CSCE 747 - Assignment 3:
Unit and Fault-Based Testing

Due Date: Tuesday, April 3rd, 11:59 PM

This assignment is worth a total of 100 points. You may discuss these problems in your teams and turn in a single submission for the team, in zip format, on Moodle. Answers must be original and not copied from online sources.

Problem 1 (55 Points)

Software engineers love caffeine, so we are planning to install a new coffee maker in the classroom. Fortunately, the CSC department at NCSU has developed control software for a shiny new CoffeeMaker, and has provided us with that code. We just have to test it.

You will be working with the JUnit testing framework to create unit test cases, find bugs and fix the CoffeeMaker code from NCSU's OpenSeminar project repository (thank you to the authors!). The example includes requirements, design and code with some seeded faults. For this exercise you are required to create unit test cases for all the application classes with JUnit, execute those against the code, detect, and fix the faults (as many as possible).

The goal for your unit tests is to achieve statement (aka: line) coverage of the code for all classes except Main (coverage must be measured with a tool, we recommend Emma, but you may use any approved tool). If you find faults (by other means) that are not detected by your tests, describe the additional tests that you would need to detect those faults. If you find portions of the code that cannot be covered by a test case, explain why those elements are not covered.

Your submission should include a test report that describes the tests (with a descriptive name and a short explanation), list of defects found, your recommended fix for each, the test results after the fix, and a coverage report. You should also turn in an electronic archive of your tests with a concise set of instructions on how to set-up and execute your tests on the original CoffeeMaker example.

Relevant links:

- CoffeeMaker code and sample tests **(do not turn in the sample tests as your own)** - http://realsearchgroup.org/SEMaterials/tutorials/coffee_maker/download/CoffeeMaker_JUnit.zip
- CoffeeMaker user stories - http://realsearchgroup.org/SEMaterials/tutorials/coffee_maker/index.html
- jUnit - http://junit.org/
  (We recommend using a Java IDE - such as Eclipse - that integrates JUnit execution tools into the development environment.)

- Emma (code coverage measurement) - http://emma.sourceforge.net/ (command line)
  http://www.eclemma.org/ (Eclipse plug-in)
- Cobertura (code coverage measurement) - http://cobertura.github.io/cobertura/

Points will be divided up as follows: 20 points for test design, 10 points for detecting faults, 15 points for the suggested fixes to the code, and 10 points for test execution and coverage reports.

Problem 2 (45 Points)

1. Generate at least four mutants for classes of your choice in the CoffeeMaker code (before you apply any of your fixes from Problem 1). You must create at least one invalid, valid-but-not-useful (non-equivalent), useful, and equivalent mutant. Each mutant must be created by applying a different mutation operator, and you must use least one mutation operator from each of the three categories in Figure 16.2 in the textbook. You do not have to use the same classes or methods for all mutant categories. Your report should include the mutated code, noting how it differs from the original code. (20 Points)
2. Assess your test suite that you created for Problem 1, with respect to the set of mutants that you derived - Are you able to kill all of the non-equivalent mutants with your test suite? If not, describe which non-equivalent mutants cannot be differentiated from the original code using your test suite, and why they cannot be differentiated. Write additional tests that can kill those non-equivalent mutants. (15 Points)
3. Identify a minimal subset of tests from your test suite that is sufficient to kill all of the non-equivalent mutants. (10 Points)