Universidade de Brasília

Programação Sistemática

Trabalho III

Adriano Torres - 11/0106032

Caio Oliveira - 14/0176713

Leandro Faria - 14/0178759

Elisabete Evaldt - 09/0093450

# Sistema de Gerenciamento de Palestras

## Histórico de Alterações

|  |  |  |  |
| --- | --- | --- | --- |
| **Data** | **Versão** | **Descrição** | **Autor** |
| 08/12/2015 | 1.0 | Criação do documento | Elisabete Evaldt |
| 09/12/2015 | 2.0 | Revisão em pares. | Leandro Faria |
|  |  |  |  |

**Palavras-chave**: Critérios de cobertura de teste, Teste de Caixa Branca, EclEmma, JaCoCoverage, JUnit.

# Relatório de Evidências de Teste – JaCoCo (EclEmma)

Este relatório tem por objetivo descrever a aplicação de teste de cobertura, realizado por meio da ferramenta JaCoCo integrado com a IDE NetBeans, realizado na suíte de testes do sistema de gerenciamento de palestras produzido durante a disciplina de Programação Sistemática, na Universidade de Brasília. JaCoCo, também conhecida como Java Code Coverage, é uma biblioteca de cobertura de código para Java, criada pela equipe EclEmma com base nas lições aprendidas e com o uso e integração de bibliotecas existentes por muitos anos. A biblioteca pode ser integrada à IDE do NetBeans através do Plug-in TikiOne JaCoCoverage.

## Ambiente de Testes

### Ambiente de Desenvolvimento Integrado

**Product Version**: NetBeans IDE 8.0.2 (Build 201411181905)

**Runtime**: Java(TM) SE Runtime Environment 1.7.0\_79-b15

**System**: Windows 8 version 6.2 running on amd64; Cp1252; en\_US (nb)

### Plugin de Teste Unitário

**Description:** JUnit

**Installed version:** 4.12

**Source:** Certified Plugins

### Plugin de Teste de Cobertura

**Description:** TikiOne JaCoCoverage (EclEmma)

**Installed version:** v1.5.2

**Source:** Certified Plugins

## Execução dos Testes

O teste de cobertura foi executado na suíte de automação de teste dos módulos criados no Trabalho II, da disciplina de Programação Sistemática. O projeto de teste unitário, Figura 1, foi criado com o auxílio da biblioteca JUnit, no qual foram criados testes para cada um dos módulos do aplicativo com o objetivo de verificar a correção e a aderência do código fonte às especificações do sistema de gerenciamento de palestras. Na fase III do trabalho, a biblioteca de teste de cobertura JaCoCo foi utilizada para aferir a cobertura dos testes criados anteriormente.

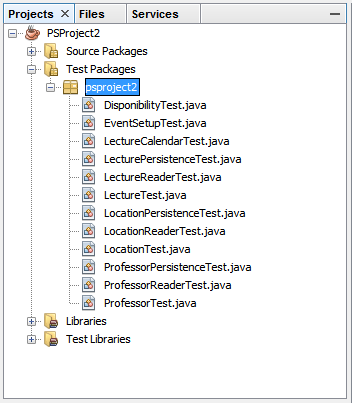


Figure 1- Visão geral da suíte de testes.

A biblioteca JaCoCo funciona de forma integrada com a IDE NetBeans, através de um Plug-in previamente instalado. Para realizar um teste de cobertura de código é necessário acionar, com um clique no botão direito, o menu de comandos. O menu então irá exibir a opção desejada. Figuras 2.

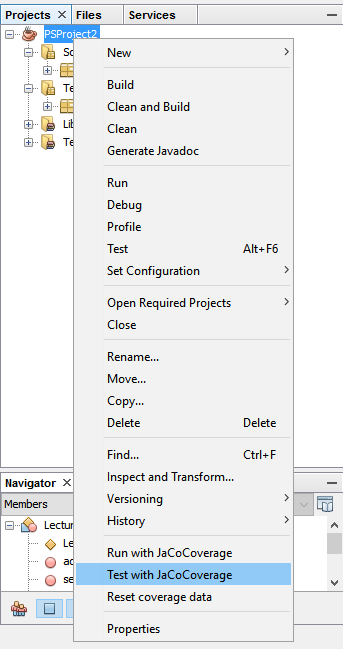


Figure 2 – Menu de comandos com a opção Test with JaCoCoverage..

Ao executar o teste de cobertura, a ferramenta apresentará um relatório navegável com o resultado da aferição. A primeira tela do relatório apresenta um sumário de todo o projeto – Figura 3.

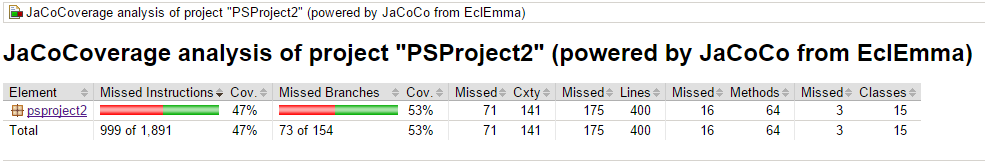


Figure 3- Tela inicial com o sumário do teste de cobertura.

A visão geral, apresenta gráficos em forma de barra que identificam o percentual de cobertura alcançada com a suíte de teste por critérios de cobertura de Instructions 47% e Branches 53%.

Na visão detalhada, é possível visualizar o percentual de cobertura por classes. Figura 4.

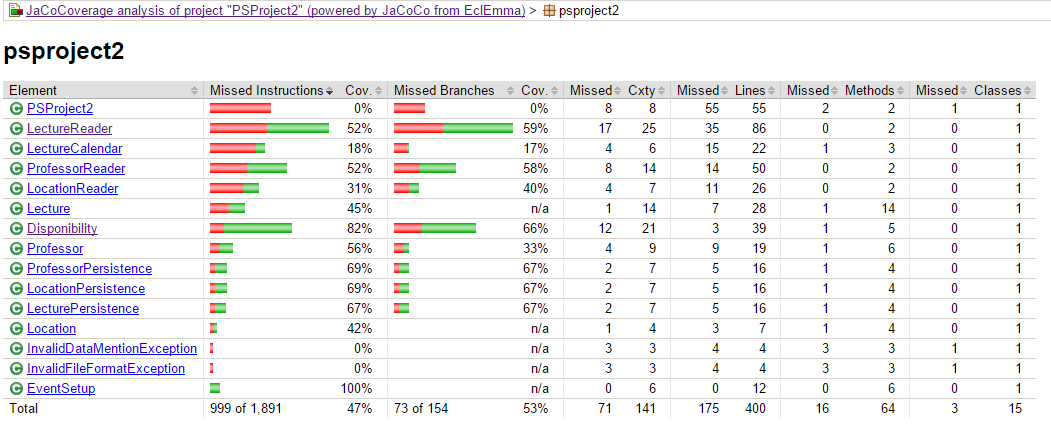


Figure 4- Visão por classes.

### Módulo de Disponibilidade (Disponibility.java)

O módulo de disponibilidade apresentou um percentual de cobertura de 82%. Apesar desse percentual estar abaixo de 100%, o valor alcançado foi satisfatório em vista do escopo definido previamente pelos casos de teste. Figura 5.

No entanto, o relatório do método populateData(String) revelou uma falta de cobertura especificada que não havia sido aplicada. A estrutura de decisão “if” somente foi testada no caso em que retorna o valor “true”. Figura 6.

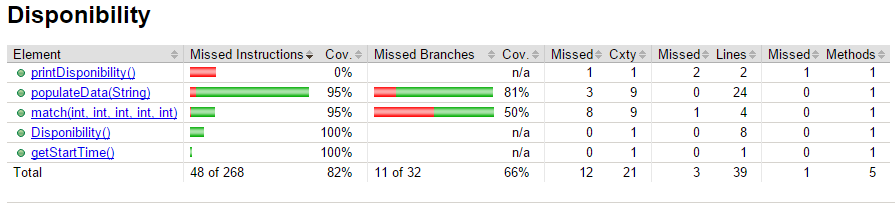


Figure 5- Cobertura do modulo Disponibilidade.

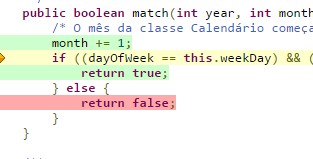


Figure 6- Falta de cobertura no módulo de disponibilidade.

### Módulo de Palestra (Lecture.java)

O módulo de palestra apresentou percentual de cobertura de 100%, apesar de o relatório da classe indicar um percentual menor devido à métodos de debug propositalmente deixados de fora. Figura 7.

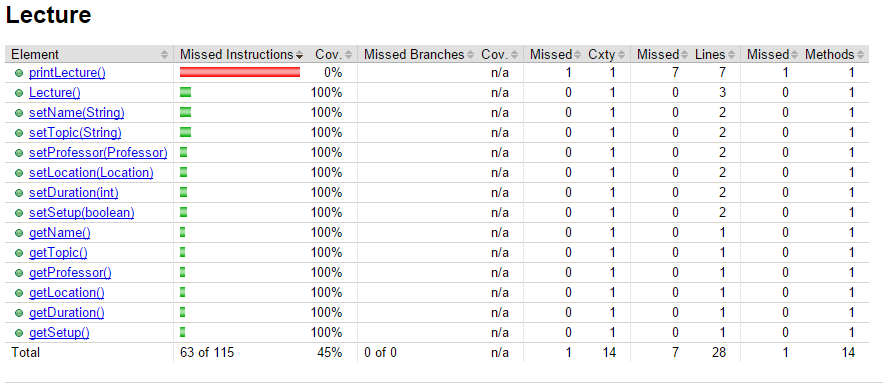


Figure 7- Resultado do teste no módulo de palestra.

### Módulo de Persistência de Palestra (LecturePersistence.java)

O módulo de persistência palestra apresentou percentual de cobertura de 100%, apesar de o relatório da classe indicar um percentual menor devido à métodos de debug propositalmente deixados de fora. Figura 8.

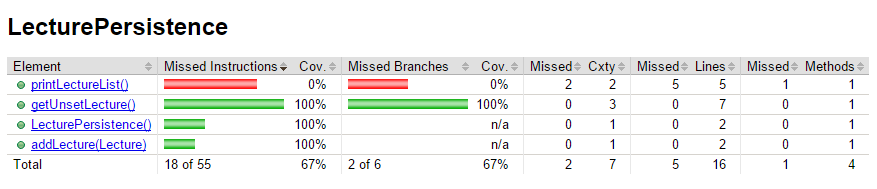


Figure 8- Resultado do teste no módulo de persistênica de palestra.

### Módulo de Leitura de Palestra (LectureReader.java)

O módulo de leitura palestra apresentou percentual de cobertura de 52%, e falhou ao deixar de cobrir os erros de exceção não especificados nos casos de teste. Figura 9 e 10.

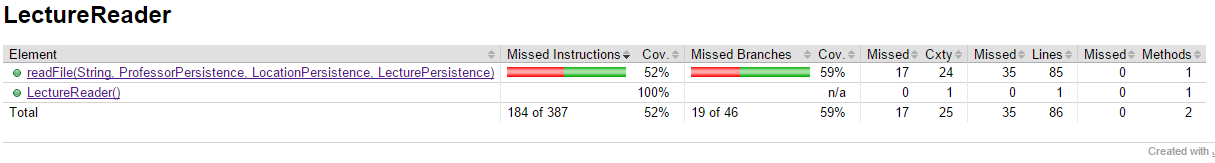


Figure 9- Resultado do teste de cobertura do módulo de leitura de palestra.

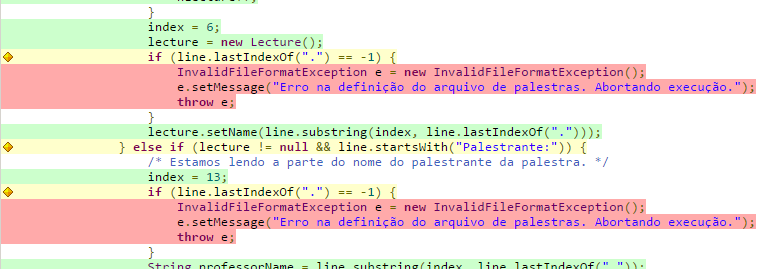


Figure 10- Falha na cobertura de casos excessões do módulo.

### Módulo de Calendário de Palestras (LectureCalendar.java)

O módulo de calendário de palestras apresentou percentual de cobertura de 24%, e foi considerado crítico pois falhou ao deixar de cobrir os erros de exceção especificados nos casos de teste dentro de uma estrutura de laço. Figura 11 e 12.

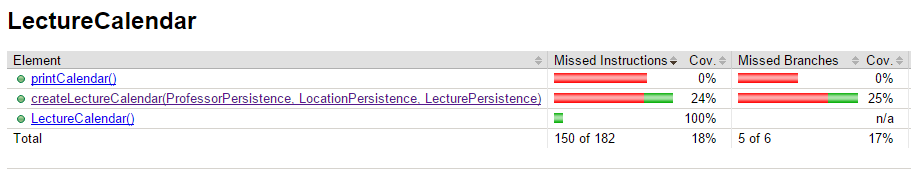


Figure 11- Resultado do teste de cobertura do módulo de calendário de palestra.

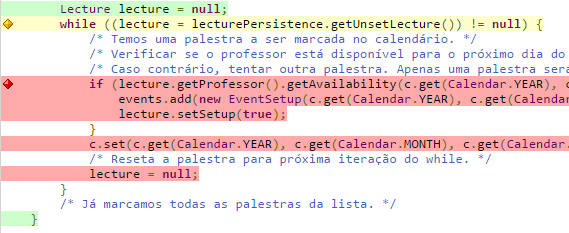


Figure 12- Falha de cobertura do módulo calendário de palestras.

### Módulo Local (Location.java)

O módulo de local apresentou percentual de cobertura de 100%, apesar de o relatório da classe indicar um percentual menor devido à métodos de debug propositalmente deixados de fora. Figura 13.

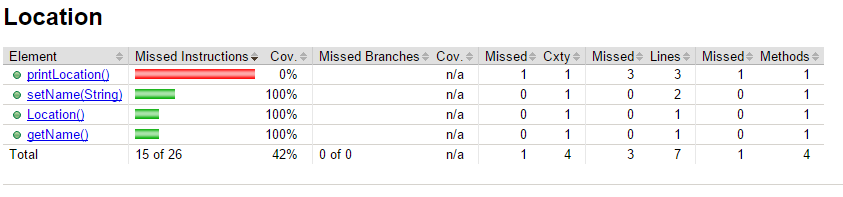


Figure 13- Resultado de teste do módulo local.

### Módulo Local – Persistência (LocationPersistence.java)

O módulo de local – persistência apresentou percentual de cobertura de 100%, apesar de o relatório da classe indicar um percentual menor devido à métodos de debug propositalmente deixados de fora. Figura 14.

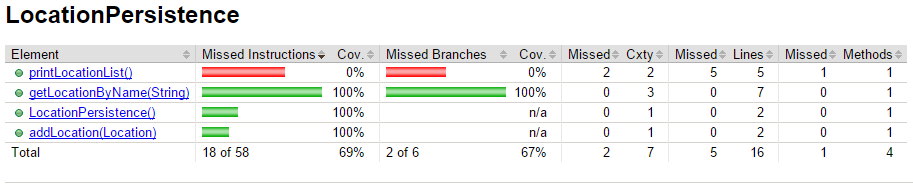


Figure 14- Resultado de teste do módulo local - persistência.

### Módulo Local – Leitura (LocationReader.java)

O módulo de local leitura apresentou percentual de cobertura baixo 30%, devido à falta ao deixar de cobrir os erros de exceção não especificados nos casos de teste. Figura 15 e 16.

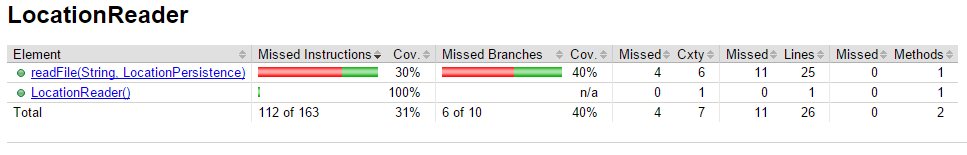


Figure 15- Resultado de teste do módulo local - leitura.

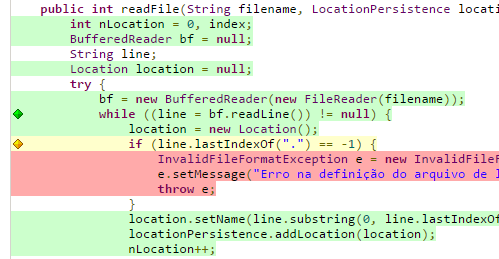


Figure 16- Falha de cobertura do módulo local - leitura.

### Módulo Professor (Professor.java)

O módulo de professor apresentou percentual de cobertura alto em torno de 88%, e falhou ao deixar de cobrir caso em que o professor não possuía disponibilidade. Figura 17 e 18.

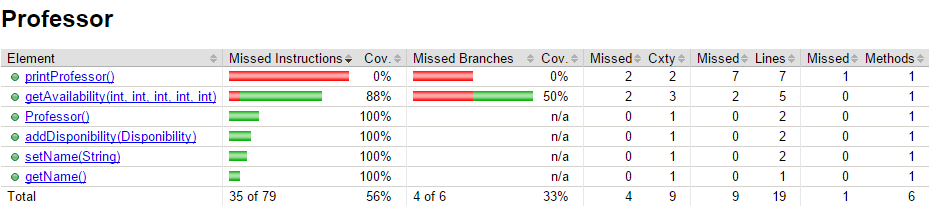


Figure 17- Resultado de teste do módulo professor.

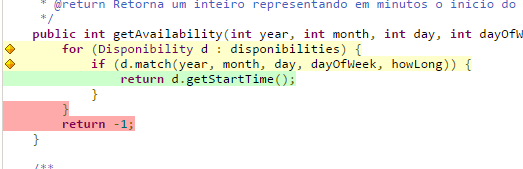


Figure 18- Falha de cobertura do módulo professor.

### Módulo Professor – Persistência (ProfessorPersistence.java)

O módulo de professor-persistência apresentou percentual de cobertura de 100%, apesar de o relatório da classe indicar um percentual menor devido à métodos de debug propositalmente deixados de fora. Figura 19.

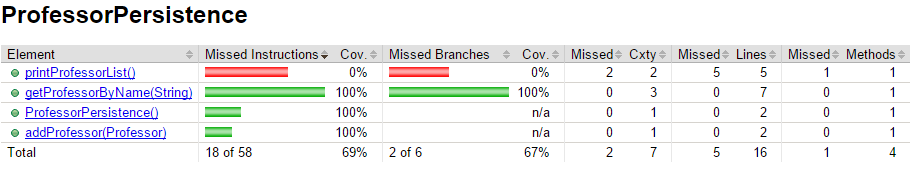


Figure 19- Resultado de teste do módulo professor-persistência.

### Módulo Professor – Leitura (ProfessorReader.java)

O módulo de professor leitura apresentou percentual de cobertura baixo 51%, devido à falta ao deixar de cobrir os erros de exceção não especificados nos casos de teste. Figura 20.

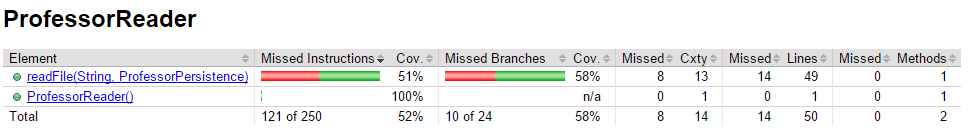


Figure 20- Resultado de teste do módulo professor-leitura.

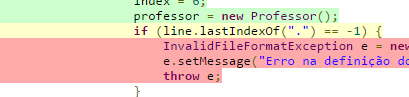


Figure 21- Falha de cobertura do módulo professor-leitura.

### Módulo de Configuração de evento (EventSetup.java)

O módulo de configuração de evento apresentou percentual de cobertura de 100%. Figura 22.

### 

Figure 22- Resultado do teste do módulo de configuração de evento.

### Módulo Principal – (PSProject2.java)

O módulo principal apresentou 0% de cobertura devido à falta de especificação de caso de teste durante a criação da suíte de teste. Caso mais grave capturado apenas com o uso da ferramenta de teste de cobertura. Figura 23.

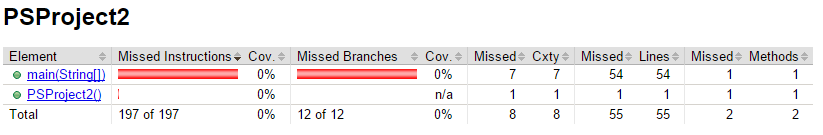


Figure 23- Resultado de teste do módulo principal.

## Avaliação

O uso da ferramenta foi de grande utilidade na análise e aferição de cobertura de testes realizadas na fase do trabalho anterior. A integração com a plataforma permitiu a rápida visualização e ação por parte dos testadores. Pudemos perceber um ganho de produtividade na aplicação do teste de cobertura e na seleção dos critérios de cobertura incomparáveis com o método manual de aferição.