CSDS 293
Software Craftsmanship
2024 Fall Semester

Programming Assignment 8

Due at the beginning of your discussion session on

Oct 21 – 25, 2024

Reading

In addition to the following topics, the quiz syllabus includes any material covered in the lectures:

- In Code Complete,
    - Section 22.3 (except "Data Flow Testing"),
    - Section 22.4 ("Which Classes Contain the Most Errors" and "Errors by Classification" only),
    - Section 22.5 ("Test-Data Generators" and "Coverage Monitors" only)
- In canvas, test cases development example and Repeated Code
- In the test-example repo, private method testing.

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.

However, a submission that avoids these problems does not necessarily qualify for high quality craftsmanship. Starting with

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

Programming

Overview

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 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 canvas under Modules, Labelled Test Cases, TestDevelopment.pdf.

For each method, the **test cases have to exhaust the typology** described in Section 22.3 (not all categories are applicable to all methods):

- Code coverage
- Branch coverage
- Data-flow (extra credit)
- Boundary
- Compound boundaries
- Bad data
- Good data

Design Document

First, create a separate test design document. The design document should be modeled after the test development example in canvas. For each method, define

- Test conditions: each with an identifier, goal, notes, and conditions (Section "Conditions" in the canvas example.)
- Tests: each with an identifier, identifiers of the satisfied conditions, and assertions (Section "Tests" in the canvas example.)

You can omit tests for methods that are automatically generated (such as some constructors, getters, setters, `equals`, `toString`, or delegate).

Additionally, describe at least one stress test for the whole code base (Section "Stress Test" in the canvas example and in Code Complete).

Implementation

Second, implement the tests. Add a comment to each test to cross reference it with the test design document. The test cases that you have developed for assignments 6 and 7 should be included and cross-referenced as "good data (legacy)".

All methods must be tested individually. 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 test-example repository that you cloned in assignment 7. You cannot use reflection to test private and protected methods.

If you plan to skip testing of an automatically generated method, you should have your IDE generate the method's documentation automatically.

Your build environment (build.xml or Makefile) should contain the following targets:

- "test" target should run all the test cases.
- "report" target should generate an html Junit and JaCoCo report for the test cases.

During the discussion, your leader should be able to run the tests and display the reports.

Canvas Resources

The Labelled Test Cases module contains an example of a test suite that has been labeled according to the textbook taxonomy.

General Considerations

This assignment concludes the 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 should be like those accepted in CSDS 132.

Discussion Guidelines

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

Submissions

Make small regular commits and push your revised code and test cases on the git repository. Tag your release and push that tag to your git repository.