OMSCS 6310 – Software Architecture & Design
Assignment #3 [150 points]: Mass Transit Simulation – Analysis & Documentation (v2)
Spring Term 2018 – Prof. Mark Moss

Submission:

- This assignment must be completed as an individual, not as part of a group.
- You must notify us via a private post on Canvas an/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.
- You must submit your work in Canvas to include all of your design artifacts/diagrams:
  - Class Diagram(s) (multiple diagram files are acceptable if it makes them more readable, but all relationships between classes must be shown clearly)
  - Object Diagram
  - ***test_mts_output.txt*** File corresponding to the Object Diagram
  - Sequence Diagram(s)
- 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 includes creating Unified Modeling Language (UML) based diagrams that accurately reflect the structure, behavior and configuration of the provided application. The provided application is a Mass Transit Simulation (MTS) system that will be used by the notional "clients" in our series of project-based assignments. The application is written in Java, and the components that you've been provided in this phase of the project series represent the "core business logic" of the overall system. The application simulates the interaction of buses moving along a route of stops, and allowing passengers to board and later depart the bus at different stops. You will be provided a fuller version of this system with more components during later phases of the project.

Disclaimer: *This scenario has been developed solely for this course. It is inspired by the Metropolitan Atlanta Rapid Transit System (MARTA) Expansion Project as part of the Georgia Tech Serve-Learn-Sustain (SLS) Program, and we will leverage some of their data and other resources. Any other similarities or differences between this scenario and any of the programs at Georgia Tech programs are purely coincidental. More information on the MARTA Expansion Project can be found at: http://serve-learn-sustain.gatech.edu/marta-expansion*

Deliverables: UML Diagrams are normally divided into three fundamental categories: Functional, Structural and Behavioral. For this assignment, you must analyze the MTS System that has been provided, and then submit specific UML Diagrams that accurately reflect the system as currently implemented. You must submit the following items:

(1) Class Diagram [50 points] – Your diagram must include all of the classes, attributes and methods as included in the MTS application, with a few exceptions as listed here. To make your class diagram more readable, you may omit very basic "setters and getters" [e.g. set_() and get_() based operations and methods], especially those of the one-line variety that only access and/or modify an object's basic attribute.

(2) Object Diagram [40 points] – An object diagram might seem similar to a Class Diagram at first glance, but there is a key difference: the Object Diagram reflects the actual objects that are created during the execution of the MTS system. Think of it as a "snapshot" of the various objects that would be instantiated from the different classes. You have been provided JAR file named ***mass_transit.jar*** that executes the system, along with a text file called ***test_mts.txt*** that contains a sequence of instructions that can be used to drive the simulation. To complete this portion of the assignment, you must execute the text file using the JAR file to generate a set of object data on your system, and then save the output in a text file named ***test_mts_output.txt***. Then, use the content of the output file to generate your Object Diagram. You must submit the output text file along with your Object diagram so that we can validate that they are consistent/matching. To execute the file, you may enter the commands manually or via the following (or a similar) command:

```
java -jar mass_transit.jar < test_mts.txt > test_mts_output.txt
```

(3) Sequence Diagram(s) [60 points] – You must generate Sequence Diagrams (sometimes referred to as an Event Diagrams) that reflect the actions contained in the simulation loop for the various MTS commands. If you can fit everything into one diagram, great; if not, then use multiple diagrams as required to ensure that the results are readable.

MTS System Description

The Mass Transit Simulation System simulates the process of using buses traveling along different routes to transport riders to different stops. The description below gives a brief, high-level overview of how the system provides these capabilities, but a large part (and intent) of this assignment also involves you taking the time to study the system more closely to better understand how it is constructed and how it operates. This is the core of the system that will also be used for the individually-based Design and group-based Implementation assignments to come.

- When the system is started, the user is provided with a command line prompt (labeled as `#main`) to enter MTS commands. The ***SimDriver class*** provides the interactive command loop that accepts and processes the MTS commands.
- The user can enter commands manually one at a time, or create a file to facilitate easier testing of a specific sequence or "script" of commands.
- There are a number of different MTS commands that are currently organized into three basic categories: Creational, Operational, and Reporting.
- The Creational category includes commands such as ***add_stop***, ***add_route***, ***add_bus***, and ***add_event***. These commands are used to create the objects in the simulation environment. The ***extend_route*** command is used to modify the route that bus takes to travel a sequence of stops.

- The *add_stop*, *add_route*, *add_bus* and *add_event* commands create those objects in the simulation domain. Each command accepts the proper attribute values in the order given in the corresponding JAVA file class listing. The ***BusSystem class*** holds the collections of objects that have been created to form the simulation environment.

- Stops are defined by the ***BusStop class***, and represent the places where riders arrive to catch a bus to a different location. Each stop has a unique ID (currently generated by the simulation system), along with a "user facing" and more descriptive Name attribute. A stop also contains a Riders attribute, which records the number of people waiting at the stop to board an available bus. Each stop also contains X- and Y-Coordinate attributes that translate to its geographical location on the travel map. The coordinate attributes are used to calculate the distance between stops, which is then used along with the Speed attribute of a bus to determine when the bus will arrive at its next stop.

- Routes are defined by the ***BusRoute class***, and represent the path along which the buses travel. Each route has a unique system-generated ID, along with more descriptive Name and Number attributes. Most importantly, each route contains an attribute that lists the stops covered by that route. The order of the list is important: normally, a bus begins at the stop listed in the first position of the list, and then progresses to each stop as ordered in the given route list. If the bus is located at the last stop on the list, it then goes back to the first stop in the list, and continues the cycle as long as required for the duration of the simulation session. Please note that a route might have the same stop listed twice, which is not necessarily an error. In these cases, the bus is likely traveling along a fixed path in alternating directions – for example, first Northbound, then Southbound, etc.

- Buses are defined by the ***Bus class***, and are used to transport passengers from a simulation perspective. Each bus has a unique system-generated ID, and travels along a certain Route. To facilitate movement along the route, the bus Route Location attribute refers to an item in the route's list of stops, rather than having the bus refer directly to the stop itself. Also, each bus has a Capacity attribute which limits the total number of passengers that it can carry at any one time, along with a Speed attribute that affects how quickly the bus travels along the route.

- The Operational category includes commands such as ***step_once*** and ***step_multi***. When used, these commands cause the simulation system to select the next event from the priority queue, and then process this event according to specific rules based on the type of event. The *step_once* command executes one event and then returns control to the user, while the *step_multi* command allows the user to execute several events in a row for convenience.
- This system uses very basic discrete-event simulation techniques to drive its operations. The ***SimQueue class*** provides a priority queue to manage system events. The events are sorted in the queue using the functionality provided in the ***SimEventComparator class***. In principle, the rank attribute of an event represents the logical time at which it will be executed, and the events are sorted based on the rank attribute. One of the events having the lowest rank in the queue will always be selected as the next event to be processed.

- Events are defined by the ***SimEvents class***, and represent the different types of actions that we will track in our simulation. Each event has a Rank attribute, which corresponds to the logical time that events will happen during the simulation, and is used to organize items in the simulation queue. Each event also has a Type attribute, and an ID attribute which is used to designate which simulation object in the system – bus, stop or route – should be used to execute the associated event actions.
- There two types of events at this point: the *move_bus event*, and the *riders_arrive event*. The *move_bus* event corresponds to the actions of a bus beginning its turn by allowing passengers to get on and off at its current stop, and then moving to the next stop on its route. The *riders_arrive* event corresponds to the actions of passengers who arrive at a departure stops at a certain time during the simulation with the intent to catch a bus to a different destination stop when possible.
- For the ***move_bus event***, we begin by envisioning the bus at its current stop. Currently, we generate random values to represent the number of riders who wish to get on and off the bus, respectively. We check certain limits to ensure that the numbers are reasonable: for example, we ensure that the number of riders getting on the bus is limited to the number of riders currently at the stop, and we also check that the number of riders who would like to board the bus don't exceed the capacity of the bus. We then update the number of passengers riding the bus, along with the number of riders still waiting at the stop.
- The final step of the *move_bus* event is to calculate the distance to the next stop on the route, and use the speed of the bus to calculate the time needed to travel to that location. This travel time is used to generate the rank for a new event, and enter that event into the simulation queue.
- For the ***riders_arrive event***, we use the functionality provided by the ***BusRiderGenerator class***. This class is very simple at this point, and basically just provides a bounded random value representing the number of new riders who have arrived at a stop at a certain point in time (when queried/called). For the sum of the event's actions, we add a random number (possibly zero) of new riders to each stop. A new *riders_arrive* event is then scheduled for a random time in the future to represent a constant and steady stream of arriving riders for the duration of the simulation run.

- The Reporting category includes the ***system_report*** command. This command is used to display the current state of various simulation objects, including the buses, stops and routes. This command will be useful when generating your Object Diagram.
- Quick note: "riders" and "passengers" are both people moving through our simulated transportation system, and there isn't intended to be any significant distinction between the two. The system currently uses the term "passengers" for people currently on a bus, as opposed to "riders" who are waiting at a stop.

Simulation System Commands

This is a quick guide to all of the MTS commands:

- **`add_stop, <Name>, <Riders>, <X-Coord>, <Y-Coord>`**

This command creates a stop object with a give <Name>, and an initial number of <Riders>, located on the travel map at the location <X-Coord>, <Y-Coord>.

- **`add_route, <Number>, <Name>`**

This command creates a route object with a given <Name> and <Number>. The route initially doesn't have any stops.

- **`extend_route, <Route ID>, <Stop ID>`**

This command appends the stop designated by <Stop ID> to the end of the route designated by <Route ID>. The ID values are the ones generated by the simulation system, not the "user facing" numbers and names entered with the commands.

- **`add_bus, <Route>, <Route Location>, <Passengers>, <Capacity>, <Speed>`**

This command creates a bus object that travels along <Route>, starting at <Route Location> with an initial number of <Passengers>. Also, the bus can hold at most <Capacity> passengers, and travels along the route at the given <Speed>.

- **`add_event, <Rank>, <Type>, <ID>`**

This command creates an event object that will be executed when the simulation reaches "logical time" <Rank>. When executed, the system will perform the actions corresponding to the event <Type> using the object designated by <ID>.

- **`step_once`**

This command executes one event from the simulation queue.

- **`step_multi, <Repetition>`**

This command executes <Repetition> events from the simulation queue.

- **`system_report`**

This command displays the attribute values of the simulation objects.

- **`quit`**

This command stops the simulation.

Evaluating Your UML Submissions

- You must generate your diagrams using an automated tool (e.g. Argo UML, LucidCharts, Microsoft Visio, etc.) so that they are as clear & legible as possible. Even PowerPoint is allowed, though this is an excellent opportunity to use a tool that is more appropriately designed for UML as opposed to a general drawing tool like PowerPoint. The choice of tool(s) is yours; however, you should do a "sanity check" to make sure that your final diagrams are readable when exported to PDF; or, if necessary, some reasonable graphical format (PNG, JPG or GIF).
- When I say "automated tool", I don't mean a tool that will automatically generate all of the diagrams for you based directly on the given source code. The assignment loses much of its effectiveness if you simply feed our source code into another application and then hand us the resulting output. The goal is for you to read the source code, work with the compiled and executable program, and generate the UML diagrams yourself to improve your understanding of the correspondence between the system and the UML diagrams that describe it in detail.
- Points will be granted based on:
  - Displaying the classes;
  - Displaying the corresponding attributes and operations for those classes;
  - Displaying supported relationships between your classes;
  - Showing generalization, aggregation and cardinalities for associations; and,
  - Displaying the data types for the values stored in the attributes and produced by the operation results.

- You must designate which version of UML you will be using – either 1.4 (the latest ISO-accepted version) or 2.0 (the latest OMG-accepted version). There are significant differences between the versions, so your diagram must be consistent with the standard you've designated. Either version is acceptable at this point in the course. We might require a specific version for some of the other UML style components later in the course, especially considering the behavioral/activity-based diagrams; and, if so, we'll let you know.
- You are permitted to add a few sentences to explain any aspects of your design that you feel need extra clarification. These sentences are optional: non-submissions will not be penalized.

Writing Style Guidelines: The style guidelines can be found on the course Udacity site, and at: https://s3.amazonaws.com/content.udacity-data.com/courses/gt-cs6310/assignments/writing.html

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.