Assignment 2

Automated Requirements-Based Unit Testing using JUnit

Due Date: Jan. 23, 2020

Yasaman Amannejad, 2020

Table of Contents

1 SUMMARY

The objective of this assignment is to introduce students to the fundamentals of automated unit testing. The most widely used unit testing tool for Java is the JUnit framework, which is a part of the XUnit framework family. After completing the assignment, students will be able to develop automated test code in JUnit and have an understanding of XUnit testing frameworks.

1.1 SOFTWARE UNDER TEST

The system to be tested for this assignment is JFreeChart. JFreeChart is an open source Java framework for chart calculation, creation and display. To get started with the JFreeChart system, download the "JFreeChart v1.0.zip" file (included in the artifact folder) from Blackboard and extract the entire archive to a known location. You can learn more about JFreeChart in Section 2.

1.2 SETUP YOUR TEST ENVIRONMENT

To set up your JFreeChart project, read Section 3. You need to add JFreeChart and a few other libraries to your project to start.

The main testing tool for this assignment is JUnit. The assignment instruction is written based on JUnit4, however, you can use JUnit 4 or 5 for completing your tasks. The JUnit library will be added to your project when creating your first JUnit test case. Instructions on how to create your first JUnit test case is provided in Section 4. You can skip this section, if you already know how to work with JUnit.

The specification of the functionalities are provided in Javadoc. You should design your test cases based on the method signatures provided in the Javadoc. Extract the Javadoc files (included in the artifact folder), and then start from index.html to see the documentations for the class under test. For more information, you can read Section 5.

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 open the Javadoc file to learn about the specification of the `org.jfree.data.Range` class. This class has 15 methods in total. You do not need to test the `hashcode` method. You will design and write your test cases for 14 methods in this class. You may divide the methods among the team members to complete. When you all finish your test codes, get together and combine all your tests in one project. Create a test suite that can run all these test cases. Complete your assignment report and submit your report and code in BlackBoard.

1.5 DELIVERABLES

- Assignment report.
- Test codes (all test cases and a test suite that runs them all). Include all your test files in a zip folder.

Submit your zip folder and your assignment report in Blackboard. Marking scheme is provided for you in Section 8.

2 SOFTWARE UNDER TEST

The system to be tested for this assignment is JFreeChart. This framework supports many different (graphical) chart types, including pie charts, bar charts, line charts, histograms, and several other chart types. To get started with the JFreeChart system, download the "JFreeChart v1.0.zip" file (included in the artifact folder) from Blackboard and extract the entire archive to a known location. More information on how to get started with these files will be provided in Section **Error! Reference source not found.**. Note that the version of JFreeChart distributed for this assignment do not correspond to the actual releases of JFreeChart. The versions have been modified for the purposes of the assignment.

The JFreeChart framework is intended to be integrated into other systems as a quick and simple way to add charting functionality to other Java applications. With this in mind, the API for JFreeChart is required to be relatively simple to understand, as it is intended to be used by many developers as an open source off-the-shelf framework. An example of different types of charts drawn using JFreeChart is shown in Figure 1.

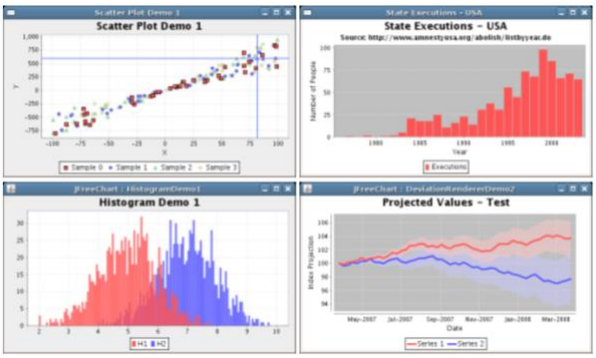

Figure 1 – Example JFreeChart graphs

While the JFreeChart system is not technically a stand-alone application, developers of JFreeChart have created several demo classes which can be executed to show some of the capabilities of the system. These demo classes have *Demo* appended to the class name. For the purpose of this assignment, full knowledge of the usage of the JFreeChart API is not particularly necessary.

The framework is grouped into two main packages, (1) `org.jfree.chart` and (2) `org.jfree.data`. Each of these two packages is also divided into several other smaller packages. For the purpose of testing in this assignment, we will be focusing on the `org.jfree.data` package.

3 SET UP YOUR PROJECT

Ensure that every team member understands the concepts in this section before moving on to the rest of the assignment.

1. If you haven't done so already, download the *JFreeChart v1.0.zip* file (included in the artifact folder) from BlackBoard. Extract the contents of the .zip file into a known location.
2. Open Eclipse.
3. 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* in the *Project Name* field, and then click *Next*.

Add the necessary java libraries

6. The *Java Settings* dialog should now be displayed. This dialog has four tabs along the top: *Source*, *Projects*, *Libraries*, and *Order and Export*. Move to the *Libraries* tab, and click the *Add External JARs (or Libraries)…* button.
7. Select the *jfreechart.jar* file from the known location and click *Open*. Click *Add External Libraries…* again, this time add all the .jar files from the *lib* directory where you have unzipped the *JFreeChart v1.0.zip* file. The Java Settings dialog should now look like Figure 2, below.

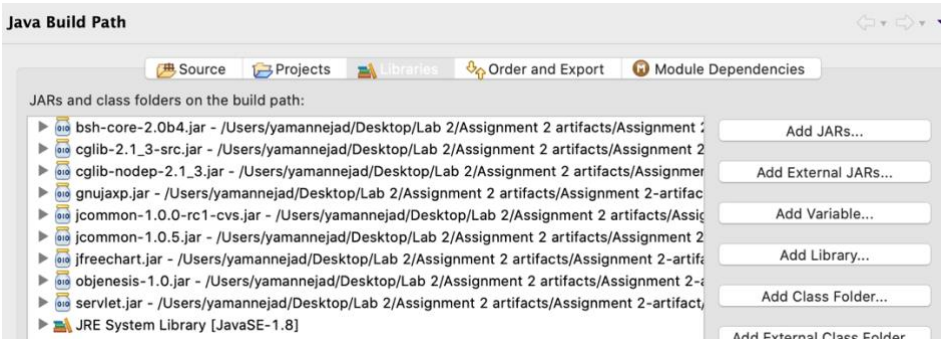

Figure 2 - The Java Settings dialog after adding required archives

8. Click *Finish*. The project (SUT) is now set up and ready for testing. To run the demo classes, in the package explorer expand the *Referenced Libraries* item in the newly created JFreeChart project, exposing the .jar files just added. Right click on the *jfreechart.jar*, and select *Run As → Java Application* (Figure 3).

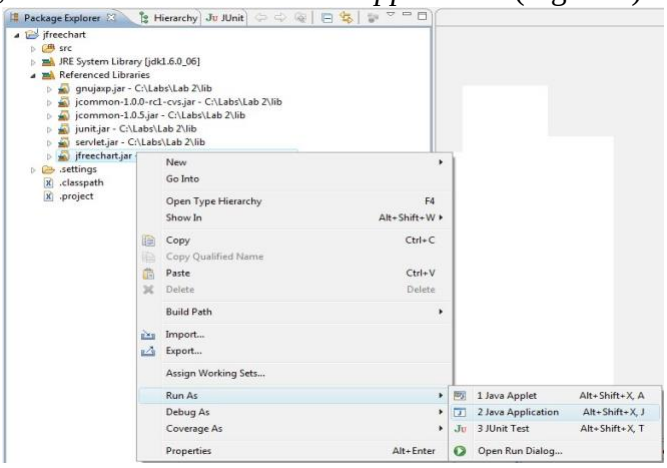

Figure 3 - Running JFreeChart

9. In the *Select Java Application* dialog, select any of the four demo applications (e.g., *CompositeDemo*), and click *OK* as shown in Figure 4.

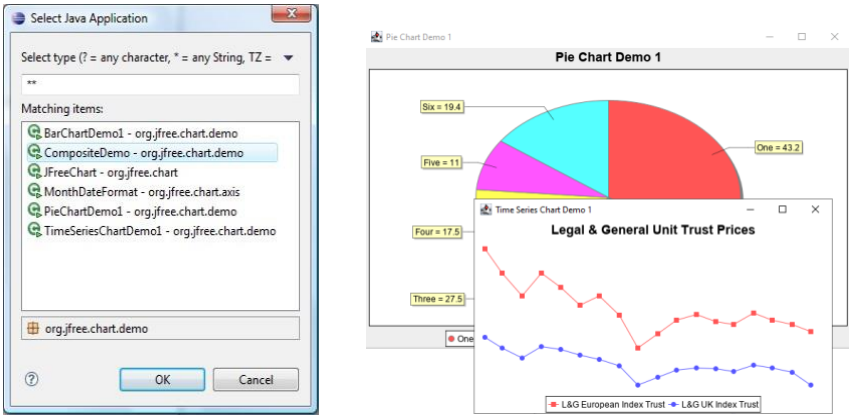

Figure 4 - The Select Java Application dialog

4 CREATE YOUR FIRST JUNIT TEST

If you have experience with creating JUnit tests, you can skip this section, and move to the instruction section.

To create a test suite containing a single unit test in JUnit, follow these steps.

10. In the package explorer, expand the *Referenced Libraries* list item to show all the archives that the project uses.
11. Expand the jfreechart.jar archive to expose all the packages that are contained in that archive.
12. Expand the `org.jfree.data` package within the archive to show all the .class files contained in that package.
13. Finally, expand the `Range.class` item to expose the class contained in that file, along with all the class' methods and fields contained in that class.

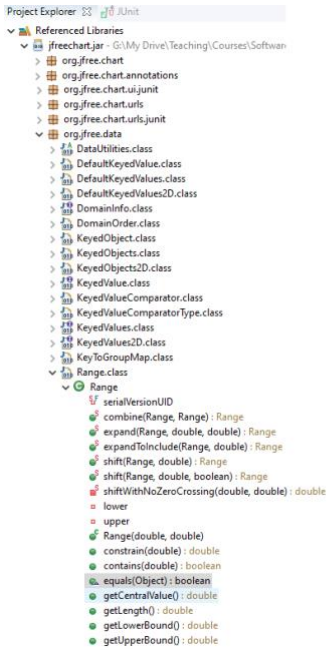

Figure 5 – Viewing the list of method inside a class

14. Click a source folder called test besides your src folder, *NEW -> Source Folder*.
15. Click on the test folder and create a JUnit test case, *File->New -> JUnit Test Case*. Type *RangeTest* in the *Name* field, '`JFreeChart/test`' as the `Source Folder`, and add '`org.jfree.data`' in the *Package* field to create your tests with the same package structure as your source code. NOTE: you can use JUnit 4 or 5 for completing this work. The syntax will be slightly different. Please refer to course slides for the differences.
16. Click *Finish*.

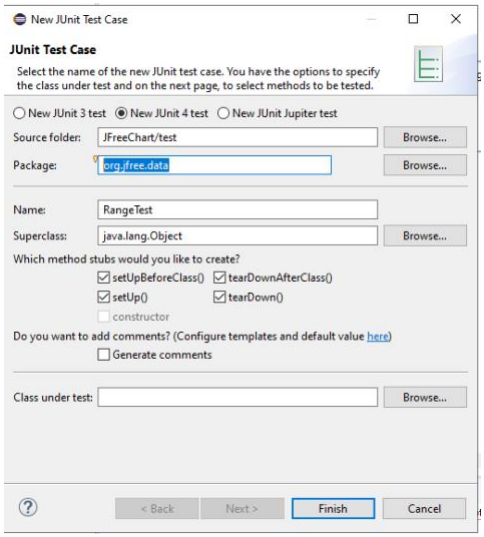

Figure 6 - Creating a new JUnit test case

17. The newly created test class (`RangeTest`) is a JUnit class which provides a unified interface for test cases and suites. A set of test cases which test similar functions of a class may all want to initialize an instance of that class and prepare for the test cases in the same way. The way that JUnit facilitates this is the `setUp()` and `tearDown()` methods. We do not need to use these method in this example. Define the `exampleRange` attribute which is required for the test cases to have access to the object. You may initialize this object in the setup method, or leave it for each test to initialize it with specific values.

```
package org.jfree.data;

import static org.junit.Assert.*;
import org.junit.After;
import org.junit.AfterClass;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Test;

public class RangeTest {
	private Range exampleRange;
	@BeforeClass
	public static void setUpBeforeClass() throws Exception {
	}
	@Before
```

```
	public void setUp() throws Exception {
	}
	@Test
	public void centralValueShouldBeZero() {
		fail("Not yet implemented");
	}
	@After
	public void tearDown() throws Exception {
	}
	@AfterClass
	public static void tearDownAfterClass() throws Exception {
	}
}
```

Figure 7 - Simple `setUp()` and `tearDown()`

18. Test cases in JUnit are individual methods which are usually identified by @Test annotation. These test cases can perform any number of steps, and should follow four phase testing (setup, exercise, verify, and teardown). The test oracle for each test case in JUnit can be implemented in several ways, using various assertions. An example assertion is:

    ```
    assertTrue(("example string").length() == 14);
    ```

    which will always pass (assuming the `length()` method is correct).

    Assertions are tested when a test method is executed, and will cause a test to fail if the assert conditions are not met. For a complete list of assertions, refer to
    JUnit 4: https://junit.org/junit4/javadoc/4.8/org/junit/Assert.html
    JUnit 5: https://junit.org/junit5/docs/5.0.1/api/org/junit/jupiter/api/Assertions.html
19. As a practice, write a simple test case for the `getCentralValue()` method as shown in Figure 8 below. The syntax for `assertEquals` in JUnit 4 is as follows:

    ```
    Public static void assertEquals(
                String message, double expected,double actual,double delta)
    ```

    `assertEquals` asserts that two values are equal. This method can also have a delta value, to allow for some variance when comparing floats.

    NOTE: The syntax of assertEqual in JUnit 5 is different. The message is not the first element.

```
package org.jfree.data;
import static org.junit.Assert.*;
import org.junit.After;
import org.junit.AfterClass;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Test;

public class RangeTest {

	private Range exampleRange;
	@BeforeClass
	public static void setUpBeforeClass() throws Exception {
	}
```

```
	@Before
	public void setUp() throws Exception {
	}

	@Test
	public void centralValueShouldBeZero() {
		exampleRange = new Range(-1, 1);
		assertEquals("The central value of -1 and 1 should be 0", 0, exampleRange.getCentralValue(), .01d);
	}

	@After
	public void tearDown() throws Exception {
	}

	@AfterClass
	public static void tearDownAfterClass() throws Exception {
	}
}
```

Figure 8 – The `RangeTest` test class after addition of a simple test case

20. Now that you have a completed test case, run the test class. To do this, right click on the `RangeTest` class in the Package Explorer and select *Run As -> JUnit Test*.

21. This will change the perspective to the JUnit perspective, and run all the tests within the `RangeTest` class. The test just written should pass, indicated by the JUnit view as shown in Figure 9 below. In JUnit a *Failure* and *Error* are similar, but differ slightly. If you get an error it means that your test method did not execute as expected (i.e., an uncaught exception), a failure, however, means that the execution was as expected, but an assertion failed.

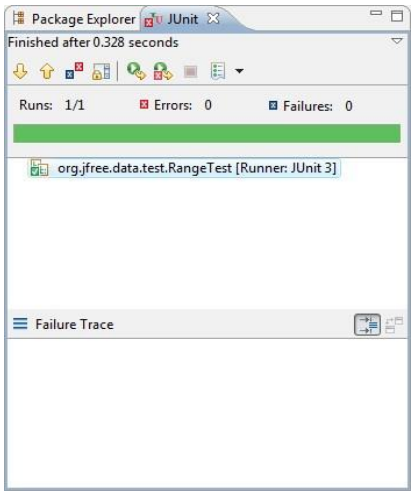

Figure 9 - JUnit view showing passed test

5 NAVIGATE JAVADOC API SPECIFICATIONS

The test generation section of this assignment will require you to generate unit tests for one class in this library based on specifications (requirements) contained in the Javadocs for those classes. If you're already familiar with Javadoc, feel free to skip this section.

Unzip the *JFreeChart–ModifiedJavadoc.zip* file and open the file *index.html*. This is the Javadoc for (a slightly modified version of) JFreeChart. The location of different elements in the Javadoc is shown in Figure 10.

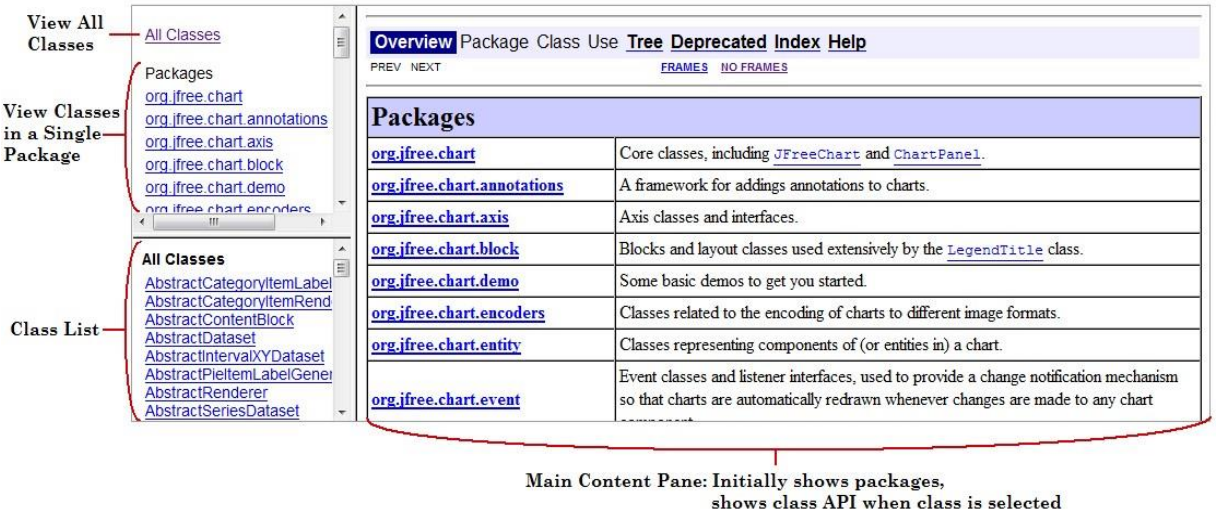

Figure 10 – Javadoc structure

Note that Javadocs can be browsed with all classes shown, or with classes filtered by package. Each of these two approaches has its usefulness. Viewing all classes is useful if you know what the class you are looking for is called, as they are ordered alphabetically. Viewing classes in a single package only is useful for when you're not sure exactly what class you're looking for, but know what area of the code it might be found in.

22. In the list of packages (top-left corner), scroll down to find the `org.jfree.chart.axis` package and click on it. This should show only a few classes in the class list (bottom left corner) now.
23. In the class list, click on the `ColorBar`. The main content pane now shows the API specifications of the `ColorBar`. Scroll down and notice the layout of the specification. At the very top is a description of the class itself (including inheritance information), followed by nested classes, attributes, methods (starting with any constructors), inherited methods, and finally the detailed specification for each method.

Take note of the information available in the *Method Summary* and *Method Detail* sections of the main content pane, as this is what you will be testing methods against (as test oracle) in the following section. For especially effective tests, however, the specifications need to be specific, precise, clear and complete.

6 INSTRUCTIONS

6.1 DEVELOPMENT OF UNIT TEST CODE

24. In this section, you will be required to create unit tests for the following class, testing them against their specifications. This class has 15 methods. You do not need to create test cases for the `hashcode` method. Take a minute to browse the Javadoc API specifications of each of these methods in Javadoc.

    - `org.jfree.data.Range` (in the package `org.jfree.data`).

25. Since you are given the requirements only, you should apply black-box test-case design techniques such equivalence classes, boundary value analysis, and decision table. When applying these techniques, make sure to follow the steps discussed in the class, e.g., first derive the domain for each input variable, then the equivalence classes, etc. You should ensure that the requirements are adequately tested.

26. It is recommend that you practice creating your test-cases on paper first, and then start automating them.

6.1.1 WRITE YOUR TEST CODE BASED ON YOUR TEST-CASE DESIGN

27. The next step is to code your test code in the JUnit framework based on the list of test cases you have designed on paper. Each test method should include one test case only. For example: `testPositiveValuesForMethodX()` and `testNegativeValuesForMethodX()`, instead of a single `testMethodX()`. This will help to keep test cases consistent, and make analysis of test case impact simpler later on. For writing your test methods, please use the example discussed earlier in Section 4.

28. If you have divided the tests and completed them individually, then upon completion of the tests, review each other's tests, looking for any inconsistencies or defects in the tests themselves.

29. Execute the tests you have created on JFreeChart v1.0.zip. If you have multiple JUnit Test Case classes (for example, one class per method under test), create one test suite that can run all your tests.

30. Note that the classes have random defects in them intentionally, and thus several of your tests should fail. **Therefore, to write your test methods, you need to follow the specifications, not the actual results.** Always remember that development code may have defects, and that is why you are testing it.

6.1.2 DISCUSSION - METHODS WITH DEPENDENCY

31. Open the Java doc for `org.jfree.data.DataUtilities` (in the package `org.jfree.data`). This class has 5 methods. Read the documentations for the methods. Some of the methods are dependent on the other interfaces, e.g., Interface `Values2D`.

32. If you were to develop test cases for this class, how can this dependency affect test cases that you develop for `DataUtilities` class. Discuss your answer in your assignment report (At this point, you do not need to develop test cases for this class).

7 SUMMARY

Upon completion of this assignment, students should have a reasonable understanding of unit testing based on requirements using the JUnit framework. Note that unit testing and JUnit are very comprehensive and it takes quite a lot of time to be an expert in them. So do not expect to be JUnit experts just by completing this assignment. If you would like to have a career path in this industry-hot topic, you will need to study this popular framework in more detail and perform more exercises to be skillful.

8 SUBMISSION

8.1 DELIVERABLES

1) Completed assignment report per group.
2) Test codes (all JUnit test case classes and a test suite that runs them all). Include all your test files in a zip folder.

Submit your zip folder and your assignment report in Blackboard.

8.2 MARKING SCHEME

Marking scheme	
Completeness (is there any missing test cases based on the test case design approach?)	35%
Correctness (do the tests actually test what they are intended to test?)	35%
Adherence to the specification in Javadoc (are tests generated only based on the specifications in the Javadoc files)	5%
Clarity (test code style and quality)	10%
Which test case design techniques have you used for designing your test cases, show one example from your test cases for each technique that you have used. (If you have not used a technique in your tests, you do not have to create an example).	10%
Discussion from Section 6.1.2 (about testing methods with dependency on external classes or interfaces)	5%
Difficulties encountered, challenges overcome, and lessons learned from performing the assignment	+1%

ACKNOWLEDGEMENTS

This activity has been originally developed in SoftQual lab and is part of a software-testing laboratory courseware available under a Creative Commons license.