SWE 437 Software Testing and Maintenance J. Offutt

**Final Exam, part 2, 14 December 2021**

For part 2 of your final exam (the tech-challenge portion), you will **analyze** a program at different levels of abstraction; **model** software components using multiple structures from our course; **design** tests according to **criteria** from our course, **implement** those tests with input values, expected results, and setup and teardown values; and either **automate** **and** **run** those tests or **run** those tests by hand.

I am providing a small Java class of less than 200 lines of executable code. The class uses two enums defined in separate classes. I am also providing you with an executable version as a web app. You will not need to know anything about web apps to test this program. You will design tests for the external interface (through a browser) and for individual methods in the class provided. The methods do not have any web dependencies.

You will create three sets of tests, each for a specific software artifact and according to a specific test criterion. The tests will be graded as follows:

1. Correctness and quality of the model (6 pts—2 per test set)
2. Correct application of the criterion specified (6 pts—2 per test set)
3. Correct implementation of the actual tests (6 pts—2 per test set)
4. Correct automated test scripts (only for part II, 2 pts)
5. Reporting of the results (3 pts—1 per test set)
6. Neatness and organization—this score will be inversely proportional to how hard I have to work to trace from the software artifact to the model to the abstract tests to the concrete tests (2 pts)
7. The total number of points is 25 (part 1 is worth 10 points, for a total of 35)

**RESOURCES**

Each student must work individually—no collaboration, no discussion during the exam period, and no help with designing the tests. I will be available for questions during the exam period. You may use any course materials, including books, the course website, the book website, the coverage web apps on the book website, slides, examples, piazza, etc. You may also use the internet, although frankly, online resources are more likely to confuse you than help you. You may NOT use any person except the professor.

**TECHNOLOGY NEEDED**

You will need to draw graphs, and may use any drawing tool, including paper and pencil. You will need access to a browser. You will need to implement and run JUnit tests. You will need to zip files to submit.

**SUBMISSION**

1. Submit your final exam by putting all files in a folder named “***MasonID*-swe437**,” and zip the folder into one zip file. You should replace “***MasonID***” with your Mason ID, or email address. Thus, my submission would be named “*offutt-swe432.zip*”. Include the following files:
   1. Your models, including graphs, tables, etc.
   2. Your abstract tests, your concrete tests (including input values and any setup and teardown values). I suggest putting all of these into one file, and CLEARLY LABELED.
   3. JUnit tests when required. When JUnit tests are included, the information from (*b*) can be included as comments in the JUnit tests.

You may also combine all files into one document and submit as PDF. NO .*pages* FILE SUBMISSIONS!

1. Submit your zip file into my dropbox file request.
2. You may resubmit up until the deadline—I will grade the most recent version.

Submit your file through this URL: **XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX**

**ACCOMMODATION**

If you need accommodation, either as specified by the DRC at the beginning of the semester, or due to the pandemic and online nature of the final, contact me by email. Online accommodations could include internet access problems or illness. If your internet or computer crashes during the exam period, contact me as SOON AS POSSIBLE through email.

**SPECIFICATIONS**

Test the program ***stringOperations.java***. You can run the program here:

<https://cs.gmu.edu:8443/offutt/servlet/stringOperationsExample.stringOperations>

You can get the source files here:

<https://cs.gmu.edu/~offutt/classes/437/stringOperationsExample/stringOpsLib.java>

<https://cs.gmu.edu/~offutt/classes/437/stringOperationsExample/stringOps.java>

<https://cs.gmu.edu/~offutt/classes/437/stringOperationsExample/orderings.java>

You will be unit-testing methods from **stringOpsLib.java**. **stringOps.java** and **orderings.java** are a shared enums that **stringOpsLib.java** needs. You **will not** need the servlet front-end.

Your task is to design, implement, and submit 3 sets of tests.

**I. Input space partitioning**

Apply ISP to the user interface for ***stringOperations***. Model this program as we have done in class examples and assignments. Find characteristics of the inputs to model the input domain. Then partition each characteristic into blocks. Use the **base choice criterion** to design abstract tests. Implement those abstract tests as input values and run the values **by hand**. (You **do not** need to automate ISP tests.) Submit your test values and a brief summary of what happened when you ran them—whether the software crashed, returned incorrect results, or returned a response that you judged to be correct.

**II. Graph-based testing**

Apply graph-based testing to **one of** two methods in **stringOpsLib.java**: ***doStringOp***() or **concatenate**(). (You choose which, do not do both.) Draw the control flow graph for the method. Annotate each edge in your graph with the predicate that causes the edge to be taken. Design tests to satisfy **edge coverage** (**EC**) on your graph. Find test paths (abstract tests) that will satisfy EC, then choose values to satisfy each test path, and finally provide expected output results. Encode the concrete test values in JUnit tests, run the tests, and capture and submit the screen result.

**III. Logic-based testing**

Apply logic-based testing to the method ***areStringsEqual***() in **stringOpsLib.java**. Draw the control flow graph for the method. Identify and write down the predicates on edges that represent branches. Design tests to satisfy **correlated active clause coverage** (**CACC**) on every predicate. Find truth assignments that will satisfy CACC (abstract tests), then choose values to satisfy each truth assignment, add values that will reach the predicate, and provide expected output results.

You do not need to automate or run these tests.

**Naming and submission**

Submit all of your test designs for parts I, II, and II. Also submit your JUnit test file and screen captures from running the test.

* Make sure your name is inside all files, including the Java file
* Include your MasonID (e.g., “offutt”) in your file name, **not** your G#
* Put your files into a folder named MasonID-swe437 and zip it before submitting