CS2110 Homework 6 - Fall 2021
Meme Magic Test

Your development build is coming along nicely. Before you release it to your users, however, you want to ensure the software is of good quality. To protect the reputation of your startup, you decide to develop a set of automated tests using JUnit.

First, set up your project:

- Create a new Java project on Eclipse, and call it "Homework 6."
- Copy and import all of your Homework 5 files into the Homework 6 project.
- Maintain a copy of Homework 5 files (i.e., do not overwrite them.)

Note: you may alternatively copy your project in Eclipse by selecting the "Homework 6" project in the Package Explorer, copy with Ctrl+c (or Command+c on mac), and paste with Ctrl+p (or Command+p on mac). On pasting, Eclipse will ask you to name the newly-pasted project.

Next, set up your testing environment (this is important!):

- Right-click your new project in the Package Explorer and choose "New" and "Source Folder". For "Folder name", type "test" and click Finish. This will provide a separate source folder for all your JUnit tests.
- Right-click the new "test" folder in Package Explorer and choose "New" and "JUnit Test Case". (If it's not in the menu there, choose "Other..." and then search for "JUnit Test Case".) This will open a window to create a new JUnit test file, and will also add JUnit to our build path.
    - Choose "New JUnit 4 test" at the top of the window, and type the Name of the test class, such as "FeedTest" to test the Feed class. Then click Finish.
      Note: you must use JUnit 4 for Gradescope to correctly run your tests.
    - You should get a warning that "JUnit 4 is not on the build path. Do you want to add it?" with the action "Add JUnit 4 library to the build path" pre-selected. Click OK to proceed.
    - That should open the new file with one test that defaults to fail. Now you're ready to implement your own tests!

See the Getting Started with Homework 6 video on Panopto (you'll need to open Panopto through Collab before clicking the link.

Learning Goals

In this assignment, we will practice:

1. Writing unit tests for Java using JUnit
2. Calculating code coverage of unit testing

Unit Testing Meme Magic

Write JUnit tests that provide at least **90%** code coverage for your project. 90% code coverage means that the tests confirm that 9 out of every 10 lines of written code is measured by the test. *(You are using a whitebox approach to testing!)*

For each Class from Homework 5, create a JUnit 4 Test named ***Class***`Test`. For example, to test the `Meme` class, create the `MemeTest` class. (The .java file for the new `MemeTest` class should reside in the test/ folder.)

For each method in your classes, create a ***methodName***`Test`. For example, to test the `deleteMeme()` method, create the `deleteMemeTest()` method.

For getter and setter methods, it is sufficient to create one method, `setAndGet`***`Variable`***`Test()`, where *Variable* is the name of the field (instance variable). For example, to test the `userName` field, you should write the `setAndGetUserNameTest()` method, which calls both the getter and setter for `userName` and asserts that the value was set correctly. For all other methods, use a separate test method to keep your tests organized.

Your code **must** pass your Unit tests. If it does not pass, code coverage will not be calculated on Gradescope.

Important Information

You can confirm that your tests are evaluating all of your code using the Coverage feature in Eclipse. **Note:** If you do not remove or comment out your main method tests, these may count as lines that need to be covered.

To access the Coverage feature, with your project open, select Run, then Coverage. The menu may vary by Eclipse version. If you don't see the Coverage option, search for coverage in help. We encourage you to set up a Run Configuration to run all your Unit test files at once. See the Getting Started with Homework 6 video on Panopto (you'll need to open Panopto through Collab before clicking the link.

Code coverage is counted as a percentage of the instructions in your classes covered by your tests. Therefore, it is important to separate the tests from the codebase. As seen in Figure 1, we currently have only 2.8% code coverage (even though it appears we have 100% coverage of our test code and 5.3% overall coverage in the project itself).

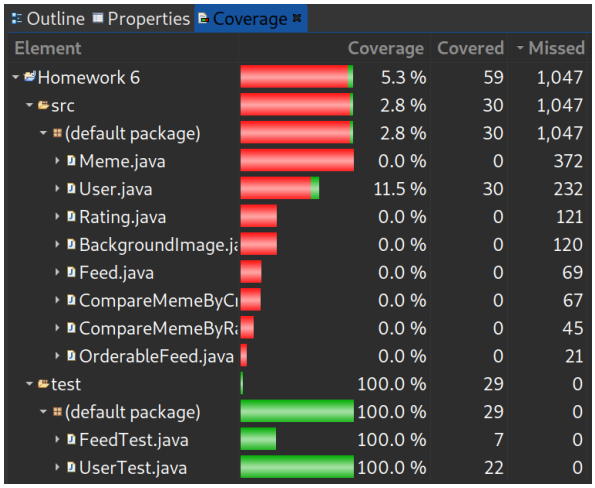

Figure 1. Screenshot of Eclipse Code Coverage display. Shows 2.8% code coverage of our classes, including 11.5% code coverage of the User class.

Additional Resources

The following resources may be helpful when completing and submitting this homework.

- JavaDoc style documentation for each of the classes and methods from Homework 5
- Video on Getting Started with Homework 6
- Video on submitting to Gradescope

Submission Information

Method and Class Naming: You must match method names, instance variable names, and data types exactly. You must use correctly formatted Java code. **Classes to perform tests MUST have names ending with "`Test`".** That is, if you want a JUnit test to run on Gradescope, it must be in a class named *Something*`Test`.

Coding Style: In real-world software development, it is paramount to create readable and easily maintainable code. That is typically achieved through the use of style and commenting guidelines. Since you will be updating this code over the next few weeks, we have provided a style guide and formatting guide that we strongly encourage you to follow:

- Coding Style Guide (includes installation instructions for Eclipse)
- Eclipse Style File

Submitting: Upload your Eclipse project (the `.java` files) to the "Homework 6 - Meme Magic Test" assignment on Gradescope. You should submit all your .java files from both your src/ and test/ directories. Gradescope will calculate code coverage for your classes and use that to calculate your final score. *If your code does not compile, it cannot check for code coverage.* We strongly encourage you to run your tests locally before submitting; you should **not** use only Gradescope to run your tests. Therefore, you may upload your code a **maximum of fifteen (15)**

times. *Note: Gradescopes code coverage values may differ slightly from Eclipse's reported coverage. We have found Gradescope to be slightly higher in most cases.*

Note: After the 15th submission, Gradescope will still allow submissions, but they will NOT be graded by the system.

Grading Rubric

Your score will be determined by lines of code coverage. The assignment will be worth a total of 100 points:

90 points - Code coverage, proportional to the amount of code coverage
50% coverage or less = 0 points -- *up to* -- 90% coverage or more = 90 points

10 points - Code readability (organized, well-indented, readable by others)