CS 0445 Data Structures
Programming Assignment 1

Online: Sunday, January 20, 2019
What is due: All **source** and **data** files, all **executable** / **.class** files, a project **write-up**, plus a completed **Assignment Information Sheet**.
How to submit: A single zip file containing all files (including all source code needed to run your program, all output files, your write-up file and your completed Assignment Information Sheet) must be submitted electronically, to the CS 0445 submission site, in the proper directory. For more submission details, see submit.html.
When it is due: All files submitted to the proper submission directory by **Monday, February 4, 2019, by 11:59PM**
Late due date: Wednesday, February 6, 2019 by 11:59PM

Submission Note: The TA will not be using any IDE to compile / run your projects. Therefore, you should submit ONLY Java source files that can be compiled and run on the command line. If you use an IDE for development, test your code on the command line before submitting. If the TA cannot compile / run your program, you will lose most of the credit even if it runs for you on your IDE.

Description and Specifications:

You have decided to use a simulation to determine two things about a bank:

1) Which method of lining up in banks (and other establishments) is better: a) a single queue that feeds all of the tellers or b) an individual queue at each teller.
2) How many tellers you should have working in your bank.

You will use a discrete-event simulation for your test, in which you will 'generate' customers at random and keep track of their progress through two fictitious banks, **SQBank, Inc.** and **MultiBank, Ltd.** You may assume each bank has K tellers on the job, and that **SQBank uses a single queue** to feed all K tellers, while **MultiBank uses K queues**, one for each teller. Each bank opens at 8AM and closes to new customers at 4PM (i.e. is open for 8 hours). However, customers already in the bank at 4PM will be allowed to complete their transactions.

Your goal is to determine the following **for each customer** at each bank:

1) customer # (order customer was **generated**)
2) time upon arrival at the bank (time customer is generated – see Implementation Details).
3) time required for transaction (in minutes – see Implementation Details)
4) whether or not the customer stays at the bank (yes or no; if the answer is no, do not calculate any of the values below. For details on how to decide, see Implementation Details)
5) queue id [-1 if customer does not have to wait, or value from 0 to K-1]
6) teller id who serves customer [value from 0 to K-1]
7) time service begins for customer
8) time customer waits for service [item 7 – item 2]
9) time service ends for customer
10) total time of customer in the bank [item 9 – item 2]

This information should be output for each customer at the end of your simulation.

In addition, you must determine the following **for each bank**:

1) The number of Queues being used (could be either 1 or K)
2) The maximum number of customers who can be waiting in line at the same time
3) The specified customer arrival rate (per hr)
4) The specified mean service time (in min)
5) The total number of customers generated

6) The total number of customers served
7) The total number of customers turned away due to long lines (see Implementation Details)

For the remaining items below, do NOT count the customers that left the bank without being served.

8) The number of customers who had to wait before being served
9) The average wait time spent by all customers in the bank (in min) (see Implementation Details)
10) The maximum wait time for any customer in the bank (in min)
11) The 1st standard deviation of the wait time for all customers in the bank (see Implementation Details)
12) The average service time of customers (measured, in min)
13) The average wait time spent by customers who had to wait (in min)
14) The average time in the system for all customers (in min)

Implementation Details

General Guidelines

- Your program must be an **event-scheduled discrete event simulation**. It must be written in Java in a good object-oriented manner. This means that your entities (Customers, Tellers) and other logical objects (ex: events) should be implemented using classes.
- All code **other than** standard classes / methods and code given out in class (ex: the SimEvent class, the PriorityQueue class) must be written by you.

Overall Program Setup

- The (fairly) simple main program for this assignment is provided for you in file Assig1.java. Read over this file vary carefully. Your SimBank class must run with this main program as is (no alterations).
- Note especially that your SimBank class must be able to represent both types of banks – that with a single queue and that with K queues. This can be done by making an ArrayList of Queue. For the single queue model you will put only a single Queue into your ArrayList. For the multi-queue model, you will put K Queues into your ArrayList, one for each of the K tellers.

Queue Management

- Each bank can accommodate a grand total of K + M customers (K customers at the K tellers plus at most M in line(s)). Clearly, for MultiBank, these M customers will be divided amongst the K queues, while in SQBank they will be in a single queue. New customers that would increase the waiting number to above M do not stay in the bank and are not processed beyond this point.
- When entering either bank, a customer will first check to see if there is an open teller. If so, the customer will immediately be served by that teller and will not have to wait in line. If more than one teller is available, the decision should be made arbitrarily.
- If no teller is available, and if fewer than M customers are already waiting (in all of the queues) then the customer will go into a queue. If M customers are already waiting then the customer will not remain in the bank.
- If a customer enters a queue in MultiBank, the shortest queue will always be chosen. If more than one queue is the same (shortest) length, the decision should be made arbitrarily.
- Once a customer is in a MultiBank queue, the customer must stay in that queue until served by a teller. In other words, a MultiBank customer in queue[i] must be waited on by teller[i]. However, an SQBank customer could be waited on by any teller.
- [Note: This policy for MultiBank is not necessarily realistic, as it may be possible for a customer to switch queues while within the bank. For more on this possibility see the Extra Credit options.]

Events

- There will be two types of events in your simulation, ArrivalEvent and CompletionLocEvent, both subclasses of SimEvent. These classes are provided for you – see SimEvent.java, ArrivalEvent.java, CompletionEvent.java and CompletionLocEvent.java.
- Each event will have a timestamp (see constructor for SimEvent). Note that SimEvent is Comparable and the compareTo() method is based on the timestamp of the event.
- All events will be placed into and removed from a PriorityQueue, called the Future Event List [FEL].
- Initially you should generate the first ArrivalEvent and place it into your FEL. Your main program will then iterate, removing the next event from the FEL with each iteration.

- If the event is an ArrivalEvent, a new Customer is generated with the given arrival timestamp. A service time (i.e. length of service) is then generated for the customer. The customer is then processed as described above. If a teller is available a CompletionLocEvent is generated for that Customer at that teller and placed into the FEL. If a teller is not available the Customer is placed into a queue. After the ArrivalEvent is processed, if the bank will still be open at the next timestamp, a new ArrivalEvent is generated and put into the FEL.
- If the event is a CompletionLocEvent the Customer at that teller is removed from the teller and processed out of the bank. If another Customer is waiting in a queue, that Customer is placed into the teller, and a new CompletionLocEvent is generated for that Customer and placed into the FEL.
- The simulation will continue running until no more events are in the FEL.

Random Numbers

- Both your Customer interarrival times and service times must be exponentially distributed. These can be generated using the RandDist class. Note that the argument to the exponential() function is the arrival rate, lambda. However, the service time for the program is specified as a mean value. The arrival rate, lambda and the mean value for the distribution have an inverse relationship (i.e. mean = 1/lambda). Keep this in mind when generating your random values.
- Also note that the arrival rate is specified in arrivals per hour, while the service time is specified in minutes. Make sure you convert so that all of your times are consistent.
- You will create a RandDist object for each of your banks, each with the same seed. This will enable the arrival and service values for both banks to be identical. Having identical pseudo-random values for both banks will make the results of your simulation more accurate.
- For more on using the RandDist class, see handout RandTest.java.
- Exponential distributions give a reasonable approximation of actual customer arrival and service values. For more information on the exponential distribution see: https://en.wikipedia.org/wiki/Exponential_distribution

Statistics

- Use the formulas below for mean and standard deviation. Assume we have N values for random variable Y (where in this case Y is the wait time of a customer) given by $y_1, y_2, \ldots, y_N$.

Mean (or expected value) = $\bar{y} = \frac{1}{N}\sum_{i=1}^{N} y_i$.

Standard Deviation = $\sigma = \sqrt{\frac{\sum_{i=1}^{N}(y_i - \bar{y})^2}{N}}$

Execution / Runs / Input / Output

- Note that in the main program, Assig1.java, the SimBank objects are initialized with several arguments, most of which are input by the user:
  - ntell = number of tellers
  - true / false = whether or not to use a single queue
  - hrs = hours the simulation will run
  - arr_rate = rate of customer arrivals per hour
  - t_min = average transaction time in minutes
  - maxq = maximum number of customers waiting (not being serviced)
  - seed = random seed to initialize RandDist object
- You should run the Assig1 program 5 times. Note that for each run, 2 SimBank objects are created, one with a single queue (true) and one with multiple queues (false).
- The following values should be the same for all 4 runs: hrs = 8, arr_rate = 30, t_min = 6, maxq = 10, seed = 80.
- The ntell value should change with each run, going from 1 to 5.
- In other words, you will run the simulation 5 times for each bank, with the only difference in the runs being the number of tellers.

- **Note:** When ntell = 1, the results for both of your banks should be identical.
- Your program output should contain a list of all of the customers who were served by the bank and another list of all of the customers who did not stay in the bank. The information for each customer should be that specified in the Description section of this document. For an example of the output see file "a1-out2.txt" on the Assignments page.
- Since your program will generate a lot of output for each run it will be helpful for analysis and grading for the output to be sent to a file. This can be done in a very easy way using redirection from the command prompt. In this way, you can write all of your output to standard output and see it as it is generated during program development. Then when you are ready to do your official runs, you can send them to files. This can be done in the following way:
  `java Assig1 > a1-out1.txt`
  The statement above will send all output from the program to file a1-out1.txt.
- Use the following names for your output files:
  `a1-out1.txt` – 1 teller
  `a1-out2.txt` – 2 tellers
  `a1-out3.txt` – 3 tellers
  `a1-out4.txt` – 4 tellers
  `a1-out5.txt` – 5 tellers
  All of these output files must be included in your .zip submission file.
- Note that when you redirect the output to a file, you will also not see the prompts for input when the program is initialized (since they will also go to the file). The input will still work – you just need to remember which values to enter at each point in the program initialization.

Write-up / Analysis:

Once you have done all of your runs, look over your results and briefly answer (in a few paragraphs) the following questions in a text document, called `a1-writeup.txt`:

1) Which method (single queue, multi-queue) appears to be preferable and why? Note specific values in your results when explaining why.
2) How many tellers do you think would be optimal for your bank? Note that this could have more than one answer, and a more informed answer may require more information from your simulation. However, try to justify your answer as best you can.

Extra Credit Options:

- As is a common practice in real life, customers may change queues if they perceive another queue to be better than theirs. For example, I choose Queue 1 because it is the shortest when I arrive, but then several customers leave another queue and that one becomes shorter. I may then decide to switch to the (newly) shorter line. For extra credit implement some reasonable queue-swapping policy for your MultiQueue bank. This should be optional, so that you can compare your results with queue-swapping to that without queue-swapping. Note that if you allow queue-swapping you will need to use something other than a Queue for your lines in the bank. A Deque might work but you should think about how the data needs to be manipulated in the lines.
- Due to the pseudo-random nature of the data, simulation results typically require several runs with the same parameters (but different data) in order to be significant. Results are also analyzed using statistical measures. This is non-trivial and requires some background in statistical analysis. You can get extra credit if you do multiple runs and do some reasonable analysis of your results (both for the runs within a single bank and for the results between the different banks). For some background on this see: https://en.wikipedia.org/wiki/Confidence_interval . Note: This will be MUCH MORE work than you will be compensated for, so don't try it unless you have a lot of time on your hands and mostly want to do it just for fun.
- Extra credit will be up to 10 points total (added to your original score out of 100). Either option, if done well could get you the full 10 points but you cannot get more than 10 no matter how much extra credit you do.