EECS 293
Software Craftsmanship
2017 Fall Semester

Programming Assignment 8

Due at the beginning of your discussion session on

October 23-27, 2017

Reading

Read Chapter 22 in Code Complete.

Grading Guidelines

The grade will primarily depend on the process to develop test cases. High marks will be given to test suite that are developed following a methodic process.

An automatic C (or less) is triggered by

- Any routine with complexity greater than 4,
- Any substantially repeated piece of code, or by
- Improperly named routines.

Starting with programming assignment 10, points will be deducted if code or branch coverage is incomplete.

Programming

First, make all changes discussed during last week's code review.

In this assignment, you will design, write, and run test cases in JUnit. *The emphasis is on a methodic and coherent process* to develop and implement test cases. You should *avoid developing a test suite in a generic and haphazard manner*. Although it is very tempting to develop test cases by error guessing, a methodic process has been discussed in the lectures and reading assignments, and should be followed

starting from this assignment. An example is given on blackboard under Course Documents, Labelled Test Cases.

For each method, the **test cases have to exhaust the typology** described in Section 22.3:

- Structured basis
- Data-flow (extra credit)
- Boundary
- Compound boundaries
- Bad data
- Good data
- At least one stress test for the whole code base

The tests must be **labelled**: add a comment to each test case stating the categories to which the test belongs (for example, a test case could be labeled: "structured basis, data flow, good data", another as "boundary", etc.). An example was covered in class and it is posted on blackboard under Course Documents, Labelled Test Cases. You can repeat the test cases that you have used in assignments 6 and 7, and presumably add new ones.

All methods must be tested individually. In particular, all protected and private methods must be tested individually. In other words, you cannot test a private or protected method solely by invoking the public methods that call it. Instead, you need to write test cases that invoke the private or protected method through a nested test hook. An example is in the testexample repository that you cloned in assignment 7. You cannot use reflection to test private and protected methods.

The exception is that you can omit tests for methods that are automatically generated (such as getters, setters, `equals`, `toString`, or delegate). If you plan to skip testing of an automatically generated method, you should have Eclipse generate the method's documentation automatically.

Your build environment should contain the following targets:

- "test" target should run all the test cases.

- "report" target should generate a html Junit and JaCoCo report for the test cases.

Blackboard Resources

The Course Documents folder contains an example of a test suite that has been labeled according to the textbook taxonomy.

General Considerations

This assignment concludes the scheduling project. In the next assignment, we will turn to a different project.

Your implementation may contain as many auxiliary private methods as you see fit, and additional helper classes may be defined. Your code should have a reasonable number of comments, but documentation is going to be the topic of a future assignment. As a general guideline at this stage of the course, comments and tests should be similar to those accepted in EECS 132.

Discussion Guidelines

The class discussion will focus on the **process** to design test cases, and on their completeness and implementation of the test cases.

Submission

Make small regular commits and push your revised code and test cases on the git repository.