HW5: Programming with queues

Due: Due 1:59PM Wednesday, Oct 8 through turnin
Total: 20pts

You may work on this assignment with one other student. A team of two members must practice pair programming. Pair programming "is a practice in which two programmers work side-by-side at one computer, continuously collaborating on the same design, algorithm, code, or test." Both members in a team must read the article All I really need to know about pair programming I learned in kindergarten. This article can be downloaded to any computer that is in the Marquette network.

Preparation

1. Queue interface and array based implementation. Refer to the textbook.
2. The simulation example in the textbook.
3. Randon number generation in Java. The method Math.random( ) returns a double value with a positive sign, greater than or equal to 0.0 and less than 1.0.

Task

Since iphone 6 and 6+ is available, an Apple store in downtown Milwaukee has seen more customers each day, and the store manager wants to hire us to figure out how he should schedule staff during the store hours.

In this assignment, you will simulate a customer service operation in the Apple store with customers being served by staff members. The store notices that on average, a certain number of customers visit the store per hour. The customers arrive at random time. They wait in line until a staff is available to help them, or may choose to go away. If there are five or more customers in the line, a new arriving customer has 50% chance to go away, and 50% chance to wait at the end of the line.

The waiting line is implemented with queue data structure. In addition, the store keeps track of customers who have been helped and customers who have chosen to go away in order.

The following are the classes you will design. You may need to design other classes of your choices.

- Class ArrayQueue<E>: array-based queue implementation with generic data type E.
    1. ArrayQueue implements the Queue Interface, which can be downloaded from the course website.
    2. ArrayQueue objects have an initial capacity, which is determined by one commandline argument. An ArrayQueue object doubles its capacity when the number of customers is greater than its capacity.

- Class Customer.
    1. A general Customer object has two attributes: the `arrival` time as `int`, and the `processTime` as `int` that the customer will need to be served. The `processTime` is a random value between 1 and 3.

    2. A customer who has been helped has an extra attribute: `startTime` as `int` when the customer is removed from the waiting line and helped.

- Class AppleStore.

```
/*
 * Constructor
 * @param qCapacity The initial capacity of the queue to be used.
 * @param simHours The number of hours that the simulation should run.
 * @param custPerHour expected number of customers to arrive per hour.
 */
 public AppleStore(int qCapacity, int simHours, int custPerHour)

/**
 * This methods performs a simulation of a store operation using a
 * queue and prints the statistics.
 * For every minute, the simulator 1) checks if there are new customers
 * arriving; 2) adds the new customer into the waiting
 * line or else records the customer who chooses to leave; 3) continues
 * to help the current customer if the current customer is not finished
 * yet, or else get the next person in the waiting line.
 * The simulator starts at minute 0, and repeats every minute until
 * it finishes the requested simulation time.
 */
 public void simulation( )

 /**
 *  print the info of all accepted customers
 */
  public void displayAcceptedCustomers()

 /**
 * print the info of all served customers
 */

  public void displayServedCustomers()

  /**
  * print the info of all waiting customers
  */
  public void displayWaitingCustomers()

 /**
 * print the info of all turned away customers
 */
 public void displayTurnAwayCustomers()
```

Test your design and implementation

Your program will not generate exactly the same outputs as the samples for the same input values due to random numbers used in the simulation. Nevertheless, your program should be able to reflect the average number of arriving customers per hour, use the same output format and output the same information.

A driver program `AppleStoreRunner.java` is provided for you. This driver program cannot be changed. Your program should work with this driver. This program takes three integer numbers as command line arguments. For example,

```
java  AppleStoreRunner 2 1 10
```

```
// the first integer (2 in the example) is the initial size for the customer queue where
// customers wait.
// the second integer (1 in the example) is the number of hours you will simulate.
// the third integer (10 in the example) is the average number of customers that come in.
```

One sample output is as follows:

```
Average waiting time: 0.3
Average line length: 0.05

Customers accepted: 10
Customer arriving@0; requesting 2
Customer arriving@6; requesting 3
Customer arriving@13; requesting 2
Customer arriving@21; requesting 2
Customer arriving@22; requesting 1
Customer arriving@30; requesting 2
Customer arriving@32; requesting 3
Customer arriving@35; requesting 1
Customer arriving@36; requesting 3
Customer arriving@37; requesting 3


Customers served: 10
Customer arriving@0; requesting 2; being waiting 0; starting@0; finishing@1
Customer arriving@6; requesting 3; being waiting 0; starting@6; finishing@8
Customer arriving@13; requesting 2; being waiting 0; starting@13; finishing@14
Customer arriving@21; requesting 2; being waiting 0; starting@21; finishing@22
Customer arriving@22; requesting 1; being waiting 1; starting@23; finishing@23
Customer arriving@30; requesting 2; being waiting 0; starting@30; finishing@31
Customer arriving@32; requesting 3; being waiting 0; starting@32; finishing@34
Customer arriving@35; requesting 1; being waiting 0; starting@35; finishing@35
Customer arriving@36; requesting 3; being waiting 0; starting@36; finishing@38
Customer arriving@37; requesting 3; being waiting 2; starting@39; finishing@41

Customers still waiting: 0

Customers turning away: 0
```

Another sample output:

```
//Samilar results as the previous example, except for the customers who are
still waiting in the line and who have turned away.
…
Customer arriving@48; requesting 1; being waiting 9; starting@57; finishing@57
Customer arriving@49; requesting 2; being waiting 9; starting@58; finishing@59


Customers still waiting: 5
Customer arriving@53; requesting 3
Customer arriving@54; requesting 1
Customer arriving@55; requesting 2
Customer arriving@56; requesting 2
Customer arriving@59; requesting 1


Customers turning away: 3
Customer arriving@30; requesting 3; leaving;  5 already waiting in line
Customer arriving@31; requesting 1; leaving;  5 already waiting in line
Customer arriving@58; requesting 1; leaving;  5 already waiting in line
```

Submission

Submit your java files including ArrayQueue.java, Customer.java, AppleStore.java, and other java files to turnin system. **Do not submit `AppleStoreRunner.java` and `Queue.java`.** The provided two java files will be used to compile and run your submission. If your submission requires different versions of `AppleStoreRunner.java` and `Queue.java`, it won't pass compilation and tests.

You should use a single command to turnin all files and separate the files with space. For example:

```
turnin -c cosc2010-Ge -p AppleStore ArrayQueue.java Customer.java AppleStore.java
```

You can submit multiple times but only the last submission will be saved by turnin. If you work in a team of two, always let the same user submit the files. The lower grade from two submissions will be used if both team members submit files.