APPLICATION 3: QUEUE

TOPIC(S)

Queue application.

READINGS & REVIEW

- Carrano, Data Structures and Abstractions with Java, Chapters 10 & 11
- Queue ADT & implementation lectures

OBJECTIVES

Be able to:

- *Utilize one or more Queues as part of a simulation.*

INSTRUCTIONS

1. **You will work in self-selected groups of 3 members (enroll in Blackboard).**
2. ***Read the assignment carefully and ask questions about anything that you don't understand (before you start).***
3. For your application:
    a. Be sure to follow Good Programming Practices.
       Your code must comply with the Coding Standards/Guidelines posted on Blackboard.
    b. Keep in mind the Guidelines on Plagiarism.
4. Do a Write-up (Analysis / Summary). Each group member must submit this individually – it goes in the ADT assignment submission, not the application assignment!
5. Prepare a PowerPoint presentation. Your group will present during the class following the posted submission due date or as announced. Your group will submit the PP presentation file as part of the group submission – it goes in the zip file/in the project root folder.
6. Submit your work by the posted due date/time. Late submissions will incur a 25% penalty per day. Non-working projects (major or minor bugs) will incur at most a 10% penalty.
7. If you'd like my assistance, please zip your entire project folder and attach it to an email message with a brief description of your question/problem. I will typically respond within a few hours. Do not expect a response after 9p (5p on the due date).

PROBLEM

Write a program to simulate a ***Train Route***.

A ***TrainRoute*** consists of a number of ***Stations***, starting and ending with a terminal station. The time that a ***Train*** needs to travel between a pair of consecutive stations on the route is determined by the relative positions of the ***Station***s on the ***TrainRoute***. The locations of the ***Station***s are specified in the configuration file. Each ***Station*** has a platform for each track (direction) – total of two per ***Station***.

Each ***Station*** has two ***Queue*****s of *Passengers*** – one for each direction of travel (inbound/outbound or clockwise/counterclockwise)**.** ***This is the only required use of Queue(s). Your reference variables must be Queue<Passenger> which will point to LinkedList<>; you are restricted to only those methods defined in interface Queue plus clear(), size() and toArray() – toArray() may only be used for debugging and optionally to log the passengers remaining on a platform at simulation end.***

Passengers are generated at random times, assigned to entry ***Station***s randomly (e.g. added to the appropriate/respective Queue) and given random destination ***Station***s. Note: the destination and entry ***Station***s are always distinct (different ***Station***s). ***Passenger***s must wait for a ***Train*** which will take them to their destination ***Station*** using the most direct path (shortest time).

Each ***Train*** stops at every ***Station*** on the ***TrainRoute***. Put ***Station***s at both ends of the ***TrainRoute*** (for circular Routes, the Stations that are logically in the first and last locations (array indices [0] and [length-1]) – the supplied configuration does this. The initial location and direction of travel for each ***Train*** is specified in the configuration file.

When a ***Train*** stops at a ***Station***, all ***Passenger***s for that destination ***Station*** disembark first then any ***Passenger***s waiting to go in the ***Train***'s direction at that ***Station*** board the ***Train*** until either no ***Passenger***s are waiting to board that ***Train***, or the ***Train*** is full.

The simulation continues for a configuration-specified amount of time.

You must ***log all relevant actions*** (e.g. a ***Passenger*** arrives at a ***Station*** destined for a ***Station***). You must use the provided ***Logger*** class/utility. The ***log (detailed) will be written to a file***; ***some information (summary) should also be written to the console***. Provide a summary of the simulation activities including the starting and ending states (where ***Train***s are on the ***TrainRoute***, how many/which ***Passenger***s are still on them, how many/which ***Passenger***s are waiting at each ***Station***, etc.); the configuration is automatically written to the log.

Each class must maintain a counter which it uses to assign each instance of that class a unique id number; id numbers are unique within a class; duplicates will exist across classes. For example, the first ***Passenger*** will have id 1, the second will be 2, etc. Similarly, the first ***Train*** instantiated will have id 1. The constructor for the class must automatically assign the next number as part of its instantiation processing. Your *toString()* method for each class must include the kind of object and *id number* (e.g. "Passenger 13").

The log file:

- contains plain text
- is named TrainSimulation.log
- is in the data folder

Keep your simulation simple:

- make it single-threaded
- make it synchronous
- do not use timers; use a counter (ticks) to track time

Your simulation must be parameterized (you must read them from a file – do not hard-code them):

- length of route
- number of trains
- number of stations
- distance between stations
- rate of passenger arrival (instantiation)
- etc.

Use the provided ***Configuration*** class/utility to parse TrainSimulation.config. Several methods are including for retrieving the configuration elements.

The configuration file:

- contains plain text
- is named TrainSimulation.config
- is in the data folder
- accepts (and ignores) comments (introduced by a '#' character through the end of the line)
- accepts and ignores blank lines
- is in the format *parameterName = value(s)* – all aspects of the simulation must be controlled by parameters specified in TrainSimulation.config.

Follow heuristics for determining classes, instance variables and methods from the problem description:

- Everything in bold above must be a class (e.g. ***TrainRoute***, ***Train***, ***Passenger***, ***Station, TrainSimulation***). This is a hard-and-fast requirement, otherwise you can use a little creative freedom on the methods and instance variables.
- Verbs are candidates for methods (e.g. objects should handle their own behavior). For example, the *Train* class could have a *board()* method.
- Attributes are candidates for instance variables.
- Has-a (part-of) relationship between classes indicates containment (instance variables in a class).
- Is-a (type-of relationships between classes indicates inheritance.
- Please ask for clarification if you're not sure what these mean.

The assignment includes:

1. An Eclipse project (zipped) containing:

    - base versions (partial implementations) of the TrainRoute, Train, and Station classes
    - fully implemented Passenger class
    - Configuration class to parse TrainSimulation.config (provided in ./data/)
    - Logger class to produce TrainSimulation.log (sample provided in ./data/)
    - Direction and Location enums, and RouteStyle class which are utility classes encapsulating corresponding functionality
    - Javadoc for all supplied classes and enums (in ./doc)

    Make certain that you add a non-Javadoc comment block identifying the project, group, and primary coder/owner of every class you modify (minimally TrainRoute.java, Train.java, Station.java) or create (Simulation.java).

2. Notes (PDF) capturing the in-class discussion about the functionality of each component of the simulation in the real world, plus additional points for consideration (most are beyond the scope of this project)

3. A PowerPoint document/template for the presentation (in the ./presentation folder).

   Unless there is a compelling reason to do otherwise, please use PowerPoint for your presentation. *If you wish to use a different program, you need to get my approval in advance*. You do not need to use the provided templates; however, your documents must include all information in them including headers (if included), identification blocks, and footers. *Make sure you update all identifying information in the footers (typically the assignment and group numbers – the file name, date, and paging update automatically).*

GOOD PROGRAMMING PRACTICES

For all classes

- In your IDE, you must name your project "DS - App 3 Queue - Group ##" where ## is your group number – do not include the double quotes – you must match the spacing and single quotes precisely.
- Put all of your classes in a package named edu.wit.dcsn.comp2000.queueapp.
- Use mnemonic/fully self-descriptive names for all class members (methods, variables, parameters, etc.).
- Make your instance variables **private**
- Include **constructors** to initialize your instance variables
- Derived class constructors should **leave initialization of super class instance variables** to the super class's constructors:
  - Remember the call to the super classes constructor is: **super(<init1>, <init2>,..)**
- If appropriate, include **Accessor** and **Mutator** methods for all instance variables *(please ask if you're not sure what these are)*
- Include a **main()** method for testing (unless it's an abstract class) and test before you proceed…
- Add comments to your code, not just so it's easier for other readers, but also so it's easier for you to remember your logic

WRITE-UP

The write-up is an individual assignment. Each group member must complete the write-up included with the ADT assignment and submit it as part of the ADT assignment, *not* the application assignment on BB.

SUBMITTING YOUR WORK

1. Make sure the course number, assignment (App 3/Queue: Train Simulation), your group number are in all your project files. For the Powerpoint presentation, list each team member's name and the specific class(es) they 'own.' ***For the source modules, include the name of the class owner – the individual primarily responsible for the design, implementation, testing, and maintenance – do not list all members in all classes – you must delegate responsibilities.*** If any other group member makes *significant* contributions, list their name(s) too.

2. Your presentation file name must be consistent with the template:
   a. Required: update/insert your group number
   b. Optional: update the date/version number

3. Your Java class files must be 'properly' named/match the class defined within.

4. You must include a comprehensive set of unit tests for your classes (except for your application). I suggest that the entire team define the test 'suite.'

   Style requirement: For unit tests, you may include a *main()* method (unit test driver) which must follow all public and private methods and inner classes which implement the API/ADT. *private* utility methods for testing must follow the *main()* method. An alternative is to use JUnit 5/Jupiter; this may require you make some methods package-visible rather than private.

   Style requirement: You must read the configuration parameters controlling the simulation from the TrainSimulation.config file. For testing and debugging purposes, you may create variations on this file (leave them in the data folder or create a test folder for them) and specify the alternate file when you instantiate the Configuration. Your application must work with the provided configuration.

   Style requirement: You are not permitted to use global variables except as explicitly noted above (id numbers, logger state).

   Style requirement: You are not permitted to use separate, distinct variables for each train, station, etc.; you must use an appropriate collection instead. You may use temporary variables to instantiate and manipulate instances.

5. Spell check and grammar check your work!

6. Make a **.zip file** for your project. ***.rar, .tar, etc. are not acceptable!*** Use Eclipse's Export… tool to create the zip file. If you are using a different IDE, instructions for alternate methods to create a zip file are posted on Blackboard.

   - Include your entire project folder (including subfolders).
   - Your PowerPoint presentation must go in the project's ./presentation folder. **Do *not* attach them directly to the Blackboard submission.**
   - In Blackboard, **attach** your solution file to the submission for this assignment – click on the assignment title to access the submission page. **Only one team member submits the project/code and PowerPoint presentation for the entire team**.

7. Each group member must complete the write-up included with the ADT assignment and submit it as the separate ADT & Application Write-up assignment, not the application assignment on BB.

GROUP PRESENTATION

1. All group members are expected to participate in the presentation.

2. Make sure your group number and all names and specific roles/class(es) 'owned' are on the title slide and the group number is in the footer of all following slides.

3. You may use any theme, template or style you wish (recommendation: keep it appropriately professional). Make sure the font size is large enough to read from the back of the room.

4. Minimally, your presentation must include all information in the provided template.

5. Presentations should take approximately 3-5 minutes, including a demonstration of your application and Q&A.

6. ***Do not read the slides to the class***. The slides should typically be short, bulleted lists of talking points. *Face the class* when speaking and *speak loudly* enough to be heard in the back of the room.

7. ***Do not include code in the slides***. If you want to show selected portions of your implementation, do so using your IDE or copy the sections of code into another application for display purposes only (again, make sure you set the font to a large enough size – try 20 pt in Eclipse, larger in IntelliJ - to be readable from the back of the room).

8. Make sure the presenter's computer is adequately charged and has the correct versions of the PowerPoint presentation and code. Prior to coming up, open the presentation and your IDE (so you can demonstrate your application) and make sure they run properly. I will provide HDMI and Thunderbolt cables/connectors.

GUIDELINES ON PLAGIARISM IN COMPUTER SCIENCE

- ***Plagiarism/cheating will result in immediate failure for this course.***
- ***The Provost's and Registrar's offices will not permit you to withdraw from a course which you failed due to cheating.***
- ***An Academic Review Board may impose additional penalties.***
- ***You will not be allowed to attend class for the remainder of the semester.***

CLEAR PLAGIARISM

A clear case of plagiarism exists if a student **passes off someone else's work as his or her own**. Examples of such behavior include (but are not limited to) the following:

- A group of students each performing a separate part of an assignment. Each student in the group then hands in the cumulative effort as his or her own work. This is different from members of a group dividing up the work then submitting it as the product of the group's efforts (as is the case with this assignment).
- A student making cosmetic alterations to another's work and then handing it in as his or her own.
- A student having another person complete an assignment and then handing it in as his or her own.
- A student handing in (as his or her own) a solution to an assignment performed by someone else from a previous offering of the course.

These are all cases of indisputable plagiarism and are characterized by the submission of work, performed by another, under one's own name.

PERMITTED COLLABORATION

Collaboration and research, where they do not constitute plagiarism, should be generally encouraged. These efforts constitute honest attempts at obtaining a deeper understanding of the problem at hand. They can also serve as a validation of a student's approach to a problem. Some examples are as follows:

- A student researching existing, public approaches to a problem presented as an assignment.
- A group of students discussing existing, public approaches to a problem presented as an assignment.
- A group of students discussing the requirements of an assignment.
- A student discussing the merits of his or her proposed solution to an assignment with the instructor or teaching assistant.

Provided no plagiarism is committed, these are all valid (and encouraged) activities.

THE GRAY AREA

The differences between the examples of plagiarism and encouraged collaboration above are those of originality and acknowledgment. In general, any work submitted without acknowledgment which is not the original work of the indicated author constitutes plagiarism. Some examples which illustrate this point follow. Note that while some of the examples do not constitute academic misconduct, they may be of questionable academic value. Also note that anyone (knowingly or through negligence) contributing to someone else's academic misconduct is also guilty of the same.

- It is acceptable to use solution proposals presented by the instructor or teaching assistant. The acknowledgment is implicit. Explicit acknowledgment is not usually required.
- It is not acceptable to use publicly available work without acknowledgment.
- It is not acceptable to use a full or partial solution obtained from or by another through any means (verbal, written, electronic, etc.), without that person's permission.
- It is not acceptable to collaborate on a single solution without acknowledgment.
- It is not acceptable to discuss solutions or partial solutions to a problem and then incorporate them without acknowledgment.
- It is acceptable to implement a standard solution to a problem without acknowledgment, but it is not acceptable to incorporate someone else's implementation without acknowledgment. Here standard solution means a commonly used data structure or algorithm.
- It is not acceptable to re-submit an assignment from another course or a previous offering of the same course without acknowledgment, regardless of authorship.
- It is not acceptable to make a solution available as an aid to others.
- ***You may not request solutions on any internet site (e.g. stackoverflow.com).***

This set of examples helps define the bounds between encouraged behavior and misconduct but does not constitute an exhaustive set.