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** |
| 07/12/2015 | 1.0 | Criação do documento | Elisabete Evaldt |
|  |  |  |  |
|  |  |  |  |

**Palavras-chave**: análise estática, FindBugs, Java, teste de software.

# Relatório de Evidências de Teste – FindBugs PlugIn

Este relatório tem por objetivo descrever a aplicação de teste de análise estática, realizado por meio do Plug-in da ferramenta FindBugs integrado com a IDE NetBeans, realizado no código fonte do sistema de gerenciamento de palestras produzido durante a disciplina de Programação Sistemática, na Universidade de Brasília.

## 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

**Description:** FindBugs Integration

**Installed version:** 1.28

**Source:** Certified Plugins

## Execução dos Testes

O teste de inspeção foi executado diretamente no projeto do código fonte, conforme mostrado na Figura 1, possibilitando a varredura de possíveis erros em todas as classes ao mesmo tempo e um relatório completo do sistema.

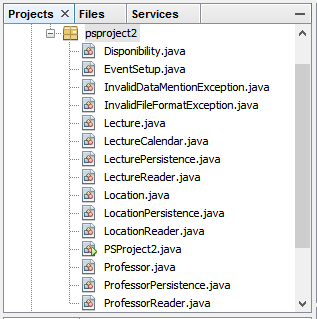


Figure 1- Visão geral do projeto contendo todo o código fonte.

A ferramenta FindBugs funciona de forma integrada com a IDE NetBeans, através de um Plug-in previamente instalado. Ao realizar um teste de inspeção de análise estática, a ferramenta deve ser selecionada na lista de ferramentas disponíveis e a inspeção irá realizar o teste de acordo com o escopo definido. As Figuras 2 e 3 ilustram esses passos.

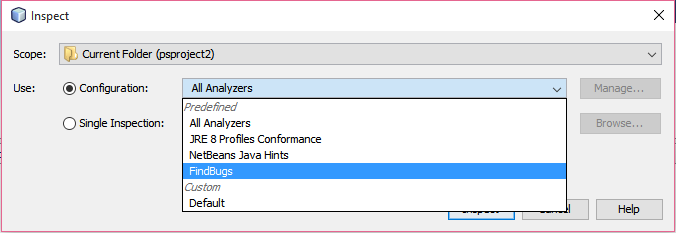


Figure 2 - Janela de configuração de teste de inspeção de código.

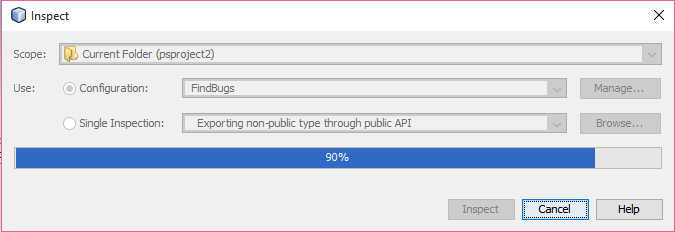


Figure 3 - Teste em execução.

Ao final, a ferramenta apresenta um relatório navegável de faltas (bugs) encontrados, passíveis de correções. No caso do sistema de gerenciamento de palestras, a ferramenta encontrou e sugeriu a correção de 41 bugs – Figura 4. Sendo eles:

11 casos de bugs classificados como “Internationalization”, no qual o código pode inibir o uso de caracteres internacionais. Considerados dentro do escopo desse projeto como de nível médio.

2 casos de bugs classificados como “Dodgy code”, no qual o código pode conduzir a erros. Considerados dentro do escopo desse projeto como de nível baixo.

28 casos de bugs classificados como “Performance”, no qual o código pode ser escrito de forma diferente para melhorar o desempenho da aplicação. Considerados dentro do escopo desse projeto como de nível baixo.

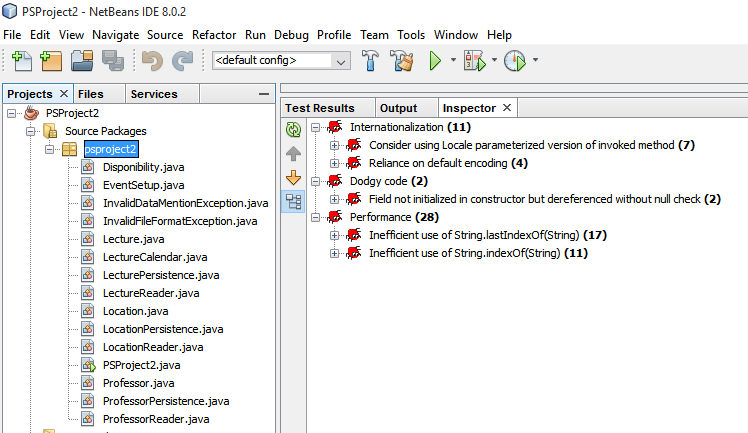


Figure 4- Tela de resultado de execução de teste.

Além da visão geral, a ferramenta apresenta uma visão de escopo de classe, no qual é possível visualizar os bugs encontrados filtrados por classe. Nessa visão, é possível navegar entre o código fonte à medida em que o bug é selecionado e visualizar a especificação do bug juntamente com a sugestão de correção que deve ser aplicada. Figura 5.

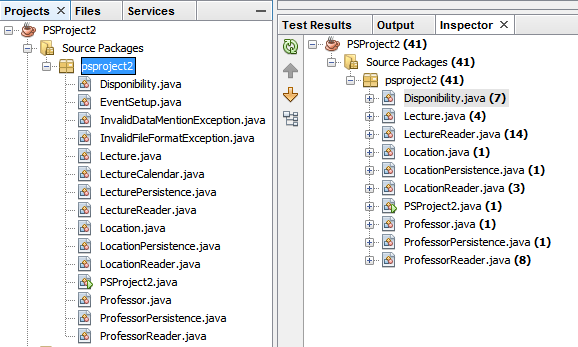


Figure 5- Visão por classes.

### Módulo de Disponibilidade (Disponibility.java)

O módulo de disponibilidade apresentou 7 faltas. Sendo 1 relacionada à Internacionalização na linha 42 e 6 relacionadas à Desempenho nas linhas 51, 60, 61, 63, 64 e 65. Figura 6.

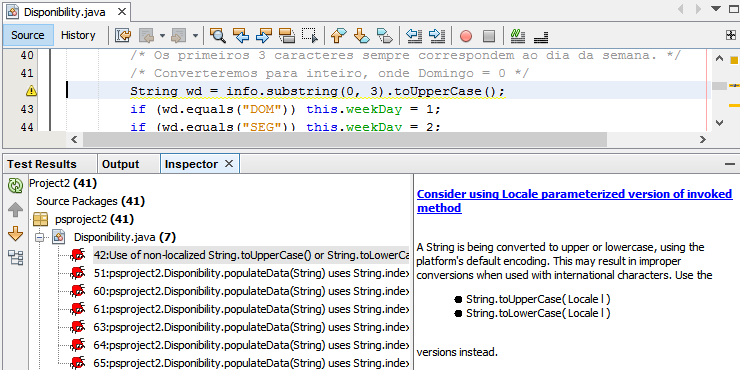


Figure 6- Resultado do teste no módulo de disponibilidade.

**Correções aplicadas:**

Linha 41: Uso de uma codificação específica do tipo toUpperCase(Locale.ROOT);

Linhas 51, 60, 61, 63, 64 e 65: Implementação do método String.indexOf('-') com aspas simples.

### Módulo de Palestra (Lecture.java)

O módulo de palestra apresentou 4 faltas. Sendo 2 relacionada à código “Desonesto” nas linhas 130 e 131; e 2 relacionadas à Internacionalização nas linhas 41 e 57. Figura 7.

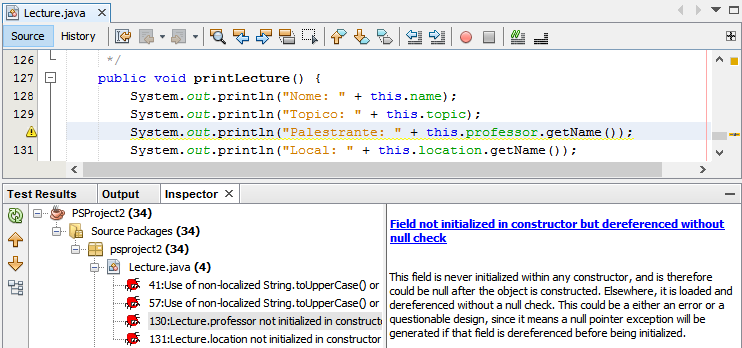


Figure 7- Resultado do teste no módulo de palestra.

**Correções aplicadas:**

Linhas 41 e 57: Uso de uma codificação específica do tipo toUpperCase(Locale.ROOT);

Linhas 130 e 131: Não foram aplicadas correções pois o bloco de código só é executado para fins de debug em ambiente de desenvolvimento.

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

O módulo de leitura de palestra apresentou 14 faltas. Sendo 13 relacionadas à Desempenho e 1 relacionada à Internacionalização na linha 38. Figura 8.

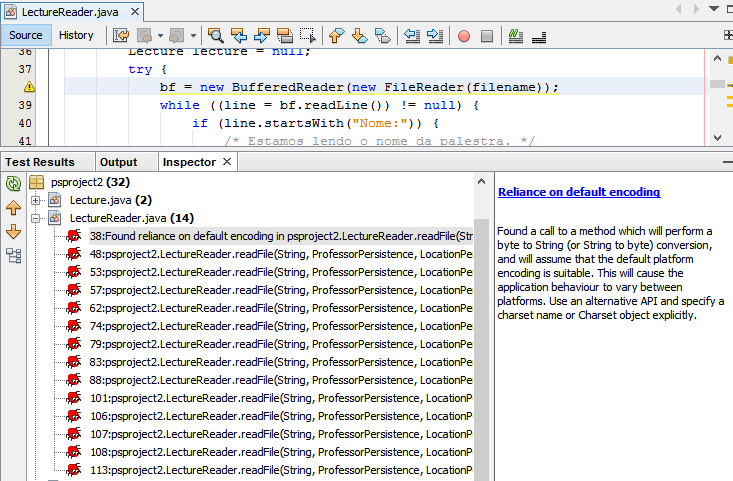


Figure 8- Resultado do teste no módulo de leitura de palestra.

**Correções aplicadas:**

Linha 38: Uso de uma codificação específica do tipo “StandardCharsets.UTF\_8” no método BufferedReader.

Linhas demais: Implementação do método String. lastIndexOf ('-') com aspas simples.

### Módulo Local (Location.java)

O módulo de local apresentou 1 falta relacionadas à Internacionalização na linha 32. Figura 9.

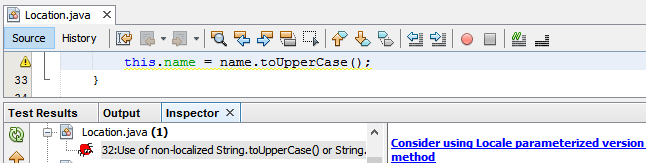


Figure 9- Resultado de teste do módulo local.

**Correções aplicadas:**

Linhas 38: Uso de uma codificação específica do tipo toUpperCase(Locale.ROOT);

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

O módulo de local apresentou 1 falta relacionadas à Internacionalização na linha 36. Figura 10.

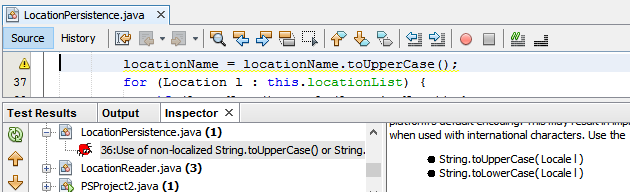


Figure 10- Resultado de teste do módulo local - persistência.

**Correções aplicadas:**

Linhas 36: Uso de uma codificação específica do tipo toUpperCase(Locale.ROOT);

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

O módulo de local apresentou 1 falta relacionadas à Internacionalização na linha 34. E 2 faltas relacionadas à Desempenho nas linhas 37 e 42. Figura 11.

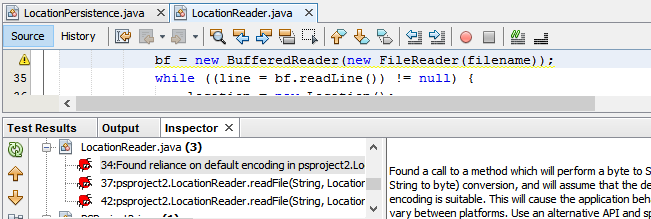


Figure 11- Resultado de teste do módulo local - leitura.

**Correções aplicadas:**

Linha 34: Uso de uma codificação específica do tipo “StandardCharsets.UTF\_8” no método BufferedReader.

Linhas 37 e 42: Implementação do método String. lastIndexOf ('-') com aspas simples.

### Módulo Principal – (PSProject2.java)

O módulo de local apresentou 1 falta relacionadas à Internacionalização na linha 36. Figura 12.

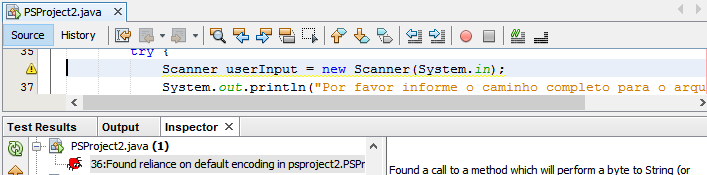


Figure 12- Resultado de teste do módulo principal.

**Correções aplicadas:**

Linha 36: Uso de uma codificação específica do tipo “StandardCharsets.UTF\_8” no método Scanner.

### Módulo Professor (Professor.java)

O módulo de professor apresentou 1 falta relacionadas à Internacionalização na linha 35. Figura 13.

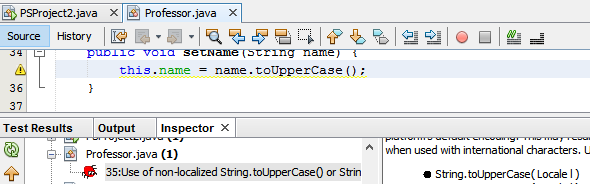


Figure 13- Resultado de teste do módulo professor.

**Correções aplicadas:**

Linhas 35: Uso de uma codificação específica do tipo toUpperCase(Locale.ROOT);

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

O módulo de professor-persistência apresentou 1 falta relacionadas à Internacionalização na linha 36. Figura 14.

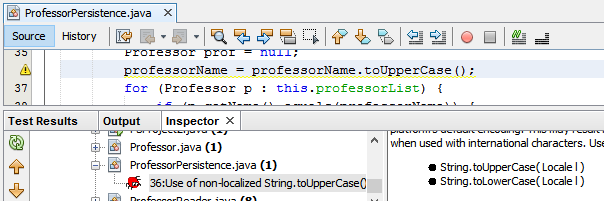


Figure 14- Resultado de teste do módulo professor-persistência.

**Correções aplicadas:**

Linhas 35: Uso de uma codificação específica do tipo toUpperCase(Locale.ROOT);

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

O módulo de professor leitura apresentou 1 falta relacionadas à Internacionalização na linha 34. E 7 faltas relacionadas à Desempenho nas demais linhas. Figura 15.

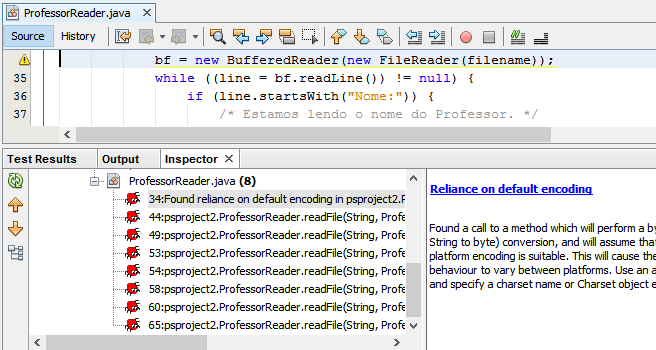


Figure 15- Resultado de teste do módulo professor-leitura.

**Correções aplicadas:**

Linha 34: Uso de uma codificação específica do tipo “StandardCharsets.UTF\_8” no método BufferedReader.

Linhas demais: Implementação do método String. indexOf ('-') com aspas simples.

## Avaliação

O uso da ferramenta foi de grande utilidade na correção de bugs imperceptíveis durante as fases de desenvolvimento anteriores. A integração com a plataforma permitiu a rápida correção através da navegabilidade entre os módulos e linhas do código fonte, além de ter acelerado na busca de soluções por fornecer uma informação detalhada do erro. Pudemos perceber um ganho de produtividade na aplicação do teste pois em um projeto de grandes proporções a integração da ferramenta é indispensável para a realização do mesmo.