A picture containing icon

Description automatically generated

CS265FZ Software Testing

Lab 4 – Statement Coverage

**There are TWO exercises to be completed.**

**Two pieces of work need to be submitted:**

1. Fill in this lab sheet and submit it to Moodle. You don’t need to attach your source code in this form. You need to upload your source code separately.
2. Submit all the required source code to Moodle. Make sure your source code is tested in Eclipse and is executable.
3. Make sure you provide detailed comments in the source code:
   1. Identify the fault(s) in the source code.
   2. How did you fixed the fault(s)?

In this lab, you will work on structural testing techniques and using *Eclemma* code coverage to see the coverage rate of your tests.

***EclEmma*** is a free Java code coverage tool (more details at <http://eclemma.org/>).

If you don’t see a similar icon in Eclipse shown in Figure 1, you will have to follow ***Part 1*** to install it, otherwise skip Part 1.

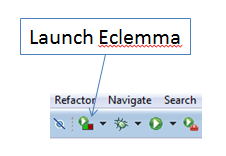


Figure 1.

**Part 1. Installing Eclemma code coverage**

In Eclipse, open ***Help*** -> ***Eclipse Marketplace***, type *EclEmma* in ***Find*** (Search tab) to search *Eclemma*, then follow the instruction to install the *Eclemma coverage* tool, as shown in Figure 2.

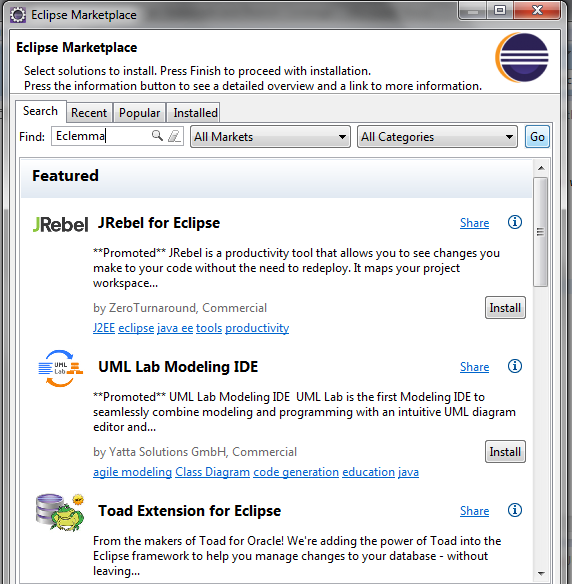


Figure 2.

**Part 2. Code coverage**

The following section explaines how *Eclemma* works, you will apply it to your tests at the end of this document.

In order to check the coverage rate, run the tests with *EclEmma* coverage ().

Then go back to the program under the test to see which part of the source code are covered by the tests as shown in Figure 3 and 4 (each line is considered as a statement).

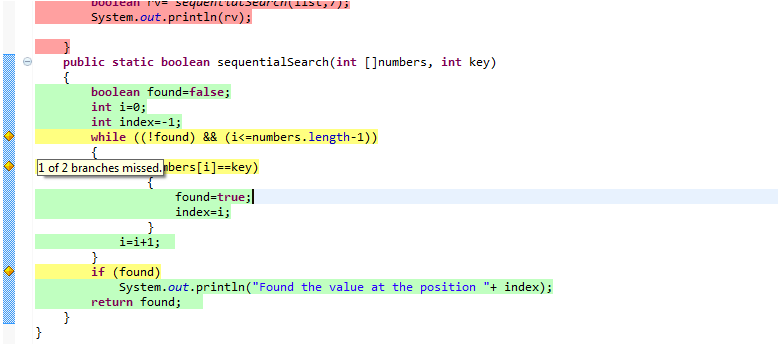
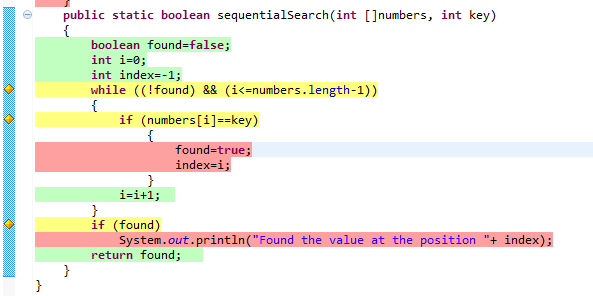


Figure 3. All code are covered by the test (yellow and green coloured)



Figured 4. Red colour indicates the code that are not covered by the test.

Click on the icon “*Link with Current Selection*” to see the coverage rate of the current program under the test (Figure 5).

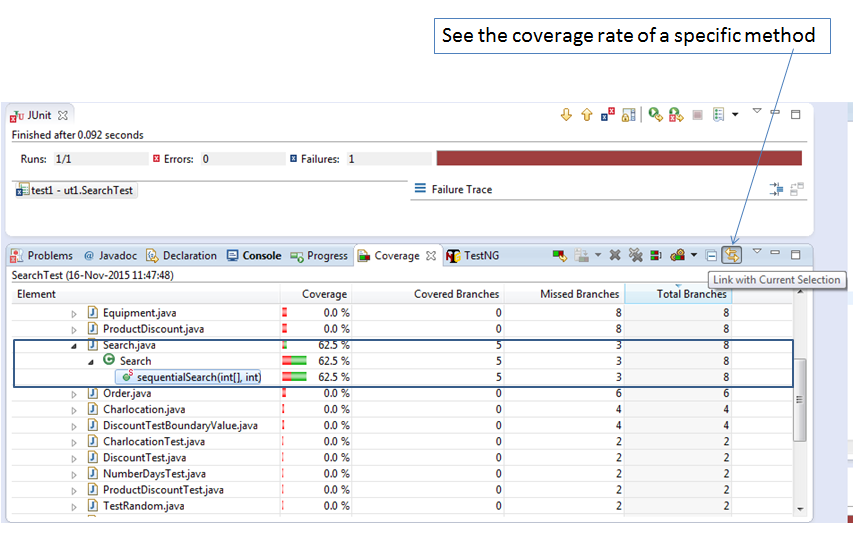


Figure 5. Detailed coverage rate.

We can change the criteria of measurement by clicking on the triangle icon as follows (Figure 6):

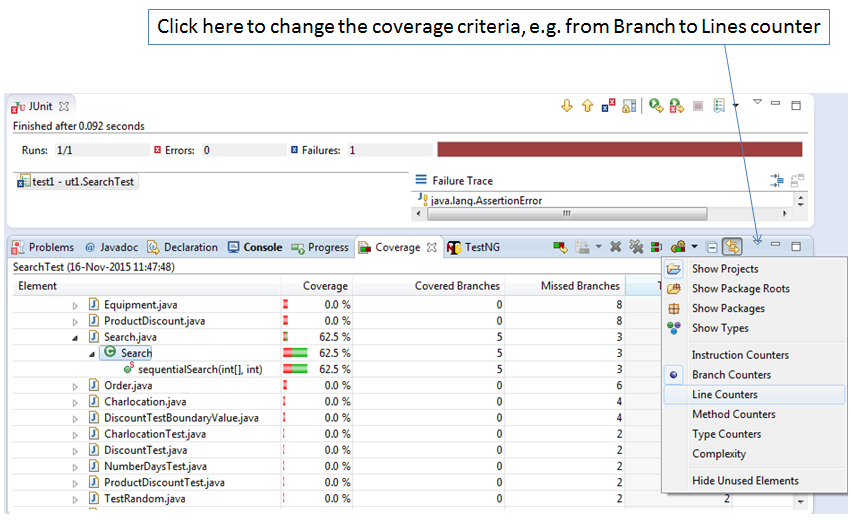


Figure 6. Change coverage feature

**Problem 1**

A program (Source Code: ***Lab4\_Program1.java***) is used to assess the cost of a car insurance policy. It takes three inputs of age, gender, and marital status. If the age entered is less than 16 or greater than 65, the program returns a premium of zero. The input for gender takes the form of ‘M’ for male and ‘F’ for female. If an incorrect value for the gender is entered the program returns a premium of zero. In general, a premium is €500. However, if a person is male, under 25 and not married then an extra €1500 is added to the premium. If the person is female or married the premium falls by €200, and if the person is aged between 45 and 65 inclusive the premium falls by €100.

* ***Task 1:***

Based on the source code (*as shown in* Figure 7), construct the Control Flow Graph of the program.

**A picture containing text

Description automatically generated**

**Figure 7**

***1..8***

***T***

***T***

***9***

***10..12***

***16***

***T***

***17***

***14..15***

***21..24***

***20***

**11**

**10**

***T***

***k***

***a***

***b***

***c***

***d***

***e***

***f***

***g***

***h***

***i***

***j***

***l***

***m***

***n***

***18***

***19***

***13***

* ***Task 2:***

From the Control Flow Graph constructed in Task 1, identify the paths that cover all nodes in the graph (The solution is not unique).

**Path1: 1 🡪 2 🡪 11**

**Path2: 1 🡪 3 🡪 4 🡪 10 🡪 11**

**Path3: 1 🡪 3 🡪 5 🡪 6 🡪 7 🡪 8 🡪 9 🡪 10 🡪 11**

* ***Task 3:***

**Based on the paths identified in Task 2 and the program specification given at the beginning of the Problem 1, generate test data for the statement coverage test.**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test ID** | **Test Cases Covered** | **Inputs** | | | **Expected Output** |
| *age* | *gender* | *married* | *premium* |
| T1.1 | SC-1, 2, 11 | 10 | M | False | 0 |
| T1.2 | SC-[1], 3, 4, 10, [11] | 20 | M | False | 2000 |
| T1.3 | SC-[1], [3], 5, 6, 7, 8. 9, [10], [11] | 50 | F | False | 200 |

* ***Task 4***

Based on the specification given above, write your testing code in JUnit 5 to test the source code of the program provided on Moodle (“***Lab4\_Program1.java***”). Make sure your test code is named as “***Lab4\_Task1.java***”.

* ***Task 5***

Based on the test results, provide the correct version of the “***Lab4\_Program1.java***”, and rename it to “***Lab4\_Program1\_Fix.java***”.

**Problem 2**

A program (**Source Code: *Lab4\_Program2.java***) combines an exam and coursework mark into a single grade. The values for exam and coursework are integers. If the exam or coursework mark is less than 50% (< 50) then the grade returned is a “Fail”. To pass the course with a “Pass, C”, the student must score between 50% and 60% in the exam (50 <= exam < 60), and at least 50% in the coursework (course >= 50). They will pass the course with “Pass, B”, if they score at least 60% in the exam (exam >= 60) and 50% in the coursework (course >= 50) and in addition to this, if the average of the exam and coursework is at least 70%, then they are awarded a “Pass, A”. Input values that are less than 0 or greater than 100 for either the exam or coursework are invalid and the program will return a message to say “Marks out of range”.

* ***Task 1:***

Based on the source code (*as shown in* Figure 8), construct the Control Flow Graph of the program.

**Text

Description automatically generated**

Figure 8

A picture containing indoor, table, clock, small

Description automatically generated

* ***Task 2:***

From the Control Flow Graph constructed in Task 1, identify the paths that cover all nodes in the graph (The solution is not unique).

**Path1: 1 🡪 2 🡪 11**

**Path2: 1 🡪 3 🡪 4 🡪 10 🡪 11**

**Path3: 1 🡪 3 🡪 5 🡪 6 🡪 10 🡪 11**

**Path4: 1 🡪 3 🡪 5 🡪 7 🡪 8 🡪 10 🡪 11**

**Path5: 1 🡪 3 🡪 5 🡪 7 🡪 9 🡪 10 🡪 11**

* ***Task 3:***

**Based on the paths identified in Task 2 and the program specification given at the beginning of the Problem 1, generate test data for the statement coverage test.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test ID** | **Test Cases Covered** | **Inputs** | | **Expected Outputs** |
| *exam* | *course* | *Result* |
| T1.1 | SC-1, 2, 11 | -100 | 55 | “Marks out of range” |
| T1.2 | SC-[1], 3, 4, 10, [11] | 25 | 55 | “Fail” |
| T1.3 | SC-[1], [3], 5, 6, [10], [11] | 55 | 55 | “Pass, C” |
| T1.4 | SC-[1], [3], [5], 7, 8, [10], [11] | 90 | 55 | “Pass, A” |
| T1.5 | SC-[1], [3], [5], [7], 9, [10], [11] | 65 | 55 | “Pass, B” |

* ***Task 4***

Based on the specification given above, write your testing code in JUnit 5 to test the source code of the program provided on Moodle (“***Lab4\_Program2.java***”). Make sure your test code is named as “***Lab4\_Task2.java***”.

* ***Task 5***

Based on the test results, provide the correct version of the “***Lab4\_Program2.java***”, and rename it to “***Lab4\_Program2\_Fix.java***”.