P.PORTO

ESCOLA
SUPERIOR
DE TECNOLOGIA
E GESTÃO

Relatório Part.1

Engenharia de Software II

Gabriel Pereira 8150115

Nuno Josefino 8150189

Rafael Vieira 8150448

Índice

Introdução

O presente trabalho surge no âmbito da Unidade Curricular de Engenharia de Software II e em específico da parte prática da mesma.

Considere-se a existência de um sistema público de aluguer de bicicletas.

O sistema é composto por vários depósitos de bicicletas espalhados pela cidade. Cada depósito contém cerca de uma dúzia de bicicletas numa área para o efeito. O depósito de bicicletas contém um determinado número de lugares devidamente numerados, sendo um lugar para cada bicicleta, onde as bicicletas estão inicialmente colocadas com um sistema "*lock/unlock*".

Este sistema encontra-se conectado ao sistema principal e funciona da seguinte forma:

- Se um utilizador está registado e tem crédito na sua conta, pode alugar uma bicicleta, o sistema "liberta" a bicicleta e o utilizador pode retirar uma bicicleta, tornando-se responsável por ela até que a devolva.;

- Quando o utilizador pretende devolver a bicicleta, escolhe um lugar disponível no depósito de bicicletas para o estacionamento da mesma. O sistema *"lock/unlock"* deteta a bicicleta automaticamente e "prende" a bicicleta. A partir desse momento o utilizador deixa de ser responsável pela bicicleta.

- Para que uma pessoa possa alugar bicicletas terá de se registar, fornecendo um conjunto de dados entre os quais o seu nome e a informação do cartão de crédito, recebendo um login (ID). Se, de seguida, o utilizador desejar alugar uma bicicleta, deve dirigir-se a um depósito e introduz, através de uma interface para o efeito, como um monitor tátil, o seu login (ID). Após as respetivas validações, o sistema seleciona uma bicicleta entre as disponíveis e "liberta-a". No momento da entrega, não há qualquer interação com qualquer interface.

Para além das funcionalidades descritas, o sistema deve acompanhar o estado de todas as bicicletas e alugueres., o utilizador deverá pagar uma determinada quantia pelo aluguer, dependendo da duração do respetivo aluguer e os utilizadores podem escolher uma bicicleta de um determinado depósito e entrega-la num outro qualquer à escolha. O sistema deverá igualmente monitorizar a manutenção das bicicletas (uma bicicleta que nunca foi alugada pode provavelmente estar avariada). Esta monitorização deverá ser realizada à distribuição das bicicletas pelos depósitos.

Os requisitos do problema apresentado encontram-se implementados na biblioteca *BikeRentalSystem*.

Este trabalho tem como objetivo a especificação de casos de teste utilizando as técnicas *Equivalence Class Partitioning* e *Boundary Value Analysis* no nível de testes unitários especificando casos de teste para cada método. Deverá ser descrito igualmente *test inputs*, *execution conditions* e *expected outputs*, assegurando que os casos de teste cobrem *Valid equivalence classes* e *Invalid equivalence classes*.

Todos estes testes deverão ser implementados e executados utilizando a *framework*: *JUnit* 5.1.

Como último objetivo define-se a elaboração de um relatório de testes, registo dos resultados dos testes especificando o *test case ID*, objeto testado, descrição, entradas, saídas esperadas e comentários relevantes. De igual forma, a implementação de cada um dos testes deverá devidamente documentado de forma a identificar cada caso de teste.

Metódo getBicycle

Tabela ECP

getBicycle		
Critério	Classe de Equivalência Válida	Classe de Equivalência Inválida
Nº inputs	3	≠3
Tipo de entradas	IDUser - int, IDDeposit - int, starttime - int	IDUser ≠ int, IDDeposit ≠ int, starttime ≠ int
Valor especifico	IDUser > 0, IDDeposit > 0, starttime >= 0	IDUser <= 0, IDDeposit <= 0, starttime < 0
	Casos de Teste	
	IDUser=1, IDDeposit=1, starttime=1	
	IDUser=, IDDeposit=1, starttime=1	
	IDUser=1 2, IDDeposit=1, starttime=1	
	IDUser="x", IDDeposit=1, starttime=1	
	IDUser=1, IDDeposit="x", starttime=1	
	IDUser=1, IDDeposit=1, starttime="x"	
	IDUser=0, IDDeposit=1, starttime=1	
	IDUser=1, IDDeposit=0, starttime=1	
	IDUser=1, IDDeposit=1, starttime=-1	

Tabela BVA

Método	getBicycle

Cenários	
1	IDUser>0, IDDeposit >0, starttime>=0
2	IDUser=<0, IDDeposit >0, , starttime>=0
3	IDUser>0, IDDeposit=<0, , starttime>=0
4	IDUser>0, IDDeposit>0, , starttime<0

Casos de Teste	Input			Output
	IDUser	**IDDeposit**	**starttime**	
1	1	1	0	identificador da bicicleta que será utilizada
2	0	1	0	-1
3	1	1	0	identificador da bicicleta que será utilizada
4	1	0	0	-1
5	1	1	-1	-1
6	1	1	0	identificador da bicicleta que será utilizada
7	1	1	1	identificador da bicicleta que será utilizada

Casos de teste

Caso de teste 0 (testGetBicycleUSERMENOSUM)

```java
/*
Teste para IDUser=-1 (enunciado incoerente pois pretendia-se testar o IDUser=0)
*/
    @Test
    public void testGetBicycleIDUSERMENOSUM() throws UserDoesNotExists {
        //Adicionar Crédito
        brs.addCredit( idUser: 0,  amount: 1);

        //Adição de um Lock
        brs.addLock( idDeposit: 1,  idLock: 1);

        //Adição de uma Bicicleta
        brs.addBicycle( idDeposit: 1,  idLock: 1,  idBike: 1);
        assertThrows(UserDoesNotExists.class, () -> {
                    //Retorna -1 se tentar requisitar uma bicicleta sem depósito
                    assertEquals( expected: -1, brs.getBicycle( IDDeposit: 1,  IDUser: -1,  startTime: 0));
                }, message: "Should throw exception: UserDoesNotExists");

        brs.getBicycle( IDDeposit: 1,  IDUser: 0,  startTime: 0);
    }
```

Caso de teste extra criado devido á falta de validações no IDUser que permite que este seja 0 (o que não deveria acontecer visto ser suposto que IDUser > 0) e para que sejam testadas todas as fronteiras foi necessário adicionar o teste para IDUser = -1.

Caso de teste 1/3/6 (testGetBicycleEXISTEBICICLETA)

```java
    // GET BICYCLE

    /*
    Teste Requisitar bicicleta existente com starttime=0
    */
    @Test
    public void testGetBicycleEXISTEBICICLETA() throws UserDoesNotExists {
        //Adicionar Crédito
        brs.addCredit( idUser: 1,  amount: 1);

        //Adição de um Lock
        brs.addLock( idDeposit: 1,  idLock: 1);

        //Adição de uma Bicicleta
        brs.addBicycle( idDeposit: 1,  idLock: 1,  idBike: 1);

        //Retorna -1 se tentar requisitar uma bicicleta sem depósito
        assertEquals( expected: 0, brs.getBicycle( IDDeposit: 1,  IDUser: 1,  startTime: 0));

       brs.getBicycle( IDDeposit: 1,  IDUser: 1,  startTime: 0);
    }
```

Caso de teste 4 (testGetBicucleSEMBICICLETASDISP)

```java
/*
Teste não há bicicletas disponiveis
 */
@Test
public void testGetBicycleSEMBICICLETASDISP() throws UserDoesNotExists {
    //Adicionar
    brs.addCredit( idUser: 1,  amount: 1);

    //Adição de um Lock
    brs.addLock( idDeposit: 1,  idLock: 1);

    //Retorna -1 se tentar requisitar uma bicicleta sem lugares ativos
    assertEquals( expected: -1, brs.getBicycle( IDDeposit: 0, IDUser: 1, startTime: 0));

    brs.getBicycle( IDDeposit: 0,  IDUser: 1,  startTime: 0);
}
```

Caso de teste 4 (testGetBicycleNAOEXISTEDEPOSITO)

```java
/*
Teste com depósito inexistente
 */
@Test
public void testGetBicycleNAOEXISTEDEPOSITO() throws UserDoesNotExists {
    //Adicionar Crédito
    brs.addCredit( idUser: 1,  amount: 1);

    //Adição de um Lock
    brs.addLock( idDeposit: 0,  idLock: 1);

    //Adição de uma Bicicleta
    brs.addBicycle( idDeposit: 1,  idLock: 1,  idBike: 1);

    //Retorna -1 se tentar requisitar uma bicicleta sem depósito
    assertEquals( expected: -1, brs.getBicycle( IDDeposit: 0, IDUser: 1, startTime: 0));

    brs.getBicycle( IDDeposit: 0,  IDUser: 1,  startTime: 0);
}
```

Caso de teste 5 (testGetBicycleSTARTTIMEMENOSUM)

```
/*
Teste com starttime=-1
 */
@Test
public void testGetBicycleSTARTTIMEMENOSUM() throws UserDoesNotExists {
    //Adicionar Crédito
    brs.addCredit( idUser: 1,  amount: 1);

    //Adição de um Lock
    brs.addLock( idDeposit: 0,  idLock: 1);

    //Adição de uma Bicicleta
    brs.addBicycle( idDeposit: 1,  idLock: 1,  idBike: 1);

    //Retorna -1 se tentar requisitar uma bicicleta sem depósito
    assertEquals( expected: -1, brs.getBicycle( IDDeposit: 0, IDUser: 1, startTime: -1));

    brs.getBicycle( IDDeposit: 0,  IDUser: 1,  startTime: 0);
}
```

Caso de teste 7 (testGetBicycleSTARTUM)

```
/*
Teste requisita bicicleta com starttime=1
 */
@Test
public void testGetBicycleSTARTUM() throws UserDoesNotExists {
    //Adicionar Crédito
    brs.addCredit( idUser: 1,  amount: 1);

    //Adição de um Lock
    brs.addLock( idDeposit: 1,  idLock: 1);

    //Adição de uma Bicicleta
    brs.addBicycle( idDeposit: 1,  idLock: 1,  idBike: 1);

    //Testar se retorna -1 se tentar requisitar uma bicicleta sem depósito
    assertEquals( expected: 0, brs.getBicycle( IDDeposit: 1, IDUser: 1, startTime: 1));

    brs.getBicycle( IDDeposit: 1,  IDUser: 1,  startTime: 1);
}
```

Metódo returnBikycle

Tabela ECP

returnBicycle		
Critério	Classe de Equivalência Válida	Classe de Equivalência Inválida
Nº inputs	3	≠3
Tipo de entradas	IDUser - int, IDDeposit - int, endtime - int	IDUser ≠ int, IDDeposit ≠ int, endtime ≠ int
Valor especifico	IDUser >= 0, IDDeposit > 0, endtime >= 0	IDUser < 0, IDDeposit <= 0, endtime < 0
	Casos de Teste	
	IDUser=1, IDDeposit=1, endtime=1	
	IDDeposit=1,IDDeposit=, endtime=1	
	IDUser =1 2, IDDeposit=1, endtime=1	
	IDUser="x", IDDeposit=1,endtime=1	
	IDUser=1, IDDeposit="x", endtime=1	
	IDUser=1, IDDeposit=1, endtime="x"	
	IDUser=-1, IDDeposit=1, endtime=1	
	IDUser=1, IDDeposit=0, endtime=1	
	IDUser=1, IDDeposit=1, endtime=-1	

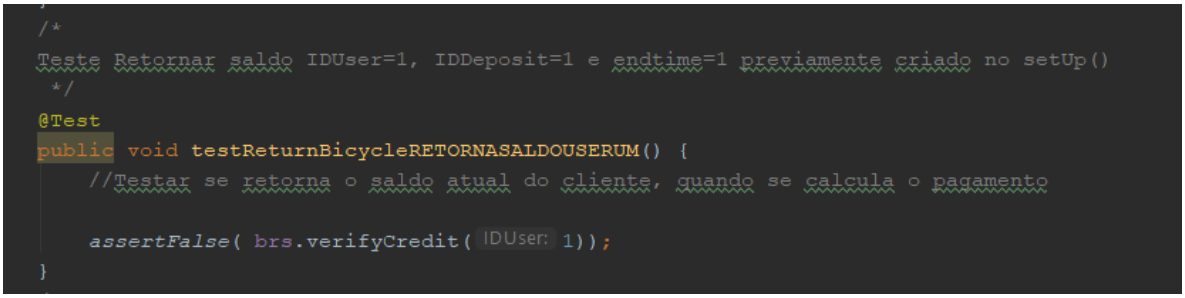
Tabela BVA

Método	returnBikycle

Cenários	
1	IDUser>=0, IDDeposit >0, endtime>=0
2	IDUser<0, IDDeposit >0, , endtime>=0
3	IDUser>0, IDDeposit=<0, , endtime>=0
4	IDUser>0, IDDeposit>0, , endtime<0

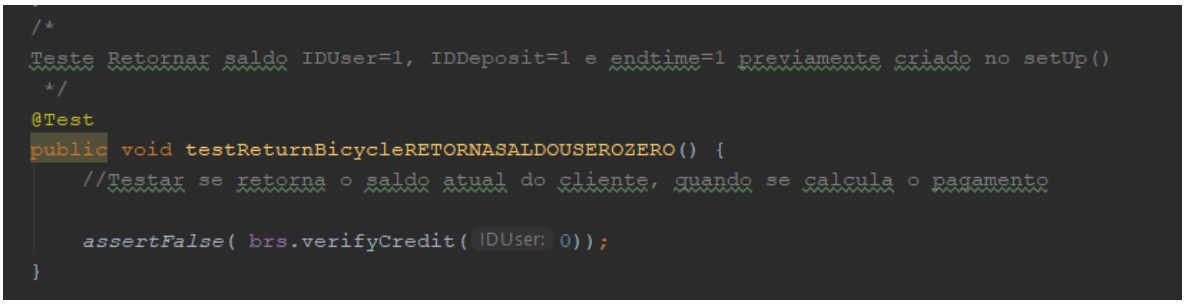
Casos de Teste	Input			Output
	IDUser	**IDDeposit**	**endtime**	
1	1	1	1	Saldo atual
2	0	1	1	Saldo atual
3	-1	1	1	-1
4	1	1	1	Saldo atual
5	1	0	1	-1
6	1	1	1	Saldo atual
7	1	1	0	Saldo atual
8	1	1	-1	-1

Caso de teste 1 (testReturnBicycleRETORNASALDOUSERUM)



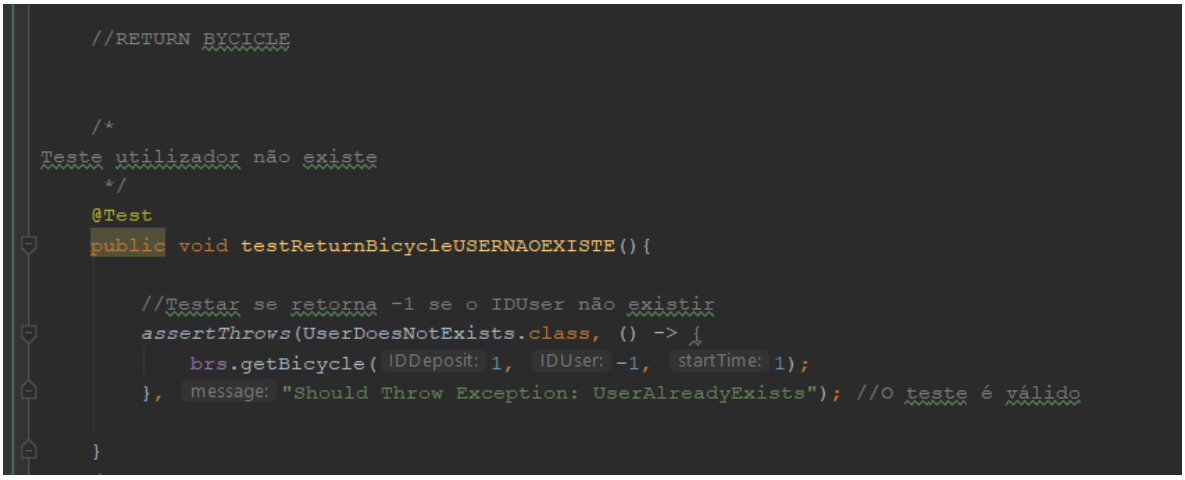
```java
/*
Teste Retornar saldo IDUser=1, IDDeposit=1 e endtime=1 previamente criado no setUp()
 */
@Test
public void testReturnBicycleRETORNASALDOUSERUM() {
    //Testar se retorna o saldo atual do cliente, quando se calcula o pagamento

    assertFalse( brs.verifyCredit( IDUser: 1));
}
```

Caso de teste 2 (testReturnBicycleRETORNASALDOUSEROZERO)



```java
/*
Teste Retornar saldo IDUser=1, IDDeposit=1 e endtime=1 previamente criado no setUp()
 */
@Test
public void testReturnBicycleRETORNASALDOUSEROZERO() {
    //Testar se retorna o saldo atual do cliente, quando se calcula o pagamento

    assertFalse( brs.verifyCredit( IDUser: 0));
}
```

Caso de teste 3 (testReturnBicycleUSERNAOEXISTE)



```java
    //RETURN BYCICLE


    /*
Teste utilizador não existe
     */
    @Test
    public void testReturnBicycleUSERNAOEXISTE(){

        //Testar se retorna -1 se o IDUser não existir
        assertThrows(UserDoesNotExists.class, () -> {
            brs.getBicycle( IDDeposit: 1, IDUser: -1, startTime: 1);
        }, message: "Should Throw Exception: UserAlreadyExists"); //O teste é válido

    }
```

Caso de teste 1/4/6 (testReturnBicycleDEPOSITEXISTE)

```
/*
Teste IDDeposit existe (TESTE QUE DEVIA PASSAR COM EXPECTED = 0)
 */
@Test
public void testReturnBicycleDEPOSITEXISTE()throws UserDoesNotExists{

    //Testar se retorna 0 se o IDDeposit  existir

    assertEquals( expected: 0, brs.returnBicycle( IDDeposit: 1,  IDUser: 1,  endTime: 1));

}
```

Neste teste, é suposto o programa retornar o valor 0 quando o IDDeposit existe, mas isto não se comprova e como tal o teste não funciona como esperado.

Caso de teste 5 (testReturnBicycleDEPOSITNAOEXISTE)

```
/*
Teste IDDeposit não existe
 */
@Test
public void testReturnBicycleDEPOSITNAOEXISTE()throws UserDoesNotExists{

    //Testar se retorna -1 se o IDDeposit não existir

   assertEquals( expected: -1, brs.returnBicycle( IDDeposit: 0,  IDUser: 1,  endTime: 1));

}
```

Caso de teste 8 (testReturnBicycleSEMLUGARESLIVRES)

```
/*
Teste sem lugares livres (endtime=-1)
 */
@Test
public void testReturnBicycleSEMLUGARESLIVRES() {


    //Senão existirem lugares de entrega livre (endtime=-1)
    assertEquals( expected: -1, brs.returnBicycle( IDDeposit: 1,  IDUser: 1,  endTime: -1));

}
```

Metódo bicycleRentalFee

Tabela ECP

bicycleRentalFee		
Critério	Classe de Equivalência Válida	Classe de Equivalência Inválida
Nº inputs	4	≠4
Tipo de entradas	rentalProgram - int, starttime - int, endtime - int, nRentals - int	rentalProgram ≠ int, starttime ≠ int, endtime ≠ int, nRentals ≠ int
Valor especifico	rentalProgram >=0, starttime>= 0, endtime >= 0, startTime<=endtime, nRentals >=0	talProgram < 0, starttime < 0, endtime < 0, startTime > endtime, nRentals < 0
	Casos de Teste	
	rentalProgram=1, starttime=0, endtime=1, nRentals=1	
	IDDeposit=1,starttime=, endtime=1, nRentals=1	
	rentalProgram=1 2, starttime=0, endtime=1, nRentals=1	
	rentalProgram="x" , starttime=0, endtime=1, nRentals=1	
	rentalProgram=1 , starttime="x", endtime=1, nRentals=1	
	rentalProgram=1 , starttime=0, endtime="x", nRentals=1	
	rentalProgram=1 , starttime=0, endtime=1 , nRentals="x"	
	rentalProgram=-1 , starttime=0, endtime=1 , nRentals=1	
	rentalProgram=1 , starttime=-1, endtime=1 , nRentals=1	
	rentalProgram=1 , starttime=0, endtime=-1 , nRentals=1	
	rentalProgram=1 , starttime=2, endtime=1 , nRentals=1	
	rentalProgram=1 , starttime=0, endtime=1 , nRentals=-1	

Tabela BVA

Método	bicycleRentalFee
Cenários	
1	rentalProgram >=0, starttime>=0, endtime>=0, starttime <=endtime, nRentals>=0
2	rentalProgram <0, starttime>=0, endtime>=0, starttime <=endtime, nRentals>=0
3	rentalProgram >=0, starttime<0, endtime>=0, starttime<=endtime, nRentals>=0
4	rentalProgram >=0, starttime>=0, endtime<0, starttime <=endtime, nRentals>=0
5	rentalProgram >=0, starttime<0, endtime>=0, starttime>endtime, nRentals>=0
6	rentalProgram >=0, starttime>=0, endtime<0, starttime <=endtime, nRentals<0

Casos de Teste	Input				Output
	rentalProgram	**starttime**	**endtime**	**nRentals**	
1	-1	1	2	1	0
2	0	1	2	1	0
3	1	1	2	1	valor referente ao programa de aluguer
4	2	1	2	1	valor referente ao programa de aluguer
5	3	1	2	1	0
6	1	-1	1	1	0
7	1	0	1	1	valor referente ao programa de aluguer
8	1	1	2	1	valor referente ao programa de aluguer
9	1	1	-1	1	0
10	1	1	0	1	0
11	1	1	1	1	valor referente ao programa de aluguer
12	1	1	2	-1	0
13	1	1	2	0	valor referente ao programa de aluguer
14	1	1	2	1	valor referente ao programa de aluguer

Caso de teste

Caso de teste 1 – testbicycleRentalfeeRENTALMENOS1

```
    // BICYCLE RENTALFEE


    /*
 Teste rentalProgram=-1
  */
    @Test
    public void testbicycleRentalfeeRENTALMENOS1() {

        assertEquals( expected: 0,brs.bicycleRentalFee( rentalProgram: -1,  initTime: 1, endTime: 1, nRentals: 1));

    }
```

Caso de teste 2 – testbicycleRentalfeeRENTALZERO

```
    /*
 Teste rentalProgram=0
 */
    @Test
    public void testbicycleRentalfeeRENTALZERO() {

        assertEquals( expected: 0,brs.bicycleRentalFee( rentalProgram: 0,  initTime: 1, endTime: 1, nRentals: 1));

    }
```

Caso teste 3/8/14 – testbicycleRentalfeeRENTALUM

```
    /*
 Teste rentalProgram=1
 */
    @Test
    public void testbicycleRentalfeeRENTALUM() {

        assertEquals( expected: 1,brs.bicycleRentalFee( rentalProgram: 1,  initTime: 1, endTime: 2, nRentals: 1));

    }
```

Caso teste 4 – testbicycleRentalfeeRENTALDOIS

```
    /*
 Teste rentalProgram=2
 */
    @Test
    public void testbicycleRentalfeeRENTALDOIS() {

        assertEquals( expected: 1,brs.bicycleRentalFee( rentalProgram: 2,  initTime: 1, endTime: 2, nRentals: 1));

    }
```

Caso teste 5 – testbicycleRentalfeeRENTALTRES

```
    /*
Teste rentalProgram=3
*/
    @Test
    public void testbicycleRentalfeeRENTALTRES() {

        assertEquals( expected: 0,brs.bicycleRentalFee( rentalProgram: 3,  initTime: 1, endTime: 2, nRentals: 1));

    }
```

Caso de teste 6 – testbicycleRentalfeeSTARTMENOSUM

```
    /*
Teste starttime=-1 (NÃO DEVIA ACEITAR STARTTIME<0)
*/
    @Test
    public void testbicycleRentalfeeSTARTMENOSUM() {

        assertEquals( expected: 0,brs.bicycleRentalFee( rentalProgram: 1,  initTime: -1, endTime: 1, nRentals: 1));

    }
```

Este teste não funciona como esperado visto que não é suposto o valor do starttime assumir valores negativos, mas quando se corre o teste este valor é aceite e não conseguimos obter o resultado esperado para este teste.

Caso de teste 7 – testbicycleRentalfeeSTARTZERO

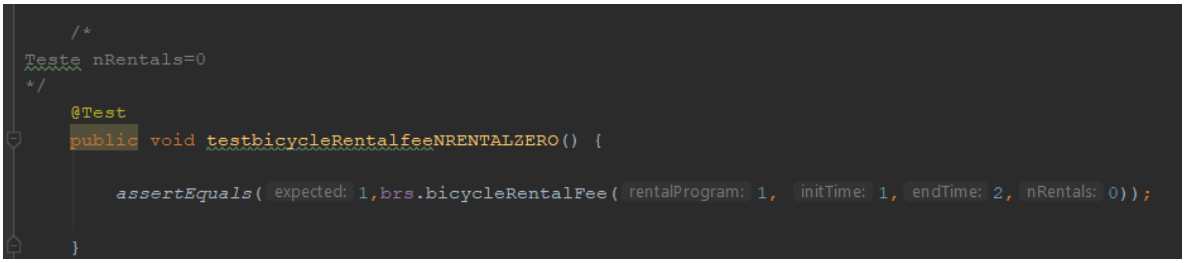
```
    /*
Teste starttime=0
*/
    @Test
    public void testbicycleRentalfeeSTARTZERO() {

        assertEquals( expected: 1,brs.bicycleRentalFee( rentalProgram: 1,  initTime: 0, endTime: 1, nRentals: 1));

    }
```

Caso de teste 9 – testbicycleRentalfeeNRENTALMENOSUM

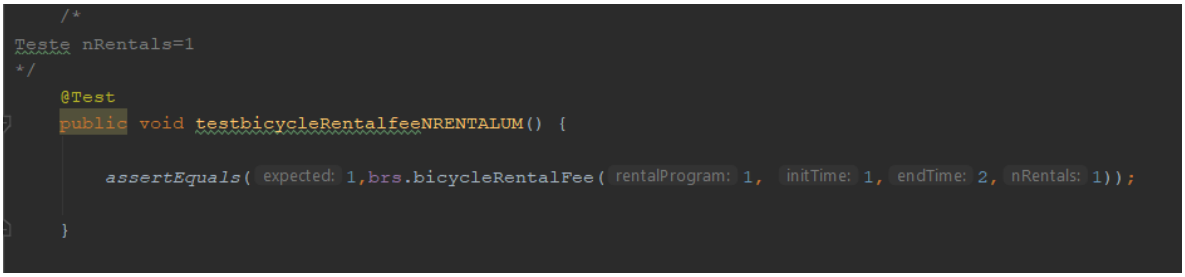
```
    /*
Teste nRentals=-1 (NAO DEVIA ACEITAR NRENTALS NEGATIVO)
*/
    @Test
    public void testbicycleRentalfeeNRENTALMENOSUM() {

        assertEquals( expected: 0,brs.bicycleRentalFee( rentalProgram: 1,  initTime: 1, endTime: 2, nRentals: -1));

    }
```

Este teste não funciona, uma vez que o valor de nRentals não é suposto ser aceite quando é negativo, mas o programa não assume este valor negativo como um erro e como tal não é possível obter o resultado esperado.

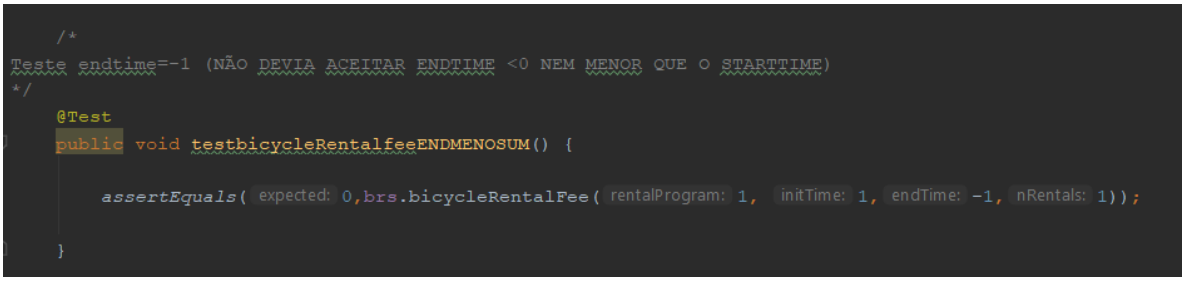
Caso de teste 10 – testbicycleRentalfeeNRENTALZERO



Caso de teste 11 – testbicycleRentalfeeNRENTALUM

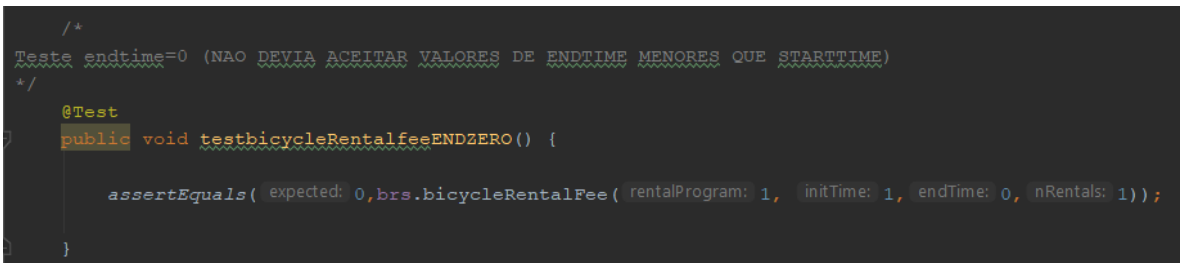


Caso de teste 12 – testbicycleRentalfeeENDMENOSUM



Não é suposto o valor da variável endtime ser aceite como negativo, assim como também não é suposto ser aceite como menor que o valor de starttime, como tal não conseguimos obter o resultado esperado para este teste.

Caso de teste 13 – testbicycleRentalfeeENDZERO



Este teste não funciona como esperado visto que o programa, ao contario do suposto, aceita um valor de endtime menor que o valor do starttime, não sendo possível obter o resultado proposto.

Metódo verifyCredit

Tabela ECP

verifyCredit		
Critério	Classe de Equivalência Válida	Classe de Equivalência Inválida
Nº inputs	1	0, >1
Tipo de entradas	IDUser - int	IDUser ≠ int
Valor especifico	IDUser >= 0	IDUser < 0
	Casos de Teste	
	IDUser=1	
	IDUser=	
	IDUser=1 2	
	IDUser="x"	
	IDUser=-1	

Tabela BVA

Método	verifyCredit
Cenários	
1	O IDUser >=0
2	O IDUser <0

Casos de Teste	Input	Output
	IDUser	
1	-1	FALSE
2	0	TRUE
3	1	TRUE

Caso de teste 1 (testVerifyCreditUSERMENOSUM)

```
// VERIFY CREDIT


/*
Teste com  IDUser=-1 e amount=1
 */
@Test
public void testVerifyCreditUSERMENOSUM() {

    //Adiciona um crédito
    brs.addCredit( idUser: -1,  amount: 1);

    assertFalse(brs.verifyCredit( IDUser: -1));
}
```

Caso de teste 2 (testVerifyCreditUSERZERO)

```
/*
Teste com  IDUser=0 e amount=1
 */
    @Test
    public void testVerifyCreditUSERZERO() {

        //Adiciona um crédito
        brs.addCredit( idUser: 0,  amount: 1);

        assertTrue(brs.verifyCredit( IDUser: 0));
    }
```

Caso de teste 3 (testVerifyCreditUSERUM)

```
Teste com  IDUser=1 e amount=1
 */
    @Test
    public void testVerifyCreditUSERUM() {

        //Adiciona um crédito
        brs.addCredit( idUser: 1,  amount: 1);

        assertTrue(brs.verifyCredit( IDUser: 1));
    }
```

Metódo addCredit

Tabela ECP

addCredit		
Critério	Classe de Equivalência Válida	Classe de Equivalência Inválida
Nº inputs	2	≠2
Tipo de entradas	IDUser - int, amount - int	IDUser ≠ int, amount ≠ int
Valor especifico	IDUser >= 0, amount > 0	IDUser < 0, amount <=0
	Casos de Teste	
	IDUser=1, amount=1	
	IDUser=1, amount=	
	IDUser=1 2, amount=1	
	IDUser="x", amount=1	
	IDUser=1, amount="x"	
	IDUser=-1, amount=1	
	IDUser=1, amount=-1	

Tabela BVA

Método	addCredit

Cenários	
1	IDUser>=0, amount > 0
2	IDUser<0, amount > 0
3	IDUser>=0, amount<=0

Casos de Teste	Input		Output
	IDUser	**amount**	
1	1	1	adicionar valor amount ao credito
2	0	1	adicionar valor amount ao credito
3	-1	1	null
4	1	1	adicionar valor amount ao credito
5	1	0	0

Caso de teste 1/4 testAddCredit1

```
// ADD CREDIT



/*
Teste utilizador com IDUser=1 e  amount=1
 */
@Test
public void testAddCredit1() {
    //Utilizador em que se adiciona créditos
    User u = brs.getUsers().get(1);

    brs.addCredit( idUser: 1,  amount: 1);

    //O crédito foi adicionado com sucesso
    assertEquals( expected: 1, u.getCredit(),  message: "Expected = 1, Actual = " + u.getCredit());
}
```

Caso de teste 2 (testAddCredit2)

```
/*
Teste utilizador com IDUser=0 e amount=1
 */
@Test
public void testAddCredit2() {
    //Utilizador em que se adiciona créditos
    User u = brs.getUsers().get(0);

    brs.addCredit( idUser: 0,  amount: 1);

    //O crédito foi adicionado com sucesso
    assertEquals( expected: 1, u.getCredit(),  message: "Expected = 1, Actual = " + u.getCredit());
}
```

Caso de teste 3 (testAddCreditMenosUM)

```
/*
Teste utilizador com IDUser=-1 e amount =1
 */
@Test
public void testAddCreditUserMenosUM(){

    assertThrows(IndexOutOfBoundsException.class, () -> {


    User u = brs.getUsers().get(-1);

    brs.addCredit( idUser: -1,  amount: 1);

    assertEquals( expected: null, u.getCredit(),  message: "Expected = null, Actual= "+ u.getCredit());

    },  message: "Shoud throw Exception: IndexOutOfBoundsException");


}
```

Caso de teste 5 (testAddCreditAmountZero)

```
/*
Teste com IDUser=1 e amount=0
 */
@Test
public void testAddCreditAmountZero() {
    //Utilizador em que se adiciona créditos
    User u = brs.getUsers().get(1);

    brs.addCredit( idUser: 1,  amount: 0);

    //O crédito foi adicionado com sucesso
    assertEquals( expected: 0, u.getCredit(),  message: "Expected =1 , Actual = " + u.getCredit());
}
```

Metódo registerUser

Tabela ECP

registerUser		
Critério	Classe de Equivalência Válida	Classe de Equivalência Inválida
Nº inputs	3	≠3
Tipo de entradas	IDUser - int, name - string, rentalProgram - int	IDUser ≠ int, name = null, rentalProgram ≠ int
Valor especifico	IDUser >= 0, name ≠ null, rentalProgram = 1 ou 2	IDUser < 0, rentalProgram ≠ 1 ou 2
	Casos de Teste	
	IDUser=1, name="jose", rentalProgram=1	
	IDUser=1,name=, rentalProgram=1	
	IDUser =1 , name="jose", rentalProgram=1 2	
	IDUser ="x" , name="jose", rentalProgram=1	
	IDUser =1 , name=, rentalProgram=1 2	
	IDUser =1 , name="jose", rentalProgram="x"	
	IDUser =-1 , name="jose", rentalProgram=1	
	IDUser =1 , name="jose", rentalProgram=3	

Tabela BVA

	registerUser
Cenários	
1	IdUser>=0 , name !=null , rentalProgram=1
2	IdUser<0 , name !=null , rentalProgram=1
3	IdUser>=0 , name =null , rentalProgram=1
4	IdUser>=0 , name !=null , rentalProgram=0
5	IdUser>=0 , name !=null , rentalProgram=1
6	IdUser>=0 , name !=null , rentalProgram=2
7	IdUser>=0 , name !=null , rentalProgram=3
8	

Casos de Teste	Input			Output
	IDUser	**name**	**rentalProgram**	
1	1	!null	1	Cria utilizador
2	-1	!null	1	Exception
3	0	!null	1	Cria utilizador
4	1	null	1	Exception
5	1	!null	0	Exception
6	1	!null	1	Cria utilizador
7	1	!null	2	Cria utilizador
8	1	!null	3	Exception

Caso de teste

Caso de teste 1 (testRegisterUserUSERJAEXISTE)

```java
/*
Teste Utilizador já existe COM IDUser=1 e rentalProgram=2
 */
    @Test
    public void testRegisterUserUSERJAEXISTE() {
        //Verificar se a excepção é lançada
        assertThrows(UserAlreadyExists.class, () -> {
            brs.registerUser( IDUser: 1,  name: "Gabriel",  rentalProgram: 2);
        },  message: "Should Throw Exception: UserAlreadyExists"); //O teste é válido

    }
```

Caso de teste 2 (testRegisterUserIDINVALIDO)

```java
/*
Teste IDUser inválido
 */
    @Test
    public void testRegisterUserIDINVALIDO() {
        //Verificar se a excepção é lançada
        assertThrows(IndexOutOfBoundsException.class, () -> {

            User u = brs.getUsers().get(-1);

            brs.registerUser( IDUser: -1,  name: "Nuno",  rentalProgram: 2);

            assertEquals( expected: null, u.getIDUser(),  message: "Expected = null, Actual= "+ u.getIDUser());

        },  message: "Shoud throw Exception: IndexOutOfBoundsException");


    }
```

Caso de teste 3 (testRegisterUserExist)

```java
    // REGISTER USER


    /*
Teste para tentar registar utilizador que já existe COM IDUser=0 e rentalProgram=1
 */
    @Test
    public void testRegisterUserExist() {

        //Excepção é lançada
        assertThrows(UserAlreadyExists.class, () -> {

            brs.registerUser( IDUser: 0,  name: "Rafael",  rentalProgram: 1);

            //O teste é válido
        },  message: "Should Throw Exception: UserAlreadyExists");

    }
```

Caso de teste 4 (testRegisterUserJAEXISTECOMZERO)

```
/*
Teste utilizador com nome=null
 */
    @Test
    public void testRegisterUserJAEXISTECOMZERO() {
        //Verificar se a excepção é lançada
        assertThrows(UserAlreadyExists.class, () -> {
            brs.registerUser( IDUser: 0, name: null, rentalProgram: 1);
        }, message: "Should Throw Exception: UserAlreadyExists"); //O teste é válido

    }
```

Caso de teste 5 (testRegisterUserRENTALZERO)

```
    /*
Teste utilizador com rentalProgram=0 (só deveria aceitar rentalProgram 1 e 2)
mas aceita tudo o que seja
 */
    @Test
    public void testRegisterUserRENTALZERO() throws UserAlreadyExists {
        //Verificar se a excepção é lançada
            brs.registerUser( IDUser: 2, name: "Rafa", rentalProgram: 0);

    }
```

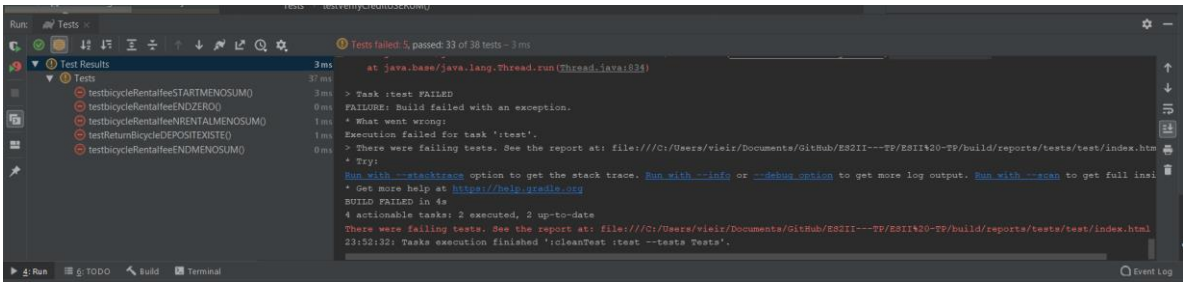
Caso de teste 6/7 (testRegisterUserValido)

```
/*
Teste para registar utilizador que ainda não existe
(TABELA BVA TEM IDUser=1 e rentalProgram=1 mas aqui foi necessário criar 2 utilizadores)
 */
@Test
public void testRegisterUserValido() throws UserAlreadyExists{

    brs.registerUser( IDUser: 2, name: "Nuno", rentalProgram: 2);

    //O teste é válido
    assertEquals( expected: 2, brs.getUsers().get(2).getIDUser());

}
```

Caso de teste 8 (testRegisterUserRENTALRES)

```
    /*
Teste utilizador com rentalProgram=3 (só deveria aceitar rentalProgram 1 e 2)
mas aceita tudo o que seja
*/
    @Test
    public void testRegisterUserRENTALTRES() throws UserAlreadyExists {
        //Verificar se a excepção é lançada
        brs.registerUser( IDUser: 2,  name: "Rafa",  rentalProgram: 3);

    }
```

Teste Gerais



Conclusão

Este projeto vem dar resposta a uma solicitação de especificação de casos de testes referente a um sistema de aluguer de bicicletas usando as técnicas *Equivalence Class Partitioning* e *Boundary Value Analysis,* descrição de *test inputs, execution conditions* e *expected outputs,* assegurando que os casos de teste cobrem *Valid equivalence classes* e *Invalid equivalence classes.*

Apesar das dificuldades iniciais na interpretação do enunciado, na configuração da *framework*: *JUnit* 5.1 e no uso das técnicas de *ECP* e de *BVA*, todos os requisitos do projeto foram possíveis de concluir por completo e com sucesso.

Anexos

Link para repositório Git: https://github.com/vieirarafael8/ES2II---TP/invitations