**CST316 Software Enterprise: Construction and Transition Spring 2013**

**Lab 2: Structural Unit Testing, and TDD January 23, 2013**

This lab is a continuation of lab1, where we used JUnit4 to create black-box, or specification-based unit tests. In this lab we will extend our tests to white-box, or structural unit tests. Then we will add a new feature using test-driven development (TDD). On the Moodle site I have provided a “[Refactored version of lab 1 given jarfile](http://swent0.asu.edu/moodle/mod/resource/view.php?id=1099)”, please download and create a new Java project using the contents of the jarfile. Please do this lab with the same partners you had last week!

**Tasks for this lab**:

*Task I: Install and Run code coverage with EclEmma*

1. In the Eclipse🡪Help🡪Install New Software menu, enter “http://update.eclemma.org”. Eclipse should come back with the EclEmma plugin in its middle box. Select the checkbox and install.
2. When done installing, Eclipse will ask you to restart, go ahead and do so.
3. After restarting Eclipse, you should see a new icon in your Eclipse toolbar, next to the Debug and Run As icons:

:::::::::Desktop:Screen shot 2013-01-21 at 9.50.40 PM.png

The icon on the left is the EclEmma one.

1. Select this icon and choose Coverage As🡪JUnit Test (if this option is grayed out then simply re-run your unit tests as before: right-click on your project, select Run As🡪JUnit Test).
2. You should see tabs in your bottom pane for *JUnit* and *Coverage*
   1. Click on *JUnit* first. You should notice 6 unit tests passing based on the refactoring I did after last week’s lab.
   2. Click on *Coverage*. Expand the Element tree down to the file level. You should see a histogram view depicting the distribution of code coverage across the project.
   3. Click around the files in banking.primitive.core, starting with Account.java, Checking.java, and Savings.java. You should notice the coloring in the editor window indicating what lines have been covered.
   4. In a separate Word document, please answer: What kind of code coverage does EclEmma provide and why?

*Task II: Import your unit tests for Checking and Savings from last week.*

1. You should notice that Checking.java (7.1%) and Savings.java (8.1%) have poor code coverage.
2. Draw control flow graphs for the Checking.withdraw and Savings.withdraw methods.
   1. Turn on line numbering in Eclipse (Preferences🡪General🡪Editors🡪Text Editors)
   2. You may hand-draw these, take a photo, and import into your Word document. USE THE LINE NUMBER as the node label for each statement in your flow graph.
   3. For each of the graphs, list the complete set of test sequences to achieve i) statement coverage, ii) edge coverage, and iii) branch coverage
3. Copy over your unit test code for Checking and Savings, which should address the *deposit* and *withdraw* methods. Put these unit tests in a file named *AccountTest.java*.
4. What is your resulting code coverage after applying these tests (note it in your Word document).
5. If your code coverage for these 2 methods on both classes is not above 90%, please improve your tests so they achieve this level or greater. Note you only have to do this for these methods, not the entire class (you can expand the class in the Coverage pane and see individual method coverage).
6. Write unit tests to cover the *closeAccount* and *saveAccounts* methods on banking.primitive.core.ServerSolution. Put your tests in new test class banking.primitive.core.ServerSolutionTest. Follow good black-box principles (understanding your test partitions, one focused test method per test objective) as well as achieving 90% or greater test coverage.

*Task III: Add a new feature using TDD*

In test-driven development, we start by writing a unit test that represents the acceptance criteria of a user story. That is, if the test passes, then we can feel confident our code meets the requirement. But the catch is you have to *write the test first, before the code!* So how can such a test pass? Answer: it won’t! (at first). Do the following:

Consider a user story: “the application will allow the user to change the name and path location of the file where it saves accounts, but cannot overwrite an existing file”.

1. Create an empty method with signature “public static boolean changeSaveFileName(String fullfilename)”
   1. Note it throws no exceptions. You may not throw a checked or unchecked exception
   2. The method will return true if the filename was changed to the input parameter, false otherwise.
   3. You may add a “public static String getSaveFileName()” if you like though it is not strictly necessary.
2. Write a unit test for ServerSolution.java to represent the expected case (the filename is updated) for changeSaveFileName.
3. Run the test, it will fail! Copy and paste the code for the test into your Word document.
4. Write the code to make the test pass.
5. Re-run and show the test passes.
6. Now consider a failure case: the filename already exists.
   1. Write a test case for this that fails. Again copy and paste into your Word document.
   2. Write the code to make it pass
   3. Re-run and show it passes
7. Now consider another failure case: the application does not have permissions to write the file to that directory.
   1. Write a test case for this that fails. Again copy and paste into your Word document.
   2. Write the code to make it pass
   3. Re-run and show it passes

For this task you should be submitting (only 1 submission for the 3 of you, please put names on top):

1. A Word doc with the answers and graphs requested in steps of Tasks I, II, and III.
2. A file named AccountTest.java, which has the unit tests for Savings and Checking implementations of deposit and withdraw.
3. A file named ServerSolutionTest.java, which includes unit tests for closeAccount and saveAccounts, plus tests for new method changeSaveFileName.
4. An updated ServerSolution.java file with the results of the TDD process resulting in changeSaveFileName.

You will be doing reflection as we did last semester, but that process will be explained separately.

At the conclusion of this lab you will be expected to start introducing unit tests in your project immediately, and continue the practice throughout the semester.