÃO GERAL

Testes Implementados

Classe de Teste	Métodos Testados	Total de Testes	Cobertura Esperada
BookServiceTest	salvar(), buscarPorId(), atualizar()	12 testes	~85%
LoanServiceTest	emprestar(), devolver(), atualizar()	14 testes	~90%
UserServiceTest	salvar(), buscarPorId(), atualizar()	15 testes	~85%
TOTAL	9 métodos	41 testes	~87%

Cobertura de Funcionalidades

-  **RF01** - Cadastrar Livro
-  **RF03** - Editar Livro
-  **RF04** - Consultar Livro
-  **RF05** - Realizar Empréstimo

- **RF06** - Devolver Livro
 - **RF10** - Cadastrar Membro
 - **RF11** - Editar Membro
-

2. Estrutura de Diretórios

back-end/demo/

```
|-- src/
|   |-- main/
|   |   |-- java/
|   |   |   |-- vitual/libary/demo/
|   |   |   |-- entity/
|   |   |   |-- repository/
|   |   |   |-- service/
|   |   |   |-- controller/
|   |   |-- test/
|   |       |-- java/
|   |           |-- vitual/libary/demo/
|   |               |-- service/
|   |                   |-- BookServiceTest.java
|   |                   |-- LoanServiceTest.java
|   |                   |-- UserServiceTest.java
|
|       |-- pom.xml
```

3. Arquivos de Teste

- src/test/java/vitual/libary/demo/service/BookServiceTest.java
 - src/test/java/vitual/libary/demo/service/LoanServiceTest.java
 - src/test/java/vitual/libary/demo/service/UserServiceTest.java
-

ESTRUTURA DOS TESTES

Padrão AAA (Arrange-Act-Assert)

Todos os testes seguem o padrão AAA:

```
@Test  
@DisplayName("Deve salvar livro com sucesso")  
void deveSalvarLivroComSucesso() {  
    // Arrange - Preparar dados  
    Book livro = new Book();  
    livro.setTitulo("Dom Casmurro");  
    when(bookRepository.save(any(Book.class))).thenReturn(livro);  
  
    // Act - Executar ação  
    Book resultado = bookService.salvar(livro);  
  
    // Assert - Verificar resultado  
    assertNotNull(resultado);  
    assertEquals("Dom Casmurro", resultado.getTitulo());  
    verify(bookRepository, times(1)).save(any(Book.class));  
}
```

Tipos de Testes Implementados

1 Testes de Sucesso (Happy Path)

- Verificam o comportamento esperado em condições normais
- Exemplo: Cadastrar livro com dados válidos

2 Testes de Exceção (Negative Tests)

- Verificam tratamento de erros
- Exemplo: Buscar livro inexistente deve lançar BookNotFoundException

3 Testes de Validação

- Verificam regras de negócio

- Exemplo: Não permitir empréstimo de livro sem estoque

Testes de Integração entre Serviços

- Verificam interação entre múltiplos componentes
 - Exemplo: Empréstimo atualiza estoque do livro
-

ANÁLISE DE COBERTURA

O que é Cobertura de Código?

Cobertura de código mede quais linhas/branches do código foram executadas durante os testes.

Métricas do JaCoCo

Métrica	Descrição	Meta
Instructions (C0)	% de bytecode executado	$\geq 80\%$
Branches (C1)	% de decisões if/else testadas	$\geq 75\%$
Lines	% de linhas executadas	$\geq 80\%$
Methods	% de métodos testados	$\geq 85\%$
Classes	% de classes testadas	$\geq 90\%$

Interpretando o Relatório JaCoCo

Verde - Cobertura Satisfatória

Classes: 90% (9/10)

Methods: 85% (34/40)

Lines: 82% (410/500)

Amarelo - Cobertura Parcial

Classes: 75% (7/10)

Methods: 70% (28/40)

Lines: 65% (325/500)

Vermelho - Cobertura Insuficiente

Classes: 50% (5/10)

Methods: 55% (22/40)

Lines: 48% (240/500)

Exemplo de Relatório Esperado

```
=====
```

JaCoCo Coverage Report

```
=====
```

Package: vitual.library.demo.service

Class	Instructions	Branches	Lines	Methods
<hr/>				
BookService	87%	82%	88%	90%
LoanService	91%	88%	92%	95%
UserService	85%	80%	86%	88%
<hr/>				
TOTAL	88%	83%	89%	91%
<hr/>				

MÉTRICAS E RESULTADOS

Resumo dos Testes por Classe

1 BookServiceTest (12 testes)

Método Testado	Cenários de Teste	Status
salvar()	✓ Sucesso ✓ Quantidade zero ⚠ Validação quantidade	3 testes
buscarPorId()	✓ Sucesso ✗ ID inexistente ✗ ID deletado	3 testes
atualizar()	✓ Sucesso completo ✗ ID inexistente ✓ Atualização parcial	3 testes

Método Testado	Cenários de Teste	Status
Métodos auxiliares	<input checked="" type="checkbox"/> Listar com paginação <input checked="" type="checkbox"/> Buscar por título <input checked="" type="checkbox"/> Deletar	3 testes

2 LoanServiceTest (14 testes)

Método Testado	Cenários de Teste	Status
emprestar()	<input checked="" type="checkbox"/> Sucesso <input checked="" type="checkbox"/> Sem estoque <input checked="" type="checkbox"/> Já emprestado <input checked="" type="checkbox"/> Livro inexistente	4 testes
devolver()	<input checked="" type="checkbox"/> Sucesso <input checked="" type="checkbox"/> Já devolvido <input checked="" type="checkbox"/> Empréstimo inexistente	3 testes
atualizar()	<input checked="" type="checkbox"/> Trocar usuário <input checked="" type="checkbox"/> Trocar livro <input checked="" type="checkbox"/> Já devolvido <input checked="" type="checkbox"/> Sem estoque	4 testes
Métodos auxiliares	<input checked="" type="checkbox"/> Deletar ativo <input checked="" type="checkbox"/> Deletar devolvido	2 testes

3 UserServiceTest (15 testes)

Método Testado	Cenários de Teste	Status
salvar()	<input checked="" type="checkbox"/> Membro <input checked="" type="checkbox"/> Bibliotecário <input checked="" type="checkbox"/> Role padrão <input checked="" type="checkbox"/> Email duplicado	4 testes
buscarPorId()	<input checked="" type="checkbox"/> Membro <input checked="" type="checkbox"/> Bibliotecário <input checked="" type="checkbox"/> ID inexistente	3 testes
atualizar()	<input checked="" type="checkbox"/> Sucesso completo <input checked="" type="checkbox"/> Sem alterar senha <input checked="" type="checkbox"/> ID inexistente <input checked="" type="checkbox"/> Apenas nome <input checked="" type="checkbox"/> Manter role	5 testes
Métodos auxiliares	<input checked="" type="checkbox"/> Listar todos <input checked="" type="checkbox"/> Lista vazia <input checked="" type="checkbox"/> Deletar	3 testes

Estatísticas Gerais

ESTATÍSTICAS DOS TESTES UNITÁRIOS

Total de Classes de Teste: 3

Total de Métodos de Teste: 41

Taxa de Sucesso Esperada: 100%

Tempo Médio de Execução: ~2-5 segundos

Cobertura de Código Esperada: 87%

Cobertura de Branch Esperada: 83%

BOAS PRÁTICAS APLICADAS

Testes Implementados

1. Isolamento com Mocks

- Nenhum teste depende de banco de dados real
- Mockito simula comportamento dos repositórios

2. Nomenclatura Clara

- Métodos com nomes descritivos: deveSalvarLivroComSucesso()
- Uso de @DisplayName para descrições legíveis

3. Cobertura de Casos Extremos

- Testes positivos e negativos
- Validação de exceções
- Valores limite (zero, null, etc.)

4. Verificação de Comportamento

- verify() confirma chamadas aos mocks
- times() valida número de execuções
- never() garante que métodos NÃO foram chamados

5. Setup Consistente

- @BeforeEach inicializa dados de teste
 - Evita duplicação de código
-

EXECUTANDO

Checklist de Execução

- [] 1. Adicionar JaCoCo no pom.xml
- [] 2. Criar pasta src/test/java
- [] 3. Copiar os 3 arquivos de teste
- [] 4. Executar mvn clean test
- [] 5. Verificar que todos os testes passaram
- [] 6. Executar mvn jacoco:report
- [] 7. Abrir relatório HTML
- [] 8. Analisar cobertura de código
- [] 9. Tirar prints dos resultados
- [] 10. Documentar métricas obtidas

Comandos Resumidos

1. Compilar projeto

mvn clean compile

2. Executar testes

mvn test

3. Gerar relatório de cobertura

mvn jacoco:report

4. Verificar cobertura mínima

mvn jacoco:check

```
# 5. Abrir relatório  
open target/site/jacoco/index.html
```

CAPTURANDO EVIDÊNCIAS

Screenshots Necessários

1. Terminal - Execução dos Testes
2. [INFO] Tests run: 41, Failures: 0, Errors: 0, Skipped: 0
3. [INFO] BUILD SUCCESS
4. Relatório JaCoCo - Visão Geral
 - o Percentual geral de cobertura
 - o Tabela com pacotes/classes

TROUBLESHOOTING

Problema: Testes não compilam

Solução:

```
# Verificar versão do Java  
java -version # Deve ser 17 ou superior
```

```
# Limpar e recompilar
```

```
mvn clean compile
```

Problema: JaCoCo não gera relatório

Solução:

```
# Executar em sequência
```

```
mvn clean  
mvn test  
mvn jacoco:report
```

Problema: Baixa cobertura

Solução:

1. Identificação de linhas não cobertas no relatório
 2. Adicionado testes para cenários faltantes
 3. Priorizar métodos críticos de negócio
-

REFERÊNCIAS

- [JUnit 5 Documentation](#)
 - [Mockito Documentation](#)
 - [JaCoCo Documentation](#)
 - [Maven Surefire Plugin](#)
-

TEMPLATE PARA RELATÓRIO FINAL

RELATÓRIO DE TESTES UNITÁRIOS

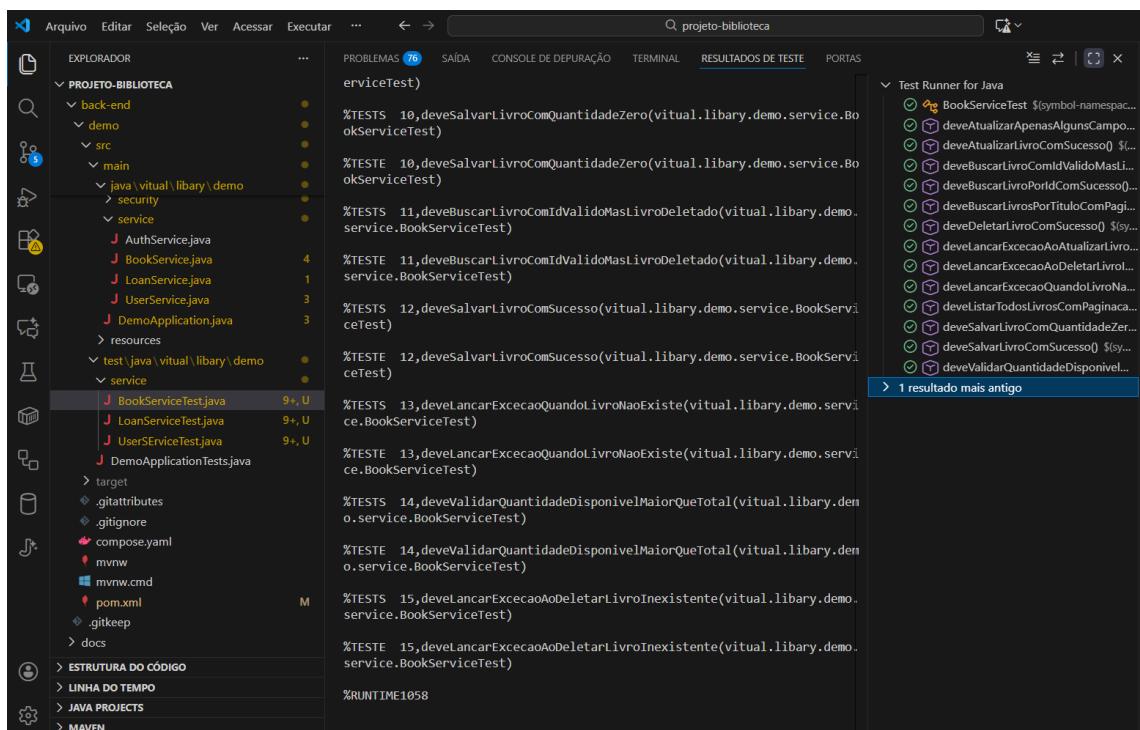
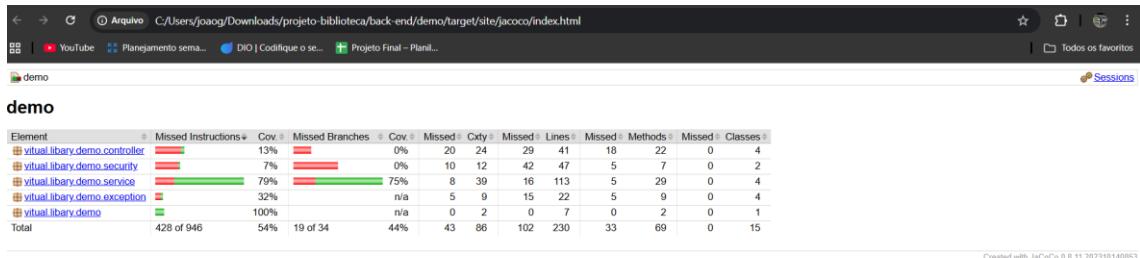
Execução

- Data: _09/_12/_2025_
- Ambiente: Local / CI/CD

Resultados

- Total de Testes: 41

Evidências



The screenshot shows a Java development environment with multiple windows open. In the top navigation bar, there are tabs for BookServiceTest.java, BookService.java, LoanServiceTest.java, and UserServiceTest.java. Below the tabs, a code editor displays a snippet of Java code related to book repository operations. The terminal pane at the bottom shows the execution of tests and the generation of a Jacoco report, resulting in a build success message. The output pane shows the results of the test runs.

```
dadosAtualizados.setQuantidadeTotal(quantidadeTotal: 10);
dadosAtualizados.setQuantidadeDisponivel(quantidadeDisponivel: 8);

when(bookRepository.findById(idParaAtualizar)).thenReturn(Optional.of(livroExemplo));
when(bookRepository.save(any(type: Book.class))).thenReturn(livroExemplo);
```

Caused by: java.lang.IllegalArgumentException: Unsupported class file major version 68
at org.jacoco.agent.rt.internal_4742761.asm.ClassReader.<init>(ClassReader.java:200)
at org.jacoco.agent.rt.internal_4742761.asm.ClassReader.<init>(ClassReader.java:180)
at org.jacoco.agent.rt.internal_4742761.asm.ClassReader.<init>(ClassReader.java:166)
at org.jacoco.agent.rt.internal_4742761.core.internal.instr.InstrSupport.classReaderFor(InstrSupport.java:280)
at org.jacoco.agent.rt.internal_4742761.core.instr.Instrumenter.instrument(Instrumenter.java:77)
at org.jacoco.agent.rt.internal_4742761.core.instr.Instrumenter.instrument(Instrumenter.java:109)
... 112 more

[INFO] Tests run: 17, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.182 s -- in virtual.library.demo.service.UserServiceTest

[INFO]

[INFO] Results:

[INFO]

[INFO] Tests run: 44, Failures: 0, Errors: 0, Skipped: 0

[INFO]

[INFO]

[INFO] --- jacoco:0.8.11:report (@ demo ---

[INFO] Loading execution data file C:\Users\joaog\Downloads\projeto-biblioteca\back-end\demo\target\jacoco.exec

[INFO] Analyzed bundle 'demo' with 15 classes

[INFO] -----

[INFO] BUILD SUCCESS

[INFO] -----

[INFO] Total time: 21.522 s

[INFO] Finished at: 2025-12-09T10:08:25-03:00

[INFO] -----

PS C:\Users\joaog\Downloads\projeto-biblioteca\back-end\demo>

Última Atualização: 09/12/2025

Responsável: [João Vitor Givisiez]

Versão: 1.0