**Trabalho Prático 2 - esii - Grupo1213**

White Box Tests

Version 3.0

21/01/2022

**Histórico de Versões**

| **Version #** | **Implemented**  **By** | **Revision**  **Date** | **Approved**  **By** | **Approval**  **Date** | **Reason** |
| --- | --- | --- | --- | --- | --- |
| 1.0 | Bruno Ferreira  Gonçalo Oliveira  Jorge Correia  Nuno Castro | 14-01-2022 | Bruno Ferreira  Gonçalo Oliveira  Jorge Correia  Nuno Castro | 14-01-2022 | Análise testes caixa branca para o módulo de transações |
| 2.0 | Bruno Ferreira  Gonçalo Oliveira  Jorge Correia  Nuno Castro | 18-01-2022 | Bruno Ferreira  Gonçalo Oliveira  Jorge Correia  Nuno Castro | 18-01-2022 | Análise testes caixa branca para o módulo de custos de envio e comparação das taxas de cobertura em relação à versão anterior. |
| 3.0 | Bruno Ferreira  Gonçalo Oliveira  Jorge Correia  Nuno Castro | 21-01-2022 | Bruno Ferreira  Gonçalo Oliveira  Jorge Correia  Nuno Castro | 21-01-2022 | Análise testes caixa branca para o módulo expedição e comparação das taxas de cobertura em relação à versão anterior. |

**Tabela de Conteúdos**

[**1. Introdução**](#_tjqz6wo46n4u) **4**

[1.1. Identificador do documento](#_30j0zll) 4

[1.2. Âmbito](#_17dp8vu) 4

[1.3. Repositório](#_3rdcrjn) 4

[1.4. Glossário](#_lkryktm1o9xx) 5

[1.5. Dados disponibilizados pelo Jacoco](#_g8bftd412n51) 6

[**2. Análise de cobertura para cada Classe**](#_u56i81slbyce) **7**

[2.1. Classe BasicLinhaTransacao](#_iw4u5ij4ksi0) 7

[2.2. Classe BasicGestaoEncomendas](#_rw6n1w84e8qg) 8

[2.3. Classe Estatísticas](#_tvgaro56x4rt) 9

[2.4. Classe Custos](#_ncyeel9oz0qf) 10

[2.5. Classe Expedições](#_rvdtp5efly4j) 11

[2.6. Classe BasicContentor](#_htai8b16b63p) 12

[2.7. Observações](#_qywm4j6gk2gg) 13

# 

# **1. Introdução**

## 1.1. Identificador do documento

WhiteBoxTests\_Grupo1213

## 1.2. Âmbito

Este documento foi criado no âmbito de apresentar os resultados dos testes de caixa branca, ou seja, apresentar os respetivos resultados de cobertura.

Após a codificação e execução dos casos de teste, identificados nos BlackBoxTests, é necessário realizar os WhiteBoxTests que consiste basicamente em testar a cobertura de ramos/instruções que os testes realizados alcançaram. Uma alta taxa de cobertura significa que uma grande parte de todo o código foi testado e, consequentemente, será menos propício a erros.

Uma vez que temos o plugin Jacoco, podemos utilizar o mesmo para analisar as informações descritas anteriormente. Além de identificar as taxas de cobertura, o Jacoco ainda indica quais as instruções que não foram cobertas.

Após o registo das instruções não cobertas, algumas delas serão eventualmente testadas para ser atingida uma maior taxa.

Numa fase inicial, foi definido pelo grupo que a taxa média de cobertura mínima seria de 65%, uma vez que existem vários métodos que ainda não podem ser testados e, consequentemente, possuem 0% de taxa de cobertura. No entanto, a taxa mínima para métodos que foram testados é de 80%, sendo que os testes devem garantir que cobrem a maior parte das instruções.

Com o desenvolver do projeto esta taxa de aceitação vai aumentando, exigindo uma maior cobertura a cada módulo.

## 1.3. Repositório

A hiperligação para o repositório onde foi realizado o trabalho é a seguinte:

<https://gitlab.estg.ipp.pt/esii.grupo1213/esii-grupo1213-tp2>

## 

## 1.4. Glossário

**Testes de software** - Um teste de software é um software que executa outro software, validando se os resultados são os esperados (teste de estado) ou se executa a sequência de eventos esperado (teste de comportamento).

Resumidamente, os testes de software permitem aos programadores verificar se a lógica do programa desenvolvido está de acordo com os requisitos.

A execução automática de testes permite identificar “bugs” resultantes de mudanças no código fonte.

**ECP** - Técnica destinada a reduzir o número de testes necessários dividindo o domínio de entrada (ou saída) em classes de dados em que os casos de teste podem ser derivados para cada operação, o “tester” deve identificar as classes de equivalência dos argumentos e os estados dos objetos

**BVA** - Técnica baseada na observação de bugs que ocorrem frequentemente em valores fronteira. É focada em testar valores especiais (null, 0, etc) e limites do domínio de entrada (ou saída) imediatamente acima e abaixo (além de ou em vez de valores intermédios)

**JUnit** - Framework open-source para realizar testes unitários para código em Java.

Permite a execução de testes de forma:

* Fácil
* Regular
* Fiável

Contém várias funcionalidades para testing entre elas a capacidade de testar cada componente de um programa de forma independente do resto do programa

**Gradle** - É uma ferramenta que permite integrar e automatizar várias tarefas relacionadas com o processo de desenvolvimento de software em várias linguagens de programação.

O Gradle determina quais os componentes do projeto que estão atualizados, evitando a recompilação de todo o projeto.

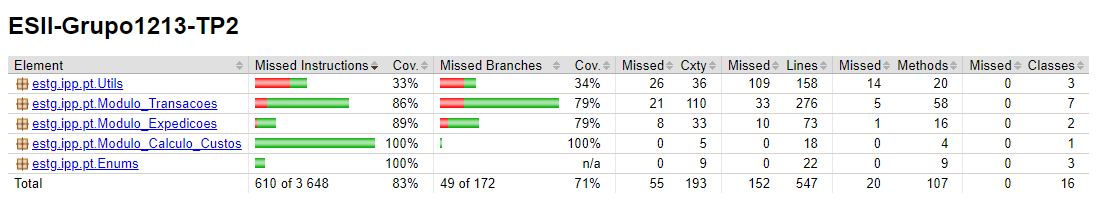
**Jacoco** - É uma biblioteca que disponibiliza taxas de cobertura e que identifica que parte do código não foi coberta.

## 1.5. Dados disponibilizados pelo Jacoco

Podemos concluir que pela análise disponibilizada pelo Jacoco, temos algumas classes que têm uma taxa de cobertura desejada pelo grupo é atingida ( 83% ).

No entanto, nem sempre significa que a cobertura tenha sido mal feita uma vez que, a metodologia seguida pelo nosso grupo implica o desenvolvimento, análise e testagem para cada módulo, pelo que alguns métodos devem apenas ser testados em módulos posteriores.

Alguns dos métodos não são testados, também pela sua baixa complexidade, tais como gets e sets, sendo que são métodos em que a taxa de erro é muito pequena pelo que, apenas foram testados alguns que implicam uma complexidade superior ao habitual.

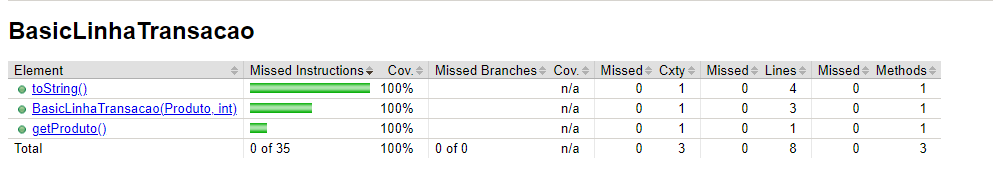


Neste relatório, para cada método, será realizada uma análise de cobertura e uma identificação de testes adicionais, caso seja necessário.

# **2. Análise de cobertura para cada Classe**

## 2.1. Classe BasicLinhaTransacao

Como podemos observar na imagem seguinte, na classe BasicLinhaTransacao foram cobertos 100% das instruções, o que significa que os testes feitos são suficientes para testar todas as instruções, o que por sua vez, ajuda a evitar a bugs.

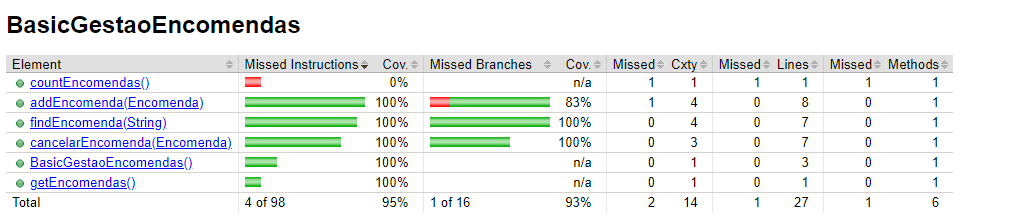


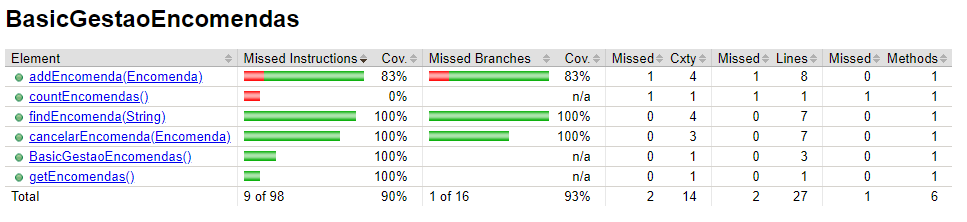
## 2.2. Classe BasicGestaoEncomendas

Em relação à versão anterior podemos observar que a taxa de cobertura de instruções desceu 17%. Esta descida deve-se ao facto de terem sido adicionadas novas instruções que apenas devem ser testadas posteriormente, no próximo módulo.

No entanto, a taxa revela-se suficiente uma vez que é superior à taxa de cobertura mínima definida pelo grupo para cada método (80%).

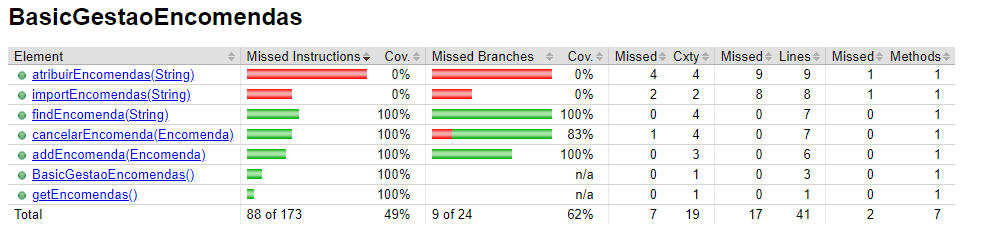
***Versões anteriores:***





Quanto à cobertura de ramos, podemos observar que se manteve nos 83%, pelo mesmo motivo apresentado na versão anterior, o facto de que será apenas testado posteriormente, no próximo módulo.

***Versão atual:***

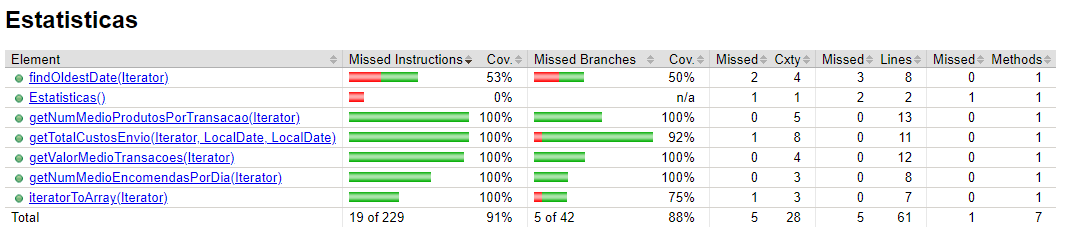


Nesta nova versão podemos observar que o método addEncomenda atingiu os 100% de cobertura. Contudo existem 2 métodos que possuem 0% de cobertura, isto acontece devido ao facto de serem métodos que serão utilizados no programa para testar a API e que por isso não foram testados..

## 2.3. Classe Estatísticas

Para esta classe foi obtida uma boa percentagem geral de cobertura.

No caso do construtor não existe cobertura visto que não deve ser possível a instanciação da classe Estatísticas.



Quanto à cobertura de ramos, apenas o método findOldestDataIterator apresenta uma taxa de cobertura inferior a 100%, devido a não ser testado diretamente.

Já o método getTotalCustosEnvio no Missed Branches está a 92% devido a falta de um teste para quando a data de uma das encomendas seja menor do que a dataInicio.

Devido a esse teste em falta, o mesmo foi adicionado e, como se pode observar, os 100% de cobertura foram atingidos.

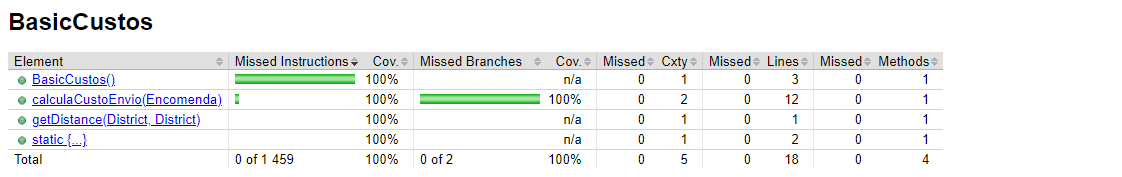
## 

## 

Nesta nova versão foi adicionado o método “getValorMedioVendasComprasPorDistrito” que possui 100% de cobertura.

## 2.4. Classe Custos

Como podemos observar na imagem seguinte, na classe BasicCustos foram cobertos 100% das instruções e ramos, o que significa que os testes feitos são suficientes para testar todas as instruções, o que por sua vez, ajuda a evitar a bugs.

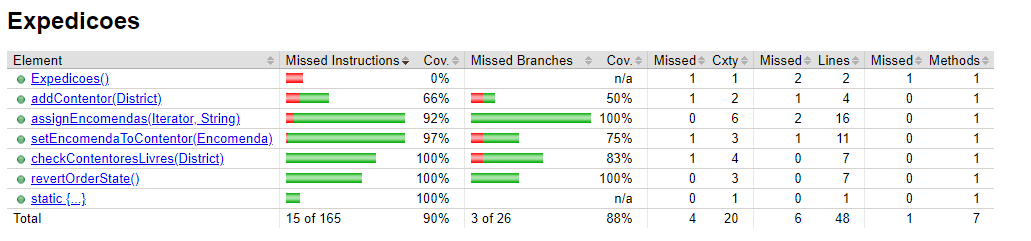


## 2.5. Classe Expedições

Como podemos observar na imagem seguinte, na classe Expedições foram cobertos 90% das instruções e 88% dos ramos, o que significa que os testes feitos são suficientes para testar uma boa parte do funcionamento desta classe.

O método addContentor possui uma cobertura menor visto que é um método invocado indiretamente pelo que não é testado se o distrito enviado é null visto que este método não pode ser chamado por outras classes.

No caso do construtor não existe cobertura visto que não deve ser possível a instanciação da classe Expedições.



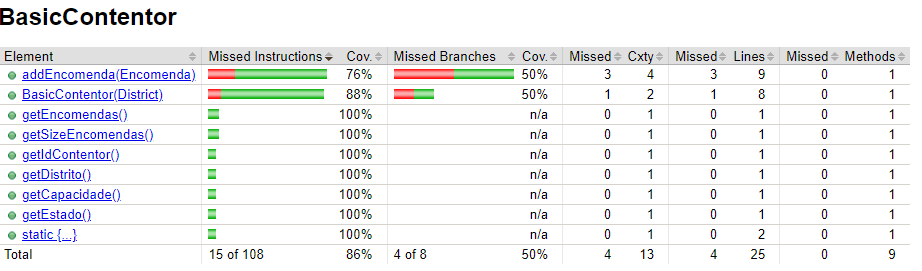
## 

## 2.6. Classe BasicContentor

Como podemos observar na imagem seguinte, na classe BasicContentor foram cobertos 86% das instruções e 50% dos ramos.

O construtor possui uma cobertura de ramos de 50% dado que não estão a ser testados métodos construtores a não ser que se encontre algum erro.

O método addEncomenda também possui uma cobertura de ramos de 50% porque é um método invocado indiretamente pela classe expedições. Como o construtor desta classe foi instanciado como protected, decidimos não criar testes específicos para este método.



## 

## 2.7. Observações

Visto todos os dados fornecidos pelo Jacoco podemos analisar alguns métodos em que a taxa de cobertura se revela bastante baixa, no entanto, isso acontece devido ao facto de que alguns métodos devem ser testados apenas posteriormente, uma vez que, não possuem testes a taxa é 0%.

**Exemplo:**

