CS6310 – Software Architecture & Design
Assignment #5 [100 points]: Mass Transit Simulation – Individual Implementation (v7)
Fall Term 2018 – Prof. Mark Moss

Submission

- This assignment must be completed as an individual, not as part of a group.
- You must submit: (1) a Java JAR file named **working_system.jar** that performs the core steps of the simulation as described below; and, (2) the corresponding Java source code for your JAR file (including external dependencies/libraries) in a ZIP file named **source_code.zip**.

- You must notify us via a private post on Canvas and/or Piazza BEFORE the Due Date if you are encountering difficulty submitting your project. You will not be penalized for situations where Canvas is encountering significant technical problems. However, you must alert us before the due date – not well after the fact. You are responsible for submitting your answers on time in all other cases.
- Please consider that uploading files to Canvas might occasionally take a long time, even in the case of seemingly "relatively small" submissions. Plan accordingly, as submissions outside of the Canvas Availability Date will likely not be accepted. You are permitted to do unlimited submissions, thus we recommend you save, upload and submit often. You should use the same file naming standards for the (optional) "interim submissions" that you do for the final submission.

Scenario

Your requirement for this assignment involves implementing a small prototype in Java that executes the core functionality of the MTS application. Your prototype must demonstrate the basic steps of (1) reading in the file containing the scenario configuration data, (2) determining which bus should be selected to move as the next event to be processed, and (3) doing the stop selection, distance and travel time calculations to determine then arrival time for the bus at its next (destination) stop. Your prototype does not have to produce a graphical display like the client provided prototype - instead, your output will be solely text-based. ***You should use the class and sequence diagrams/models that you developed in Assignment #2 to drive your implementation as much as possible.***

Deliverables

You must develop a prototype that performs the following tasks:
Step 1: Read the data from the provided scenario configuration file
loop for twenty (20) iterations:

- Step 2: Determine which bus should be selected for processing (based on lowest arrival time)
- Step 3: Determine which stop the bus will travel to next (based on the current location and route)
- Step 4: Calculate the distance and travel time between the current and next stops
- Step 5: Display the output line of text to the display
- Step 6: Update system state and generate new events as needed

end loop

The clients will provide examples of the input and output file formats, along with an explanation of the desired analytical tasks. You must provide the actual Java source code, along with all references

to any external packages that you used to develop your system, in a ZIP file named **source_code.zip**. You must also provide an executable JAR file named **working_system.jar** to support testing and evaluation of your system.

Evaluation

- We will evaluate the correctness of your system against multiple test cases. The clients have provided some basic test cases that should be used to evaluate the correctness of your system. These test cases can be used to ensure that your code is producing the correct output.
- Formatting is important! Use the matching characters for the output strings, and don't put extra spaces between elements of the output strings. Strings that do not match the correct output because of formatting (syntax) errors will receive significant penalties.

- Your total score of 100 points will be a combination of:
  - 50 points for your program's performance against the given (client's) test cases
  - 35 points for your program's performance against the new and unseen test cases
  - 15 points for the formatting of the submission and related issues
- The client has provided twelve test cases, and your output must match their output exactly to receive full credit. The cases are divided into the following categories based on the general level of complexity:
  - Four (4) single bus, single route cases at 2 points each
  - Three (3) multiple bus, single route cases at 4 points each
  - Five (5) multiple bus, multiple route cases at 6 points each
- We will also generate more complex test cases that will NOT be provided to you in advance. The scenario configuration commands within the test cases will not change, but the file contents will, so please don't "hardcode" any of the current test case contents into your program. There will be seven (7) "new" test cases at 5 points each.
- The remaining fifteen points will allocated to the submission of the program: correct formatting of the submission, correct output formatting, working source code that compiles to the submitted JAR file, and any other related issues.

- The given cases are numbered, and generally proceed from simpler, smaller cases to larger, more complex cases. A possible solution for each test case is also given; for example, the solution for **test2_connector_singlebus.txt** is given as **test2_solution.txt**. Remember that events that occur at the same logical time may be executed in any order and still be correct.

- Here's a relatively small example, using the **test0_instruction_demo.txt** test case:

```
> cat test0_instruction_demo.txt
add_depot,0,DEP/Repair Depot,0.2,0.28
add_stop,5,NEP/NE Perimeter,0,0.13,0.24
add_stop,11,ART/Arts Studio,0,0.11,0.31
add_stop,16,NTE/North End,0,0.2,0.21
add_route,52,2,UpperHalf
add_route,54,4,LowerHalf
add_route,56,6,RoundTrip
```

```
extend_route,52,16
extend_route,52,5
extend_route,54,11
extend_route,54,5
extend_route,56,16
extend_route,56,5
extend_route,56,11
extend_route,56,5
add_bus,67,52,0,0,15,10000,10000,27
add_bus,68,54,0,0,15,10000,10000,40
add_bus,69,56,1,0,10,10000,10000,60
add_event,0,move_bus,67
add_event,20,move_bus,68
add_event,0,move_bus,69
```

Note that bus 68 does not start immediately as the scenario begins. Your program should match the contents of the **test0_solution.txt** file as shown here, with some possible differences based on the order of execution (*updated as of v5):

```
> java -jar working_system.jar test0_instruction_demo.txt
b:67->s:5@12//p:0/f:0
b:69->s:11@6//p:0/f:0
b:69->s:5@12//p:0/f:0
b:67->s:16@24//p:0/f:0
b:69->s:16@18//p:0/f:0
b:69->s:5@24//p:0/f:0
b:68->s:5@28//p:0/f:0
b:69->s:11@30//p:0/f:0
b:67->s:5@36//p:0/f:0
b:68->s:11@36//p:0/f:0
b:69->s:5@36//p:0/f:0
b:68->s:5@44//p:0/f:0
b:69->s:16@42//p:0/f:0
b:67->s:16@48//p:0/f:0
b:69->s:5@48//p:0/f:0
b:68->s:11@52//p:0/f:0
b:69->s:11@54//p:0/f:0
b:67->s:5@60//p:0/f:0
b:68->s:5@60//p:0/f:0
b:69->s:5@60//p:0/f:0
```

Since buses 67 and 69 do begin at the same time, the first two lines could occur in either order and still be a valid solution. In fact, this example includes a number of cases where two or more events occur at the same time, leading to multiple correct solutions. For example, note the last three lines of the solution, where all three buses will arrive at their next location at time 60. If the scenario were to continue from this point, the simulation system would be allowed to execute any of these three events next.

- One possible way to test your program is to use the **diff** function. The diff function compares two files to identify the differences between them, and is usually implemented in most Linux/Unix

and Mac OS systems. You can store the results of running your program on the test case, and then compare your results to the answers using commands similar to the following:

```
java -jar working_system.jar test0_instruction_demo.txt > results.txt
diff test0_solution.txt results.txt
```

Windows OS system also normally include the **FC** command that performs a similar function, as well as the opportunity to import the **cgywin** package or similar libraries. Please feel free to use the Linux **man** command, or other Internet resources, to learn more.

- There is no guarantee that these test cases are comprehensive in the sense that your program is completely correct if it passes the given test cases. You are welcome (and encouraged) to generate your own test cases as well. You are also permitted to share test cases with your classmates, but the test cases only – not specific code nor implementation details.

- You are also welcome to use the **marta_sim_13in_testing** (or **marta_sim_23in_testing**) JAR file to generate and compare the output from your own test cases against the output from your program. Also, remember that the client's prototype has fuel management and depot functionality built-in.

- A few recommendations when creating your own test cases:
  - Make sure that you create a depot stop as ID 0 at some reasonable location. The specific location of the depot is irrelevant, but we recommend a location somewhere near the "center of mass" of your other stops.
  - Give all buses a very large Initial Fuel value (e.g. 10,000 miles) and equivalent Fuel Capacity.
  - Keep the distances between your stops comparable to the test cases that have been given. You are welcome to create larger scale distances between stops, but be cognizant of the amount of fuel that will be needed to avoid refueling effects.

- You may assume that the commands in our test cases are organized in the following order: **add_depot**, **add_stop**, **add_route**, **extend_route**, **add_bus** and **add_event**. You are encouraged (but not required) to organize the commands in your test cases in the same manner.

Submission Details

- You must submit your source code in a ZIP file named **source_code.zip** with a project structure as described below. You must include all of your source *.java code – you do not need to include the (compiled) *.class files. Also, you must submit (separately) a runnable JAR file of your program named **working_system.jar** to aid in evaluating correctness.

- We will test your program by copying it into a directory with an existing test case, and then executing it at a (Linux) terminal prompt with the command:

```
java -jar working_system.jar <name of test file>
```

- Your program should display the output directly to "standard output" on the Linux Terminal – not to a file, special console, etc. Also, note that the test file must be located in the same directory

where your JAR is currently located. More importantly, we will copy your JAR file to different locations during our testing processes, so be sure that your program reads the test files from the current local directory, and that your program is NOT hard-coded to read from a fixed directory path or other structure.

- We've provided a virtual machine (VM) for you to use to develop your program, but you are not required to use it for development if you have other preferences. We do recommend, however, that you test your finished system on the VM before you make your final submission. The VM will be considered the "execution environment standard" for testing purposes, and the occasionally received excuses of "…it worked on my (home) computer…" will not be sufficient.

- On a related note, it's good to test your finished system on a separate machine if possible, away from the original development environment. Unfortunately, we do receive otherwise correct solutions that fail during our tests because of certain common errors:
    - forgetting to include one or more external libraries, etc.
    - having your program read test files embedded files in its own JAR package;
    - having your program read files from a specific, hardcoded (and non-existent) folder

- Many modern Integrated Development Environments (IDEs) such as Eclipse offer very straightforward features that will allow you to create a runnable JAR file fairly easily. Also, please be aware that we might recompile your source code to verify the functionality & evaluation of your solution compared to your submitted source code.

Closing Comments & Suggestions

This is the information that has been provided by the customer so far. We (the OMSCS 6310 Team) will likely conduct an Office Hours where you will be permitted to ask us questions in order to clarify the client's intent, etc. We will answer some of the questions, but we will not necessarily answer all of them. Also, though this current version of the system if "the core" of the system going forward, our clients will very likely add, update, and possibly remove some of the requirements over the span of the course. One of your main tasks will be to ensure that your architectural documents and related artifacts remain consistent with the problem requirements – and with your system implementations – over time.

Quick Reminder on Collaborating with Others

Please use Piazza for your questions and/or comments, and post publicly whenever it is appropriate. If your questions or comments contain information that specifically provides an answer for some part of the assignment, then please make your post private first, and we (the OMSCS 6310 Team) will review it and decide if it is suitable to be shared with the larger class. Best of luck on to you this assignment, and please contact us if you have questions or concerns.