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

**Lab 1: Unit Testing January 16, 2013**

In this lab you will use JUnit to perform unit testing a simple Banking app. In the zip file provided on the class webpage is an Eclipse project you can load into normal Eclipse. Unzip the zip into a directory and then import the project under Package Explorer by doing a right-click, Import, “Existing Projects into Workspace”.

**Tasks for this lab**:

*Task I: Run the application and unit tests*

1. Run the application by running banking.gui.Main my.properties. Jot down notes for any strange or unexpected behaviors.
2. Review the javadocs. Draw a class diagram of the classes in package in banking.primitive.core. Draw a statechart for the states an Account object may be in. You may draw these by hand and take a photo and insert into your Word doc.
3. You will notice 2 Junit tests, AccountServerTest and AccountServerTest2. Run these by right-clicking on them and doing “Run As…” and select Junit test.
4. Inspect the AccountServerTest. Answer/do the following:
   1. It tests 2 methods. In you opinion does it cover the boundary cases for the methods in question?
   2. What are the equivalence partitions for the testGetAccount test?
   3. The testNewAccounts test is incomplete, there is a note “FOR LAB” at the bottom. Add the required tests (hint: you need 4 assert statements).
5. Create a TestSuite AccountServerTestSuite named that runs both of the test classes together.

For this task you should be submitting a Word doc, an updated AccountServerTest, and the TestSuite class AccountServerTestSuite

*Task II: Black-box testing of Savings.java and Checking.java*

1. Create a unit test class (JUnit 4) called SavingsTest.java (use the eclipse tools)
2. Create a unit test class called CheckingTest.java
   1. NOTE: there may be potential to share some code between these two test classes
   2. HINT: Remember to write positive, negative, and exception tests

Start by writing the test method for “deposit” on each subclass, then do “withdraw” when those are done. Consider the principle of writing focused tests – one per each “principle” or “equivalence partition.”