ICS 45J: Programming in Java – Winter 2017 Lab 4
Deadline: 3/10/2017

This lab will focus on utilizing Multithreading, File IO, Arrays, and ArrayLists. You will simulate a city train system consisting of 5 train stations and 5 trains with their own train tracks. Your trains will be labeled 0,1,2,3 and 4. Additionally your train stations will be 0,1,2,3 and 4. (The labels of 0 – 4 are meant to make it easier to handle the indexes of our arrays since they are 0 based).

Configuration

The arrival rate and destination of passengers can be configured using a text file named "TrainConfig.txt". The configuration file will have the following format:

```
1000
100
2 4 100;5 2 300
3 0 500;1 4 200
5 0 200;2 1 500;3 3 600
4 0 200
2 3 300;6 2 100;4 0 40
```

The 1st line represents the duration of the entire simulation (number of simulated seconds).
The 2nd line represents how many milliseconds will count as one of our simulated seconds.
Lines 3 – 7 represent the passenger arrival rate of each station in the train track network. The general format to define passenger behavior is `[num_passengers station_destination rate_of_occurrence]`. Each train station can have many behaviors defined and each one will be separated by a semicolon (`;`). For example, looking at the sample TrainConfig.txt file above states it will be interpreted as follows:

- The simulation will be **1000** simulated seconds long. **(Line 1)**
- Each simulated second will be 100 milliseconds in real-time. **(Line 2)**
- Train station **0** will have 2 people requesting to go to the 4th train station every 100 simulated seconds and 5 people requesting to go to the 2nd train station every 300 simulated seconds.**(Line 3)**
- Train station 1 will have 3 people requesting to go to the 0th train station every 500 simulated seconds and 3 people requesting to go to the 3rd train station every 600 simulated seconds. **(Line 4)**
- Train station 2 will have 5 people requesting to go to the 0th train station every 200 simulated seconds, 2 people requesting to go to the 1st train station every 500 simulated seconds, and 3 people requesting to go to the 3rd train station every 600 simulated seconds. **(Line 5)**
- Train station 3 will have 4 people requesting to go the 0th train station every 200 simulated seconds. **(Line 6)**
- Train station 4 will have 2 people requesting to go to the 3rd train station every 300 simulated seconds, 6 people requesting to go to the 2nd train station every 100 simulated seconds, and 4 people requesting to go to the 0th train station every 40 simulated seconds. **(Line 7)**

Assumptions Regarding Configuration File:

- The configuration file will not contain passengers who are attempting to go to the current station they are at.
- You may assume the file is correctly formatted and has legal values for the simulation.

Train Rules

Each of the 5 train will be executed in a separate thread. Each train will have access to a TrainSystemManager object keeping state of the train stations. Each train can detect if a passenger is waiting at any given time for every train station in the train system via the TrainSystemManager. Several rules dictate the behavior of each train and are defined as:

- Each train will run in its own thread.
- Each train will check the state of the train system and passengers without sleeping.
- All trains start on the 0th train station.
- A train will only head to a train station to load passengers if it currently does not have any passengers and is not moving towards a train station.
- By default, a train will load all passengers requesting to go up first.
    - o If a train reaches a train station to load passengers and there are no passengers that want to go up, then the train will load all

passengers that want to go down.

- The train must drop off current passengers to the train station that it is closest to.
    - For example, if the train at train station 2 has two passengers requesting to go to train station 3 and train station 4, then it will stop at train station 3 first and then train station 4 (not 4 and then 3).
- If a train has no passengers to drop off, then it will continuously check all train stations to see if any passengers are waiting for a train and no other train is approaching that train station for passenger pickup.
    - No two trains will approach a train station for passenger pickup at the same time.
- If a train detects that there are passengers waiting on a train station, and no other train is currently approaching a train station for passenger pickup, then it will go towards that train station to load passengers.
- A train takes 10 simulated seconds to load / unload passengers.
- A train takes 5 simulated seconds to pass through one train station.
- A train does not have a max capacity of passengers.

You probably noticed that this is not a very efficient way for a train to operate (and can encounter annoying situations where people going down may have to wait until everyone going up is loaded). There are many ways to optimize the efficiency of the trains, but for the purposes of this lab, I want the focus to be on Java constructs and not train optimization to help simply the lab a bit.

Simulation State
As the simulation progresses, there are several pieces of information to keep track of (and will be useful for testing and checking consistency as discussed later):

For each train station in the train system:

- Keep track of the **total** number of passengers requesting a trains ride on the station.
    - This is defined by the TrainConfig.txt file behavior.
- Keep track of the **total** number of passengers that exited a train on the train station.
- Keep track of the **current** number of passengers waiting for a train on the train station.
- Keep track of the train **currently** heading towards the train station for passenger pickup.

For each train:

- Keep track of the **total** number of passengers that entered the train throughout the simulation.
- Keep track of the **total** number of passengers that exited this train on a specific train station.
- Keep track of the **current** number of passengers heading to any train station.

Organization
Some guidelines for structuring your code are (but not limited to):

- Lab4.java
    - Contains the main method that constructs a trainSimulation object, starts the simulation, and prints out some statistics (mainly the Simulation State described above).
- TrainSimulation.java
    - Class that sets up and controls the simulation. Some things to keep track of:
        - Train system components such as the train threads and the TrainSystemManager.
        - Simulation variables such as the total simulation time and the simulated second rate.
        - Configuration of the passenger behavior for each train station. This includes maintaining some structure (consider an ArrayList of ArrayLists of PassengerArrivals objects for each train station) to keep track of the passenger arrival behavior throughout the simulation.
    - Public Methods
        - start()
            - Runs the main simulation loop including incrementing the simulated time and managing passenger arrival behavior. The simulation ends when the current simulation time is greater than the total simulation time defined in trainConfig.txt
        - printTrainState()
            - Prints the state of the Trains and train stations described in **Simulation State** above.

    - Any other methods you deem necessary (such as a specific method to read the trainConfig.txt file to define the passenger arrivals).
- TrainSystemManager.java
  - Class that allow trains to manipulate the state of the train stations. There will only be one TrainSystemManager object constructed in trainSimulation. This object is shared among all train threads.
    - TrainSystemStations station[] trainStations – An array of Train system stations representing the state of all train stations in the train system.
  - Public Methods
    - Any methods you may need to access and manipulate the state of the Train stations. Since this object is shared among many threads, consider implementing the proper locking / synchronization mechanisms here.
- TrainStation.java
  - Class that represents the state of a specific train station in the train system including:
    - int[] totalDestinationRequests – An array where the ith element is representing the number of passengers who has requested to go to the ith train station throughout the simulation.
    - int[] arrivedPassengers – An array where the ith element is representing the number of passengers who have arrived on this train station from a train throughout the simulation.
    - int[] passengerRequests – An array where the ith element represents the number of people who currently want to travel to the ith train station of the train system.
    - int approachingtrain – The train ID that is currently heading to the train station for passenger pickup. The value, -1, can be used to represent no train currently heading in that direction.
  - Public Methods
    - Any getter / setter methods deemed necessary to maintain the correct state throughout the simulation.
- Train.java
  - Class representing a train and its behavior. This class implements the Runnable interface in order to run in its own thread. Some fields to maintain are:
    - int trainID – A unique ID (0 – 4) for a specific train.
    - int currentTrain station – The current train station the train is at.
    - int numPassengers - The current number of passengers in the train.
    - int totalLoadedPassengers – Total number of passengers this train loaded.
    - Int totalUnloadedPassengers – Total number of passengers this train unloaded.
    - ArrayList< trainEvent> moveQueue – Contains trainEvents that define the movement of a train and the anticipated time of arriving at a destination.
    - int[] passengerDestinations – An array where the ith element represents the number of current passengers who's destination is the ith train station.
    - TrainSystemManager manager – A train system manager provides methods to update the state of the train system via public methods. An instance of this manager object is shared among all train threads and is created in trainSimulation.
  - Public Methods
    - Train(int trainID, TrainSystemManager manager)
      - Constructor that takes in an ID and a TrainSystemManager object.
    - run()
      - Required method to implement since Train.class implements Runnable. Should continuously loop and process trainEvents in the moveQueue.
    - Any other methods to perform the train behavior such as creating trainEvents, managing the moveQueue, and updating its current state.

- TrainEvent.java
  - Class representing an event that the train is scheduled to perform. Events in this case contain information including the expected time that the event will occur. An event represents a train's destination train station and expected simulated arrival

time. Some fields this should contain are:
    - int destination – the train station that the train needs to move to.
    - expectedArrival – expected simulated time that the train will spend on that train station (including loading / unloading passengers and train station traversal time).
  - Public Methods
    - Any getters / setters necessary to maintain the state of a train Event.
- PassengerArrival.java
  - Class used to represent the passenger arrival behavior extracted from TrainConfig.txt.
    - int numPassengers – Represents the number of passengers that will request a train for this specific behavior.
    - int destinationTrain station – Represents the desired destination train station of the passengers.
    - int timePeriod – Represents the periodic time period these passengers will request train access.
    - int expectedTimeOfArrival – Represents the simulated time where the next batch of passengers will enter the simulation.
  - Public Methods
    - Any getters / setters necessary to maintain the state of a PassengerArrival object.
- SimClock.java
  - A class used to represent the simulated time. A SimClock only needs to tick forward and the clock starts at time 0. This class will only contain a single int value representing simulated time.
  - Public Methods
    - SimClock()
      - Initializes simulated time to 0.
    - public static void tick()
      - Increments the simulated time by 1.
    - Public static int getTime()
      - Returns the value of the simulated time.

Consistency
Since we are dealing with multi-threaded behavior and shared data among the entire simulation. You will need to apply any Locks and/or synchronized mechanisms discussed in class. Think about what may cause data dependency hazards and be sure to provide the appropriate safeguards to avoid them. Note that the shared memory object among the train threads is the TrainSystemManager. Be sure you're synchronizing access to this object in order to prevent race conditions.

Testing
For the testing portion of this lab, I would like students to provide a report (a .pdf file – there isn't a page limit, but should only include relevant content) containing sections of console output illustrating your functionality works as expected and the state of the simulation is consistent. Using TrainSystemManager along with the state of your Trains is useful to see if your simulation is working correctly or if you may be encountering any data dependency hazards. When running your simulation, you will need to output some information to the console and use this to support your report (for easily readable output, always put the simulated time when an event occurred at the start of an output line):

- Number of passengers entering a specific train station and requesting to go to the i^{th} train station.
- Any train action including:
  - A specific train heading to a specific train station to pickup passengers.
  - A specific train heading to a specific train station to unload passengers.
  - A specific train reaching a specific train station to pickup passengers. This includes the number of passenger(s) and the passenger'(s) destination station(s).
  - A specific train reaching a specific train station to unload passengers. This includes the number of passenger(s) exiting on the specific train station.
- After your simulation is over, print out the statistics for each train station and train. You can use this in your report to validate correct functionality in your program.