**CS 1181** **Programming Assignment 3**

**Store Simulation**

**Reminder: No STATIC variables. The ONLY STATIC method allowed is main.**

**Note: In order to make this project more doable as a 1 week effort, I am providing part of the solution. I am providing a complete Customer class as well as a complete OrderedCustomerList class. More details are provided below. See the Javadoc for these classes for the complete description of the two classes.**

One common use of computers is to run simulations. A simulation is a program designed to model a real system of interest. By setting parameters and selecting appropriate input values, results are produced that mimic the behavior of the real system. If the simulation is a valid representation of the real system, it can even predict the behavior of the real system.

In this project, you will construct a program to simulate a store. The goal is to use your simulation to analyze the customer’s shopping experience given a number of express (12 items or less) checkout lanes, and a number of checkout lanes serving any size order. The owner would like to determine if there are enough or too many express lanes, and if there are enough or too many lanes serving any size order.

In this simulation, customers are characterized by the following information (1) their time of arrival to the store, (2) their order size and (3) the average time it takes them to select an item. For example a customer might arrive at the store 2 minutes after it opens. The customer shops and selects 15 items and takes 0.5 minutes to find and select each item. We simulate this as follows:

Customer arrival at simulation time 2.0 minutes

Customer completes shopping at arrival time + order size \* item selection time

In this case, the customer spends 15 \* 0.5 = 7.5 minutes shopping which means the customer is done shopping at simulation time 9.5 minutes. Once a customer finishes shopping, the customer moves to the shortest checkout lane (the one with the fewest customers waiting). Since this customer has more than 12 items, the customer selects one of the regular (non-express) checkout lanes. The time a customer waits to check out is determined by how long it takes the other customers using the lane to checkout. The length of time for a customer to check out is determined by the number of items in the customer’s order and the speed of the cashier for the selected lane.

**What data is important to the simulation?**

When designing a simulation, the most important question is what data must be produced by the simulation? In this case, we want to capture the shopping experience of the customer and be able to analyze the performance of each checkout lane. From an object oriented perspective, there are three main classes described in this problem, the customer, the checkout lane, and the simulation.

**The Customer Class**

Each customer will have a specific *number of items* they intend to shop for, and a specific *number of minutes* it will take to select each item. This data must be provided for each customer that enters the store.

There are several points in time that are significant to capturing the customer’s experience. All times are relative to when the store opens which is considered time 0. The times that are significant to the customer experience include:

Arrival time When the customer first enters the store – must be provided.

Shopping done time When the customer has completed selecting their items.

It is also when the customer selects a checkout lane.

Checkout done time When the customer has finished checking out and leaves the store.

In addition to the above items, each customer keeps the following data:

* The checkout lane selected (identified by a number)
* The length of time needed for the selected lane to check out the order

The following statements help to understand when different customer data items are manipulated.

* The number of items, time to select an item, and arrival time are all known at the start of the simulation for every customer.
* The time when a customer is done shopping can be calculated from data available when the customer data is read into the simulation.
* The selection of which checkout lane to enter is determined at the simulation time at which a customer is done shopping. Selection of which checkout lane is determined by the shortest checkout lane of the correct type (regular or express).
* The time to process the customer’s order and the time that checkout will be done can be determined from the state of the selected checkout lane at the time the customer selects the lane (See checkout lane data below).

The simulation **events** in the lifetime of a customer in the store are when the customer arrives, when shopping is done, and when checkout is completed. The simulation will need to provide code to coordinate activities associated with each of these three events.

* Arrival time – handled when the customer data is loaded into the simulation
* Shopping done – coordinate activity between customer and checkout lane
* Checkout done – coordinate activity between customer and checkout lane

**Checkout Lane Information**

There are several data items needed to quantify the operation of a checkout lane. The following list describes those items:

* *Time to check out an item* – this item is cashier dependent and must be provided in the initial data.
* *Time to process a payment* – this item is also cashier dependent and must be provided in the initial data.
* *Customers serviced* – the total number of customers processed thru a checkout lane.
* *Customers waiting* – the number of customers currently waiting to be checked out. This is affected every time a customer selects a lane and when checkout is completed for a customer.
* *Maximum wait line* – the maximum value attained by the customers waiting item.

In addition to the above lane information, the simulation must determine the average number of customers waiting to be checked out, and the average length of time a customer is waiting to be checked out. The following identifies the checkout lane data needed to determine these values.

* To process the average number of customers waiting for a checkout lane, we need to know the number of customers serviced by a checkout lane (which we already have), and the *total number of customers waiting at the time each customer selects the lane*. The average customers waiting for a lane is the *total customers waiting / customers serviced*. Therefore, each checkout lane must keep a *running total of the number of customers waiting when each customer selects the lane.*
* The average customer wait to be checked out by a lane is perhaps the most important statistic. How much time is a customer waiting before it is his or her turn to be checked out? More than any other statistic, this will be a reflection of the customer’s shopping experience. In order to calculate this, a checkout lane must keep a *running total of the wait time for each customer* processed thru a checkout lane. The average customer wait will be the *total wait time / customers serviced*. The customer wait timeis calculated by the customer as *the time the customer completed checkout minus the time the customer completed shopping minus the time to process the customer’s order.*

One additional piece of checkout lane information is needed for running the simulation.

* The simulation time when the checkout lane will complete checking out all the customers currently waiting.

This is also *the time when the checkout lane could start checking out a customer* that just selected the lane. For example, if it is 11:00, and it will take 20 minutes to check out the waiting customers, then a new customer selecting the lane would begin checking out at 11:20.

**Simulation Structure**

A Simulation class should be written to orchestrate all the activities needed to simulate the operation of this store. The following describes the major parts of the Simulation class.

Recommendation: the constructor for this class takes 2 file name parameters, one for the customer data file, one for the lane data file.

**Step 1: Load the customer data.**

Recommendation: Call a method to do this from the constructor – pass the customer data file name into this method.

All the customer data exists in a file that must be read to initialize part of the simulation. Each line of the file contains the arrival time of a customer in minutes, the number of items the customer needs to shop for, and the average time the customer needs to select each item in minutes.

Sample CustomerArrivals.txt data:

3.5 18 1.0

6.1 76 0.5

25.4 6 0.75

etc.

The method that loads the customer data will create a Customer object for each line of the file. When the Customer object is created, the time the customer will complete shopping is determined. The customer object must be inserted into an **ordered list** based on this shopping done time. When all customer data has been read, the ordered list will contain all the customer objects organized in the order in which they will complete shopping.

**An OrderedCustomerList class is provided that knows how to insert Customer objects in order based on the customer’s next significant event time. See the Javadoc for the OrderedCustomerList class for details.**

**Step 2: Load the checkout lanes**

Recommendation: Call a method to do this from the constructor – pass the lane data file name into this method.

All the checkout lane data exists in a file that must be read to create the checkout lane objects. The format of the file is as follows:

number of regular checkout lines (> 0)

for each regular checkout line you will have the following:

checkout time per item (in minutes), time to process a customer payment (in minutes)

number of express checkout lines (> 0)

for each express checkout line you will have the following:

checkout time per item (in minutes), time to process a customer payment (in minutes)

Sample data:

3

0.05 2.5

0.10 3.0

0.08 2.2

1

0.20 1.0

Since there are two different types of checkout lanes, both of which do exactly the same thing, the simplest way to set this up is by having *two different arrays of checkout objects*. One array is for regular checkout lanes, and the other array is for express checkout lanes.

The method that initializes the checkout lane objects reads the number of checkout lanes from the file, and creates an array of checkout lane objects of that size. Then reads a line of data for each checkout lane, creates a checkout lane object, and adds it to the array. This will be done twice, once for regular checkout lanes, and once for express checkout lanes.

**Step 3: Verify that you have loaded everything correctly!**

After steps 1 and 2 are completed, the Simulation class object will contain an OrderedCustomerList object, and two arrays of checkout lane objects. To verify that customers and checkout lanes were initialized properly, a *dumpData* method must be added to the Simulation class to output the initialized customer data and checkout lane data. The OrderedListClass provides a method to output data from all Customer objects in the list using the toString method of the Customer class. The CheckOutLane class must provide a toString method to output the information about CheckOutLane objects. The dumpData method can loop through the regular and express checkout arrays, outputting lane data.

**Step 4: Run the simulation**

The Simulation class must have a method that actually runs the simulation loop. There are two events that the simulation will need to process:

* Customer is done shopping and selects a checkout lane
* Customer checkout is complete

The simulation loop runs until there are no customers left in the OrderedCustomerList. The first customer in the list represents the next customer event to be processed. When the simulation loop begins, all the customer objects in the list represent customers that have completed shopping and need to select a checkout lane. The first customer object in the list is the first customer to complete shopping.

To process the list, take the first customer off the list and determine which of the two customer events this customer object is experiencing. The OrderedCustomerList class provides a *getNext* method that takes the first customer off the list and returns that object.

If the customer is experiencing the shopping done event, then the customer’s checkout done time will still be 0. To handle this shopping done event, the simulation must select a checkout lane for this customer. The Customer class provides a *useExpressLane(max items allowed)* method that returns true if the customer can use an express lane. Based on which lane type the customer needs to use, search for the checkout lane with the smallest number of customers waiting. Store the index of the selected checkout lane in the customer object (setter provided). Pass the customer object to the selected checkout lane object’s method responsible for adding a customer to the checkout lane. *(See* ***Shopping Done Event*** *section for what the lane must do.)* Once the checkout lane has updated its internal state for the new customer, and updated the customer state, the customer object must be added back to the customer list using the insert method of the OrderedCustomerList class.

If the customer is experiencing the checkout done event, use the checkout lane number from the customer to index the appropriate checkout lane array to locate the checkout lane object that this customer is using. Pass the customer object to the checkout lane’s method for handling when a customer is done checking out.

The checkout lane’s method for handling checkout done must update the checkout lane’s customers waiting and customers serviced data. It must also update the running sum of customer wait times by adding the waiting time for this customer to the running sum as previously described in the Checkout Lane Information section.

After the checkout lane processes the customer done event, output the data from the customer object that is now leaving the store. This customer has completed the simulation and is not reinserted into the list.

**Shopping Done Event**

The method in the checkout lane to handle adding a customer to the lane must update the number of customers waiting in line and maximum line length appropriately. It must also add the current number of customers waiting to the running total of the customers waiting in the line when a customer enters the checkout lane (before incrementing the number of waiting customers). This method also calculates the length of time needed to check out this customer’s order and stores that information into the customer object.

The checkout lane must keep track of when it will be available to start checking out each arriving customer. Initially this time is 0 (when the store opens). If the number of waiting customers (after being incremented) is 1, then the *next available start time* for the next new customer is the current customer’s shopping done time plus the length of time needed to process the current customer’s order. If there is more than 1 customer waiting, then the *next available start time* becomes the time to check out the current customer added to the *next available start time.*

The checkout lane’s *next available start time* is also the current customer’s checkout complete time. The lane must update the current customer object so its checkout done time is set to the *next available start time.*

**Display Final Statistics**

At the end of the simulation, the lane statistics described earlier must be displayed. Below is an example of the output from the end of a simulation run on the provided data files. It includes output from the last few customers to complete checkout.

Customer{ oSize = 72, itemTime = 0.9, arrivedAt: 918.97, shoppingDone: 983.77, checkOutDone: 4277.42

Customer{ oSize = 46, itemTime = 1.4, arrivedAt: 921.23, shoppingDone: 985.63, checkOutDone: 4285.02

Customer{ oSize = 44, itemTime = 1.0, arrivedAt: 944.87, shoppingDone: 988.87, checkOutDone: 4292.42

Customer{ oSize = 71, itemTime = 1.1, arrivedAt: 912.55, shoppingDone: 990.65, checkOutDone: 4302.52

Customer{ oSize = 63, itemTime = 1.2, arrivedAt: 915.64, shoppingDone: 991.24, checkOutDone: 4311.82

Customer{ oSize = 58, itemTime = 1.1, arrivedAt: 930.38, shoppingDone: 994.18, checkOutDone: 4320.62

Customer{ oSize = 39, itemTime = 1.2, arrivedAt: 949.12, shoppingDone: 995.92, checkOutDone: 4327.52

Customer{ oSize = 55, itemTime = 1.5, arrivedAt: 917.20, shoppingDone: 999.70, checkOutDone: 4336.02

Customer{ oSize = 71, itemTime = 1.3, arrivedAt: 908.21, shoppingDone: 1000.51, checkOutDone: 4346.12

Customer{ oSize = 64, itemTime = 1.4, arrivedAt: 916.44, shoppingDone: 1006.04, checkOutDone: 4355.52

Customer{ oSize = 68, itemTime = 1.0, arrivedAt: 938.98, shoppingDone: 1006.98, checkOutDone: 4365.32

Customer{ oSize = 46, itemTime = 1.3, arrivedAt: 949.54, shoppingDone: 1009.34, checkOutDone: 4372.92

Customer{ oSize = 65, itemTime = 1.3, arrivedAt: 931.83, shoppingDone: 1016.33, checkOutDone: 4382.42

Customer{ oSize = 74, itemTime = 1.3, arrivedAt: 922.65, shoppingDone: 1018.85, checkOutDone: 4392.82

Customer{ oSize = 72, itemTime = 1.3, arrivedAt: 949.75, shoppingDone: 1043.35, checkOutDone: 4403.02

Customers Serviced Max Line Length Average Line Length Average Cust Wait

Reg: 647 446 219.144 1110.788

Reg: 697 446 220.595 917.015

Reg: 664 446 221.660 1028.681

Reg: 581 446 221.348 1655.926

Reg: 626 445 221.110 1263.014

Exp: 313 4 0.550 0.842

Exp: 192 3 0.380 0.730

Exp: 119 3 0.319 1.032

**What to submit:**

**Create one ZIP file which contains your entire Netbeans project. Copy and paste your CheckOutLane class, your SimulationManager class, and your test class into a single Notepad or Wordpad file. Submit your ONE ZIP file and your ONE text file to the dropbox.**

**Rubric (100 pts) – No points if it does not compile.**

* **Check Out Lane class (35)**
  + **Constructor (2)**
  + **Three getter methods (3)**
  + **Handles adding a customer to the lane correctly updating statistics (10)**
  + **Handles customer checkout completion correctly updating statistics (8)**
  + **Method to calculate average number of customers waiting to check out (4)**
  + **Method to calculate the average customer wait time (4)**
  + **toString method 4)**
* **Simulation Manager class (50)**
  + **Load customers into the ordered customer list (8)**
  + **Load checkout lanes into the ordered customer list (8)**
  + **Method to output initial customer list and checkout lane data (4)**
  + **Simulation loop (30)**
    - **Detect and handle customer shopping done event (12)**
    - **Detect and handle customer checkout done event (10)**
    - **Display customers after checkout done (2)**
    - **Display final statistics (6)**
* **Main test class (5)**
  + **Creates a Simulation Class object**
  + **Invokes methods on that object**
* **Proper commenting and code formatting standards followed (10)**
  + **All classes have class level Javadoc comment**
  + **All methods have Javadoc comments including parameters and return types**