

INDIVIDUAL ASSESSMENT SSE3305-1: TESTING REPORT OF ELECTRICITY BILL CALCULATOR

Ma ZhiYuan

Table of Contents

Introduction	3
INDIVIDUAL ASSESSMENT SSE3305-1: TESTING REPORT OF ELECTRICITY BILL	
CALCULATOR	4
Schedule of Activities and Tasks	4
Test Analysis & Design.....	5
Global Test	5
Test Script	5
Partial test	7
Test in Residential	8
Test in Non-Residential	11
Test Implementation.....	14
Global Test	14
Partial Test.....	15
Summary report.....	19
Appendix I - Screenshots	22
Appendix II – Test Source code	24

Introduction

The aim of the report is to show the result of testing the given program, Electricity Bill Calculator, by using both black-box and white-box testing techniques and referring the requirement, source code and design information. This report will contain test strategy (the method will be used on testing program), test script (including test cases identified by using black-box/white-box technique), test class, and Coverage information. in addition, at the end of this document, it will give the information to justify why the solution which has been used in this report is the most efficient one and prove that the tested code is ready for deployment.

Keywords: Test strategy, Test design, Test Script, Test class, Coverage information

INDIVIDUAL ASSESSMENT SSE3305-1: TESTING REPORT OF ELECTRICITY BILL CALCULATOR

The aim of the report is to show the result of testing the given program, Electricity Bill Calculator, by using both black-box and white-box testing techniques and referring the requirement, source code and design information. This report will contain test strategy (the method will be used on testing program), test script (including test cases identified by using black-box/white-box technique), test class, and Coverage information. In addition, at the end of this document, it will give the information to justify why the solution which has been used in this report is the most efficient one and prove that the tested code is ready for deployment.

Schedule of Activities and Tasks

The following table shows the set of tasks necessary to prepare for and perform testing for Electricity Bill Calculator.

No.	Task	Activity	Target Date
1.	Test Planning	Complete planning	28 May, 2021
2.	Test Analysis & Design	Prepare test script	29 May, 2021 – 31 May, 2021
3.	Test Implementation	Perform and record test	1 June, 2021 – 3 June, 2021
4.	Summary report	Summarize test result	4 June, 2021 – 5 June, 2021
5.	Submit Assessment	Submit to <i>Putra blast</i>	6 June, 2021 (DDL in 7 June)

Figure 1: Schedule Table of Activities and Tasks

Test Analysis & Design

From the perspective of the overall System Flowchart (individual assessment specification page number 5), there are total three main branches running on inside the program, which include their own modules with functions.

$$\text{Independent Path} = e - n + 2$$

Known from diagram: node = 10, edge = 11.

Therefore, Independent path = 3.

Considering the complexity of the system to be tested; for example, when TYPE OF RESIDENTIAL = "NON-RESIDENTIAL", the option TYPE OF SECTOR will appear. In addition, each branch has its own modules, and the functions of the modules would not interact with each other. Therefore, the system would not be tested as a whole at once, and the testing direction will be divided into partial sections. Then, the testing process will be performed on global aspect and partial aspect as each branch independently and orderly.

Global Test

The global test mainly focuses on whether there are problems in the main system aspect, for example, whether the system will call or run the module expected after inputting the attributes.

Test Script

Testing Path:

According to the explanation above,

Path 1: back

Path 2: not back && residential

Path 3: not back && non-residential

The test script based on the identified independent path.

TC#	Path	Back (Boolean)	Type of Residential	Electricity Consumption	Tariff Code	Type of Sector	Expected Output
1	1	TRUE	-	-	-	-	(Main page)
2	2	FALSE	Residential	100	-	-	allow to Residential result page
3	2	FALSE	Residential	-	-	-	Consumption invalid
4	2	FALSE	Residential	-100	-	-	Consumption invalid
5	3	FALSE	Non-Residential	100	-	-	Tariff Code and Type of Sector invalid
6	3	FALSE	Non-Residential	100	Commercial	-	Tariff Code Invalid
7	3	FALSE	Non-Residential	100	-	Tariff B: Low Voltage Commercial	Type of Sector Invalid
8	3	FALSE	Non-Residential	100	Commercial	Tariff B: Low Voltage Commercial	allow to Non-Residential result page
9	3	FALSE	Non-Residential	-	Commercial	Tariff B: Low Voltage Commercial	Consumption Invalid
10	3	FALSE	Non-Residential	-100	Commercial	Tariff B: Low Voltage Commercial	Consumption Invalid

Figure 2: Test case from Flowchart

But it also can be test using Decision Table.

	T1	T2	T3	T4	T5	T6
State of Residential	R	N	N	N	-	-
Type of Sector	default	void	-	Commercial	-	-
Tariff Code	default	-	void	Tariff B	-	-
Electricity Consumption	≥ 0	-	-	≥ 0	< 0	void
Calculate the bill of Residential	X					
Calculate the bill of non-Residential				X		
Display Result	X			X		
Invalid Input		X	X		X	X

Figure 3: Decision Table of Global test

The test script based on the Decision Table.

- TC1: State of Residential = 'Residential' and Electricity Consumption ≥ 0 and the rest don't care;
- TC2: State of Residential = 'non-Residential' and Type of Sector is void and the rest don't care;
- TC3: State of Residential = 'non-Residential' and Tariff Code is void and the rest don't care;
- TC4: State of Residential = 'non-Residential' and Type of Sector is Commercial and Tariff Code is Tariff B;

- TC5: Electricity Consumption < 0 and the rest don't care
- TC6: Electricity Consumption is void and the rest don't care

TC#	Input				Expected Output
	Type of Residential	Type of Sector	Tariff Code	Electricity Consumption	
1	Residential	-		200	Calculate the bill of Residential and Display Result
2	Non-Residential	void	-	200	Invalid Input
3	Non-Residential	-	void	200	Invalid Input
4	Non-Residential	Commercial	Tariff B	200	Calculate the bill of non-Residential and Display Result
5	-	-		-200	Invalid Input
6	-	-		void	Invalid Input

Figure 4: Test cases of Decision Table

From what we got here, obviously, decision table is more simply and efficient then the path testing because comparing to the test case basing on path, the one basing on decision table has lesser cases.

The test technique going to be used is ***Decision table***.

However, there is still a problem that controller of this program cannot be test by JUnit directly. In order to finish the test and cover the program as much as possible, the global test will be divided into 2 parts. The first part is use JUnit to test the function *residential* and *nonresidential* basing on the input and output. The second is test program from UI operation to see whether the controller can get input and transfer into correct method or cannot. Then compare these 2 parts together to see if the results are same or not.

Partial test

The Partial test mainly focuses on the modules and their functions.

Just as the accordion from test path above, path 1 which can be done in global testing; Therefore, path 2 and path 3 as 2 branches will be tested in partial test.

Section#	Type	Module
S1	-	-
S2	Residential	<i>calcNoDiscBilR, calcTax</i>
S3	Non-Residential	<i>calcNoDiscBilNR, calcTaxNR</i>

Figure 5: Branches table

Test in Residential

When the user enters with Residential, the system is supposed to calculate the quote request amount as specified in assessment specification Figure 1 based on the electricity rate and usage inputted by the user.

calcNoDiscBilR

The role of this part in the program is to get the consumption (the only input) and calculate the residential Electricity Bill and details such as *First 200 kWh(1-200 kWh)*, *Next 100 kWh(201-300 kWh)*, *Next 300 kWh(301-600 kWh)*, *Next 300 kWh(601-900 kWh)*, and *Next 901 kWh* onwards.

$$\text{First 200 kwh} = (\text{consumption} * 21.80) / 100$$

$$\text{Next 100 kwh} = ((\text{consumption} - 200) * 33.40) / 100$$

$$\text{Next 300 kwh}_1 = ((\text{consumption} - 300) * 51.60) / 100$$

$$\text{Next 300 kwh}_1 = ((\text{consumption} - 600) * 54.60) / 100$$

$$\text{Next balance kwh} = ((\text{consumption} - 900) * 57.60) / 100$$

Figure 6: Formulas in calcNoDiscBilR

Black-Box Testing

Input Domain – Valid Partition:

TC 1: $0 \leq \text{consumption} \leq 200$

TC 2: $201 \leq \text{consumption} \leq 300$

TC 3: $301 \leq \text{consumption} \leq 600$

TC 4: $601 \leq \text{consumption} \leq 900$

TC 5: $901 \leq \text{consumption}$

Input Domain – Invalid Partition:

TC 6: $\text{consumption} < 0$

- Equivalence Partitioning

TC#	Input	Expected Output					
	Consumption	Electricity Bill	First 200 kWh	Next 100 kWh	Next 300 kWh	Next 300 kWh	Next 901 kWh
1	120	26.16	26.16	0	0	0	0
2	240	56.96	43.6	13.36	0	0	0
3	450	154.4	43.6	33.4	77.4	0	0
4	700	286.4	43.6	33.4	154.8	54.6	0
5	1000	452.7	43.6	33.4	154.8	163.8	57.1
6	-100	Invalid Consumption					

Figure 7: Test cases by Equivalence Partitioning of calcNoDiscBilR

- Boundary Value Analysis

TC#	Input	Expected Output					
	Consumption	Electricity Bill	First 200 kWh	Next 100 kWh	Next 300 kWh	Next 300 kWh	Next 901 kWh
6	-1	Invalid Input					
1	0	0	0	0	0	0	0
	1	0.22	0.22	0	0	0	0
	199	43.38	43.38	0	0	0	0
	200	43.6	43.6	0	0	0	0
2	201	43.93	43.6	0.33	0	0	0
	202	44.27	43.6	0.67	0	0	0
	299	76.67	43.6	33.07	0	0	0
	300	77	43.6	33.4	0	0	0
3	301	77.52	43.6	33.4	0.52	0	0
	302	78.03	43.6	33.4	1.03	0	0
	599	231.28	43.6	33.4	154.28	0	0
	600	231.8	43.6	33.4	154.8	0	0
4	601	232.35	43.6	33.4	154.8	0.55	0
	602	232.89	43.6	33.4	154.8	1.09	0
	899	395.05	43.6	33.4	154.8	163.25	0
	900	395.6	43.6	33.4	154.8	163.8	0
5	901	396.17	43.6	33.4	154.8	163.8	0.57
	902	396.74	43.6	33.4	154.8	163.8	1.14

Figure 8: Test cases by Boundary Value Analysis of calcNoDiscBilR

CalcTax

The role of this part in the program is to get the consumption, electricity bill (from calcNoDiscBilR) and disc (written in source code and always be 0), and calculate the tax bill.

$$\text{Tax Bill} = (\text{Electricity Bill} - 231.80) * 0.06$$

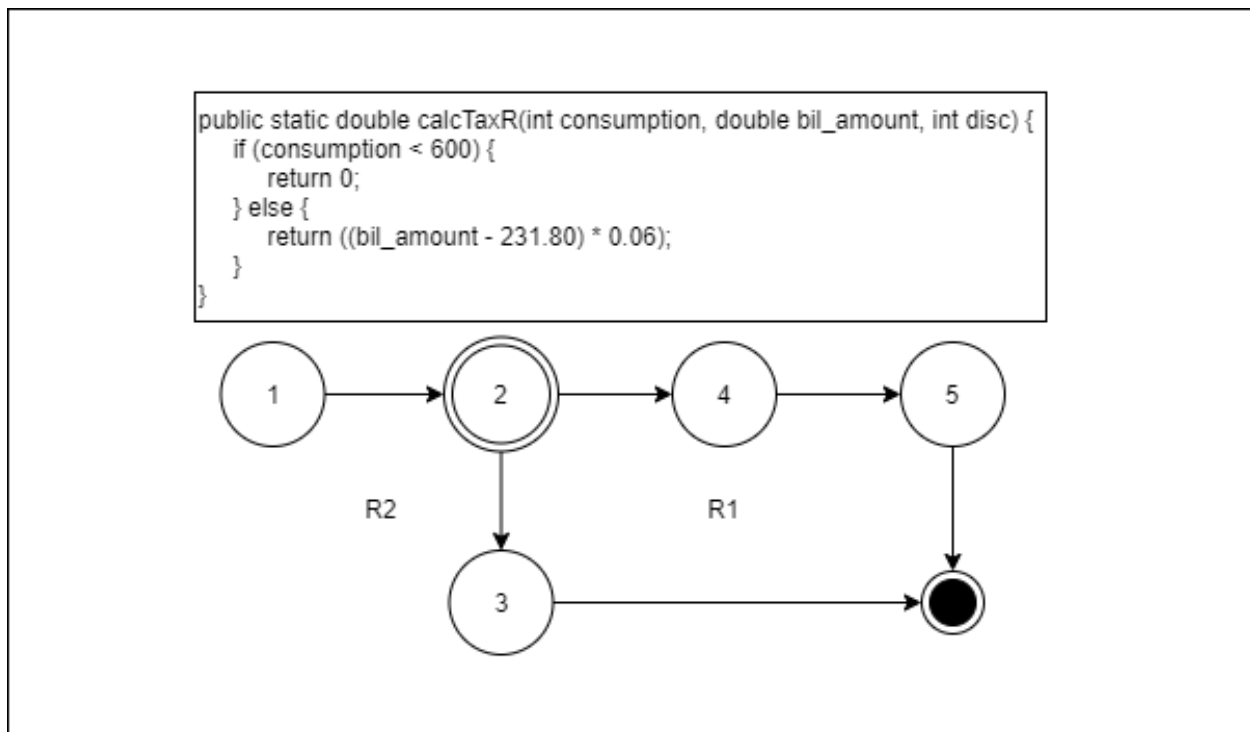
White-Box Testing

Figure 9: Flow Graph Notation of calcTax(R)

$$V(G) = \text{edges} - \text{nodes} + 2$$

$$V = 6 - 6 + 2 = 2$$

$$V(G) = P + 1$$

$$V = 1 + 1 = 2$$

Therefore, there should be 2 independence paths of this part of program.

Path 1: 1-2-4-5-end

Path 2: 1-2-3-end

TC#	Input		Expected Output
	Consumption	Electricity Bill	Tax
1	600	231.8	0
2	1000	452.7	13.25

Figure 10: Test case of calcTax(R)

Test in Non-Residential

When the user enters with Non-Residential, the system is supposed to calculate the quote request amount as specified in assessment specification Figure 2 based on the electricity rate and usage inputted by the user.

Note: although there are several attributes such as *vtype*, *vsector* and *vtariff* which are required when user accessing Non-Residential calculator, those attributes would not participate into calculation function and they only have one default choice (the rest haven't been implemented) for each one of them. Therefore, those attributes would not be involved in following test cases.

calcNoDiscBilNR

If the part which haven't been implemented are ignored, the flowchart will be simplified and the testing will mainly focus on *TARIFF B* part.

The role of this part in the program is to get the consumption (the only input) and calculate the non-residential Electricity Bill and details such as *First 200 kWh* and *Next 200 kWh* onwards.

Black-Box Testing

Input Domain – Valid Partition:

TC 1: $0 \leq \text{consumption} \leq 200$

TC 2: $201 \leq \text{consumption}$

Input Domain – Invalid Partition:

TC 6: consumption < 0

- Equivalence Partitioning

TC#	Input	Expected Output		
	Consumption	Electricity Bill	First 200 kWh	After 200 kWh
1	120	52.2	52.2	0
2	240	107.36	87	20.36
3	-100	Invalid Consumption		

Figure 11: Test cases by Equivalence Partitioning of calcNoDiscBilNR

- Boundary Value Analysis

TC#	Input	Expected Output		
	Consumption	Electricity Bill	First 200 kWh	After 200 kWh
1	0	0	0	0
	1	0.435	0.435	0
	199	86.57	86.57	0
	200	87	87	0
2	201	87.51	87	0.509
	202	88.018	87	1.018
3	-1	Invalid Consumption		

Figure 12: Test cases by BVA of calcNoDiscBilNR

calcTaxNR

The role of this part in the program is to get the consumption, electricity bill (from calcNoDiscBilR) and disc (written in source code and always be 0), and calculate the tax bill.

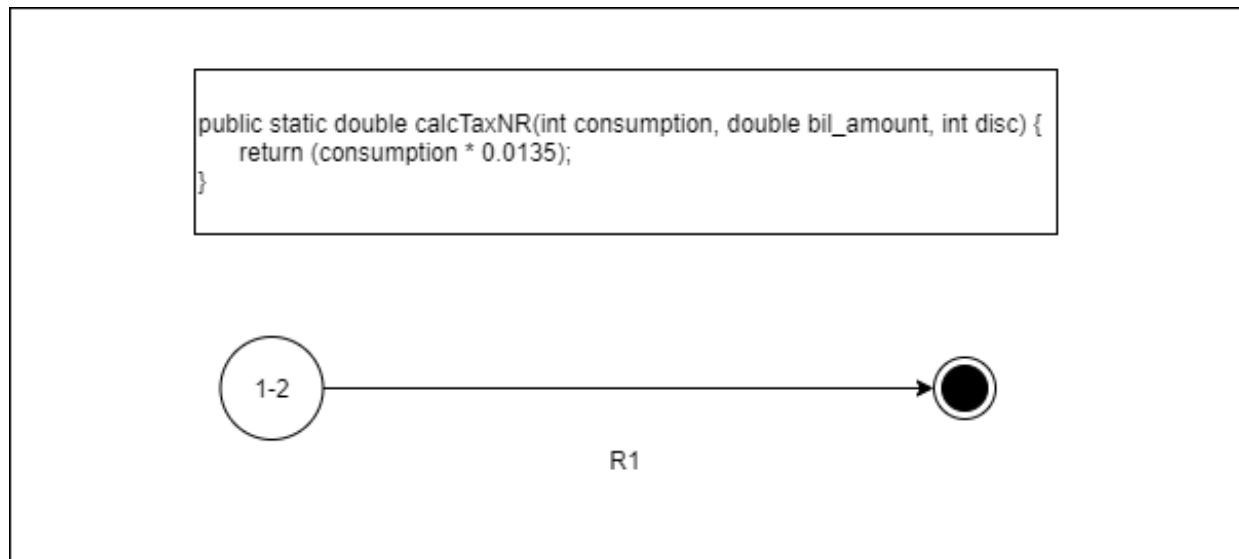


Figure 13: Flow Graph Notation of calcTaxNR

Obviously, there is only one independence path of this part of program.

Therefore, this part will be combined with calcNoDiscBiNR.

- Equivalence Partitioning

TC#	Input	Expected Output			
	Consumption	Electricity Bill	First 200 kWh	Next 200 kWh	Tax
1	120	52.2	0	0	1.62
2	240	107.36	87	20.36	3.24
3	-100	Invalid Consumption			

Figure 14: Test cases by EP of combination

- Boundary Value Analysis

TC#	Input	Expected Output			
	Consumption	Electricity Bill	First 200 kWh	After 200 kWh	Tax
1	0	0	0	0	0
	1	0.44	0.44	0	0.0135
	199	86.57	86.57	0	2.6865
	200	87	87	0	2.7
2	201	87.51	87	0.51	2.7135
	202	88.018	87	1.02	2.727
3	-1	Invalid Consumption			

Figure 15: Test cases by BVA of combination

Test Implementation

This section of document focuses on the testing implementation of the given program, the Electricity Bill Calculator, basing on the techniques and test design which have been done in the previous section of the document. The software of testing is Junit.

Global Test

TC#	Input				Expected Output	Actual Output	Pass/Fail
	Type of Residential	Type of Sector	Tariff Code	Electricity Consumption			
1	Residential	none		600	Calculate the bill of Residential and Display Result	Calculate the bill of Residential and Display Result	P
2	Non-Residential	void	-	600	Invalid Input	java.lang.NullPointerException: Cannot invoke "String.equals(Object)" because "vsector" is null	F
3	Non-Residential	Commercial	void	600	Invalid Input	java.lang.NullPointerException: Cannot invoke "String.equals(Object)" because "vtariff" is null	F
4	Non-Residential	Commercial	Tariff B	600	Calculate the bill of non-Residential and Display Result	Calculate the bill of non-Residential and Display Result	P
5	Non-Residential	Commercial	Tariff B	-600	Invalid Input	Your Electricity Bill: -87.0 Your ICPT (RM0.0135 per kWh): -2.7 Your Total Bill: -89.7	F
6	Residential	none		void	Invalid Input	Invalid Input	P

Figure 16: Test cases by Condition Table of Global Test after testing

For the output of test case 2 & 3, the system should notice and stop user to run the program before they done the selection of *Type of Sector* and *Tariff Code*. However, what the system gave is `NullPointerException` instead of notification like what it does in *Electricity Consumption* (when users ignore to input it, the system will not continue). For test case 5, it is a data-filtering mistake, which means that the *Electricity Consumption* should not be a negative

number. Obviously, there is no condition to be act when user inputting negative number as consumption, so the result will also be wrong.

*The result of operation UI test will be shown in **Summary Report** (page 21).*

Partial Test

There are differences in the number of the digits after the decimal point (for example, 1.018 and 1.02) in different program modules. The decimal values used in all functions as running and returning data are not defined the digits, except for the two functions *residential ()* and *nonresidential ()*. In addition, the program specification does not give any requirement for digits number of double values. However, in order to make the test more efficient, the decimals in the calculation process are always in accordance with the 2 decimal places (example: 2.3333333 = 2.33).

The test implementation of calcNoDiscBilR

TC#	Input	Expected Output						Actual Output						Pass/Fail
	Consumption	Electricity Bill	First 200 kWh	Next 100 kWh	Next 300 kWh	Next 300 kWh	Next 900 kWh	Electricity Bill	First 200 kWh	Next 100 kWh	Next 300 kWh	Next 300 kWh	Next 901 kWh	
1	120	26.16	26.16	0	0	0	0	26.16	0	0	0	0	0	F
2	240	56.96	43.6	13.36	0	0	0	56.96	43.6	13.36	0	0	0	P
3	450	154.4	43.6	33.4	77.4	0	0	154.4	43.6	33.4	77.4	0	0	P
4	700	286.4	43.6	33.4	154.8	54.6	0	286.4	43.6	33.4	154.8	54.6	0	P
5	1000	452.7	43.6	33.4	154.8	163.8	57.1	452.7	43.6	33.4	154.8	163.8	57.1	P
6	-100	0	0	0	0	0	0	-21.8	0	0	0	0	0	F

Figure 17: Test cases by EP of calcNoDiscBilR after testing

The problem in test case 1 is that the program didn't show the correct value of *First 200 kWh*.

```

78     }
79     } else {
80         bil_amount = (consumption * 21.80) / 100;
81         consumption = consumption - 200;
82     }

```

When reviewing the source code of this part, from line 79 to 82, it shows that the calculation result is directly assign to the attribute `bil_amount` and the attribute `first200kwh` has not been

changed. But actually, the *bil_amount* and *first200kwh* should have the same value 26.16 when consumption is not greater than 200.

And same problem as previous for negative value in test case 6.

TC#	Input	Expected Output						Actual Output						Pass/Fail
	Consumption	Electricity Bill	First 200 kWh	Next 100 kWh	Next 300 kWh	Next 300 kWh	Next 901 kWh	Electricity Bill	First 200 kWh	Next 100 kWh	Next 300 kWh	Next 300 kWh	Next 901 kWh	
6	-1	0	0	0	0	0	0	-0.22	0	0	0	0	0	F
1	0	0	0	0	0	0	0	0	0	0	0	0	0	P
	1	0.22	0.22	0	0	0	0	0.22	0	0	0	0	0	F
	199	43.38	43.38	0	0	0	0	43.38	0	0	0	0	0	F
	200	43.6	43.6	0	0	0	0	43.6	0	0	0	0	0	F
2	201	43.93	43.6	0.33	0	0	0	43.93	43.6	0.33	0	0	0	P
	202	44.27	43.6	0.67	0	0	0	44.27	43.6	0.67	0	0	0	P
	299	76.67	43.6	33.07	0	0	0	76.67	43.6	33.07	0	0	0	P
	300	77	43.6	33.4	0	0	0	77	43.6	33.4	0	0	0	P
3	301	77.52	43.6	33.4	0.52	0	0	77.52	43.6	33.4	0.52	0	0	P
	302	78.03	43.6	33.4	1.03	0	0	78.03	43.6	33.4	1.03	0	0	P
	599	231.28	43.6	33.4	154.28	0	0	231.28	43.6	33.4	154.28	0	0	P
	600	231.8	43.6	33.4	154.8	0	0	231.8	43.6	33.4	154.8	0	0	P
4	601	232.35	43.6	33.4	154.8	0.55	0	232.35	43.6	33.4	154.8	0.55	0	P
	602	232.89	43.6	33.4	154.8	1.09	0	232.89	43.6	33.4	154.8	1.09	0	P
	899	395.05	43.6	33.4	154.8	163.25	0	395.05	43.6	33.4	154.8	163.25	0	P
	900	395.6	43.6	33.4	154.8	163.8	0	395.6	43.6	33.4	154.8	163.8	0	P
5	901	396.17	43.6	33.4	154.8	163.8	0.57	396.17	43.6	33.4	154.8	163.8	0.57	P
	902	396.74	43.6	33.4	154.8	163.8	1.14	396.74	43.6	33.4	154.8	163.8	1.14	P

Figure 18: Test cases by BVA of calcNoDiscBilR after testing

The test implementation of calcTaxR

TC#	Input		Expected Output	Actual Output	Pass/Fail
	Consumption	Electricity Bill	Tax	Tax	
1	600	231.8	0	0	P
2	1000	452.7	13.25	13.25	P

Figure 19: Test cases of calcTaxR after testing

TC#	Input		Expected Output	Actual Output	Pass/Fail
	Consumption	Electricity Bill	Tax	Tax	
1	600	231.8	0	0	P
2	601	232.35	0.03	0.03	P

	602	232.89	0.07	0.07	P
--	-----	--------	------	------	---

Figure 20: Test cases by BVA of calcTaxR after testing

Seems all the test case are passed. But there is still a problem when it showing the coverage of the partial source code. Obviously, the method is not fully covered.

```

149 // To calculate 6% on bill amount. The tax is charged for consumption more than
150 // 600kWh
151 public static double calcTaxR(int consumption, double bil_amount, int disc) {
152     if (consumption < 600) {
153         return 0;
154     } else {
155         return ((bil_amount - 231.80) * 0.06);
156     }
157 }

```

However, according to the program specification (page 10), the condition should be *consumption* < 601 or *consumption* <= 600.

The test implementation of calcNoDiscBilNR and calcTaxNR

TC#	Input	Expected Output				Actual Output				Pass/Fail
	Consumption	Electricity Bill	First 200 kWh	After 200 kWh	tax	Electricity Bill	First 200 kWh	After 200 kWh	tax	
1	120	52.2	52.2	0	1.62	52.2	0	0	1.62	F
2	240	107.36	87	20.36	3.24	107.36	87	20.36	3.24	P
3	-100	0	0	0	0	-43.5	0	0	-3.5	F

Figure 21: Test cases by EP of calcNoDiscBilNR after testing

TC#	Input	Expected Output				Actual Output				Pass/Fail
	Consumption	Electricity Bill	First 200 kWh	After 200 kWh	tax	Electricity Bill	First 200 kWh	After 200 kWh	tax	
1	0	0	0	0	0	0	0	0	0	P
	1	0.44	0.44	0	0.0135	0.44	0	0	0.014	F
	199	86.57	86.57	0	2.69	86.57	0	0	2.69	F
	200	87	87	0	2.7	87	0	0	2.7	F
2	201	87.51	87	0.51	2.71	87.44	0	0	2.71	F
	202	88.02	87	1.02	2.73	88.02	87	1.02	2.73	P
3	-1	0	0	0	0	-0.435	0	0	-0.0135	F

Figure 22: Test cases by BVA of calcNoDiscBilNR after testing

The problem in test case 1-2, 1-3, 1-4 and 2-1 is that the program didn't show the correct value of *First 200 kWh* just as what happened in *the test implementation of calcNoDiscBilR*. And same problem as previous for negative value in test case 3.

```

133     if (consumption > 201) {
134         first200kwh = 200 * 0.435;
135         consumption = consumption - 200;
136         nextbalancekwh = consumption * 0.509;
137         bil_amount = first200kwh + nextbalancekwh;

```

But in test case 2-1, there are another problem that the *Electricity Bill* is wrong. When reviewing the source code of this part, from line 133, it shows the condition is written as *consumption > 201*. But the condition should be written as *consumption => 201* as what is shown in program specification.

Additional test for round

TC#	Input		Expected Output	Actual Output	Pass/Fail
	value	place			
1	0.33333333	-1	Invalid Input	Invalid Input	P
2	0.33333333	0	0	0	P
3	0.33333333	1	0.3	0.3	P
4	0.33333333	2	0.33	0.33	P

Figure 23: Test cases by BVA of *round* after testing

This internal method has not been found any problems.

Summary report

After testing and analyzing the software, many problems were discovered one by one.

Information about specific issues is listed in the table below:

ORDER	PROBLEM	SECTION	DETAIL
1	Void selection	Global Test	Figure 16: Test cases by Condition Table of Global Test after testing, test case 2 & 3 (page 14)
2	Negative value	Global Test	Figure 16: Test cases by Condition Table of Global Test after testing, test case 5 (page 14)
3	Incorrect First 200 kWh	Partial Test	Figure 17: Test cases by EP of <i>calcNoDiscBilR</i> after testing, test case 1 (page 15) Figure 18: Test cases by BVA of <i>calcNoDiscBilR</i> after testing, test case 1-2, 1-3, 1-4 (page 16)
4	Negative value	Partial Test	Figure 17: Test cases by EP of <i>calcNoDiscBilR</i> after testing, test case 6 (page 15) Figure 18: Test cases by BVA of <i>calcNoDiscBilR</i> after testing, test case 6 (page 16)
5	Wrong condition	Partial Test	Figure 20: Test cases by BVA of <i>calcTaxR</i> after testing, test case 1 (page 16)
6	Incorrect First 200 kWh	Partial Test	Figure 21: Test cases by EP of <i>calcNoDiscBilNR</i> after testing, test case 1 (page 17); Figure 22: Test cases by BVA of <i>calcNoDiscBilNR</i> after testing, test case 1-2, 1-3, 1-4 (page 17);
7	Negative value	Partial Test	Figure 21: Test cases by EP of <i>calcNoDiscBilNR</i> after testing, test case 3 (page 17); Figure 22: Test cases by BVA of <i>calcNoDiscBilNR</i> after testing, test case 3 (page 17);
8	Wrong condition	Partial Test	Figure 22: Test cases by BVA of <i>calcNoDiscBilNR</i> after testing, test case 2-1 (page 17);

Figure 24: Report Table I

Order	Method	State	Remark
1	calcTaxNR(int, double, int)	abnormal	If the input is not negative number, then it won't have problem
2	calcTaxR(int, double, int)	abnormal	Wrong condition: consumption < 600
3	round(double, int)	normal	-
4	calcNoDiscBilNR(int)	abnormal	incorrect value of First 200 kWh;

Order	Method	State	Remark
5	calcNoDiscBilR(int)	abnormal	wrong condition: consumption > 201; negative number incorrect value of First 200 kWh; wrong condition: consumption < 201; negative values
6	nonresidential(String, String, String, int)	normal	-
7	residential(String, int)	normal	-

Figure 25: Report Table II

For all the tests which have been done are about class *EBService_Imp*, the main class which controlling the core of the whole program. For global test, this test actually is designed for test both *EBService_Imp* (residential and nonresidential parts) and *EBController*. Because *EBController* as a special part in this program which cannot be test by JUnit directly, the tests of it is performed on UI operation to check if it works.








































TC#	DETAIL	OUTPUT	PASS/FAIL
1	State of Residence: <div>Residential</div> Electricity Consumption <div>600</div>	Your Electricity Bill: 231.8 Your Tax Bill: 0.0 Your Total Bill: 231.8 Detail Calculation of Your Electricity Bill	P
2	Type of Residential: <div>Non-Residential</div> Type of Sector: <div>Choose type of sector</div> Electricity Consumption <div>600</div>	Type Exception Report Message Cannot invoke "String.equals(Object)" because "vsector" is null Description The server encountered an unexpected condition that prevent Exception <pre>java.lang.NullPointerException: Cannot invoke "String.equals(C com.sse3305.service.EBService_Imp.nonresidential (EBSe com.sse3305.EBController.doPost (EBController.java:44) javax.servlet.http.HttpServlet.service (HttpServlet.ja javax.servlet.http.HttpServlet.service (HttpServlet.ja org.apache.tomcat.websocket.server.WsFilter.doFilter(W</pre> Note The full stack trace of the root cause is available in the server logs.	F
3	Type of Residential: <div>Non-Residential</div> Type of Sector: <div>Commercial</div> Tariff Code: <div>Choose tariff code</div> Electricity Consumption <div>600</div>	Type Exception Report Message Cannot invoke "String.equals(Object)" because "vtariff" is null Description The server encountered an unexpected condition that prevent Exception <pre>java.lang.NullPointerException: Cannot invoke "String.equals(com.sse3305.service.EBService_Imp.nonresidential (EBSe com.sse3305.EBController.doPost (EBController.java:44) javax.servlet.http.HttpServlet.service (HttpServlet.ja javax.servlet.http.HttpServlet.service (HttpServlet.ja org.apache.tomcat.websocket.server.WsFilter.doFilter(W</pre> Note The full stack trace of the root cause is available in the server logs.	F















TC#	DETAIL	OUTPUT	PASS/FAIL
4	State of Residence: <div>Non-Residential</div> Type of Sector: <div>Commercial</div> Tariff Code: <div>Tariff B: Low Voltage Cc</div> Electricity Consumption <div>600</div>	***** Your Electricity Bill: 290.6 Your ICPT (RM0.0135 per kWh): 8.1 Your Total Bill: 298.7 Detail Calculation of Your Electricity Bill *****	P
5	State of Residence: <div>Non-Residential</div> Type of Sector: <div>Commercial</div> Tariff Code: <div>Tariff B: Low Voltage Cc</div> Electricity Consumption <div>-600</div>	Your Electricity Bill: -261.0 Your ICPT (RM0.0135 per kWh): -8.1 Your Total Bill: -269.1 Detail Calculation of Your Electricity Bill	F
6	Type of Residential: <div>Residential</div> Electricity Consumption <div>Enter the electricity con:</div>	Type of Residential: <div>Residential</div> Electricity Consumption <div>Enter the electricity con:</div> <div>Caution! 请填写此字段。</div> <div>Back</div>	P

Figure 26: Global test case after UI operation testing

However, the test case 1 and 4 can prove that the program can call the correct method to run next step when user input the information in a right way. As you can see the results are same as Figure 16: Test cases by Condition Table of Global Test after testing in page 14. In addition, from the result we can know that, there is no checking if text field is void and no condition for preventing negative values.

Appendix I - Screenshots

Intro	Picture
The screenshot of the Junit test result of Test cases by Condition Table of Global Test (<i>nonresidential/ residential</i>) after testing	 Global Test 1 (0.000 s)  Global Test 2 (0.001 s)  Global Test 3 (0.001 s)  Global Test 4 (0.000 s)  Global Test 5 (0.001 s)  Global Test 6 (0.001 s)
The screenshot of the Junit test result of Test cases by EP of <i>calcNoDiscBiIR</i> after testing	 Equivalence Partitioning of calcNoDiscBiIR Test 1 (0.001 s)  Equivalence Partitioning of calcNoDiscBiIR Test 2 (0.000 s)  Equivalence Partitioning of calcNoDiscBiIR Test 3 (0.001 s)  Equivalence Partitioning of calcNoDiscBiIR Test 4 (0.001 s)  Equivalence Partitioning of calcNoDiscBiIR Test 5 (0.000 s)  Equivalence Partitioning of calcNoDiscBiIR Test 6 (0.001 s)  Equivalence Partitioning of calcNoDiscBiIRNR Test 1 (0.001 s)  Equivalence Partitioning of calcNoDiscBiIRNR Test 2 (0.001 s)  Equivalence Partitioning of calcNoDiscBiIRNR Test 3 (0.001 s)
The screenshot of the Junit test result of Test cases by BVA of <i>calcNoDiscBiIR</i> after testing	 BVA of calcNoDiscBiIR Test 2 number 1 (0.000 s)  BVA of calcNoDiscBiIR Test 2 number 2 (0.000 s)  BVA of calcNoDiscBiIR Test 2 number 3 (0.000 s)  BVA of calcNoDiscBiIR Test 2 number 4 (0.000 s)  BVA of calcNoDiscBiIR Test 3 number 1 (0.000 s)  BVA of calcNoDiscBiIR Test 3 number 2 (0.000 s)  BVA of calcNoDiscBiIR Test 3 number 3 (0.000 s)  BVA of calcNoDiscBiIR Test 3 number 4 (0.000 s)  BVA of calcNoDiscBiIR Test 4 number 1 (0.000 s)  BVA of calcNoDiscBiIR Test 4 number 2 (0.000 s)  BVA of calcNoDiscBiIR Test 4 number 3 (0.000 s)  BVA of calcNoDiscBiIR Test 4 number 4 (0.000 s)  BVA of calcNoDiscBiIR Test 5 number 1 (0.000 s)  BVA of calcNoDiscBiIR Test 5 number 2 (0.000 s)  BVA of calcNoDiscBiIR Test 6 number 1 (0.001 s)  BVA of calcNoDiscBiIR Test 1 number 1 (0.000 s)  BVA of calcNoDiscBiIR Test 1 number 2 (0.000 s)  BVA of calcNoDiscBiIR Test 1 number 3 (0.000 s)  BVA of calcNoDiscBiIR Test 1 number 4 (0.001 s)
The screenshot of the Junit test result of Test cases of <i>calcTaxR</i> after testing	 calcTaxR Test 1 (0.001 s)  calcTaxR Test 2 (0.001 s)
The screenshot of the Junit test result of Test cases by BVA of <i>calcTaxR</i> after testing	 BVA of calcTaxR Test 2 number 1 (0.000 s)  BVA of calcTaxR Test 2 number 2 (0.001 s)  BVA of calcTaxR Test 1 (0.011 s)

Intro	Picture
The screenshot of the Junit test result of Test cases by EP of <i>calcNoDiscBilNR</i> after testing	 Equivalence Partitioning of calcNoDiscBilNR Test 1 (0.000 s)  Equivalence Partitioning of calcNoDiscBilNR Test 2 (0.000 s)  Equivalence Partitioning of calcNoDiscBilNR Test 3 (0.000 s)
The screenshot of the Junit test result of Test cases by BVA of <i>calcNoDiscBilNR</i> after testing	 BVA of calcNoDiscBilNR Test 1 number 1 (0.000 s)  BVA of calcNoDiscBilNR Test 1 number 2 (0.000 s)  BVA of calcNoDiscBilNR Test 1 number 3 (0.001 s)  BVA of calcNoDiscBilNR Test 1 number 4 (0.000 s)  BVA of calcNoDiscBilNR Test 2 number 1 (0.000 s)  BVA of calcNoDiscBilNR Test 2 number 2 (0.000 s)  BVA of calcNoDiscBilNR Test 3 number 1 (0.001 s)
The screenshot of the Junit test result of Additional testing (<i>round</i>)	 round Test 1 (0.000 s)  round Test 2 (0.000 s)  round Test 3 (0.000 s)  round Test 4 (0.000 s)

Total test cases wrote in `EBService_ImpTest.java`















Finished after 0.19 seconds

Runs: 50/50

Errors: 0

Failures: 19

Coverage

Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
ElectricityBill	 80.5 %	2,769	671	3,440
src/test/java	 80.3 %	2,334	573	2,907
src/main/java	 81.6 %	435	98	533
com.sse3305	 0.0 %	0	98	98
com.sse3305.service	 100.0 %	435	0	435
EBService_Imp.java	 100.0 %	435	0	435
EBService_Imp	 100.0 %	435	0	435
calcTaxNR(int, double, int)	 100.0 %	5	0	5
calcTaxR(int, double, int)	 100.0 %	11	0	11
round(double, int)	 100.0 %	17	0	17
calcNoDiscBilNR(int)	 100.0 %	48	0	48
calcNoDiscBilR(int)	 100.0 %	150	0	150
nonresidential(String, String)	 100.0 %	95	0	95
residential(String, int)	 100.0 %	94	0	94

Appendix II – Test Source code

Source code of class *EBService_ImpTest*:

```
package com.sse3305.service;

import static org.junit.jupiter.api.Assertions.*;

import java.util.ArrayList;
import java.math.BigDecimal;
import java.math.RoundingMode;
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Test;

class EBService_ImpTest {

    //copy from the main program used to keep format of double values
    public static double round(double value, int places) {
        if (places < 0)
            throw new IllegalArgumentException();

        BigDecimal bd = BigDecimal.valueOf(value);
        bd = bd.setScale(places, RoundingMode.HALF_UP);
        return bd.doubleValue();
    }

    //assume the program already run in residential/non- residential method (pre-condition: the
    //controller can assign correct method when accept values from JSP UI). The result of this Junit global test
    //will be compared with the UI operation test.
    @Test
    @DisplayName("Global Test 1")
    void Global_Test1() {
        EBService_Imp imp = new EBService_Imp();
        int veconsumption = 600;
        String vtype = "Residential";
        String vsector = "";
        String vtariff = "";
        boolean real = false;
        try{
            ArrayList<Double> EbillR = imp.residential(vtype, veconsumption);
        }catch (Exception e) {
            real = true;
        }
        assertEquals(false, real);
    }

    @Test
    @DisplayName("Global Test 2")
    void Global_Test2() {
        EBService_Imp imp = new EBService_Imp();
        int veconsumption = 600;
        String vtype = "Non-Residential";
        String vsector = "";
        String vtariff = "Tariff B";
        boolean real = false;
        try{
            ArrayList<Double> EbillR = imp.nonresidential(vtype, vsector, vtariff,
veconsumption);
        }catch (Exception e) {
            real = true;
        }
        assertEquals(false, real);
    }

    @Test
    @DisplayName("Global Test 3")
    void Global_Test3() {
```



```

        EBSERVICE_Imp imp = new EBSERVICE_Imp();
        int veconsumption = 600;
        String vtype = "Non-Residential";
        String vsector = "Commercial";
        String vtariff = "";
        boolean real = false;
        try{
            ArrayList<Double> EbillR = imp.nonresidential(vtype, vsector, vtariff,
veconsumption);
        }catch(Exception e) {
            real = true;
        }
        assertEquals(false, real);
    }

    @Test
    @DisplayName("Global Test 4")
    void Global_Test4() {
        EBSERVICE_Imp imp = new EBSERVICE_Imp();
        int veconsumption = 600;
        String vtype = "Non-Residential";
        String vsector = "Commercial";
        String vtariff = "Tariff B";
        boolean real = false;
        try{
            ArrayList<Double> EbillR = imp.nonresidential(vtype, vsector, vtariff,
veconsumption);
        }catch(Exception e) {
            real = true;
        }
        assertEquals(false, real);
    }

    @Test
    @DisplayName("Global Test 5")
    void Global_Test5() {
        EBSERVICE_Imp imp = new EBSERVICE_Imp();
        int veconsumption = -600;
        String vtype = "Non-Residential";
        String vsector = "Commercial";
        String vtariff = "Tariff B";
        boolean real = false;
        try{
            ArrayList<Double> EbillR = imp.nonresidential(vtype, vsector, vtariff,
veconsumption);
            if(EbillR.get(0) < 0)real = true;
        }catch(Exception e) {
            real = true;
        }
        assertEquals(false, real);
    }

    @Test
    @DisplayName("Global Test 6")
    void Global_Test6() {
        EBSERVICE_Imp imp = new EBSERVICE_Imp();
        int veconsumption = 0;
        String vtype = "Residential";
        String vsector = "";
        String vtariff = "";
        boolean real = false;
        try{
            ArrayList<Double> EbillR = imp.residential(vtype,veconsumption);
        }catch(Exception e) {
            real = true;
        }
        assertEquals(false, real);
    }
}

```

```

@Test
@DisplayName("Equivalence Partitioning of calcNoDiscBilNR Test 1")
void calcNoDiscBilNR_EP_Test1() {
    EBServise_Imp imp = new EBServise_Imp();
    int veconsumption = 120;
    ArrayList<Double> bilNR_detail = imp.calcNoDiscBilNR(veconsumption);
    double eb = 52.2;
    double f2 = 52.2;
    double a2 = 0;
    assertEquals(f2, round(bilNR_detail.get(0),2));
    assertEquals(a2, round(bilNR_detail.get(1),2));
    assertEquals(eb, round(bilNR_detail.get(2),2));
    double tax = round(imp.calcTaxNR(veconsumption, eb, 0),2);
    double t = 1.62;
    assertEquals(t, tax);
}

@Test
@DisplayName("Equivalence Partitioning of calcNoDiscBilNR Test 2")
void calcNoDiscBilNR_EP_Test2() {
    EBServise_Imp imp = new EBServise_Imp();
    int veconsumption = 240;
    ArrayList<Double> bilNR_detail = imp.calcNoDiscBilNR(veconsumption);
    double eb = 107.36;
    double f2 = 87;
    double a2 = 20.36;
    assertEquals(f2, round(bilNR_detail.get(0),2));
    assertEquals(a2, round(bilNR_detail.get(1),2));
    assertEquals(eb, round(bilNR_detail.get(2),2));
    double tax = round(imp.calcTaxNR(veconsumption, eb, 0),2);
    double t = 3.24;
    assertEquals(t, tax);
}

@Test
@DisplayName("Equivalence Partitioning of calcNoDiscBilNR Test 3")
void calcNoDiscBilNR_EP_Test3() {
    EBServise_Imp imp = new EBServise_Imp();
    int veconsumption = -100;
    ArrayList<Double> bilNR_detail = imp.calcNoDiscBilNR(veconsumption);
    double eb = 0;
    double f2 = 0;
    double a2 = 0;
    assertEquals(f2, round(bilNR_detail.get(0),2));
    assertEquals(a2, round(bilNR_detail.get(1),2));
    assertEquals(eb, round(bilNR_detail.get(2),2));
    double tax = round(imp.calcTaxNR(veconsumption, eb, 0),2);
    double t = 0;
    assertEquals(t, tax);
}

@Test
@DisplayName("BVA of calcNoDiscBilNR Test 1 number 1")
void calcNoDiscBilNR_BVA_Test1_1() {
    EBServise_Imp imp = new EBServise_Imp();
    int veconsumption = 0;
    ArrayList<Double> bilNR_detail = imp.calcNoDiscBilNR(veconsumption);
    double eb = 0;
    double f2 = 0;
    double a2 = 0;
    assertEquals(f2, round(bilNR_detail.get(0),2));
    assertEquals(a2, round(bilNR_detail.get(1),2));
    assertEquals(eb, round(bilNR_detail.get(2),2));
    double tax = round(imp.calcTaxNR(veconsumption, eb, 0),2);
    double t = 0;
    assertEquals(t, tax);
}

```

```

@Test
@DisplayName("BVA of calcNoDiscBilNR Test 1 number 2")
void calcNoDiscBilNR_BVA_Test1_2() {
    EBSERVICE_imp imp = new EBSERVICE_imp();
    int veconsumption = 1;
    ArrayList<Double> bilNR_detail = imp.calcNoDiscBilNR(veconsumption);
    double eb = 0.44;
    double f2 = 0.44;
    double a2 = 0;
    assertEquals(f2, round(bilNR_detail.get(0),2));
    assertEquals(a2, round(bilNR_detail.get(1),2));
    assertEquals(eb, round(bilNR_detail.get(2),2));
    double tax = round(imp.calcTaxNR(veconsumption, eb, 0),2);
    double t = 0.0135;
    assertEquals(t, tax);
}

@Test
@DisplayName("BVA of calcNoDiscBilNR Test 1 number 3")
void calcNoDiscBilNR_BVA_Test1_3() {
    EBSERVICE_imp imp = new EBSERVICE_imp();
    int veconsumption = 199;
    ArrayList<Double> bilNR_detail = imp.calcNoDiscBilNR(veconsumption);
    double eb = 86.57;
    double f2 = 86.57;
    double a2 = 0;
    assertEquals(f2, round(bilNR_detail.get(0),2));
    assertEquals(a2, round(bilNR_detail.get(1),2));
    assertEquals(eb, round(bilNR_detail.get(2),2));
    double tax = round(imp.calcTaxNR(veconsumption, eb, 0),2);
    double t = 2.69;
    assertEquals(t, tax);
}

@Test
@DisplayName("BVA of calcNoDiscBilNR Test 1 number 4")
void calcNoDiscBilNR_BVA_Test1_4() {
    EBSERVICE_imp imp = new EBSERVICE_imp();
    int veconsumption = 200;
    ArrayList<Double> bilNR_detail = imp.calcNoDiscBilNR(veconsumption);
    double eb = 87;
    double f2 = 87;
    double a2 = 0;
    assertEquals(f2, round(bilNR_detail.get(0),2));
    assertEquals(a2, round(bilNR_detail.get(1),2));
    assertEquals(eb, round(bilNR_detail.get(2),2));
    double tax = round(imp.calcTaxNR(veconsumption, eb, 0),2);
    double t = 2.7;
    assertEquals(t, tax);
}

@Test
@DisplayName("BVA of calcNoDiscBilNR Test 2 number 1")
void calcNoDiscBilNR_BVA_Test2_1() {
    EBSERVICE_imp imp = new EBSERVICE_imp();
    int veconsumption = 201;
    ArrayList<Double> bilNR_detail = imp.calcNoDiscBilNR(veconsumption);
    double eb = 87.51;
    double f2 = 87;
    double a2 = 0.51;
    assertEquals(f2, round(bilNR_detail.get(0),2));
    assertEquals(a2, round(bilNR_detail.get(1),2));
    assertEquals(eb, round(bilNR_detail.get(2),2));
    double tax = round(imp.calcTaxNR(veconsumption, eb, 0),2);
    double t = 2.72;
    assertEquals(t, tax);
}

```

```

@Test
@DisplayName("BVA of calcNoDiscBilNR Test 2 number 2")
void calcNoDiscBilNR_BVA_Test2_2() {
    EBSERVICE_imp imp = new EBSERVICE_imp();
    int veconsumption = 202;
    ArrayList<Double> bilNR_detail = imp.calcNoDiscBilNR(veconsumption);
    double eb = 88.02;
    double f2 = 87;
    double a2 = 1.02;
    assertEquals(f2, round(bilNR_detail.get(0),2));
    assertEquals(a2, round(bilNR_detail.get(1),2));
    assertEquals(eb, round(bilNR_detail.get(2),2));
    double tax = round(imp.calcTaxNR(veconsumption, eb, 0),2);
    double t = 2.73;
    assertEquals(t, tax);
}

@Test
@DisplayName("BVA of calcNoDiscBilNR Test 3 number 1")
void calcNoDiscBilNR_BVA_Test3_1() {
    EBSERVICE_imp imp = new EBSERVICE_imp();
    int veconsumption = -1;
    ArrayList<Double> bilNR_detail = imp.calcNoDiscBilNR(veconsumption);
    double eb = 0;
    double f2 = 0;
    double a2 = 0;
    assertEquals(f2, round(bilNR_detail.get(0),2));
    assertEquals(a2, round(bilNR_detail.get(1),2));
    assertEquals(eb, round(bilNR_detail.get(2),2));
    double tax = round(imp.calcTaxNR(veconsumption, eb, 0),2);
    double t = 0;
    assertEquals(t, tax);
}

@Test
@DisplayName("Equivalence Partitioning of calcNoDiscBilR Test 1")
void calcNoDiscBilR_EP_Test1() {
    EBSERVICE_imp imp = new EBSERVICE_imp();
    int veconsumption = 120;
    ArrayList<Double> bilNR_detail = imp.calcNoDiscBilR(veconsumption);
    double eb = 26.16;
    double f200 = 26.16;
    double n100 = 0;
    double n300 = 0;
    double n300_2 = 0;
    double n900 = 0;
    assertEquals(f200, round(bilNR_detail.get(0),2));
    assertEquals(n100, round(bilNR_detail.get(1),2));
    assertEquals(n300, round(bilNR_detail.get(2),2));
    assertEquals(n300_2, round(bilNR_detail.get(3),2));
    assertEquals(n900, round(bilNR_detail.get(4),2));
    assertEquals(eb, round(bilNR_detail.get(5),2));
}

@Test
@DisplayName("Equivalence Partitioning of calcNoDiscBilR Test 2")
void calcNoDiscBilR_EP_Test2() {
    EBSERVICE_imp imp = new EBSERVICE_imp();
    int veconsumption = 240;
    ArrayList<Double> bilNR_detail = imp.calcNoDiscBilR(veconsumption);
    double eb = 56.96;
    double f200 = 43.6;
    double n100 = 13.36;
    double n300 = 0;
    double n300_2 = 0;
    double n900 = 0;
    assertEquals(f200, round(bilNR_detail.get(0),2));
    assertEquals(n100, round(bilNR_detail.get(1),2));
    assertEquals(n300, round(bilNR_detail.get(2),2));
}

```

```

        assertEquals(n300_2, round(bilNR_detail.get(3),2));
        assertEquals(n900, round(bilNR_detail.get(4),2));
        assertEquals(eb, round(bilNR_detail.get(5),2));
    }

    @Test
    @DisplayName("Equivalence Partitioning of calcNoDiscBilR Test 3")
    void calcNoDiscBilR_EP_Test3() {
        EBSERVICE_imp imp = new EBSERVICE_imp();
        int veconsumption = 450;
        ArrayList<Double> bilNR_detail = imp.calcNoDiscBilR(veconsumption);
        double eb = 154.4;
        double f200 = 43.6;
        double n100 = 33.4;
        double n300 = 77.4;
        double n300_2 = 0;
        double n900 = 0;
        assertEquals(f200, round(bilNR_detail.get(0),2));
        assertEquals(n100, round(bilNR_detail.get(1),2));
        assertEquals(n300, round(bilNR_detail.get(2),2));
        assertEquals(n300_2, round(bilNR_detail.get(3),2));
        assertEquals(n900, round(bilNR_detail.get(4),2));
        assertEquals(eb, round(bilNR_detail.get(5),2));
    }

    @Test
    @DisplayName("Equivalence Partitioning of calcNoDiscBilR Test 4")
    void calcNoDiscBilR_EP_Test4() {
        EBSERVICE_imp imp = new EBSERVICE_imp();
        int veconsumption = 700;
        ArrayList<Double> bilNR_detail = imp.calcNoDiscBilR(veconsumption);
        double eb = 286.4;
        double f200 = 43.6;
        double n100 = 33.4;
        double n300 = 154.8;
        double n300_2 = 54.6;
        double n900 = 0;
        assertEquals(f200, round(bilNR_detail.get(0),2));
        assertEquals(n100, round(bilNR_detail.get(1),2));
        assertEquals(n300, round(bilNR_detail.get(2),2));
        assertEquals(n300_2, round(bilNR_detail.get(3),2));
        assertEquals(n900, round(bilNR_detail.get(4),2));
        assertEquals(eb, round(bilNR_detail.get(5),2));
    }

    @Test
    @DisplayName("Equivalence Partitioning of calcNoDiscBilR Test 5")
    void calcNoDiscBilR_EP_Test5() {
        EBSERVICE_imp imp = new EBSERVICE_imp();
        int veconsumption = 1000;
        ArrayList<Double> bilNR_detail = imp.calcNoDiscBilR(veconsumption);
        double eb = 452.7;
        double f200 = 43.6;
        double n100 = 33.4;
        double n300 = 154.8;
        double n300_2 = 163.8;
        double n900 = 57.1;
        assertEquals(f200, round(bilNR_detail.get(0),2));
        assertEquals(n100, round(bilNR_detail.get(1),2));
        assertEquals(n300, round(bilNR_detail.get(2),2));
        assertEquals(n300_2, round(bilNR_detail.get(3),2));
        assertEquals(n900, round(bilNR_detail.get(4),2));
        assertEquals(eb, round(bilNR_detail.get(5),2));
    }

    @Test
    @DisplayName("Equivalence Partitioning of calcNoDiscBilR Test 6")
    void calcNoDiscBilR_EP_Test6() {
        EBSERVICE_imp imp = new EBSERVICE_imp();

```

```

        int veconsumption = -100;
        ArrayList<Double> bilNR_detail = imp.calcNoDiscBilR(veconsumption);
        double eb = 0;
        double f200 = 0;
        double n100 = 0;
        double n300 = 0;
        double n300_2 = 0;
        double n900 = 0;
        assertEquals(f200, round(bilNR_detail.get(0),2));
        assertEquals(n100, round(bilNR_detail.get(1),2));
        assertEquals(n300, round(bilNR_detail.get(2),2));
        assertEquals(n300_2, round(bilNR_detail.get(3),2));
        assertEquals(n900, round(bilNR_detail.get(4),2));
        assertEquals(eb, round(bilNR_detail.get(5),2));
    }

@Test
@DisplayName("BVA of calcNoDiscBilR Test 1 number 1")
void calcNoDiscBilR_BVA_Test1_1() {
    EBSservice_Imp imp = new EBSservice_Imp();
    int veconsumption =0;
    ArrayList<Double> bilNR_detail = imp.calcNoDiscBilR(veconsumption);
    double eb = 0;
    double f200 = 0;
    double n100 = 0;
    double n300 = 0;
    double n300_2 = 0;
    double n900 = 0;
    assertEquals(f200, round(bilNR_detail.get(0),2));
    assertEquals(n100, round(bilNR_detail.get(1),2));
    assertEquals(n300, round(bilNR_detail.get(2),2));
    assertEquals(n300_2, round(bilNR_detail.get(3),2));
    assertEquals(n900, round(bilNR_detail.get(4),2));
    assertEquals(eb, round(bilNR_detail.get(5),2));
}

@Test
@DisplayName("BVA of calcNoDiscBilR Test 1 number 2")
void calcNoDiscBilR_BVA_Test1_2() {
    EBSservice_Imp imp = new EBSservice_Imp();
    int veconsumption =1;
    ArrayList<Double> bilNR_detail = imp.calcNoDiscBilR(veconsumption);
    double eb = 0.22;
    double f200 = 0.22;
    double n100 = 0;
    double n300 = 0;
    double n300_2 = 0;
    double n900 = 0;
    assertEquals(f200, round(bilNR_detail.get(0),2));
    assertEquals(n100, round(bilNR_detail.get(1),2));
    assertEquals(n300, round(bilNR_detail.get(2),2));
    assertEquals(n300_2, round(bilNR_detail.get(3),2));
    assertEquals(n900, round(bilNR_detail.get(4),2));
    assertEquals(eb, round(bilNR_detail.get(5),2));
}

@Test
@DisplayName("BVA of calcNoDiscBilR Test 1 number 3")
void calcNoDiscBilR_BVA_Test1_3() {
    EBSservice_Imp imp = new EBSservice_Imp();
    int veconsumption =199;
    ArrayList<Double> bilNR_detail = imp.calcNoDiscBilR(veconsumption);
    double eb = 43.38;
    double f200 = 43.38;
    double n100 = 0;
    double n300 = 0;
    double n300_2 = 0;
    double n900 = 0;
    assertEquals(f200, round(bilNR_detail.get(0),2));

```

```

        assertEquals(n100, round(bilNR_detail.get(1),2));
        assertEquals(n300, round(bilNR_detail.get(2),2));
        assertEquals(n300_2, round(bilNR_detail.get(3),2));
        assertEquals(n900, round(bilNR_detail.get(4),2));
        assertEquals(eb, round(bilNR_detail.get(5),2));
    }

@Test
@DisplayName("BVA of calcNoDiscBilR Test 1 number 4")
void calcNoDiscBilR_BVA_Test1_4() {
    EBSservice_Imp imp = new EBSservice_Imp();
    int veconsumption = 200;
    ArrayList<Double> bilNR_detail = imp.calcNoDiscBilR(veconsumption);
    double eb = 43.6;
    double f200 = 43.6;
    double n100 = 0;
    double n300 = 0;
    double n300_2 = 0;
    double n900 = 0;
    assertEquals(f200, round(bilNR_detail.get(0),2));
    assertEquals(n100, round(bilNR_detail.get(1),2));
    assertEquals(n300, round(bilNR_detail.get(2),2));
    assertEquals(n300_2, round(bilNR_detail.get(3),2));
    assertEquals(n900, round(bilNR_detail.get(4),2));
    assertEquals(eb, round(bilNR_detail.get(5),2));
}

@Test
@DisplayName("BVA of calcNoDiscBilR Test 2 number 1")
void calcNoDiscBilR_EP_Test2_1() {
    EBSservice_Imp imp = new EBSservice_Imp();
    int veconsumption = 201;
    ArrayList<Double> bilNR_detail = imp.calcNoDiscBilR(veconsumption);
    double eb = 43.93;
    double f200 = 43.6;
    double n100 = 0.33;
    double n300 = 0;
    double n300_2 = 0;
    double n900 = 0;
    assertEquals(f200, round(bilNR_detail.get(0),2));
    assertEquals(n100, round(bilNR_detail.get(1),2));
    assertEquals(n300, round(bilNR_detail.get(2),2));
    assertEquals(n300_2, round(bilNR_detail.get(3),2));
    assertEquals(n900, round(bilNR_detail.get(4),2));
    assertEquals(eb, round(bilNR_detail.get(5),2));
}

@Test
@DisplayName("BVA of calcNoDiscBilR Test 2 number 2")
void calcNoDiscBilR_EP_Test2_2() {
    EBSservice_Imp imp = new EBSservice_Imp();
    int veconsumption = 202;
    ArrayList<Double> bilNR_detail = imp.calcNoDiscBilR(veconsumption);
    double eb = 44.27;
    double f200 = 43.6;
    double n100 = 0.67;
    double n300 = 0;
    double n300_2 = 0;
    double n900 = 0;
    assertEquals(f200, round(bilNR_detail.get(0),2));
    assertEquals(n100, round(bilNR_detail.get(1),2));
    assertEquals(n300, round(bilNR_detail.get(2),2));
    assertEquals(n300_2, round(bilNR_detail.get(3),2));
    assertEquals(n900, round(bilNR_detail.get(4),2));
    assertEquals(eb, round(bilNR_detail.get(5),2));
}

@Test
@DisplayName("BVA of calcNoDiscBilR Test 2 number 3")

```

```

void calcNoDiscBilR_EP_Test2_3() {
    EBService_Imp imp = new EBService_Imp();
    int veconsumption = 299;
    ArrayList<Double> bilNR_detail = imp.calcNoDiscBilR(veconsumption);
    double eb = 76.67;
    double f200 = 43.6;
    double n100 = 33.07;
    double n300 = 0;
    double n300_2 = 0;
    double n900 = 0;
    assertEquals(f200, round(bilNR_detail.get(0),2));
    assertEquals(n100, round(bilNR_detail.get(1),2));
    assertEquals(n300, round(bilNR_detail.get(2),2));
    assertEquals(n300_2, round(bilNR_detail.get(3),2));
    assertEquals(n900, round(bilNR_detail.get(4),2));
    assertEquals(eb, round(bilNR_detail.get(5),2));
}

@Test
@DisplayName("BVA of calcNoDiscBilR Test 2 number 4")
void calcNoDiscBilR_EP_Test2_4() {
    EBService_Imp imp = new EBService_Imp();
    int veconsumption = 300;
    ArrayList<Double> bilNR_detail = imp.calcNoDiscBilR(veconsumption);
    double eb = 77;
    double f200 = 43.6;
    double n100 = 33.4;
    double n300 = 0;
    double n300_2 = 0;
    double n900 = 0;
    assertEquals(f200, round(bilNR_detail.get(0),2));
    assertEquals(n100, round(bilNR_detail.get(1),2));
    assertEquals(n300, round(bilNR_detail.get(2),2));
    assertEquals(n300_2, round(bilNR_detail.get(3),2));
    assertEquals(n900, round(bilNR_detail.get(4),2));
    assertEquals(eb, round(bilNR_detail.get(5),2));
}

@Test
@DisplayName("BVA of calcNoDiscBilR Test 3 number 1")
void calcNoDiscBilR_EP_Test3_1() {
    EBService_Imp imp = new EBService_Imp();
    int veconsumption = 301;
    ArrayList<Double> bilNR_detail = imp.calcNoDiscBilR(veconsumption);
    double eb = 77.52;
    double f200 = 43.6;
    double n100 = 33.4;
    double n300 = 0.52;
    double n300_2 = 0;
    double n900 = 0;
    assertEquals(f200, round(bilNR_detail.get(0),2));
    assertEquals(n100, round(bilNR_detail.get(1),2));
    assertEquals(n300, round(bilNR_detail.get(2),2));
    assertEquals(n300_2, round(bilNR_detail.get(3),2));
    assertEquals(n900, round(bilNR_detail.get(4),2));
    assertEquals(eb, round(bilNR_detail.get(5),2));
}

@Test
@DisplayName("BVA of calcNoDiscBilR Test 3 number 2")
void calcNoDiscBilR_EP_Test3_2() {
    EBService_Imp imp = new EBService_Imp();
    int veconsumption = 302;
    ArrayList<Double> bilNR_detail = imp.calcNoDiscBilR(veconsumption);
    double eb = 78.03;
    double f200 = 43.6;
    double n100 = 33.4;
    double n300 = 1.03;
    double n300_2 = 0;
}

```



```

        double n900 = 0;
        assertEquals(f200, round(bilNR_detail.get(0),2));
        assertEquals(n100, round(bilNR_detail.get(1),2));
        assertEquals(n300, round(bilNR_detail.get(2),2));
        assertEquals(n300_2, round(bilNR_detail.get(3),2));
        assertEquals(n900, round(bilNR_detail.get(4),2));
        assertEquals(eb, round(bilNR_detail.get(5),2));
    }

@Test
@DisplayName("BVA of calcNoDiscBilR Test 3 number 3")
void calcNoDiscBilR_EP_Test3_3() {
    EBSservice_Imp imp = new EBSservice_Imp();
    int veconsumption = 599;
    ArrayList<Double> bilNR_detail = imp.calcNoDiscBilR(veconsumption);
    double eb = 231.28;
    double f200 = 43.6;
    double n100 = 33.4;
    double n300 = 154.28;
    double n300_2 = 0;
    double n900 = 0;
    assertEquals(f200, round(bilNR_detail.get(0),2));
    assertEquals(n100, round(bilNR_detail.get(1),2));
    assertEquals(n300, round(bilNR_detail.get(2),2));
    assertEquals(n300_2, round(bilNR_detail.get(3),2));
    assertEquals(n900, round(bilNR_detail.get(4),2));
    assertEquals(eb, round(bilNR_detail.get(5),2));
}

@Test
@DisplayName("BVA of calcNoDiscBilR Test 3 number 4")
void calcNoDiscBilR_EP_Test3_4() {
    EBSservice_Imp imp = new EBSservice_Imp();
    int veconsumption = 600;
    ArrayList<Double> bilNR_detail = imp.calcNoDiscBilR(veconsumption);
    double eb = 231.8;
    double f200 = 43.6;
    double n100 = 33.4;
    double n300 = 154.8;
    double n300_2 = 0;
    double n900 = 0;
    assertEquals(f200, round(bilNR_detail.get(0),2));
    assertEquals(n100, round(bilNR_detail.get(1),2));
    assertEquals(n300, round(bilNR_detail.get(2),2));
    assertEquals(n300_2, round(bilNR_detail.get(3),2));
    assertEquals(n900, round(bilNR_detail.get(4),2));
    assertEquals(eb, round(bilNR_detail.get(5),2));
}

@Test
@DisplayName("BVA of calcNoDiscBilR Test 4 number 1")
void calcNoDiscBilR_EP_Test4_1() {
    EBSservice_Imp imp = new EBSservice_Imp();
    int veconsumption = 601;
    ArrayList<Double> bilNR_detail = imp.calcNoDiscBilR(veconsumption);
    double eb = 232.35;
    double f200 = 43.6;
    double n100 = 33.4;
    double n300 = 154.8;
    double n300_2 = 0.55;
    double n900 = 0;
    assertEquals(f200, round(bilNR_detail.get(0),2));
    assertEquals(n100, round(bilNR_detail.get(1),2));
    assertEquals(n300, round(bilNR_detail.get(2),2));
    assertEquals(n300_2, round(bilNR_detail.get(3),2));
    assertEquals(n900, round(bilNR_detail.get(4),2));
    assertEquals(eb, round(bilNR_detail.get(5),2));
}

```

```

@Test
@DisplayName("BVA of calcNoDiscBilR Test 4 number 2")
void calcNoDiscBilR_EP_Test4_2() {
    EBSERVICE_imp imp = new EBSERVICE_imp();
    int veconsumption = 602;
    ArrayList<Double> bilNR_detail = imp.calcNoDiscBilR(veconsumption);
    double eb = 232.89;
    double f200 = 43.6;
    double n100 = 33.4;
    double n300 = 154.8;
    double n300_2 = 1.09;
    double n900 = 0;
    assertEquals(f200, round(bilNR_detail.get(0),2));
    assertEquals(n100, round(bilNR_detail.get(1),2));
    assertEquals(n300, round(bilNR_detail.get(2),2));
    assertEquals(n300_2, round(bilNR_detail.get(3),2));
    assertEquals(n900, round(bilNR_detail.get(4),2));
    assertEquals(eb, round(bilNR_detail.get(5),2));
}

@Test
@DisplayName("BVA of calcNoDiscBilR Test 4 number 3")
void calcNoDiscBilR_EP_Test4_3() {
    EBSERVICE_imp imp = new EBSERVICE_imp();
    int veconsumption = 899;
    ArrayList<Double> bilNR_detail = imp.calcNoDiscBilR(veconsumption);
    double eb = 395.05;
    double f200 = 43.6;
    double n100 = 33.4;
    double n300 = 154.8;
    double n300_2 = 163.25;
    double n900 = 0;
    assertEquals(f200, round(bilNR_detail.get(0),2));
    assertEquals(n100, round(bilNR_detail.get(1),2));
    assertEquals(n300, round(bilNR_detail.get(2),2));
    assertEquals(n300_2, round(bilNR_detail.get(3),2));
    assertEquals(n900, round(bilNR_detail.get(4),2));
    assertEquals(eb, round(bilNR_detail.get(5),2));
}

@Test
@DisplayName("BVA of calcNoDiscBilR Test 4 number 4")
void calcNoDiscBilR_EP_Test4_4() {
    EBSERVICE_imp imp = new EBSERVICE_imp();
    int veconsumption = 900;
    ArrayList<Double> bilNR_detail = imp.calcNoDiscBilR(veconsumption);
    double eb = 395.6;
    double f200 = 43.6;
    double n100 = 33.4;
    double n300 = 154.8;
    double n300_2 = 163.8;
    double n900 = 0;
    assertEquals(f200, round(bilNR_detail.get(0),2));
    assertEquals(n100, round(bilNR_detail.get(1),2));
    assertEquals(n300, round(bilNR_detail.get(2),2));
    assertEquals(n300_2, round(bilNR_detail.get(3),2));
    assertEquals(n900, round(bilNR_detail.get(4),2));
    assertEquals(eb, round(bilNR_detail.get(5),2));
}

@Test
@DisplayName("BVA of calcNoDiscBilR Test 5 number 1")
void calcNoDiscBilR_EP_Test5_1() {
    EBSERVICE_imp imp = new EBSERVICE_imp();
    int veconsumption = 901;
    ArrayList<Double> bilNR_detail = imp.calcNoDiscBilR(veconsumption);
    double eb = 396.17;
    double f200 = 43.6;
    double n100 = 33.4;

```

```

        double n300 = 154.8;
        double n300_2 = 163.8;
        double n900 = 0.57;
        assertEquals(f200, round(bilNR_detail.get(0),2));
        assertEquals(n100, round(bilNR_detail.get(1),2));
        assertEquals(n300, round(bilNR_detail.get(2),2));
        assertEquals(n300_2, round(bilNR_detail.get(3),2));
        assertEquals(n900, round(bilNR_detail.get(4),2));
        assertEquals(eb, round(bilNR_detail.get(5),2));
    }

@Test
@DisplayName("BVA of calcNoDiscBilR Test 5 number 2")
void calcNoDiscBilR_EP_Test5_2() {
    EBSservice_Imp imp = new EBSservice_Imp();
    int veconsumption = 902;
    ArrayList<Double> bilNR_detail = imp.calcNoDiscBilR(veconsumption);
    double eb = 396.74;
    double f200 = 43.6;
    double n100 = 33.4;
    double n300 = 154.8;
    double n300_2 = 163.8;
    double n900 = 1.14;
    assertEquals(f200, round(bilNR_detail.get(0),2));
    assertEquals(n100, round(bilNR_detail.get(1),2));
    assertEquals(n300, round(bilNR_detail.get(2),2));
    assertEquals(n300_2, round(bilNR_detail.get(3),2));
    assertEquals(n900, round(bilNR_detail.get(4),2));
    assertEquals(eb, round(bilNR_detail.get(5),2));
}

@Test
@DisplayName("BVA of calcNoDiscBilR Test 6 number 1")
void calcNoDiscBilR_EP_Test6_1() {
    EBSservice_Imp imp = new EBSservice_Imp();
    int veconsumption = -1;
    ArrayList<Double> bilNR_detail = imp.calcNoDiscBilR(veconsumption);
    double eb = 0;
    double f200 = 0;
    double n100 = 0;
    double n300 = 0;
    double n300_2 = 0;
    double n900 = 0;
    assertEquals(f200, round(bilNR_detail.get(0),2));
    assertEquals(n100, round(bilNR_detail.get(1),2));
    assertEquals(n300, round(bilNR_detail.get(2),2));
    assertEquals(n300_2, round(bilNR_detail.get(3),2));
    assertEquals(n900, round(bilNR_detail.get(4),2));
    assertEquals(eb, round(bilNR_detail.get(5),2));
}

@Test
@DisplayName("calcTaxR Test 1")
void calcTaxR_Test1() {
    EBSservice_Imp imp = new EBSservice_Imp();
    int veconsumption = 600;
    ArrayList<Double> bilNR_detail = imp.calcNoDiscBilR(veconsumption);
    double eb = 231.8;
    double t = 0;
    double tax = imp.calcTaxR(veconsumption, eb, 0);
    assertEquals(t, round(tax,2));
}

@Test
@DisplayName("calcTaxR Test 2")
void calcTaxR_Test2() {
    EBSservice_Imp imp = new EBSservice_Imp();
    int veconsumption = 1000;
    ArrayList<Double> bilNR_detail = imp.calcNoDiscBilR(veconsumption);

```

```

        double eb = 452.7;
        double t = 13.25;
        double tax = imp.calcTaxR(veconsumption, eb, 0);
        assertEquals(t, round(tax,2));
    }

    @Test
    @DisplayName("BVA of calcTaxR Test 1")
    void BAV_calcTaxR_Test1() {
        EBSERVICE_imp imp = new EBSERVICE_imp();
        int veconsumption = 600;
        ArrayList<Double> bilNR_detail = imp.calcNoDiscBilR(veconsumption);
        double eb = 231.8;
        double t = 0;
        double tax = imp.calcTaxR(veconsumption, eb, 0);
        assertEquals(t, round(tax,2));
    }

    @Test
    @DisplayName("BVA of calcTaxR Test 2 number 1")
    void BAV_calcTaxR_Test2_1() {
        EBSERVICE_imp imp = new EBSERVICE_imp();
        int veconsumption = 601;
        ArrayList<Double> bilNR_detail = imp.calcNoDiscBilR(veconsumption);
        double eb = 232.35;
        double t = 0.03;
        double tax = imp.calcTaxR(veconsumption, eb, 0);
        assertEquals(t, round(tax,2));
    }

    @Test
    @DisplayName("BVA of calcTaxR Test 2 number 2")
    void BAV_calcTaxR_Test2_2() {
        EBSERVICE_imp imp = new EBSERVICE_imp();
        int veconsumption = 602;
        ArrayList<Double> bilNR_detail = imp.calcNoDiscBilR(veconsumption);
        double eb = 232.89;
        double t = 0.07;
        double tax = imp.calcTaxR(veconsumption, eb, 0);
        assertEquals(t, round(tax,2));
    }

    //test round method
    @Test
    @DisplayName("round Test 1")
    void round_Test1() {
        EBSERVICE_imp imp = new EBSERVICE_imp();
        double value = 0.333333;
        int places = -1;
        double e;
        boolean real = false;
        try {
            e = imp.round(value, places);
            //assertEquals(3, e);
        } catch (Exception e1) {
            real = true;
        }
        assertEquals(false, real);
    }

    @Test
    @DisplayName("round Test 2")
    void round_Test2() {
        EBSERVICE_imp imp = new EBSERVICE_imp();
        double value = 0.333333;
        int places = 0;
        double e;

```

```
        boolean real = false;
        try {
            e = imp.round(value, places);
            assertEquals(0, e);
        } catch (Exception e1) {
            real = true;
        }
        assertEquals(false, real);
    }

    @Test
    @DisplayName("round Test 3")
    void round_Test3() {
        EBSERVICE_imp imp = new EBSERVICE_imp();
        double value = 0.333333;
        int places = 1;
        double e;
        boolean real = false;
        try {
            e = imp.round(value, places);
            assertEquals(0.3, e);
        } catch (Exception e1) {
            real = true;
        }
        assertEquals(false, real);
    }

    @Test
    @DisplayName("round Test 4")
    void round_Test4() {
        EBSERVICE_imp imp = new EBSERVICE_imp();
        double value = 0.333333;
        int places = 2;
        double e;
        boolean real = false;
        try {
            e = imp.round(value, places);
            assertEquals(0.33, e);
        } catch (Exception e1) {
            real = true;
        }
        assertEquals(false, real);
    }
}
```