Relatório Part.1

Engenharia de Software II

Gabriel Pereira 8150115

Nuno Josefino 8150189

Rafael Vieira 8150448



# Índice

[Índice 2](#_Toc530441511)

[Introdução 6](#_Toc530441512)

[Metódo getBicycle 8](#_Toc530441513)

[Tabela ECP 8](#_Toc530441514)

[Tabela BVA 8](#_Toc530441515)

[Casos de teste 9](#_Toc530441516)

[Caso de teste 0 (testGetBicycleUSERMENOSUM) 9](#_Toc530441517)

[Caso de teste 1/3/6 (testGetBicycleEXISTEBICICLETA) 10](#_Toc530441518)

[Caso de teste 4 (testGetBicucleSEMBICICLETASDISP) 10](#_Toc530441519)

[Caso de teste 4 (testGetBicycleNAOEXISTEDEPOSITO) 11](#_Toc530441520)

[Caso de teste 5 (testGetBicycleSTARTTIMEMENOSUM) 11](#_Toc530441521)

[Caso de teste 7 (testGetBicycleSTARTUM) 12](#_Toc530441522)

[Metódo returnBikycle 12](#_Toc530441523)

[Tabela ECP 12](#_Toc530441524)

[Tabela BVA 12](#_Toc530441525)

[Caso de teste 1 (testReturnBicycleRETORNASALDOUSERUM) 14](#_Toc530441526)

[Caso de teste 2 (testReturnBicycleRETORNASALDOUSEROZERO) 14](#_Toc530441527)

[Caso de teste 3 (testReturnBicycleUSERNAOEXISTE) 14](#_Toc530441528)

[Caso de teste 1/4/6 (testReturnBicycleDEPOSITEXISTE) 15](#_Toc530441530)

[Caso de teste 5 (testReturnBicycleDEPOSITNAOEXISTE) 15](#_Toc530441532)

[Caso de teste 8 (testReturnBicycleSEMLUGARESLIVRES) 15](#_Toc530441534)

[Metódo bicycleRentalFee 16](#_Toc530441536)

[Tabela ECP 16](#_Toc530441537)

[Tabela BVA 16](#_Toc530441538)

[Caso de teste 17](#_Toc530441539)

[Caso de teste 1 – testbicycleRentalfeeRENTALMENOS1 17](#_Toc530441540)

[Caso de teste 2 – testbicycleRentalfeeRENTALZERO 17](#_Toc530441541)

[Caso teste 3/8/14 – testbicycleRentalfeeRENTALUM 17](#_Toc530441542)

[Caso teste 4 – testbicycleRentalfeeRENTALDOIS 18](#_Toc530441543)

[Caso teste 5 – testbicycleRentalfeeRENTALTRES 18](#_Toc530441544)

[Caso de teste 6 – testbicycleRentalfeeSTARTMENOSUM 18](#_Toc530441545)

[Caso de teste 7 – testbicycleRentalfeeSTARTZERO 18](#_Toc530441546)

[Caso de teste 9 – testbicycleRentalfeeNRENTALMENOSUM 19](#_Toc530441547)

[Caso de teste 10 – testbicycleRentalfeeNRENTALZERO 19](#_Toc530441548)

[Caso de teste 11 – testbicycleRentalfeeNRENTALUM 19](#_Toc530441549)

[Caso de teste 12 – testbicycleRentalfeeENDMENOSUM 19](#_Toc530441550)

[Caso de teste 13 – testbicycleRentalfeeENDZERO 20](#_Toc530441551)

[Metódo verifyCredit 20](#_Toc530441553)

[Tabela ECP 20](#_Toc530441554)

[Tabela BVA 20](#_Toc530441555)

[Caso de teste 1 (testVerifyCreditUSERMENOSUM) 20](#_Toc530441556)

[Caso de teste 2 (testVerifyCreditUSERZERO) 21](#_Toc530441557)

[Caso de teste 3 (testVerifyCreditUSERUM) 21](#_Toc530441558)

[Metódo addCredit 22](#_Toc530441559)

[Tabela ECP 22](#_Toc530441560)

[Tabela BVA 22](#_Toc530441561)

[Caso de teste 1/4 testAddCredit1 23](#_Toc530441562)

[Caso de teste 2 (testAddCredit2) 23](#_Toc530441563)

[Caso de teste 3 (testAddCreditMenosUM) 23](#_Toc530441564)

[Caso de teste 5 (testAddCreditAmountZero) 24](#_Toc530441565)

[Metódo registerUser 24](#_Toc530441566)

[Tabela ECP 24](#_Toc530441567)

[Tabela BVA 24](#_Toc530441568)

[Caso de teste 25](#_Toc530441569)

[Caso de teste 1 ( testRegisterUserUSERJAEXISTE) 25](#_Toc530441570)

[Caso de teste 2 (testRegisterUserIDINVALIDO) 25](#_Toc530441571)

[Caso de teste 3 (testRegisterUserExist) 26](#_Toc530441572)

[Caso de teste 4 (testRegisterUserJAEXISTECOMZERO) 26](#_Toc530441573)

[Caso de teste 5 (testRegisterUserRENTALZERO) 26](#_Toc530441574)

[Caso de teste 6/7 (testRegisterUserValido) 27](#_Toc530441575)

[Caso de teste 8 (testRegisterUserRENTALRES) 27](#_Toc530441576)

[Teste Gerais 28](#_Toc530441577)

[Conclusão 28](#_Toc530441578)

# 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*.

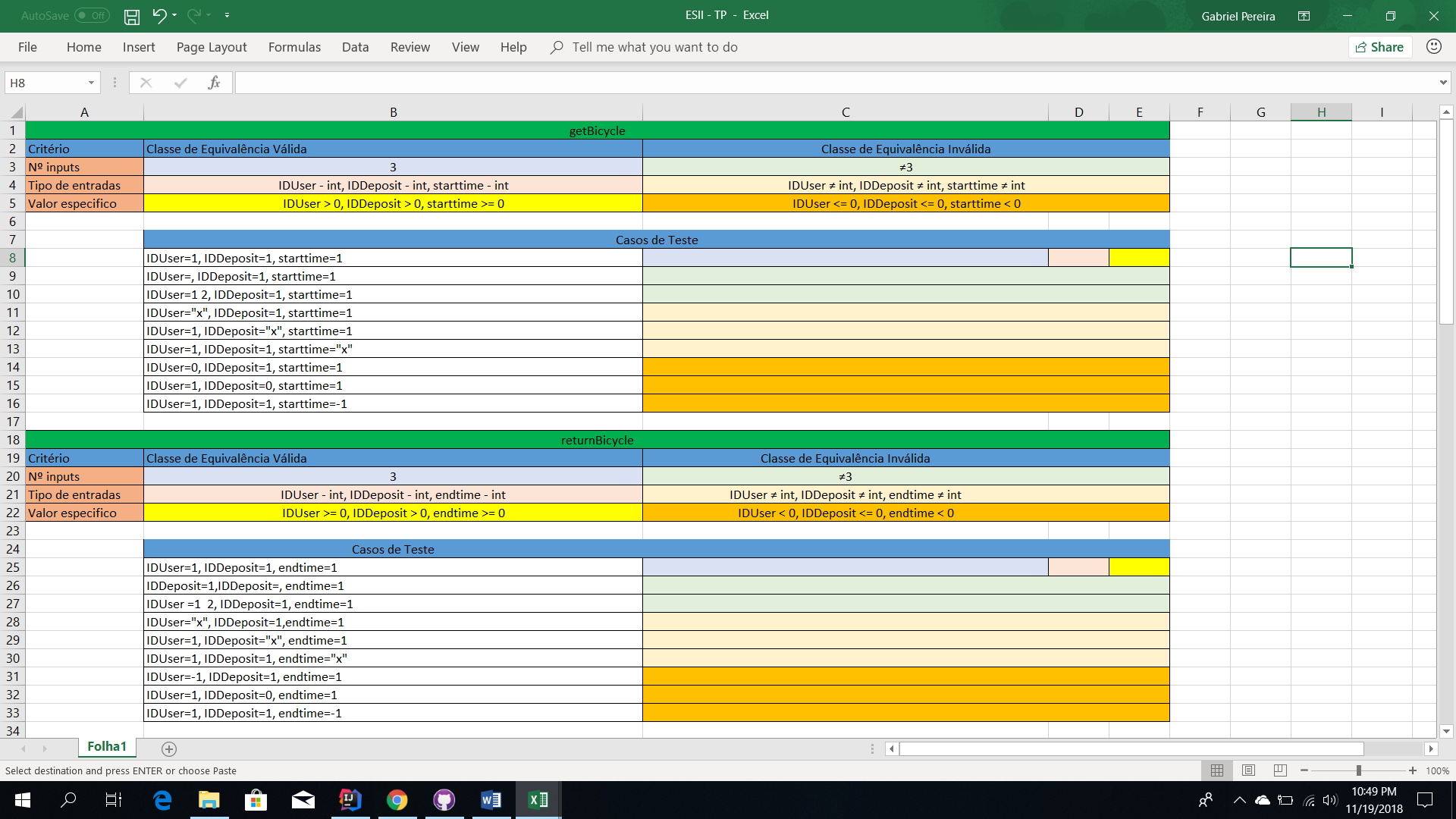
Este trabalho tem como objetivo a especificação de casos de teste utilizando as técnicas *Equivalence Class Partitioning* e *Boundary Value Analysis* no nível de testes unitários especificando casos de teste para cada método. Deverá ser descrito igualmente *test inputs*, *execution conditions* e *expected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

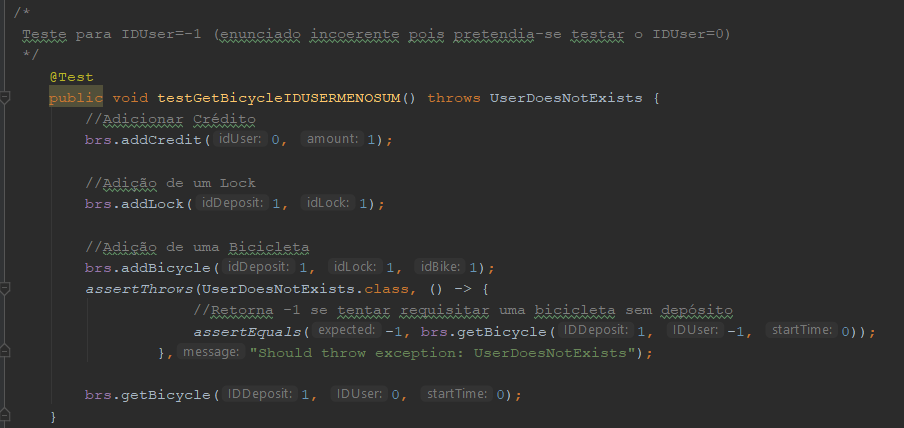


## Tabela BVA

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 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 |

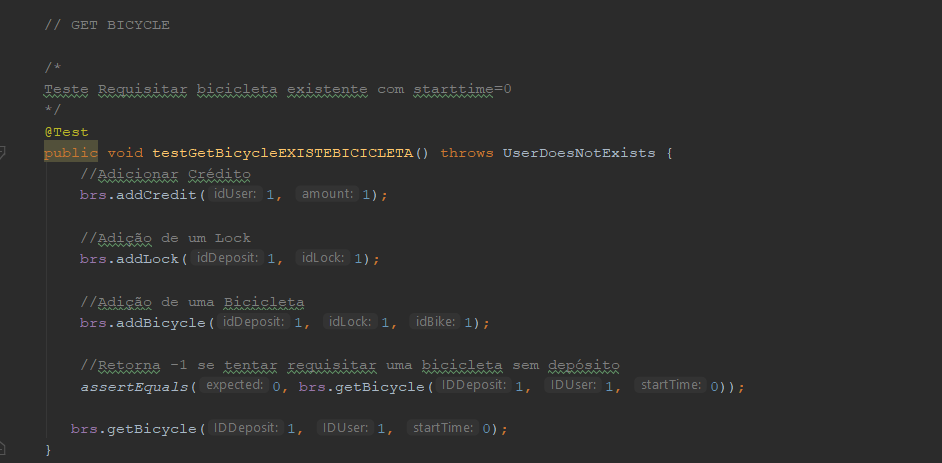
## Casos de teste

## Caso de teste 0 (testGetBicycleUSERMENOSUM)

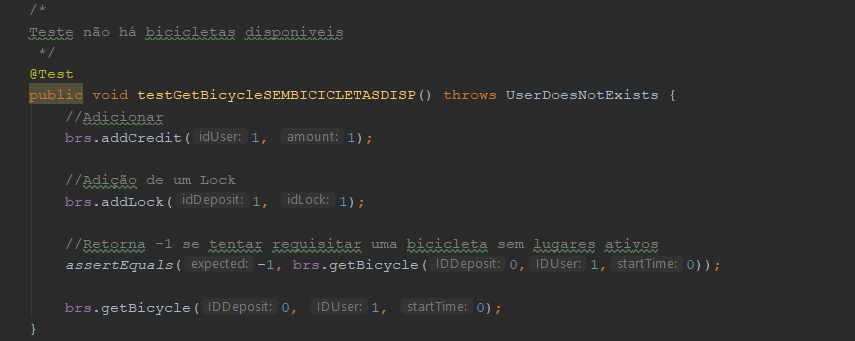


Caso de teste extra criado devido á falta de validações no IDUser que permite que este seja 0 (o que não deveria acontecer visto ser suposto que IDUser > 0) e para que sejam testadas todas as fronteiras foi necessário adicionar o teste para IDUser = -1.

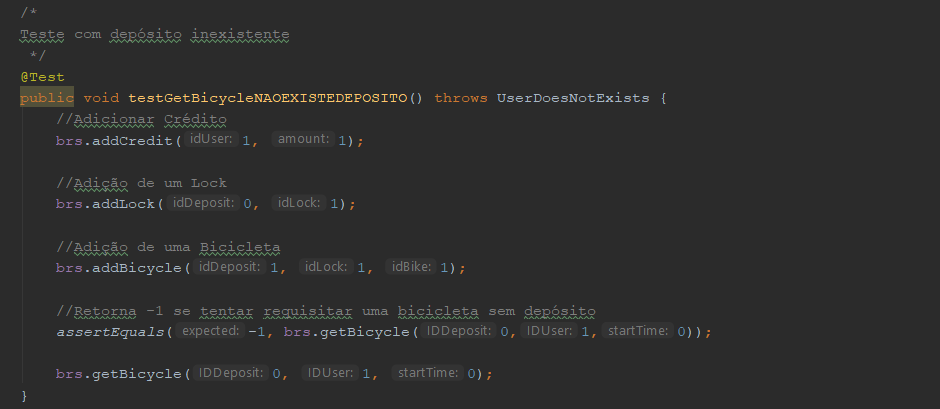
## Caso de teste 1/3/6 (testGetBicycleEXISTEBICICLETA)



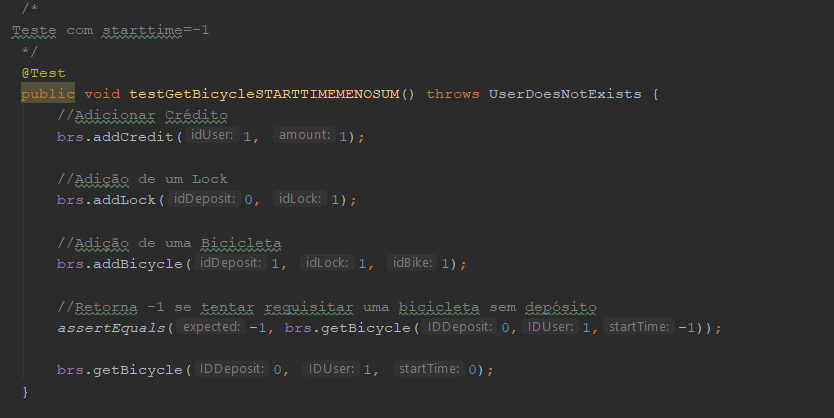
## Caso de teste 4 (testGetBicucleSEMBICICLETASDISP)



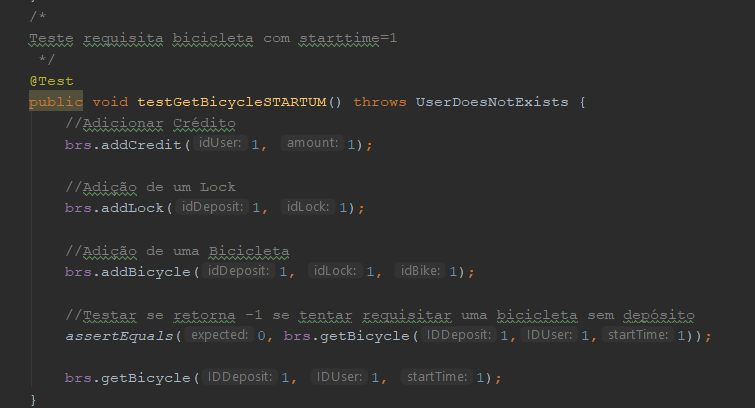
## Caso de teste 4 (testGetBicycleNAOEXISTEDEPOSITO)



## Caso de teste 5 (testGetBicycleSTARTTIMEMENOSUM)

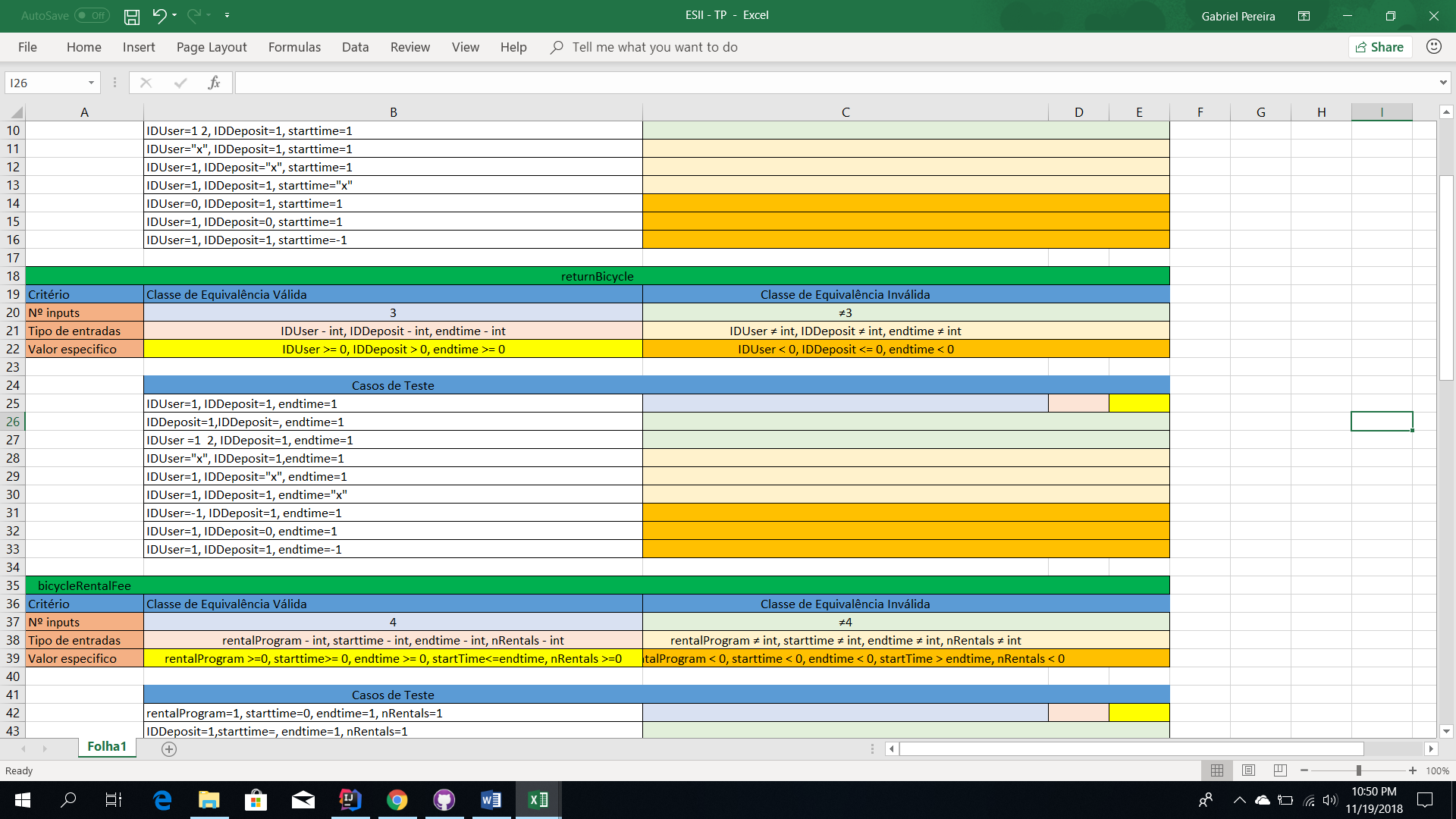


## Caso de teste 7 (testGetBicycleSTARTUM)



# Metódo returnBikycle

## Tabela ECP



## 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)

## Caso de teste 2 (testReturnBicycleRETORNASALDOUSEROZERO)

## Caso de teste 3 (testReturnBicycleUSERNAOEXISTE)

## 

## Caso de teste 1/4/6 (testReturnBicycleDEPOSITEXISTE)

## 

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)

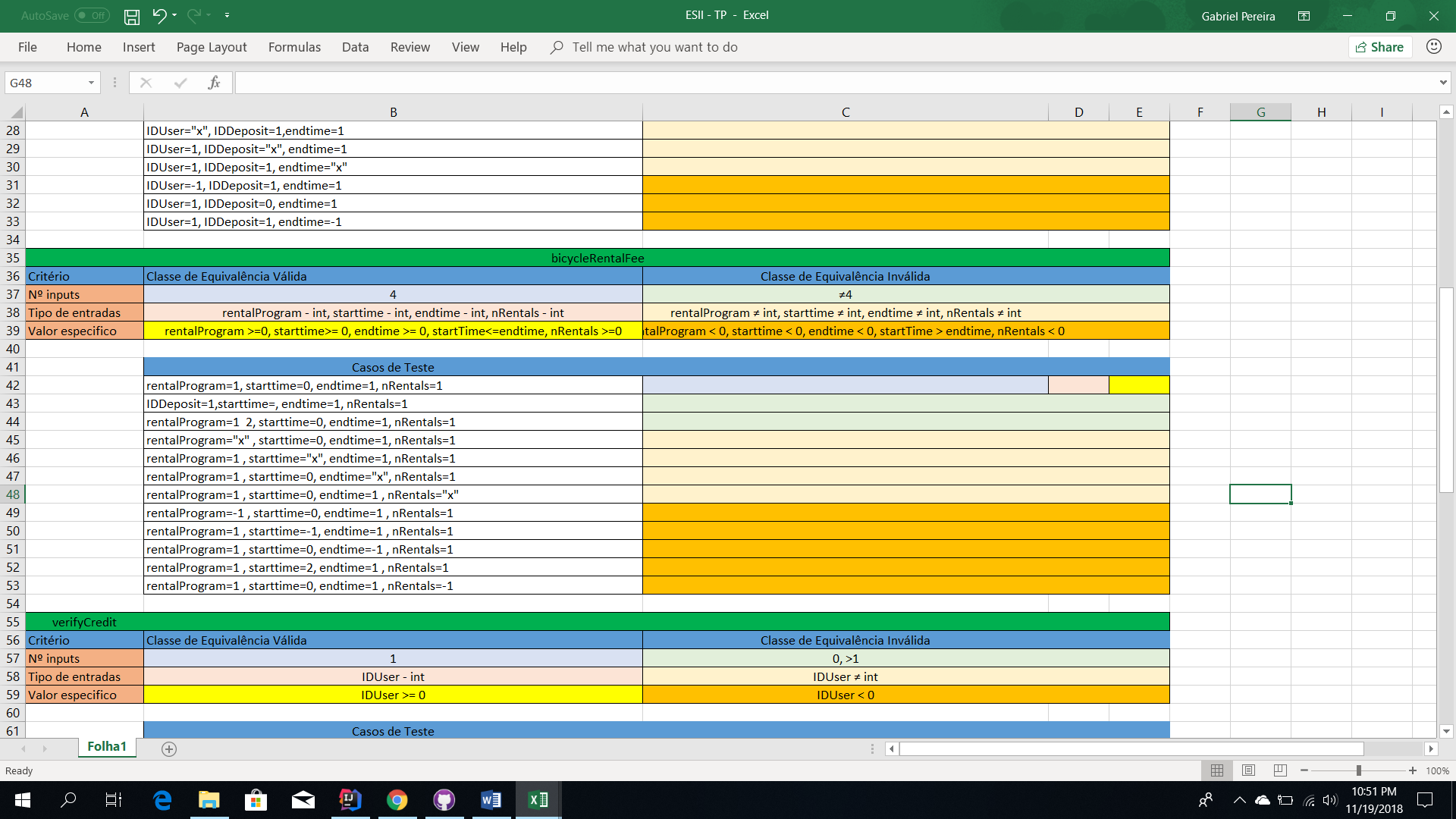
## 

## Caso de teste 8 (testReturnBicycleSEMLUGARESLIVRES)

## 

# Metódo bicycleRentalFee

## Tabela ECP

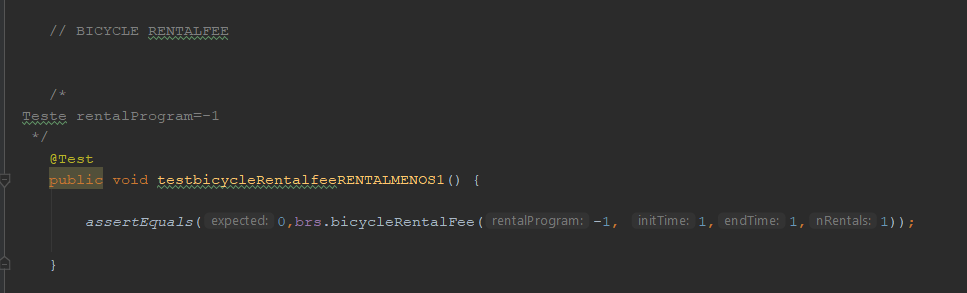


## Tabela BVA

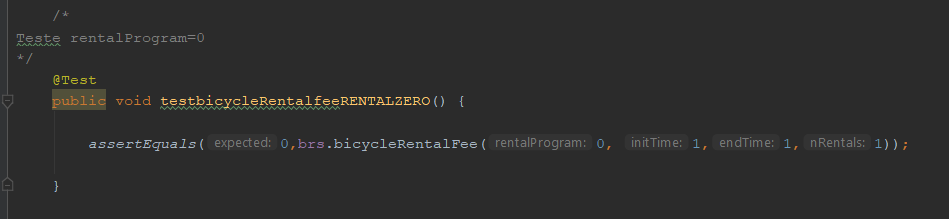
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Método | bicycleRentalFee | | | | |
|  |  |  |  |  |  |
| Cenários |  |  |  |  |  |
| 1 | rentalProgram >=0, starttime>=0, endtime>=0, starttime <=endtime, nRentals>=0 | | | | |
| 2 | rentalProgram <0, starttime>=0, endtime>=0, starttime <=endtime, 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

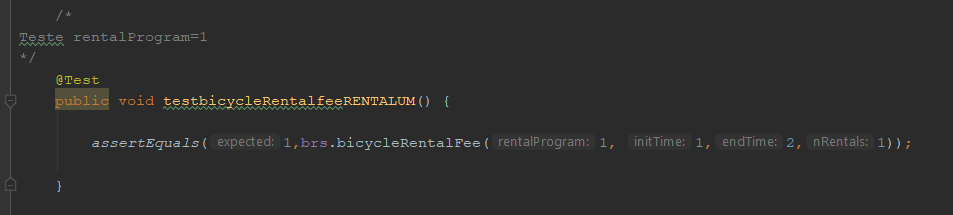
## Caso de teste 1 – testbicycleRentalfeeRENTALMENOS1



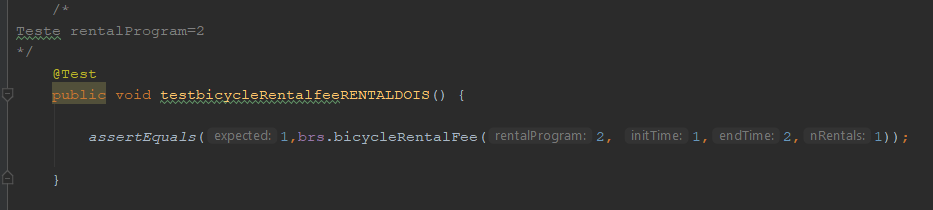
## Caso de teste 2 – testbicycleRentalfeeRENTALZERO



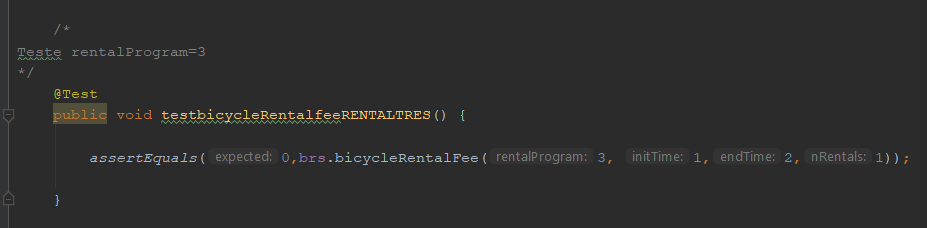
## Caso teste 3/8/14 – testbicycleRentalfeeRENTALUM



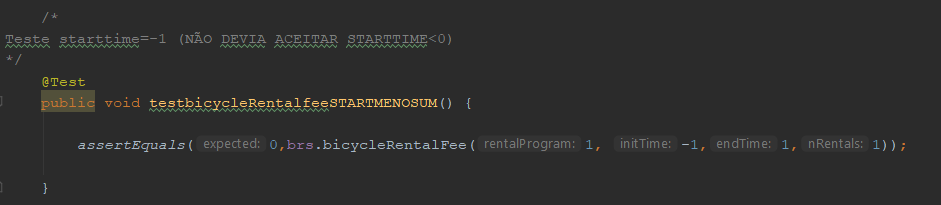
## Caso teste 4 – testbicycleRentalfeeRENTALDOIS



## Caso teste 5 – testbicycleRentalfeeRENTALTRES

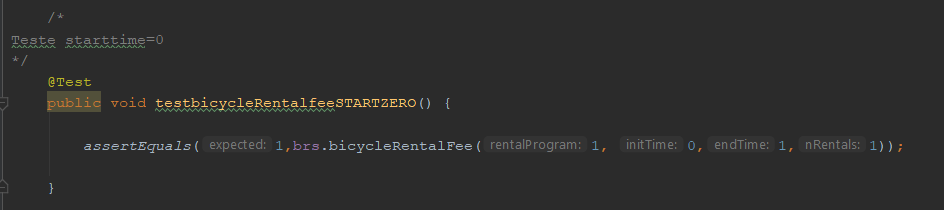


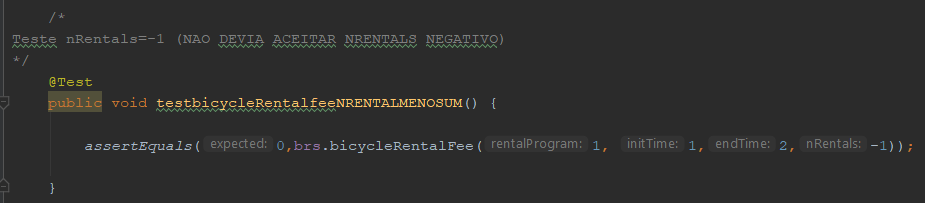
## Caso de teste 6 – testbicycleRentalfeeSTARTMENOSUM



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.

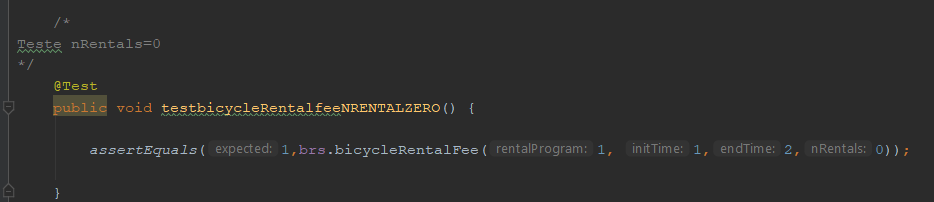
## Caso de teste 7 – testbicycleRentalfeeSTARTZERO



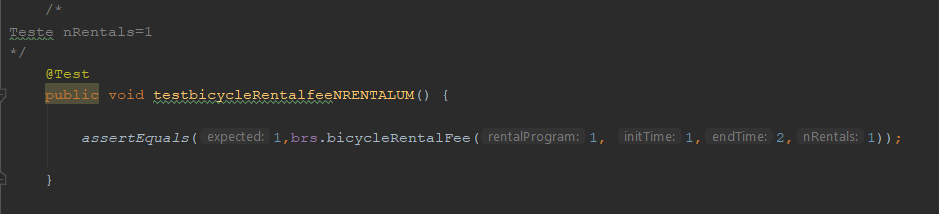
Caso de teste 9 – testbicycleRentalfeeNRENTALMENOSUM 

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.

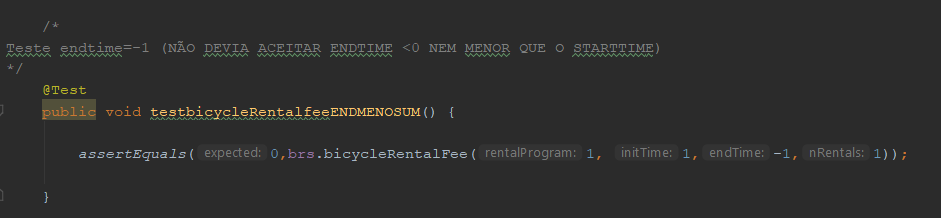
## Caso de teste 10 – testbicycleRentalfeeNRENTALZERO



## Caso de teste 11 – testbicycleRentalfeeNRENTALUM

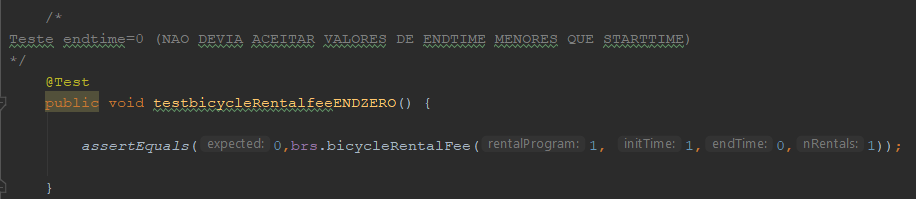


## Caso de teste 12 – testbicycleRentalfeeENDMENOSUM



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.

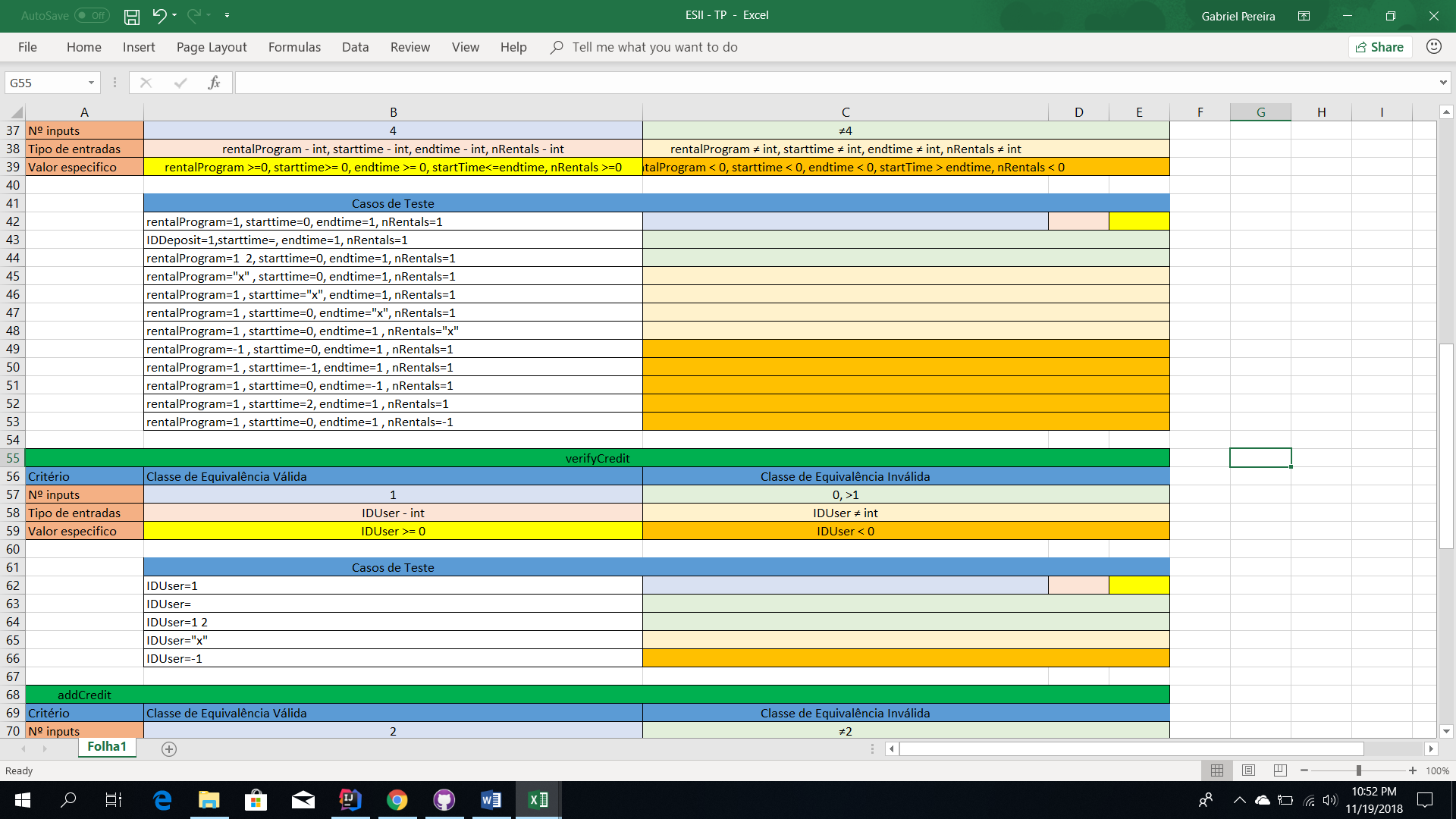
## Caso de teste 13 – testbicycleRentalfeeENDZERO



## 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

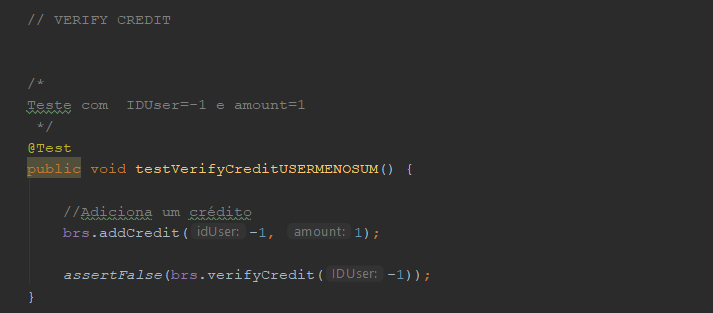
## Tabela ECP



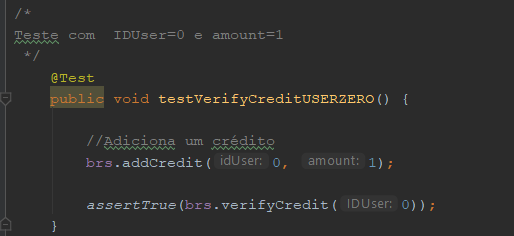
## Tabela BVA

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 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 |

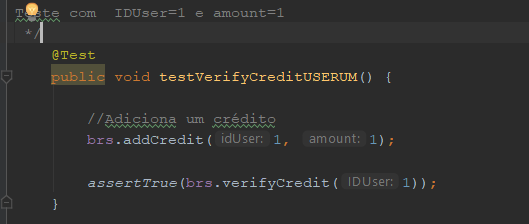
## Caso de teste 1 (testVerifyCreditUSERMENOSUM)



## Caso de teste 2 (testVerifyCreditUSERZERO)

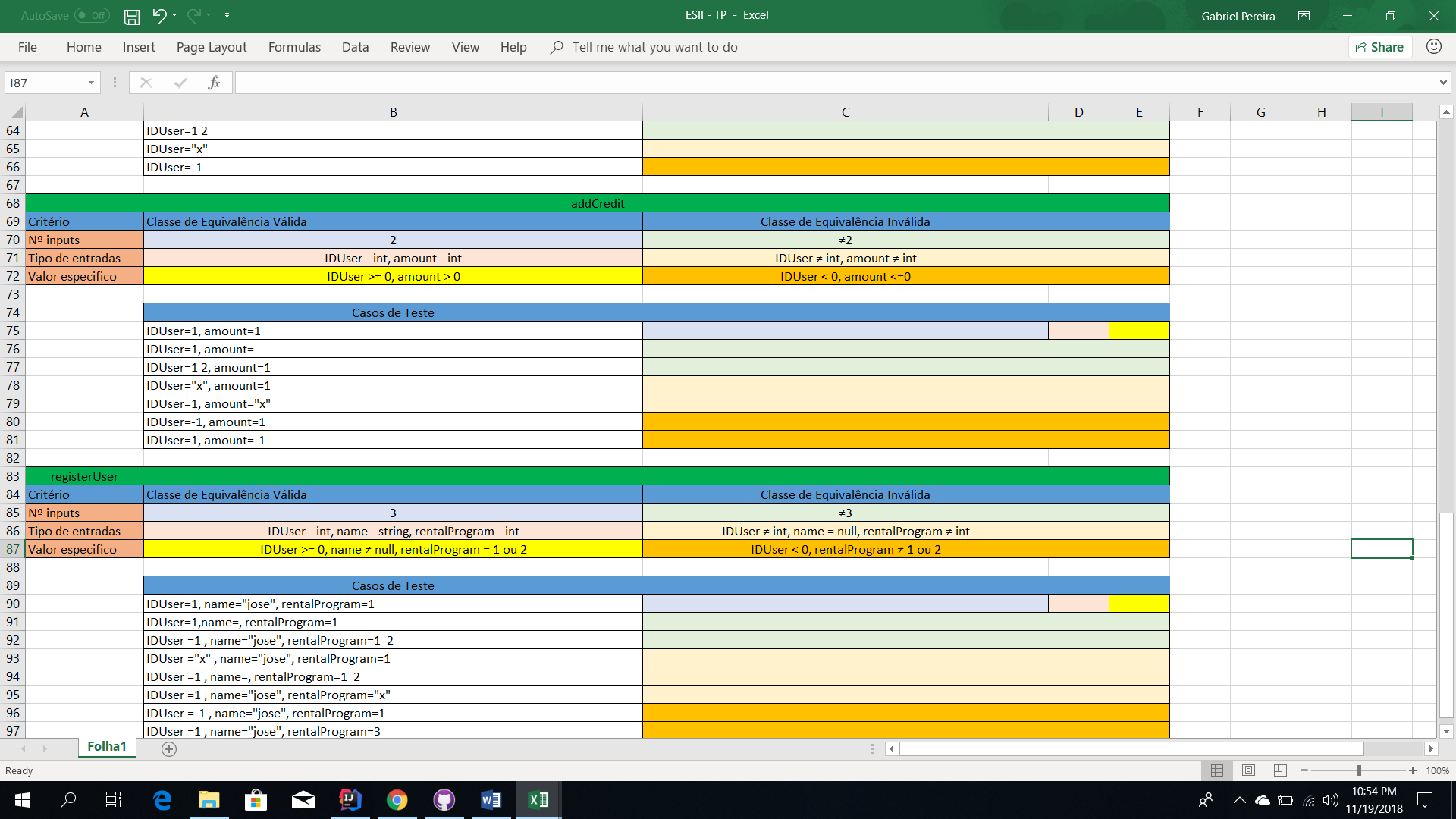


## Caso de teste 3 (testVerifyCreditUSERUM)



# Metódo addCredit

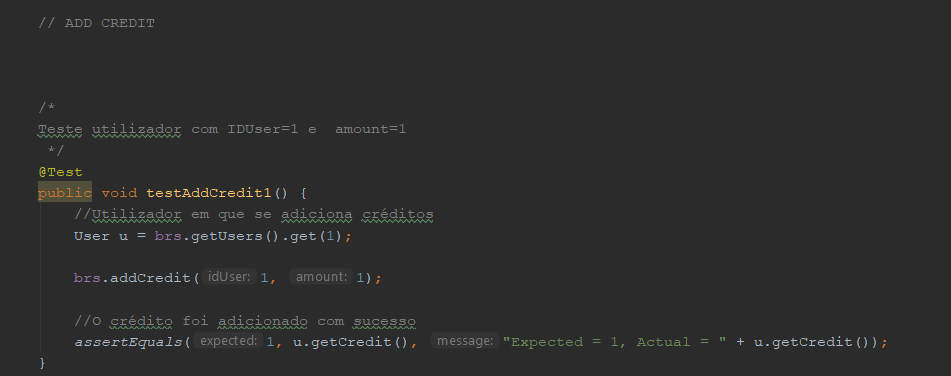
## Tabela ECP



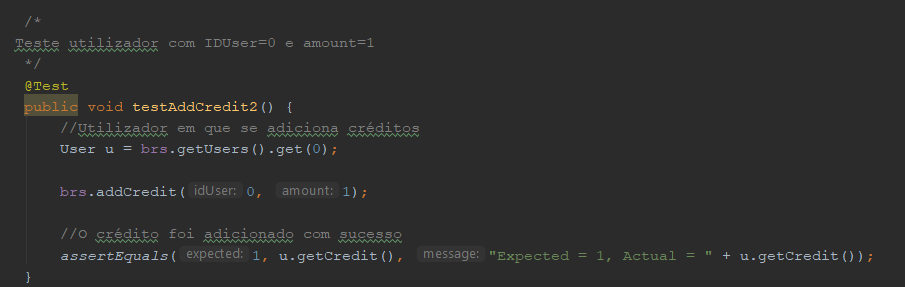
## Tabela BVA

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 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



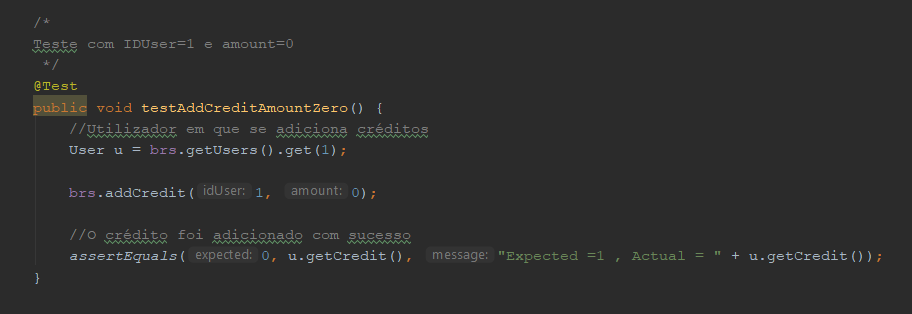
## Caso de teste 2 (testAddCredit2)



## Caso de teste 3 (testAddCreditMenosUM)

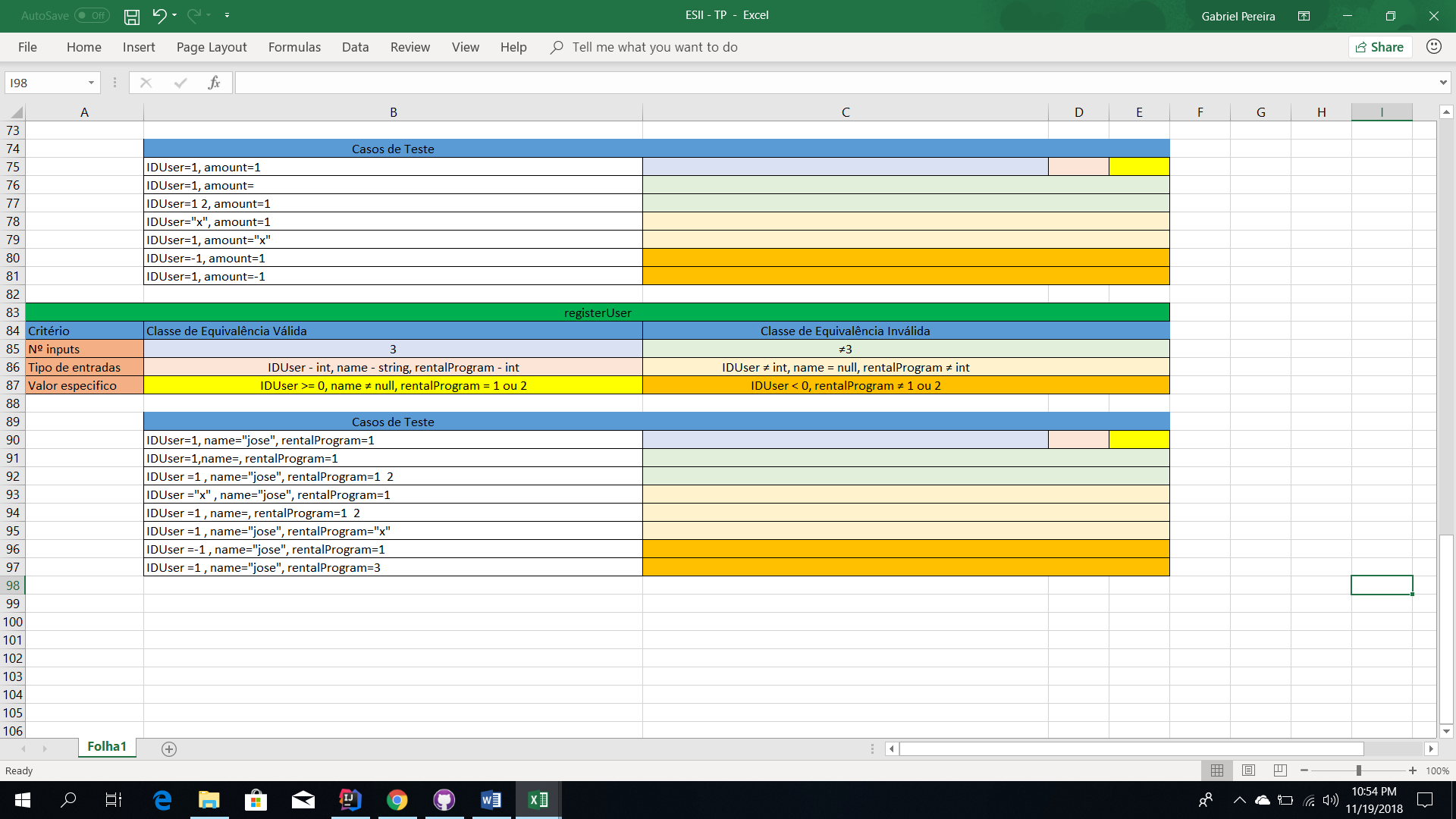


## Caso de teste 5 (testAddCreditAmountZero)



# Metódo registerUser

## Tabela ECP

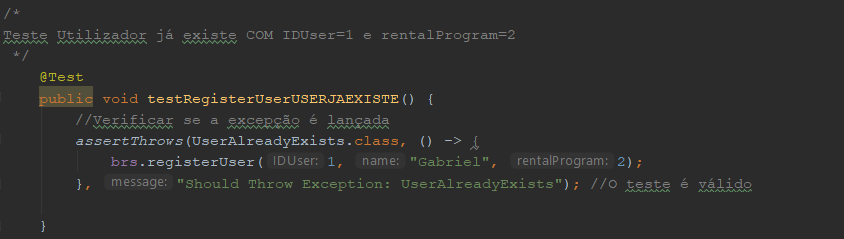


## Tabela BVA

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 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

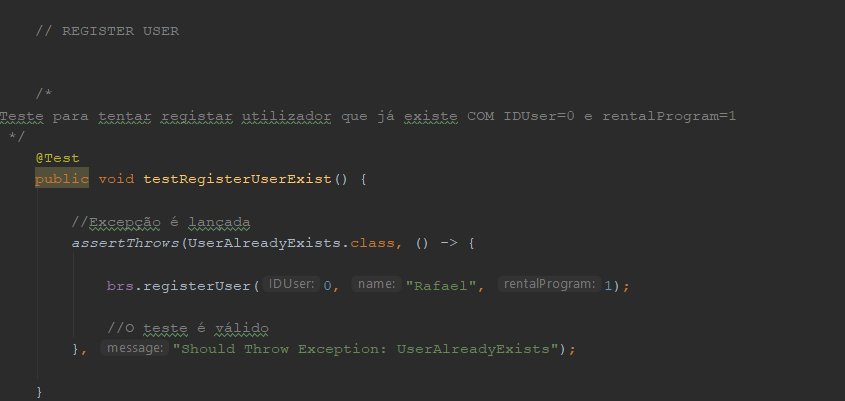
## Caso de teste 1 ( testRegisterUserUSERJAEXISTE)



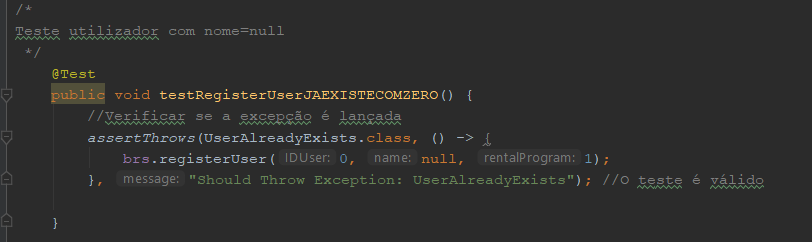
## Caso de teste 2 (testRegisterUserIDINVALIDO)



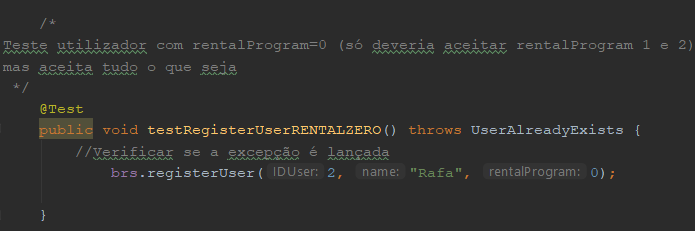
## Caso de teste 3 (testRegisterUserExist)



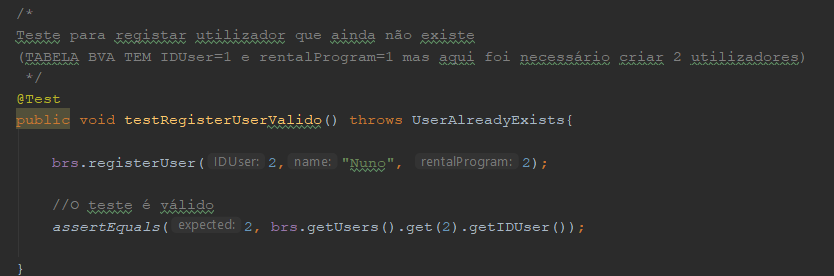
## Caso de teste 4 (testRegisterUserJAEXISTECOMZERO)



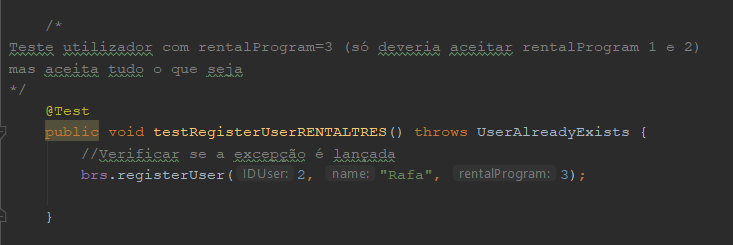
## Caso de teste 5 (testRegisterUserRENTALZERO)



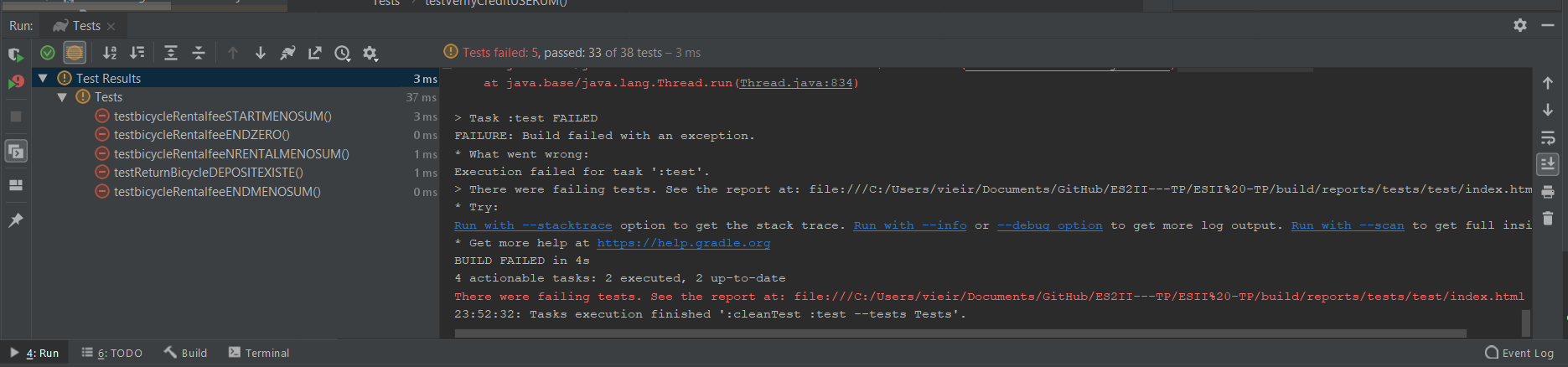
## Caso de teste 6/7 (testRegisterUserValido)



## Caso de teste 8 (testRegisterUserRENTALRES)



# 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