Assignment 4

Code Coverage & White-box testing

Due Date: Feb. 6, 2020

Yasaman Amannejad, 2020

Table of Contents

1 SUMMARY

This assignment has a similar focus to assignment 2, as it is once again unit testing. Unit testing will be performed using JUnit in Eclipse. As with the previous assignments, students will start by familiarizing themselves with the usage of the testing tools followed by implementation (enhancement) of the test suite. The major difference between the testing being performed in this assignment and assignment (#2) is that this assignment shows the students a different technique in deciding what test cases to develop. To develop the test cases in this assignment, instead of basing it on the requirements of the code, students will base their test cases on the control and data flow of the code for the SUT.
The objectives of this assignment are to introduce students to the concepts of determining the adequacy of a white-box test suite based on code coverage. In white-box testing, it is important to measure the adequacy of a test suite based on completeness defined by the portion of the code which is exercised. This definition can take several forms, including control-flow coverage criteria: statement (or node) coverage, branch (or edge) coverage, condition coverage, path coverage or data-flow coverage criteria.

After completing the assignment, students will be able:

- To use code coverage tools to measure test adequacy and become aware of similar tools for other programming environments
- To gain an understanding of how data-flow coverage works and be able to calculate it by hand

1.1 SOFTWARE UNDER TEST

The system to be tested for this assignment is JFreeChart, the same SUT used in Assignment #2. To get started with the JFreeChart system, download and use the "JFreeChart v2.0.zip" file.

1.2 SETUP YOUR TEST ENVIRONMENT

Create a new project and copy-past the source files from the given zip folder into the source folder of your new project. Then, add the given libraries to the class path of your project. Then, copy all your tests from assignment 2 and assignment 3 into this new project. You should be able to run your tests as before. Now, you are ready to continue to work on this assignment.

If you need more details for setting up your project, please read Section 2.

1.3 GROUP WORK

This assignment should be completed in groups that you formed in the first assignment. The report will also be completed as a group.

1.4 ASSIGNMENT PROCESS

Control flow - In this assignment, you will first calculate the test coverage for the test cases that you developed in the previous assignments. Then, you will write more test cases to increase the control flow coverage.

Data flow - For only the `Range.constrain(double)` method of the class `Range` under the `org.jfree.data` package, calculate the data-flow coverage by tracing through the execution of each of your test cases manually. Complete your assignment report and submit your report and test codes in BlackBoard.

Finally, each member of the team must complete the self-assessment and teamwork evaluation, individually: https://forms.gle/ojvJhnqmNUraBRrr7

1.5 DELIVERABLES

- All test codes (Zip your test folder and submit it in Blackboard)
- Assignment report
- Self-assessment and teamwork evaluation (*Only this part requires an **individual** submission*).

Marking scheme is provided for you in Section 4.

2 SET UP YOUR PROJECT

2.1 CREATE AN ECLIPSE PROJECT

1. If you haven't done so already, download the JFreeChart v2.0.zip file from BlackBoard.
2. Extract the contents of the .zip file into a known location.
3. Open Eclipse. Open the *New Project* dialog by selecting the *File -> New -> Project…*
4. Ensure that *Java Project* is selected and click *Next.*
5. The dialog should now be prompting for the project name. Enter *JFreeChart_A4* in the *Project Name* field.
6. Then, extract the source files from JFreeChart v2.0.zip and add the source files into the source folder of you project.

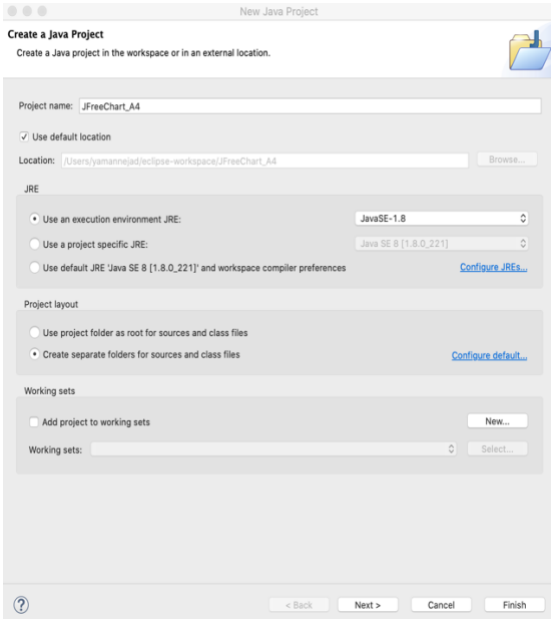


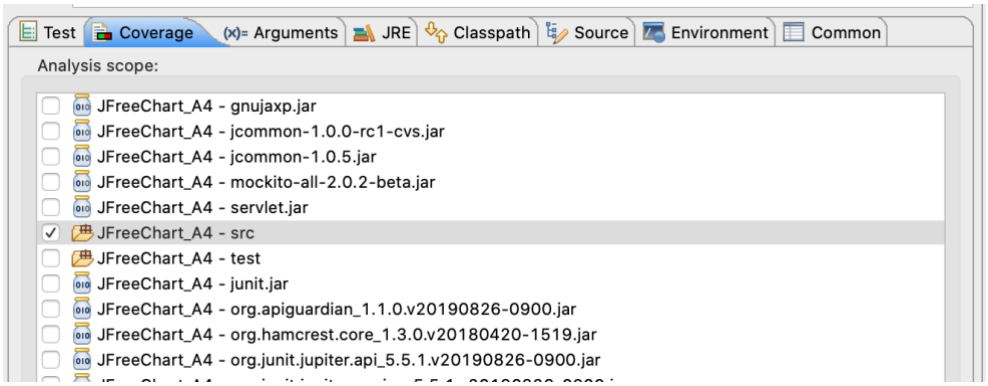
Figure 1 - New Java Project dialog with name and source path filled in

7. Add the given jar (library) files to the class path of your project.
8. You can run the demo applications, by running the classes under *org.jfree.chart.demo*.
9. The project (SUT) is now set up and ready.

2.2 IMPORT A TEST SUITE

For the purpose of demonstrating the abilities of coverage tools, part of the test suite developed in assignment 2 and assignment 3 will be used.

10. Create a source folder in your project and call it “test”.
11. Add JUnit library to your project (you can simply add it by creating a JUnit test case).
12. Copy and paste your test codes from assignment 2 and assignment 3 into test folder of your new project or follow the following steps to import them.
13. Now, you should be able to run your tests.



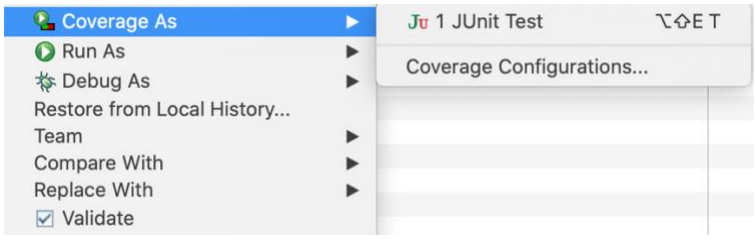
3 INSTRUCTIONS

In this section, you will be required to create (or improve) unit tests for several classes. The classes to be tested are: `org.jfree.data.DataUtilities`, `org.jfree.data.Range`, and `org.jfree.data.DefaultKeyedValues`. Note that although the focus in adequacy criteria has changed (it is now on source code), to develop the test cases the test oracle should still be derived from the requirements (as contained in the Javadocs of the SUT – you cannot assume that the output of the code under test is correct).

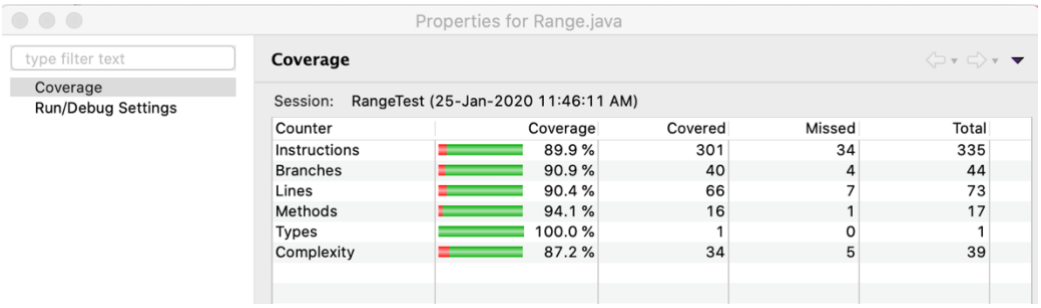
3.1 MEASURING TEST COVERAGE

14. **Measuring coverage** - Using a coverage analysis tool such as EclEmma (it should be included in Eclipse, by default), calculate the coverage for your **classes under test**.

    You can run your tests with the coverage tool:



Then, go to the class under test and open the properties of your class and check the coverage metrics.



3.2 MEASURE DATA FLOW COVERAGE MANUALLY

15. To become more familiar with data flow coverage and achieve a deeper understanding of how coverage calculations work, calculate the coverage for an individual method by hand.

16. For only the `Range.constrain(double)` method of the class `Range` under the `org.jfree.data` package, calculate the data-flow coverage by tracing through the execution of each of your test cases manually. This will need to be included in your report. For an example of how to do this, see Appendix.

3.3 TEST CASE DEVELOPMENT

17. You should expand your test cases to meet the following coverage for each of the classes under test.

    a Minimum coverage:

    i 100% method coverage

    ii 90% statement coverage

    iii 80% branch coverage

Notes:

1) If the coverage tool that you are using does not support any of the above metrics, replace it with another metric that the tool supports.
2) If you cannot meet any of these criteria, you should explain it in your report.
3) Even after you meet the above requirements, it is recommended that you continue designing new test cases to increase the coverage of your tests, as much as you can.
4) The classes under test have random defects in them intentionally, and thus several of your tests should fail. Therefore, to develop test oracles in your test code, you need to follow the specifications, not the actual results by the SUT code.

Upon completion of the tests, review each others tests, looking for any inconsistencies or defects in the tests themselves. Measure the code coverage (only control flow metrics as listed above) of your entire test suite, and record detailed coverage. Include this information (preferably in a tabular form) in your assignment report.

4 SUBMISSION

4.1 DELIVERABLES

- All test codes (Zip your test folder and submit it in Blackboard)
- Assignment report
- Self-assessment and teamwork evaluation (*Only this part requires an individual submission*).

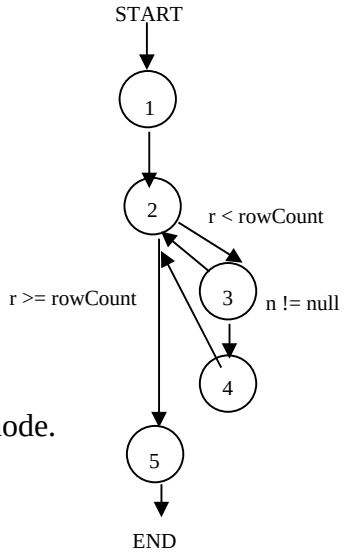
4.2 GRADING SCHEMA

Marking Scheme	
Code coverage: Adherence to the minimum coverage requirement. - Include the coverage information for each class in your assignment report.	20%
Clarity (Test code style, etc.) of the test cases for the newly developed codes or the updated ones.	5%
Adherence to requirements (do they test the code against the requirements?)	5%
Correctness (do the tests actually test what they are intended to test?)	10%
Manual data-flow coverage calculations for `Range.constrain(double)` method.	15%
If you have added any new test case in this step of the assignment, include one of them in your assignment report and describe how this test was not possible to be developed based on Black-box testing	15%
A comparison on the advantages and disadvantages of requirements-based test generation and coverage-based test generation.	15%
Self-assessment and teamwork evaluation (Individual submission)	15%
Any difficulties encountered, challenges overcome, and lessons learned from performing the assignment.	+1%

5 APPENDIX

For demonstration purposes, the `DataUtilities.calculateColumnTotal` method is analyzed next. To start with, the code is inspected, and the control/data-flow are shown.

```
public static double calculateColumnTotal(Values2D data, int column) {
    double total = 0.0;
    int rowCount = data.getRowCount();
    for (int r = 0; r < rowCount; r++) {
       Number n = data.getValue(r, column);
       if (n != null) {
          total += n.doubleValue();
       }
    }
    return total;
}
```



From the control-flow graph, we find all definitions and uses at each node.

Node	Defines	c-uses	p-uses
1	data, column, tota rowCount	data	
2	r	r	r, rowCount
3	n	data, column	n
4	total	total, n	
5		total	

From here, the table can be further refined to include all the definition-clear-use paths, with the computation uses and the predicate uses clarified.

- *dcu(v, i)* is the set of definition-clear-computation-use paths if the variable v is defined at node i
- *dpu(v, i)* is the set of definition-clear-predicate-use paths if the variable v is defined at node i

Node	dcu(v, i)	dpu(v, i)
1	dcu(data, 1) = {3} dcu(column, 1) = {3} dcu(total, 1) = {4, 5}	
2	dcu(r, 2) = {2, 3}	dpu(rowCount, 1) = {(2, 3), (2, 5)} dpu(r, 2) = {(2, 3), (2, 5)}
3	dcu(n, 3) = {4}	dpu(n, 3) = {(3,4),(3,2)}
4	dcu(total, 4) = {4,5}	
5		
	Total number of def-clear c-use paths to cover: 9	Total number of def-clear p-use pat to cover: 6

At this stage, the all uses, all computation uses or all predicate uses can be calculated by tracing the nodes covered and calculating the percentage of uses covered. For example, if for a particular test case, nodes 1, 2, 5 are executed (when the test input is an empty set of Value2D data), then the following definition-clear paths from the above table will be covered:

- def of total in node 1 and its c-use at node 5: 1-2-5: 1 instance in above table
- def of rowCount in node 1 and p-use in edge (2, 5): 1-2-5: 1 instance in above table
- def of r in node 2 and p-use in edge (2, 5): 1-2-5: 1 instance in above table

Thus, 3 out of the total 15 (=9+6) definition-clear paths would be covered, yielding an all-uses coverage ratio of 20% (=3/15).