Project

Decision Code Coverage

➢ Part 1: Statement and Decision coverage

1) Description of the tool used and types of coverages it provides:

- The tool used for the statement and decision coverage is **EclEmma** [1].

- It is a free tool provided in Eclipse. EclEmma is based on JaCoCo, which provides code coverage analysis [1].

- It is already provided along with Eclipse installation under Eclipse Public License.

- It is mainly based on Vlad Roubstov's EMMA library [1].

- Eclemma shows execution status in the eclipse. The execution line color represents the coverage. For instance, **Green** lines are **fully** executed, **red** lines are **not** executed and **yellow** lines are **partially** executed [2].

- EclEmma provides **Statement** and **Branch** (also called Decision) Code coverage [3].

- It shows coverage in form of **percentages**, where we can also check the (number of) **total instructions**, **covered instructions**, and **missed instructions**.

- We can expand functions and test cases to check which **test case** covers **how many instructions**.

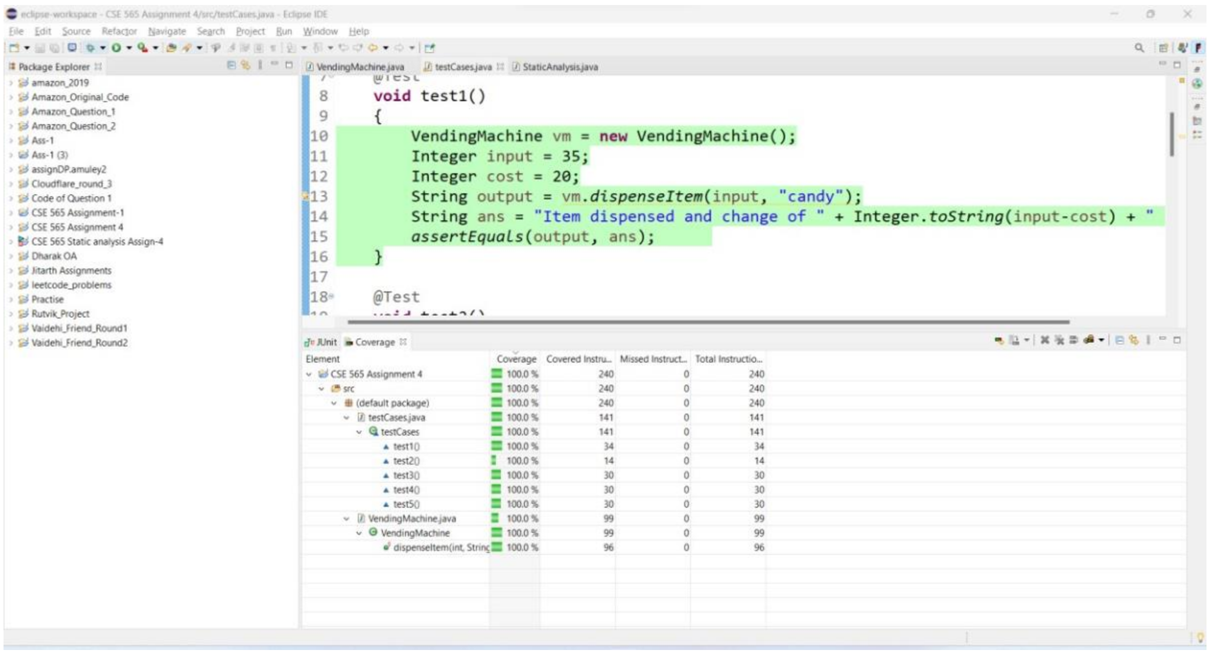
2) **Set of Junit test cases developed by a student. Submit Junit cases in java format.** (Java file is also attached to the submission)

- testCases.java [Package: Default package]

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


class testCases {

   @Test
   void test1()
   {
         VendingMachine vm = new VendingMachine();
         Integer input = 35;
         Integer cost = 20;
         String output = vm.dispenseItem(input, "candy");
         String ans = "Item dispensed and change of " + 
Integer.toString(input-cost) + " returned";
         assertEquals(output, ans);
   }

   @Test
   void test2()
   {
         Integer input = 25;
         //Integer cost = 25;
         String output = VendingMachine.dispenseItem(input, 
"coke");
         String ans = "Item dispensed.";
         assertEquals(output, ans);
   }

   @Test
   void test3()
   {
         Integer input = 35;
         Integer cost = 45;
         String output = VendingMachine.dispenseItem(input, 
"coffee");
         String ans = "Item not dispensed, missing " + 
Integer.toString(cost-input) + " cents. Can purchase candy or 
coke.";
         assertEquals(output, ans);
   }
   @Test
   void test4()
   {
         Integer input = 22;
         Integer cost = 45;
         String output = VendingMachine.dispenseItem(input, 
"coffee");
         String ans = "Item not dispensed, missing " + 
Integer.toString(cost-input) + " cents. Can purchase candy.";
         assertEquals(output, ans);
   }

   @Test
   void test5()
   {
         Integer input = 15;
         Integer cost = 45;
         String output = VendingMachine.dispenseItem(input, 
"coffee");
         String ans = "Item not dispensed, missing " + 
Integer.toString(cost-input) + " cents. Cannot purchase item.";
         assertEquals(output, ans);
   }
}
```

3) Screenshot showing the coverage achieved for the test cases developed.

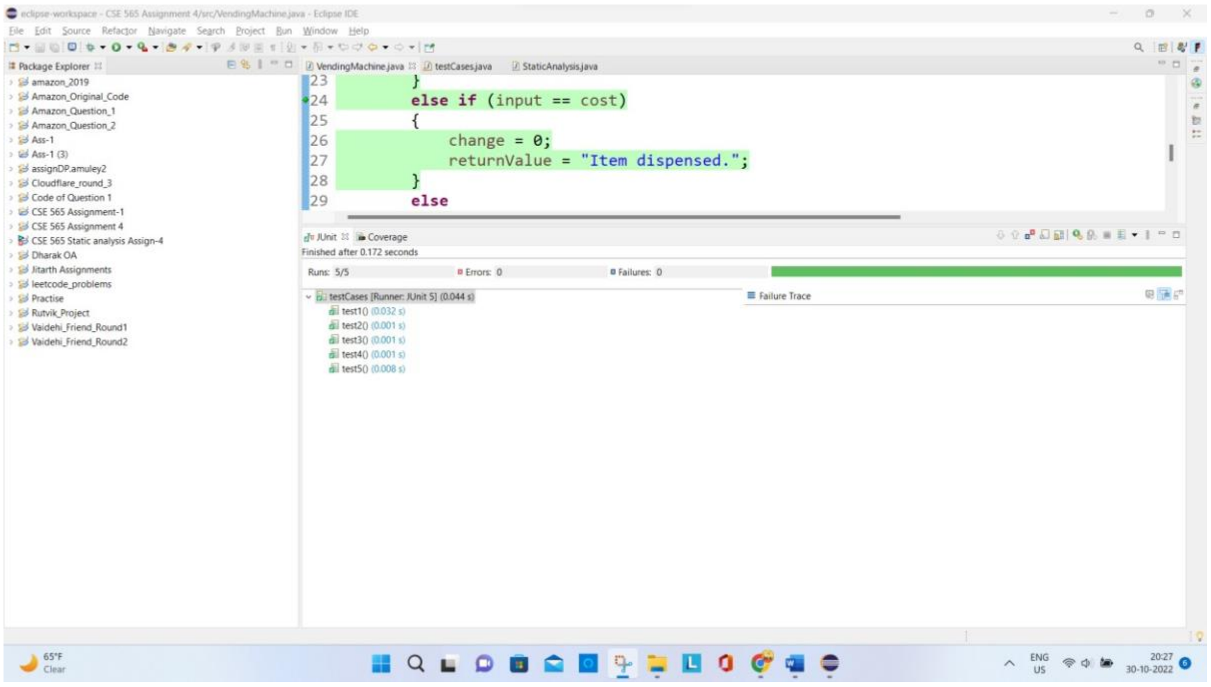
[Coverage: 100%]



```
    void test1()
    {
        VendingMachine vm = new VendingMachine();
        Integer input = 35;
        Integer cost = 20;
        String output = vm.dispenseItem(input, "candy");
        String ans = "Item dispensed and change of " + Integer.toString(input-cost) + "
        assertEquals(output, ans);
    }

    @Test
```

Element	Coverage	Covered Instru...	Missed Instruct...	Total Instructio...
CSE 565 Assignment 4	100.0 %	240	0	240
src	100.0 %	240	0	240
(default package)	100.0 %	240	0	240
testCases.java	100.0 %	141	0	141
testCases	100.0 %	141	0	141
test1()	100.0 %	34	0	34
test2()	100.0 %	14	0	14
test3()	100.0 %	30	0	30
test4()	100.0 %	30	0	30
test5()	100.0 %	30	0	30
VendingMachine.java	100.0 %	99	0	99
VendingMachine	100.0 %	99	0	99
dispenseItem(int, String	100.0 %	96	0	96

[Test cases Passed]



```
            }
            else if (input == cost)
            {
                change = 0;
                returnValue = "Item dispensed.";
            }
            else
```

```
Finished after 0.172 seconds
Runs: 5/5        Errors: 0        Failures: 0

testCases [Runner: JUnit 5] (0.044 s)
    test1() (0.032 s)
    test2() (0.001 s)
    test3() (0.001 s)
    test4() (0.001 s)
    test5() (0.008 s)
```

4) Your evaluation of the Tool’s usefulness:

- I found EclEmma very useful when it comes to checking code coverage.

- It is in-built into Eclipse (provided under Eclipse Public License]. We can create a JUnit test file that automatically adds the **JUnit5 library** to build a path by right-clicking on **Package -> New -> JUnit test**

- We can write our code and right-click on the file and click on **Coverage as -> JUnit** test which will pop up the coverage window.

- To conclude, the tool provides branch coverage (and obviously, statement coverage) and it is **very easy to use**. **No configuration procedure** is needed [3].

- Additionally, it also highlights which statements or branches are fully executed, partially executed, or not executed at all; along with the number of Instructions covered.

References:

[1] https://www.eclemma.org/#:~:text=EclEmma%20is%20a%20free%20Java,be%20analyzed%20for%20code%20coverage

[2] https://www.eclipse.org/community/eclipse_newsletter/2015/august/article1.php

[3] https://stackoverflow.com/questions/48266017/what-kind-of-test-coverage-criteria-eclipse-uses

[4] https://www.tutorialspoint.com/software_testing_dictionary/anomaly.htm