**Testes de *software* com *IntelliJ***

***Engenharia de Software II***

**GRUPO 9**

André Ribeiro nº 8150291

Cristiana Monteiro nº 8150489

Engenharia Informática – 2º Ano

Escola Superior de Tecnologia e Gestão Politécnico do Porto

*Novembro de 2018*

Índice

[1 - Âmbito e referências 3](#_Toc530424124)

[1.1 - Âmbito 3](#_Toc530424125)

[1.2 - Referências 3](#_Toc530424126)

[2 - Definições 3](#_Toc530424127)

[3 - Atividades e testes unitários 3](#_Toc530424128)

[3.1 - Planear a abordagem geral 4](#_Toc530424129)

[3.1.1 - Planear *inputs* 4](#_Toc530424130)

[3.1.2 - Planear *tasks* 4](#_Toc530424131)

[3.1.3 - Planear *outputs* 4](#_Toc530424132)

[3.2 - Determinar os recursos a serem testados 4](#_Toc530424133)

[3.3 - Código implementado 4](#_Toc530424134)

[3.3.1 - Classe *AllTests* 5](#_Toc530424135)

[3.3.2 - Classe *MyTests* 5](#_Toc530424136)

[3.4 - Executar os procedimentos de teste 5](#_Toc530424137)

[3.4.1 - *GET BICYCLE()* 5](#_Toc530424138)

[3.4.2 - *RETURN BICYCLE()* 7](#_Toc530424139)

[3.4.3 - *VERIFY CREDIT()* 8](#_Toc530424140)

[3.4.4 - *ADD CREDIT()* 9](#_Toc530424141)

[3.4.5 - *REGISTER USER()* 10](#_Toc530424142)

[3.5 - Verificar as terminações 12](#_Toc530424143)

[3.5.1 - Verificar *inputs* 12](#_Toc530424144)

[3.5.2 - Verificar *tasks* 12](#_Toc530424145)

[3.5.3 - Verificar *outputs* 13](#_Toc530424146)

[3.6 - Analisar os testes 13](#_Toc530424147)

[3.6.1 - Analisar *inputs* 13](#_Toc530424148)

[3.6.2 - Analisar *tasks* 13](#_Toc530424149)

[3.6.3 - Analisar *outputs* 13](#_Toc530424150)

[4 - Anexos 14](#_Toc530424151)

[4.1 - Anexo *AllTests* 14](#_Toc530424152)

[4.2 - Anexo *MyTests* 14](#_Toc530424153)

# 1 - Âmbito e referências

É apresentada, neste capítulo, uma breve contextualização e pesquisa realizada para o desenvolvimento deste trabalho prático e constituição do presente relatório.

## 1.1 - Âmbito

Na disciplina de engenharia de *software* II foi proposto a realização de um trabalho prático com vista a realização de uma bateria de testes de *software* recorrendo às ferramentas lecionadas na aula entre eles:

* *IntelliJ IDEA*
* *Gradle*
* Plataformas de controlo de *software*

## 1.2 - Referências

[http://www.vogella.com/tutorials/JUnit/article.html](http://www.vogella.com/tutorials/JUnit/article.html?fbclid=IwAR2vRnx5vWYXEHh2YOoKYVEJJt_lHtWKZekQtKIGZL3x7CceVJcFRsn7xrc)  
[https://www.jetbrains.com/help/idea/2016.3/creating-test-methods.html](https://www.jetbrains.com/help/idea/2016.3/creating-test-methods.html?fbclid=IwAR2VmlxYVaS663btHMq0dBidwxk4STV1eQ4iqCwd6waKfYlc_mD-pL8vCuM)  
[https://www.toptal.com/java/getting-started-with-junit](https://www.toptal.com/java/getting-started-with-junit?fbclid=IwAR1j5jv08MgqPtw0K5iCik5vRRHYfAf3DyA5dZ7mimVBAKHthQNOo4Rkwk8)  
[https://junit.org/junit5/docs/current/user-guide/](https://junit.org/junit5/docs/current/user-guide/?fbclid=IwAR3UcNCjIILUwcZh06J8XKFwbVJIMGnXU1WsMEsa6yR_POUBMIBH0pTYoFY)  
[https://www.tutorialspoint.com/junit/index.htm](https://www.tutorialspoint.com/junit/index.htm?fbclid=IwAR3erBMD6g09p6g-Sk7t1DCkyXnR6aND32qj_bYspElNQgpG0eqtphCnw4E)  
[https://github.com/brunobmo/ESII\_Exercicio](https://github.com/brunobmo/ESII_Exercicio?fbclid=IwAR07lC8S9RvQCY0Ux8UvSRBWXxo95__nSaRotsj85KWK6xPpIT5QJMqY6ts)

# 2 - Definições

*Input* – traduzido significa entrada. É o ato de entrada de informação/instruções num sistema em que este o irá processar.

*Output* – traduzido significa saída. É designado algo de output quando ao fim de um processo o sistema devolve algo ao utilizador.

*Tasks* – consideramos neste trabalho as *tasks* como casos especiais em todas as circunstâncias em que estas são mencionadas.

# 3 - Atividades e testes unitários

## 3.1 - Planear a abordagem geral

Neste ponto 3.1 serão descritos os planeamentos de *inputs*, *tasks* e *outputs* com vista a realização dos vários testes.

### 3.1.1 - Planear *inputs*

Será elaborado uma bateria de testes a cada método referido no enunciado como objeto de análise, de forma a obter determinados resultados e cobrir o maior número de possibilidades.

### 3.1.2 - Planear *tasks*

Em conjunto com a bateria de testes elaborada (acima referido) para cada método, estas terão determinados casos especiais a serem testados.

### 3.1.3 - Planear *outputs*

Após o planeamento dos inputs e analisando o *java doc* fornecido para este trabalho prático, os *outputs* esperados deverão estar de acordo com esse mesmo planeamento e especificações indicadas anteriormente.

## 3.2 - Determinar os recursos a serem testados

Os recursos definidos como objetos de teste/análise foram:

* *GET BICYCLE()*
* *RETURN BICYCLE()*
* *VERIFY CREDIT()*
* *ADD CREDIT()*
* *REGISTER USER()*

## 3.3 - Código implementado

O seguinte tópico retrata as implementações efetuadas ao longo do processo de testes do *software*.

### 3.3.1 - Classe *AllTests*

Esta classe foi inicialmente criada para efetuar os testes comuns às diversas classes existentes.

O código desta está disponibilizado na categoria anexos presente neste relatório (4.1 - Anexo *AllTests*).

### 3.3.2 - Classe *MyTests*

Esta classe foi inicialmente criada para efetuar os testes principais das várias classes constituintes no presente trabalho prático.

O código desta está disponibilizado na categoria anexos presente neste relatório (4.2 - Anexo *MyTests*).

## 3.4 - Executar os procedimentos de teste

Nos testes abaixo apresentados, foram estudados os casos suscetíveis de ocorrer erros, ou seja, testados os valores válidos e inválidos e ainda os limites. Desta forma foram analisados os principais casos onde podem ocorrer erros.

### 3.4.1 - *GET BICYCLE()*

|  |  |  |
| --- | --- | --- |
| Caso de Uso | Válido | Inválido |
| ***ID User* > 0** | ]0, +00[ | ]-00, 0[ |
| Tipo de entrada | *Integer* | *Integer* |
| Número de entradas | 3 | 3 |

|  |  |  |  |
| --- | --- | --- | --- |
| ***ID USER*** | Testes *inputs* | Testes execução | *Outputs* esperados |
| Teste 1 | -3 |  | Não deve funcionar |
| Teste 2 | -1 |  | Não deve funcionar |
| Teste 3 | 0 |  | Não deve funcionar |
| Teste 4 | 1 |  | Deve funcionar |
| Teste 5 | 3 |  | Deve funcionar |
| Teste 6 | 123 |  | Deve funcionar |
| Teste 7 | *NULL* |  | Não deve funcionar |

|  |  |  |
| --- | --- | --- |
| Caso de Uso | Válido | Inválido |
| **Deposito > 0** | ]0, +00[ | ]-00, 0[ |
| Tipo de entrada | *Integer* | *Integer* |
| Número de entradas | 3 | 3 |

|  |  |  |  |
| --- | --- | --- | --- |
| **Deposito** | Testes *input* | Testes execução | *Outputs* esperados |
| Teste 1 | -3 |  | Não deve funcionar |
| Teste 2 | -1 |  | Não deve funcionar |
| Teste 3 | 0 |  | Não deve funcionar |
| Teste 4 | 1 |  | Deve funcionar |
| Teste 5 | 3 |  | Deve funcionar |
| Teste 6 | 123 |  | Deve funcionar |
| Teste 7 | *NULL* |  | Não deve funcionar |

### 3.4.2 - *RETURN BICYCLE()*

|  |  |  |
| --- | --- | --- |
| Caso de Uso | Válido | Inválido |
| ***ID User* >= 0** | [0, +00[ | ]-00, 0[ |
| Tipo de entrada | *Integer* | *Integer* |
| Número de entradas | 3 | 3 |

|  |  |  |  |
| --- | --- | --- | --- |
| ***ID USER*** | Testes *input* | Testes execução | *Outputs* esperados |
| Teste 1 | -3 |  | Não deve funcionar |
| Teste 2 | -1 |  | Não deve funcionar |
| Teste 3 | 0 |  | Deve funcionar |
| Teste 4 | 1 |  | Deve funcionar |
| Teste 5 | 3 |  | Deve funcionar |
| Teste 6 | 123 |  | Deve funcionar |
| Teste 7 | *NULL* |  | Não deve funcionar |

|  |  |  |
| --- | --- | --- |
| Caso de Uso | Válido | Inválido |
| **Deposito > 0** | ]0, +00[ | ]-00, 0[ |
| Tipo de entrada | *Integer* | *Integer* |
| Número de entradas | 3 | 3 |

|  |  |  |  |
| --- | --- | --- | --- |
| **Deposito** | Testes *input* | Testes execução | *Outputs* esperados |
| Teste 1 | -3 |  | Não deve funcionar |
| Teste 2 | -1 |  | Não deve funcionar |
| Teste 3 | 0 |  | Não deve funcionar |
| Teste 4 | 1 |  | Deve funcionar |
| Teste 5 | 3 |  | Deve funcionar |
| Teste 6 | 123 |  | Deve funcionar |
| Teste 7 | *NULL* |  | Não deve funcionar |

### 3.4.3 - *VERIFY CREDIT()*

|  |  |  |
| --- | --- | --- |
| Caso de Uso | Válido | Inválido |
| ***ID User* > 0** | ]0, +00[ | ]-00, 0[ |
| Tipo de entrada | *Integer* | *Integer* |
| Número de entradas | 3 | 3 |

|  |  |  |  |
| --- | --- | --- | --- |
| **Deposito** | Testes *input* | Testes execução | *Outputs* esperados |
| Teste 1 | -3 |  | Não deve funcionar |
| Teste 2 | -1 |  | Não deve funcionar |
| Teste 3 | 0 |  | Não deve funcionar |
| Teste 4 | 1 |  | Deve funcionar |
| Teste 5 | 3 |  | Deve funcionar |
| Teste 6 | 123 |  | Deve funcionar |
| Teste 7 | *NULL* |  | Não deve funcionar |

### 3.4.4 - *ADD CREDIT()*

|  |  |  |
| --- | --- | --- |
| Caso de Uso | Válido | Inválido |
| ***ID User* >= 0** | [0, +00[ | ]-00, 0[ |
| Tipo de entrada | *Integer* | *Integer* |
| Número de entradas | 3 | 3 |

|  |  |  |  |
| --- | --- | --- | --- |
| ***ID USER*** | Testes *input* | Testes execução | *Outputs* esperados |
| Teste 1 | -3 |  | Não deve funcionar |
| Teste 2 | -1 |  | Não deve funcionar |
| Teste 3 | 0 |  | Deve funcionar |
| Teste 4 | 1 |  | Deve funcionar |
| Teste 5 | 3 |  | Deve funcionar |
| Teste 6 | 123 |  | Deve funcionar |
| Teste 7 | *NULL* |  | Não deve funcionar |

|  |  |  |
| --- | --- | --- |
| Caso de Uso | Válido | Inválido |
| ***Amount* > 0** | ]0, +00[ | ]-00, 0[ |
| Tipo de entrada | *Integer* | *Integer* |
| Nº de entradas | 3 | 3 |

|  |  |  |  |
| --- | --- | --- | --- |
| ***Amount*** | Testes *input* | Testes execução | *Outputs* esperados |
| Teste 1 | -3 |  | Não deve funcionar |
| Teste 2 | -1 |  | Não deve funcionar |
| Teste 3 | 0 |  | Não deve funcionar |
| Teste 4 | 1 |  | Deve funcionar |
| Teste 5 | 3 |  | Deve funcionar |
| Teste 6 | 123 |  | Deve funcionar |
| Teste 7 | *NULL* |  | Não deve funcionar |

### 

### 3.4.5 - *REGISTER USER()*

|  |  |  |
| --- | --- | --- |
| Caso de Uso | Válido | Inválido |
| ***ID User* >= 0** | [0, +00[ | ]-00, 0[ |
| Tipo de entrada | *Integer* | *Integer* |
| Número de entradas | 3 | 3 |

|  |  |  |  |
| --- | --- | --- | --- |
| ***ID* *USER*** | Testes *input* | Testes execução | *Outputs* esperados |
| Teste 1 | -3 |  | Não deve funcionar |
| Teste 2 | -1 |  | Não deve funcionar |
| Teste 3 | 0 |  | Deve funcionar |
| Teste 4 | 1 |  | Deve funcionar |
| Teste 5 | 3 |  | Deve funcionar |
| Teste 6 | 123 |  | Deve funcionar |
| Teste 7 | *NULL* | *Error* | Não deve funcionar |
| Teste 8 | *ID\_EXISTENTE* | *UserAlreadyExists* | Deve devolver uma exceção |

|  |  |  |
| --- | --- | --- |
| Caso de Uso | Válido | Inválido |
| ***Name* != *NULL*** | != *NULL* | *NULL* |
| Tipo de entrada | *String* | *String* |
| Nº de entradas | 1 | 1 |

|  |  |  |  |
| --- | --- | --- | --- |
| ***Name* != *NULL*** | Testes *input* | Testes execução | *Outputs* esperados |
| Teste 1 | *NULL* |  | Não deve funcionar |
| Teste 2 | != *NULL* |  | Deve funcionar |
| Teste 3 | 0 |  | Não deve funcionar |

|  |  |  |
| --- | --- | --- |
| Caso de Uso | Válido | Inválido |
| ***RENTAL PROGRAM* = 1 ou 2** | 1 || 2 | ]-00, 1[ U ]2, +00[ |
| Tipo de entrada | *Integer* | *Integer* |
| Número de entradas | 3 | 3 |

|  |  |  |  |
| --- | --- | --- | --- |
| ***RENTAL* *PROGRAM* = 1 ou 2** | Testes *input* | Testes execução | *Outputs* esperados |
| Teste 1 | -3 |  | Não deve funcionar |
| Teste 2 | -1 |  | Não deve funcionar |
| Teste 3 | 0 |  | Não Deve funcionar |
| Teste 4 | 1 |  | Deve funcionar |
| Teste 5 | 2 |  | Deve funcionar |
| Teste 6 | 3 |  | Não deve funcionar |
| Teste 7 | 123 |  | Não deve funcionar |
| Teste 8 | *NULL* |  | Não deve funcionar |

## 3.5 - Verificar as terminações

No 3.5 poderemos visualizar as verificações efetuadas nos quadros anteriormente apresentados podendo numa próxima fase fazer conclusões relativas aos testes efetuados nos respetivos métodos.

### 3.5.1 - Verificar *inputs*

Realização da verificação da informação recebida pelo sistema que irá retornar uma “resposta” para análise tendo em conta todas as especificações dos métodos e testes a realizar.

### 3.5.2 - Verificar *tasks*

Realização da verificação da informação pelo sistema que irá retornar uma “resposta” para análise tendo em conta todos casos especiais a serem testados em todos os métodos e testes a realizar.

### 3.5.3 - Verificar *outputs*

Realização da verificação do retorno do sistema à informação anteriormente enviada sendo esta um caso especial de teste ou simplesmente um teste normal.

## 3.6 - Analisar os testes

Verificação e analise de todos os testes dos métodos efetuados com conclusões.

### 3.6.1 - Analisar *inputs*

Nesta fase, poderemos verificar que foi realizada o envio da informação dos *inputs* referente a cada teste de cada um dos métodos a realizar.

### 3.6.2 - Analisar *tasks*

Em todos os métodos existem casos especiais a serem testados. Estes são:

* *NULL*
* 0 (Número zero)

### 3.6.3 - Analisar *outputs*

Na análise de *outputs*, poderemos verificar o resultado do *input* enviado para a realização de um determinado teste num determinado método.

# 4 - Anexos

## 4.1 - Anexo *AllTests*

import Models.Bike;

import Models.Deposit;

import Models.Lock;

import Models.User;

import org.junit.runner.RunWith;

import org.junit.runners.Suite;

@RunWith(Suite.class)

@Suite.SuiteClasses({

BikeRentalSystem.class,

User.class,

Deposit.class,

Lock.class,

Bike.class

})

public class ALLTests {

}

## 4.2 - Anexo *MyTests*

import Exceptions.UserAlreadyExists;

import Exceptions.UserDoesNotExists;

import Models.User;

import org.junit.Test;

import org.junit.jupiter.api.Assertions;

import org.junit.jupiter.api.BeforeEach;

import static org.junit.Assert.assertEquals;

public class MyTests {

private BikeRentalSystem user;

@BeforeEach

public void setUp() throws UserAlreadyExists {

user = new BikeRentalSystem(1);

user.registerUser(30, "andre", 1);

}

@Test

public void getName() {

User person = new User(11, "andre", 11);

String ResName = person.getName();

assertEquals("andre", ResName);

}

@Test

public void IDUser() {

User id = new User (2, "andre", 23);

assertEquals(id.getIDUser(), 2);

// assertEquals(id.getIDUser(), -6); // Teste com valores negativos!

}

@Test

public void getBicycleTestUserInvalido() throws UserAlreadyExists, UserDoesNotExists {

BikeRentalSystem user = new BikeRentalSystem(1);

user.registerUser(2, "andre", 1);

Assertions.assertThrows(UserDoesNotExists.class, () -> user.getBicycle(1, 1, 1));

Assertions.assertThrows(UserDoesNotExists.class, () -> user.getBicycle(1, 0, 1));

Assertions.assertThrows(UserDoesNotExists.class, () -> user.getBicycle(1, -1, 1));

}

@Test

public void verifyCredit() throws UserAlreadyExists {

int idUser = 1;

int rentalFee = 1;

BikeRentalSystem user = new BikeRentalSystem(rentalFee);

user.registerUser(idUser, "andre", 1);

user.addCredit(idUser, 10);

Assertions.assertTrue(user.verifyCredit(idUser));

}

@Test

public void verificaSeExisteID() {

User id = new User (30, "andre", 1);

Assertions.assertThrows(UserAlreadyExists.class, () -> user.registerUser(30, "andre", 1));

}

}