Assignment 3

Whitebox Testing & Continuous Integration

Due Date: March 4, 2021

Yasaman Amannejad, 2021

1 SUMMARY

The objectives of this assignment are to introduce students to the concepts of determining the adequacy of a whitebox test suite based on code coverage, and practice working with continuous integration.

1.1 SOFTWARE UNDER TEST

The software under test for this assignment is a different version of JFreeChart. 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 into this new project. You should be able to run your tests as before (do not forget to import JUnit library). Now, you are ready to continue developing more test cases.

You may create a Github project for your code at this point or you can create it after you finish writing your test cases. You must create a workflow action in your repository to run your tests every time a new change is added to your project.

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

In this assignment, you will first calculate the code 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.

1.5 DELIVERABLES

- All test codes (Zip your test folder and submit it in Blackboard)
- Assignment report (in Blackboard)
- Share your Github project with Github user: comp3505

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_A3* in the *Project Name* field. Click finish and "Do not Create" module-info.
6. Then, extract the source files from JFreeChart v2.0.zip and add the source files into the source folder of you project.

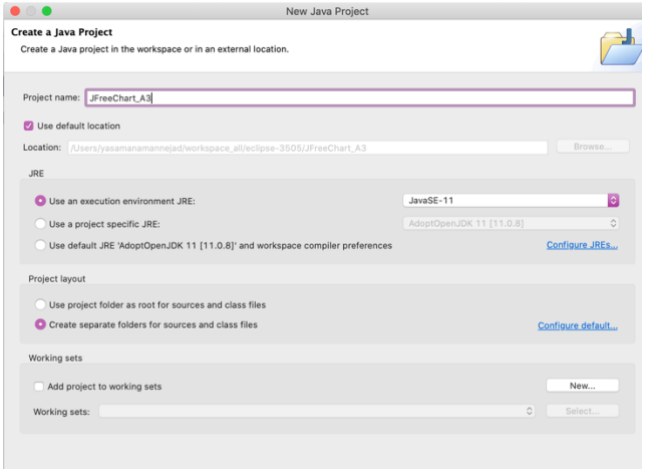


Figure 1 - New Java Project dialog with name and source path filled in

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 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 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

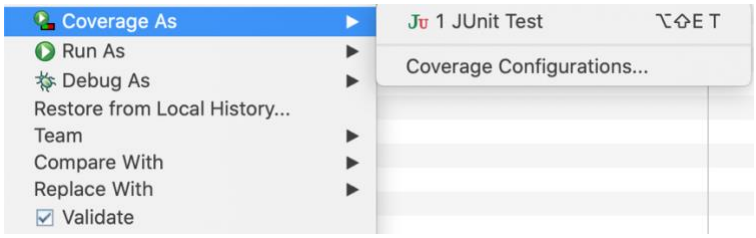
In this section, you will be required to create (or improve) unit tests for the two classes of the project. The classes to be tested are: `org.jfree.data.DataUtilities`, `org.jfree.data.Range`.

Note that although the focus in adequacy criteria has changed (it is now on source code), to develop the test cases the expected values in assertions, i.e., test oracles, 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).

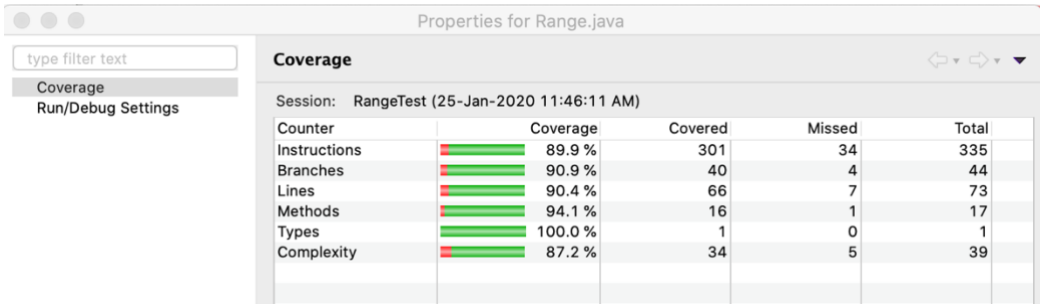
3.1 MEASURING TEST COVERAGE

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** (not your test class) and open the properties of your class and check the coverage metrics.



Counter	Coverage	Covered	Missed	Total
Instructions	89.9 %	301	34	335
Branches	90.9 %	40	4	44
Lines	90.4 %	66	7	73
Methods	94.1 %	16	1	17
Types	100.0 %	1	0	1
Complexity	87.2 %	34	5	39

3.2 TEST CASE DEVELOPMENT

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

    Minimum coverage:

    i 100% method coverage

    ii 90% statement coverage

    iii 70% branch coverage

Upon completion of the tests, review each other's 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 the coverage information (preferably in a tabular form) in your assignment report.

Notes:

1) 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. The provided values are *minimum* criteria.
2) The classes under test have random defects in them intentionally, and thus several of your tests should still fail. Therefore, to develop test oracles in your test code, you need to follow the specifications, not the actual results by the SUT.
3) 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.

3.3 CONTINUOUS INTEGRATION

16. Follow the instructions given for setting up a continuous integration workflow in Github and create a Git action that can run all your tests when a change is submitted to your Github repository (using push or pull request).

17. When you set up your workflow, the tests will fail. This is due to incorrect implementation of the code in SUT. Take a screenshot of your workflow and the failing tests.

18. Each member of the team should fix at least one method in SUT and push the changes to the repository (using push or pull request). This should reduce the number of the failed tests in the repository. Take other screenshots of the workflow and include them in your report. The team members can continue and fix all errors in the SUT, but this step is option.

4 SUBMISSION

4.1 DELIVERABLES

- All test codes (Zip your test folder and submit it in Blackboard)
- Assignment report (in Blackboard)
- Share your Github project with Github user: comp3505

4.2 GRADING SCHEMA

Marking Scheme	
Improving the test code issues based on feedback from assignment 2	5%
Code coverage: Adherence to the minimum coverage requirement. - Include the coverage information for each class in your assignment report.	10%
Clarity (Test code style, etc.) of the new test cases.	5%
Adherence to requirements (do they test the code against the requirements?)	15%
Correctness (do the tests actually test what they are intended to test?)	10%
If you have added any new test case in this step of the assignment, include a few of them (e.g., 3 test cases) in your assignment report and describe how this test was not possible to be developed based on Blackbox testing	15%
A comparison on the advantages and disadvantages of requirements-based test generation and coverage-based test generation.	10%
Continuous integration workflow	20%
Fixing the SUT (at least one method per team member) and re-running the workflow in Github actions. Contribution of each team member on Github must be visible.	10%
Any difficulties encountered, challenges overcome, and lessons learned from performing the assignment.	+1%