**Assignment 3 - Code Review & White**

**Box Unit Testing**

Version: March 29, 2019

**Due: Monday, April 1st, 11:59pm**

# Objectives

* Understand how to review code and to integrate reviews into source code control.
* Understand white-box testing
* Understand the various forms of code coverage
* Practice best practices in white box unit testing in JUnit and the EclEmma plugin.

## Setup

You will need a few things for this assignment:

* The EclEmma Eclipse plug-in (either in Gradle or Eclipse both will be explained)
* You will maintain a separate documentation file.
* This assignment is meant to be used with JUnit 4. For this assignment I expect you to stick with this version. The Gradle file included is meant for JUnit 4. For your project you can choose if you want to use JUnit 4 or 5. I stick with JUnit 4 since I think it is a bit easier to get started, especially in Gradle.

In this assignment we will look into Code Reviews and revisit Unit Tests but now as Whitebox test.

## Setup

You will need a few things for this assignment:

* You will maintain a separate documentation file.
* This assignment is meant to be used with JUnit 4. For this assignment I expect you to stick with this version. The Gradle file included is meant for JUnit 4. For your project you can choose if you want to use JUnit 4 or 5. I stick with JUnit 4 since I think it is a bit easier to get started, especially in Gradle.

**The master branch DOES NOT CHANGE in this assignment!**

# Task 1: Perform Code Review (10 points)

1. For this assignment we will work with the same code as in assignment 2.
2. Create a new branch based on the "Blackbox" branch (not based off of master) called "Review".
3. Push the review file (given on Canvas) to GitHub as Word document.
4. Import the project into your IDE (or of course use from last time).
5. Review the source code files for conformance to the defects provided in the coding standards document (Coding standards, for general quality practices - Code Smells, and for logic errors). Log any defects using the MS-Word form. You (individually) should at least log 7 defects in the given log file. Be sure to provide the information in that form (ID, Location, Problem description Category (CS, CG, FD or MD defects that do not fit in the other categories). You should log at least one defect for each of the first three categories (CS, CG, FD). It does not count as different defects if you state CS 1 a couple of times (file header). Make sure you find different errors and do not list the same ones over and over.
6. When you are done commit and push your review form to GitHub (overwriting the old one).

# Task 2: Fixing the defects (10 points)

1. Select 3 defects from your Word document from Task 1 (they have to be from different categories).
2. Pick one bug and resolve the defect locally on your computer.
   1. Commit the changes with a good commit message naming the defect id and a description.
   2. Push the changes to your remote repository (to the Review branch).
3. Repeat the steps in 2 for the second and third bug.

Now you should have 3 branches: master (unchanged), Blackbox with your assignment 2 tests and Review with your review changes.

# Task 3: White-box testing (with some black box) - 30 Points

Create a new branch, base this branch off of your "Review" branch and name it "Whitebox". Delete the GivenBlackBox.java test (so your build will pass when you run "gradle build").

Your task is to test the Course.java class. You will see that some methods have some specification of what they should do included (this gives you an idea of what the method is supposed to do - so some black box information).

1. Look at the given GivenWhiteBox.java test. It gives you an idea how you can test.
2. Draw a control flow graph for the calculateAverageWithoutMinWithoutMax() method.
3. Turn on line numbering in Eclipse (Window -> Preferences -> General ->

Editors -> Text Editors)

1. You may hand-draw these, take a photo, and import into your document. USE THE LINE NUMBER as the node label for each statement in your flow graph.
2. List in your document which sequences you need to achieve i) node coverage and ii) edge coverage. Remember that you try to reach each coverage with the least amount of test cases.
3. Create a new test class called CourseTest.java. Write all your test cases into this class (GivenWhiteBox.java should NOT change):
   1. Write a test method for EACH INDIVIDUAL TEST SEQUENCE you identified in 1.c. Yes, some test sequences may address both node and edge coverage and so only need to be written once. Please put a header comment clearly indicating what test sequences the test method addresses.
   2. In case you find any errors in the algorithms - correct them and put SER316start and SER316-end to clearly mark what you changed. Also add a comment why you think this needed changing.
4. Also test the functionality based on the description of the method to make sure it works as it should. In case you find any errors in the the algorithm - correct them and put SER316-start and SER316-end to clearly mark what you changed. Also add a comment why you think this needed changing.
5. You will see two more methods that have comments above them. Create test cases for these as well (no node graph needed) and try to find if they work as intended (specified). Fix them (with marking your changes as before) if needed.
6. All your tests in this test class should pass at the end.
7. Code coverage you can skip the Eclipse parts (-E- and run it through Gradle – see Task 4) you will have to do item c) though (either through Eclipse or through

Gradle)

* 1. -E- Add the EclEmma plugin for Eclipse (in case it is not included in your Eclipse version yet). You can add this to Eclipse by choosing one of the options here: http://eclemma.org/installation.html. Personally I prefer the Update Site through the Eclipse Market place, search for EclEmma.

If you use a different IDE then you might not be able to use EclEmma then you should answer the questions and do the task in the enumerated list after doing the Gradle part in Task 4! Please make a comment about it in your document under Task 3 Code Coverage and continue with Task 4 and when you are done go back here and answer the questions under c).

* 1. -E- Once the plugin has been installed and you have restarted Eclipse, run your Unit Tests again. Then you can view your code coverage results. Go to Window -> Show View -> Other -> Java -> Coverage. For running the test and coverage again right-click on your test suit class in the Package Explorer, choose ”Coverage As...” and ”JUnit Test”, which will re-run your unit tests and also give you a Coverage View in the bottom pane.
  2. In your document:
     1. Cut and paste your code coverage pane and include it in your document (or report when done through Gradle).
     2. Answer: What is the overall code coverage for the program?
     3. Answer: What is the code coverage for Course.java with your tests from CourseTest.java?
     4. If it is not already, include more Unit Tests into your test class (CourseTest.java) until you reach code coverage of at least 95% for the 3 methods you tested. (again make sure a test only counts as test, if it tests something, fix errors if you find any)
     5. Answer: What is the Code Coverage you were able to reach for each of them?
     6. Take a screenshot of your code coverage and paste it into your document (or report when done through Gradle). .

# Task 4: Gradle - 5 Points

This is basically the same as in assignment 2 but now there will actually be some worth while code coverage. Leave the Gradle file as as (it should still be the original one).

**Step 1:** You already have a Gradle file in your project. Run "gradle build" and then "gradle test". Look at all the things that are in the build folder. Especially what you find in the reports folder.

**Step 2:** The Gradle file already includes Jacoco. Jacoco is part of the EclEmma Plugin and can also give you information about code coverage. Your test results, which were created by ”gradle test” showed you the tests you have run and if they were successful. Now we also want to see the code coverage (which will not be a lot yet).

Now run "gradle jacocoTestReport". This should generate an HTML test report informing you about the code coverage, find the folder where this report is included in your build directory.

Take a screenshot of your command window, your file structure and your name showing somewhere on the screen (or asurite or something to identify you). Include this screenshot in your document. Put this under Task 4.

# Submission

On Canvas submit

1. GitHub link to your private repo
2. Your PDF document including

*•* All answers and screenshots mentioned above.