CSE 4495 - Assignment 2:
Unit and Structural Testing

Due Date: As per instructions provided in the new eLMS

You may discuss these problems in your teams and turn in a single submission for the team (zipped archive) on eLMS. **Answers must be original and not copied from online sources.**

Cover Page: On the cover page of your assignment, include the name of the course, the date, your group name, and a list of your group members.

Problem 1 - Unit Testing (20 Marks)

You have acquired the code for a coffee maker system online. The system describes as follows:
"Software engineers love caffeine, so we are planning to install a new coffee maker in the classroom. Fortunately, the CSC department at North Carolina State University (NCSU) has developed control software for a shiny new CoffeeMaker and has provided us with that code."

Now before you go on and install it in UIU, as responsible SQA engineers your job 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 (thanks to the authors!). The example code comes with some seeded faults.

The core functionality of the system is defined by the user interface (offered by the Main class).

Based on your exploration of the system and its functionality, you will write unit tests using JUnit for C**offeeMaker, Inventory, Recipe,** and **RecipeBook** classes (excluding Main and the exceptions), execute those tests against the code, detect faults, and fix as many of the faults as possible.

The source code is available on eLMS:

Your submission should include:

1) **A document containing:**
   a) **Test descriptions**
      - Describe the unit tests that you have created, including a description of what each test is intended to do and how it serves a purpose in verifying system functionality. Your tests must cover the major system functionality, including both normal usage and erroneous input.
   b) **Instructions on how to set-up and execute your tests**
      - (if you used any external libraries other than JUnit itself, or did anything non-obvious when creating your unit tests).
   c) List of faults found, along with a recommended fix for each, and a list of which of your test cases expose the fault.
2) **Unit tests implemented using the JUnit framework (Java files from the test package).**

If you find faults by other means, such as exploratory testing, that are not detected by your unit tests, you should try to create unit tests that expose those faults.

Relevant links:

- JUnit guide: https://junit.org/junit5/docs/current/user-guide/#writing-tests
- Instructions for executing JUnit tests in your IDE of choice: https://junit.org/junit5/docs/current/user-guide/#running-tests
- More instructions for running JUnit tests in IntelliJ IDEA: https://www.jetbrains.com/help/idea/junit.html

I recommend using IntelliJ - that would make it easier to integrate JUnit into the development environment.

Marks will be distributed up as follows:

- **30% points for test descriptions,**
- **40% points for unit tests,**
- **20% points for detecting faults, and**
- **10% points for the suggested fixes to the codes**.

(See Next Page for Bonus Task)

Bonus - Structural Coverage (5 Marks)

After testing the CoffeeMaker using your knowledge of the functionality of a coffee machine and your own intuition, you have decided to also use the source code as the basis of additional unit tests intended to strengthen your existing testing efforts.

You have identified the following methods in particular as worthy of attention:

- CoffeeMaker::makeCoffee
- Inventory::addSugar
- Recipe::setPrice
- RecipeBook::addRecipe

Either design new unit tests to achieve full Branch Coverage over these four methods, or if you already have achieved full Branch Coverage over these methods in Problem 1, identify the subset of tests that achieve this coverage.

- **In either case, describe what specific code elements each test covers, and explain exactly why the chosen inputs cover those elements in the manner needed for Branch Coverage.**
- **Add a section at the end of your document reporting current coverage and the newly written cases to achieve full coverage.**

To measure coverage in IntelliJ, see https://www.jetbrains.com/help/idea/code-coverage.html. You may use any of the three coverage runners. In Eclipse, use EclEmma: https://www.eclemma.org/. If using the command line, use EMMA: http://emma.sourceforge.net/.

Both the IntelliJ coverage runner and EclEmma can output a report (e.g., https://www.jetbrains.com/help/idea/viewing-code-coverage-results.html#coverage-in-editor). Be sure that you include all files from the generated report (i.e., not just index.html)