

## COMP 250 Assignment 2

**Posted:** Saturday, Feb. 12, 2011

**Due:**

- Question 1 (WebCT): Sunday, Feb. 27, 2011 at 23:59.
- Question 2 (hard copy): Monday, Feb. 28 in class (or see WebCT for alternative hand in instructions)

### **General Instructions**

- The T.A.s handing this assignment are Faiyaz, Jonathan, and Juan. Their office hours and location will be posted on the public web page.
- Use the WebCT discussion boards for clarification questions only.
- Do not change any of the given code. Only add code where instructed. Do not change the package name, as this slows down the grading.
- The starter code that is provided includes comments before each method that allow you to generate Javadoc. You are encouraged to learn Javadoc and practice using it, e.g. it will allow you to better understand the class specifications given in the Java API ([click here](#))

For instructions on how to generate javadoc in Eclipse, see ([click here](#))

- Use Java naming conventions for variable names ([click here](#))
- You are expected to comment your code so that the grader can easily follow what the code is doing. Points will be removed for poor style (improper indentation, non-existing or unhelpful comments)
- Use WebCT to hand in your code. Include a README file for any issues that you wish the grader to be aware of.
- **Late assignment policy:** Late assignments will be accepted up to only 3 days late and will be penalized by 20 points per day. If you submit one minute late, this is equivalent to submitting 23 hours and 59 minutes late, etc.

## Question 1 (50 points)

Queues arise in many real situations where you have clients that share a server. Examples include customers at Tim Horton's, phone callers on a Sympatico help line, airplanes waiting to take off or land at Trudeau, and many more. Queues also are often used in computer science, for example, your computer's operating system (a server) needs to coordinate how the various programs that are running on the computer (clients) share time slices of the central processor. Other examples include web servers and mail servers, the clients being the computers on a network.

Here we will simulate a simple queuing system. We will use the terms customer and teller as in a bank simulation, rather than the general terms client and server.

We will compare the performance of two common *scheduling algorithms* for queuing systems:

- *(standard) queue* – “first come, first served”
- *round robin* - first come, first served *except* that a customer is served for at most  $T$  consecutive time units. If a customer has received  $T$  consecutive time units and if it needs more service time then the customer must return to the back of the queue. For example, a customer that requires  $mT$  units of service time will have to pass through the queue  $m$  times. With this policy, customers that require more service time will end up spending more time in the queue.

The queuing system has the following properties.

- There is a clock that counts time steps.
- At most one new customer may arrive at the queuing system at any time step, and the arrival occurs at the beginning of the time step. If a new customer does arrive at any given time step, it enters the customer queue, even if the queue is empty. The probability of a customer arrival at any time  $t$  is  $P_{\text{ARRIVAL}}$  and the probability of no arrival is thus  $1 - P_{\text{ARRIVAL}}$ . This arrival probability is constant and independent of arrival events in the past. Think of an unfair coin being tossed at each time to decide whether a new customer arrives or not. In probability theory, such “coin tosses” are called *Bernoulli trials*.
- There is one teller. During any time step, the teller is considered to be either busy or not busy, namely serving a customer or not.
- Different customers require different numbers of service time units. Again, this is modeled by Bernoulli trials, namely if a customer is being served at some time, then the probability that it is finished being served after that time is  $P_{\text{DEPARTURE}}$ . We require that  $P_{\text{ARRIVAL}} < P_{\text{DEPARTURE}}$  since otherwise the queue would tend to grow without bound.

You are given fully implemented classes **Customer**, **Teller** and **Clock** as well as partially implemented classes **CustomerQueue** and **RunSimulation**.

The main method of the class **RunSimulation** computes statistics that can be used to compare the performance for different P\_ARRIVAL and P\_DEPARTURE settings, and for standard queue versus round-robin with parameter T. For example, it computes:

- the arrival rate (the average number of arrivals per unit time -- in theory, this is P\_ARRIVAL)
- the number of customers served
- the fraction of time that the server is busy. (This is equivalent to the average number of clients being served at any time.)
- the mean queue length
- the average amount of time that clients spend waiting in the queue.
- the average amount of time that a client spends being served
- etc

## Requirements

You must complete the methods in the **CustomerQueue** and **RunSimulation** classes, according to the given specifications.

### A. (10 points)

Complete the **enqueue**, **dequeue**, **isEmpty** and **size** methods of the **CustomerQueue** class. The queue is implemented using a linked list of type **Customer**. You are allowed to use any `java.util.LinkedList` methods. For the complete documentation, see [\(here\)](#).

### B. (30 points)

Complete the partially implemented **main** method of the **RunSimulation** class to incorporate the simulation of a bank queue. The simulation should be performed in the following order.

1. Check if the teller is busy. If yes, check whether the customer being served is finished i.e. should depart the system. If yes, first update the queue duration (the total amount of time the customer has spent in the queue) and then, move this customer to a separate list of serviced customers which is used for gathering statistics about customers (see below). Otherwise, if the customer that is being served is not finished, check if the customer has been served for T consecutive time units and in that case send that customer to the back of the customer

queue and set the teller's status to not busy. Note that in the standard queue (as opposed to the round robin queue), we may assume  $T$  is infinity (or a large number) since there is no limit on consecutive service times. This allows us to use the same program to model the two types of scheduling policies. The constant `MAX_SERVICE_TIME` in the class **Teller** represents the value of  $T$  for a particular scheduling algorithm.

2. Check if a new customer arrives at the system. If so, the customer joins the customer queue.
3. If the teller is not busy and if there is at least one customer in the queue, then the teller should welcome the next customer, i.e. remove it from the queue.
4. if the teller is busy, then it serves the customer for that time step and updates the quantities for number of customers in the system (`sumCustomersInSystem`) and the queue lengths (`sumQueueLengths`) for statistics gathering.

**c. (5 points)**

Complete the partially implemented method `serviceDurationVsQueueDuration` of the **RunSimulation** class. This method prints the mean waiting times of the customers as a function of their actual service duration. As the customers' service duration can be distributed over a wide range, the method only considers the customers having service duration in the range from 1 to (`maxServiceDurationToConsider - 1`).

**d. (5 points)**

Run the **RunSimulation** class for both the queue scheduling schemes. What is the key difference between the standard queue and a round-robin queue results that you observe? Write your answer in a file named `qu4_prog.txt`.

**Important Notes:**

- The simulation of customer arrivals and departures is modeled using random number generators. Naturally, if you run your code multiple times, the results will be different. However, if you run the code for a large number of time steps (10,000,000 to 100,000,000 time steps ), the results should not vary too much between different runs.
- To make it easier for you to verify the correctness of your simulation methods, we output theoretical expected values for the quantities. If your simulation's experimental results are close to these theoretical bounds, your code is doing fine. However, do not expect the experimental results to match exactly with the theoretical results.
- The theoretical results are only applicable when the `P_ARRIVAL` and `P_DEPARTURE` are very small and `P_ARRIVAL < P_DEPARTURE`. In the **Customer** class, we define

the values of P\_ARRIVAL and P\_DEPARTURE. It's recommended that you do not change these values.

- The value of T (MAX\_SERVICE\_TIME) is set to 5 for round-robin scheduling and to a very large number ( $2^{31} - 1$ ) for the standard scheduling. It's recommended that you do not change these values.

### You need to submit:

CustomerQueue.java    RunSimulation.java    q4\_prog.txt

### Question 2 (50 points)

Give a formal proof of each of the following, by applying the definitions given in class. You are not allowed to use "limits".

A.  $t(n) = \sqrt{31n + 12n \log n + 57}$  is  $O(\sqrt{n} \log n)$ .

B.  $t(n) = (n - 3 \log n)^{1.6} + 5n^{1.5} + 7$  is  $\Omega(n^{1.6})$ .

Solve the following recurrences. You may assume for simplicity that  $n$  is a power of 2, and you may take the base case to be  $t(1) = c$ .

C.  $t(n) = 5t\left(\frac{n}{2}\right) + n$

D.  $t(n) = t\left(\frac{n}{2}\right) + 3 \log n$

E.  $t(n) = 2t\left(\frac{n}{2}\right) + n^2$

Hand in a hard copy of your answers. Your answers must be clearly legible.

Have fun. Good luck!