HOMEWORK - SPRING 2014

HOMEWORK 4 - due Tuesday, March 25 no later than 5:00PM

REMINDERS:

- **Be sure your code follows the coding style for CSE214.**
- **Make sure you read the warnings about academic dishonesty. *Remember, all work you submit for homework or exams MUST be your own work.***
- **Login to your grading account and click "Submit Assignment" to upload and submit your assignment.**
- **You may use any Java API class that you wish.**
- **You may use Scanner, InputStreamReader, or any other class that you wish for keyboard input.**

In this homework assignment, you will implement a simulation of service in a fast food restaurant. In this restaurant, customers have the option of placing their order from an electronic kiosk or at the cashier (with a human). The kiosks are faster on average, but have a certain probability of failure. When a kiosk fails, a cashier must stop taking orders to go and fix it. Each kiosk and cashier has a line, which is represented as a queue. In this assignment, a time unit is one second. To develop this simulation, you will implement the following classes:

1. public class Customer - **This class represents one customer. Each customer must keep track of the time on the simulation clock when he/she entered the restaurant, as well as the time it will take to serve the customer (a random number generated when the Customer enters the restaurant).**

2. public class CustomerQueue – **Objects of this class will be used to store the line of customers in the Cashier and Kiosk classes. If you use a Java API class, you must use inheritance (extend a Java API class) to simplify the class definition. Consider implementing the Queue interface and accessing CustomerQueue objects through a Queue reference. The CustomerQueue class must provide the following methods:**

- public CustomerQueue() - constructor
- public void enqueue(Customer c) – enqueue a customer
- public Customer dequeue() – remove and return the customer at the head of the queue
- public Customer peek() – return the customer at the head of the queue without removing it
- public int size() – returns the number of customers in the queue
- public boolean isEmpty() – determines if the queue is empty

3. public class Cashier – **This class represents a Cashier. A Cashier must keep track of the customers**

lined up for it using a Queue. A Cashier must also keep track of the amount of time until it is done fixing a kiosk (which will be 0 if the cashier is not fixing a kiosk at the moment), as well as a reference to the kiosk that it is fixing (or null if it is not fixing any kiosk at the moment). Provide the following methods:

- public void assignToKiosk(Kiosk k, int seconds) – Assign this Cashier to fix the given Kiosk. Set the time remaining in the repair to the given number of seconds. Set this Kiosk's "broken" field to true. During this time, the cashier cannot serve customers (see act() method). Throw an exception if this cashier is already fixing a kiosk.

- public int act() – If this cashier is currently fixing a kiosk, decrement the time remaining on the repair. If after decrementing, the time remaining is 0, then set the "broken" field of the kiosk to false. If this cashier is not currently fixing a kiosk, decrement the time remaining to serve the customer at the head of the queue. If after decrementing, the time remaining is 0, dequeue the customer so that next time act() is called, the next customer will be served. If a customer completes an order, return the time he spent in line. If no customer completes an order, return -1.

- Getters and setters as appropriate

4. public class Kiosk – **This class represents a Kiosk. A Kiosk must keep track of the customers lined up for it using a Queue of ints, just as a Cashier. In addition, a Kiosk must keep track of whether it is broken, but NOT the amount of time remaining in the repair (that information is kept in the Cashier class). Provide the following methods:**

- public void break(), public void fix() – set the broken/fixed status of this Kiosk appropriately

- public int act() – if this kiosk is broken, do nothing. Otherwise, decrement the time remaining to serve the customer at the head of the queue. If after decrementing, the time remaining is 0, dequeue the customer so that next time act() is called, the next customer will be served. If a customer completes an order, return the time he spent in line. If no customer completes an order, return -1.
- Getters and setters as appropriate

5. public class Simulator – **Your simulator class will take the following parameters in its constructor and will not use specific hard-coded numbers, so simulations with different parameters can be run easily:**

- int num_cashiers – number of cashiers

- int num_kiosks – number of automated kiosks

- int min_cashier_time – the minimum amount of time a cashier must take to serve a customer

- int max_cashier_time – the maximum amount of time a cashier can take to serve a customer

- int min_kiosk_time – the minimum amount of time a customer must take to place an order at a kiosk

- int max_kiosk_time – the maximum amount of time a customer can take to place an order at a kiosk

- int min_repair_time – the minimum amount of time a repair must take

- int max_repair_time – the maximum amount of time a repair can take
- double arrival_prob – the probability of a customer arriving each second
- double kiosk_prob – the probability that a newly arrived customer will choose to place his/her order at a kiosk, rather than with a cashier
- double malfunction_prob – the probability that a kiosk will fail each second (this number should be small, like .01)

The following instance variables will be necessary:

- Cashier[] cashiers – an array of the cashiers in this restaurant. The length should be num_cashiers
- Kiosk[] kiosks – an array of the kiosks in this restaurant. The length should be num_kiosks
- int orders – the number of orders taken in the simulation so far
- int totalTimeWaited – the total time waited by all customers served in the simulation so far

The simulator class only requires the following instance methods (of course, additional methods are okay if they make the coding simpler).

- public int simulate(int duration) – runs a simulation with this Simulator's parameters on an initially empty restaurant (no customers) for the given duration of time, and returns the average time waited for customers who were served. You will need to do the following for each time unit:
    - Decide whether a customer will arrive in this time unit (using arrival_prob). If so, decide whether the customer will use a kiosk or cashier to place his/her order (using kiosk_prob), randomly choose one specific kiosk/cashier (with uniform probability), randomly decide how long the customer will take to serve (an int in the range [min_cashier_time, max_cashier_time] or [min_kiosk_time, max_kiosk_time] as appropriate) and insert the customer into that line.
    - Decide whether each kiosk will malfunction or not (using malfunction_prob for each kiosk). Note the malfunction probability applies to *each kiosk individually,* so multiple kiosks can malfunction in the same time unit. If a kiosk malfunctions, decide how long the repair will take (an int in the range [min_repair_time, max_repair_time]) and randomly assign it to a cashier (with uniform probability).
    - Allow each kiosk and cashier to act by calling its act() method. This takes care of either serving customers or fixing kiosks as appropriate, provided the act() methods are correctly written. Be sure to increment the number of orders taken and the total time waited if any act() method returns a time.
- Getters and setters as appropriate
- It is highly recommended to write methods for generating random integers within the ranges necessary,

as well as random booleans for arrival of customers and malfunction of kiosks. See the BooleanSource class in the lecture notes.

- Additionally, implement a main method that prompts the user for the simulation parameters and duration of simulation, instantiates a Simulator, runs the simulation, displays the results, then gives the user the option of running another simulation, or quitting.

Notes:

- We will only test with num_cashiers > num_kiosks, so you do not have to worry about what happens if a kiosks malfunctions and there are no cashiers available to fix it.
- We will not test with any minimum time = 0.
- You can ignore customers that remain in queues at the end of the simulation (for purposes of calculating average wait time)

SAMPLE INPUT/OUTPUT:

```
Output is black, user input is red, comments are green.

Enter the number of cashiers: 2
Enter the number of kiosks: 1
Enter the minimum cashier service time: 3
Enter the maximum cashier service time: 5
Enter the minimum kiosk service time: 1
Enter the maximum kiosk service time: 4
Enter the minimum repair time: 4
Enter the maximum repair time: 8
Enter the arrival probability: .3
Enter the probability for customers to use a kiosk: .6
Enter the probability of kiosk malfunction: .1
Enter the duration of the simulation: 20 //short simulation for sample IO

// Queues are printed from back to front, so the first number is last in line.
// Customers will be printed as (time_entered, time_to_serve)

Time = 1
No customer arrives
Cashier 1: []
Cashier 2: []
Kiosk 1: []
Customers served: 0
Total time waited: 0

Time = 2
A customer arrives and chooses kiosk 1. Time to serve = 3.
Cashier 1: []
Cashier 2: []
Kiosk 1: [(2, 3)]
Customers served: 0
```

```
Total time waited: 0

Time = 3
A customer arrives and chooses kiosk 1. Time to serve = 4
Cashier 1: []
Cashier 2: []
Kiosk 1: [(3, 4), (2, 2)]
Customers served: 0
Total time waited: 0

Time = 4
No customers arrive.
Cashier 1: []
Cashier 2: []
Kiosk 1: [(3, 4), (2, 1)]
Customers served: 0
Total time waited: 0

Time = 5
No customers arrive.
Cashier 1: []
Cashier 2: []
Kiosk 1: [(3, 4)] One customer served this time unit
Customers served: 1
Total time waited: 3 //The current time is t = 5. The customer arrived at t = 2. 5 -
2 = 3

Time = 6
A customer arrives and chooses Cashier 1. Time to serve = 4
Cashier 1: [(6, 4)]
Cashier 2: []
Kiosk 1: [(3, 3)]
Customers served: 1
Total time waited: 3

Time = 7
No customers arrive.
Kiosk 1 malfunctions. Cashier 1 is selected to fix it. Time to fix = 6
Cashier 1: [(6, 4)] Busy for 6 more seconds.
Cashier 2: []
Kiosk 1: [(3, 3)] Broken
Customers served = 1
Total time waited: 3

Time = 8
No customers arrive.
Cashier 1: [(6, 4)] Busy for 5 more seconds.
Cashier 2: []
Kiosk 1: [(3, 3)] Broken
Customers served = 1
Total time waited: 3

Time = 9
No customers arrive.
Cashier 1: [(6, 4)] Busy for 4 more seconds.
Cashier 2: []
Kiosk 1: [(3, 3)] Broken
```

```
Customers served = 1
Total time waited: 3

Time = 10
No customers arrive.
Cashier 1: [(6, 4)] Busy for 3 more seconds.
Cashier 2: []
Kiosk 1: [(3, 3)] Broken
Customers served = 1
Total time waited: 3

Time = 11
A customer arrives and chooses Cashier 2. Time to serve = 5
Cashier 1: [(6, 4)] Busy for 2 more seconds.
Cashier 2: [(11, 5)]
Kiosk 1: [(3, 3)] Broken
Customers served = 1
Total time waited: 3

Time = 12
No customer arrives.
Cashier 1: [(6, 4)] Busy for 2 more seconds.
Cashier 2: [(11, 4)]
Kiosk 1: [(3, 3)] Broken
Customers served = 1
Total time waited: 3

Time = 13
No customer arrives.
Cashier 1: [(6, 4)] Busy for 1 more seconds.
Cashier 2: [(11, 3)]
Kiosk 1: [(3, 3)] Broken
Customers served = 1
Total time waited: 3

Time = 14
No customer arrives.
Cashier 1: [(6, 4)] Done fixing
Cashier 2: [(11, 2)]
Kiosk 1: [(3, 3)] Now fixed
Customers served = 1
Total time waited: 3

Time = 15
A customer arrives and chooses Kiosk 1. Time to serve = 4
Cashier 1: [(6, 3)]
Cashier 2: [(11, 1)]
Kiosk 1: [(3, 2)]
Customers served = 1
Total time waited: 3

Time = 16
No customer arrives.
Cashier 1: [(6, 2)]
Cashier 2: [] One customer served this time unit
Kiosk 1: [(3, 1)]
Customers served = 2
```

```
Total time waited: 8 //5 was added to total time waited (16 - 11 = 5)

Time = 17
A customer arrives and chooses Kiosk 1. Time to serve = 3
Cashier 1: [(6, 1)]
Cashier 2: []
Kiosk 1: [(17, 3)] One customer served this time unit
Customers served = 3
Total time waited: 22 //14 was added to the total time waited (17 - 3 = 14)

Time = 18
A customer arrives and chooses Kiosk 1. Time to serve = 1
Cashier 1: [] One customer served this time unit
Cashier 2: []
Kiosk 1: [(18, 1), (17, 2)]
Customers served = 4
Total time waited: 34 //12 was added to the total time waited (18 - 6 = 12)

Time = 19
A customer arrives and chooses Kiosk 1. Time to serve = 1
Cashier 1: []
Cashier 2: []
Kiosk 1: [(19, 1), (18, 1), (17, 1)]
Customers served = 4
Total time waited: 34

Time = 20
No customers arrive.
Cashier 1: []
Cashier 2: []
Kiosk 1: [(19, 1), (18, 1)] One customer served this time unit
Customers served = 5
Total time waited: 37 //2 was added to the total time waited (20 - 17 = 3)

5 customers served. Average waiting time was 7.4 seconds per customer.
```

Course Info | **Schedule** | **Sections** | **Announcements** | **Homework** | **Exams** | **Help/FAQ** | **Grades** | **HOME**