Project 2 Report

Christian Hollar

11/10/20

Introduction

Lab 3 required building an event simulated program that represents a coffee shop. The input txt file included with the lab had arrival times of customers for a coffee shop. The goal of the lab was to determine whether the coffee shop could hold the customers, the amount of customers served, the average time it took for the customer to be served, the maximum wait time it took for the customer to be served, and the net profit of the coffee shop for the day. This was completed by utilizing an event-driven simulation. The data structures included are ArrayList, LInkedList(orArrayDeque), Queue, and PriorityQueue. ArrayList was primarily used to store all completed transactions. The Queue was initialized using a LinkedList or ArrayDeque. The Queue represented the wait line in the form of a FIFO queue. The PriorityQueue was used to hold each event for the event driven simulation.

Approach

The first portion of the approach includes two new objects. They are customer and cashier classes. The customer is generated by the parent class customer setting. Customer setting takes in the first 5 lines of the .txt file and generates the boundaries for time and profit. Each time a new Customer is created, the abstract class CustomerSettings will generate a unique value for the interaction time and the profit. The cashier object holds a customer and status. The customer refers to the current customer that the cashier is serving. If the cashier status is true it signals it should take the next available customer. False status simply means the cashier is busy.

The next portion of the approach was having an event object. The event objects would be stored in a priorityqueue. It would hold variables int type, customer c, date d, and int cashV. The type variable represented what the event was. The number 1 corresponds to a new

attempted customer. The number 2 represents a customer ending its interaction with the cashier. The date represents when the event takes place, and cashV represents which cashier the customer visited.

The concept of the program first begins with creating a list of events of type 1 based on the input of customers. This is sorted in a PriorityQueue based on the date of the event. A method will evaluate each event using poll(). First it will check if there is space available within the wait line. If so, the customer will be entered in the FIFO queue which represents the waistline. At the beginning of each while() cycle of the method, it will check if any cashiers are available. If they are, the event will be assigned a cashier number and the clock will begin. This clock will add the randomly generated time to the current event date. An event of type 2 will be created using this time signaling the parting time for the customer. Event 2s will be evaluated by adding that customer to the completed arraylist and then changing the corresponding cashier status back to true to indicate the cashier is ready to serve another customer using the cashV variable.

Methods

Beginning with the Customer class, the methods included are getProfit(), getXTime(), and exitTime(). getProfit() does not require a parameter, and returns the unique integer value corresponding to profit for each customer. getXTime() does not require a parameter, and returns the unique integer value corresponding to the amount of time the customer would spend with the cashier. exitTime() requires a date which represents the time at which the customer started the cashier interaction. It then returns the end time of the interaction by adding the time spent with the cashier to the input time.

Moving forward to the Event class, the methods included are compareTo and getDate(). compareTo() is required for the ordering of the elements within the PriorityQueue. The compareTo() method is used to order the events by their individual date variable. getDate() returns the date which the Event would take place.

The next class with methods is the Controller class. It includes loadStudent(), cashierList(), and runThrough. The loadStudent class is responsible for taking in the file using the scanner utility. An ArrayList of type String named asetter is utilized for holding each line of the file. The first 5 lines of the file are removed because they do not correspond to a new customer. Then for the remaining strings within the asetter ArrayList, they are converted into dates of the form (HH:mm:ss) and delivered into an ArrayList of type Date named dates. This is completed through the SimpleDateFormat text input and the parse method included with the Date utility. Both of these conversions require for loops and try catch statements. The for loops are mainly used to cycle through each line or element of the input text file or arraylist. The try catch blocks are required for Scanner methods and the imported parse method. The final step includes just a for loop. For each cycle of the for loop, a new customer is created using a date from the dates arraylist. Then each time a customer is created, a new event is created of type 1 with the same customer and his shop entrance time. These are all type 1 events because they are each new potential customers.

For the cashierList() method, this generates an ArrayList of type cashier. The number of cashiers included in the ArrayList is from the input value. Each of the cashiers are created with status true because the cashier is ready for a new customer.

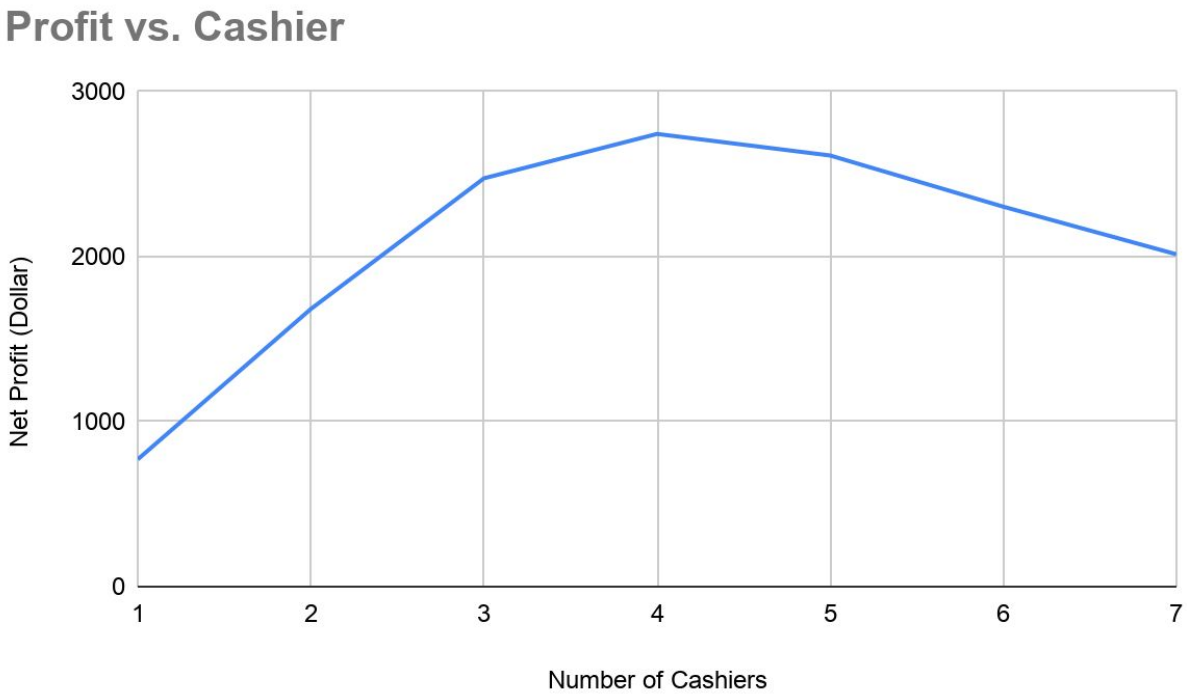
The runThrough() method does the main work. It evaluates the priorityQueue until it is empty using a while loop. First it checks to see if any of the cashiers contained in the cashierList() arraylist have a status of true. If there is, the first customer from the FIFO queue will be paired with the customer using poll(). This will remove the customer from the waistline. In addition, a type 2 event is generated indicating the time at which this customer will be leaving. This is done from taking the time generated from CustomerSettings for each customer and adding it to the current event date. If the top of the priorityQueue is an event of type one, it will check to see if the wait line can hold it. If it can't, overflow will increase by one and the PriorityQueue will move to the next event. If it can, the customer will be added to the wait line. If

the top of the priorityQueue is an event of type two, it will add it to the completed customer list and set the corresponding cashier back to a true status.

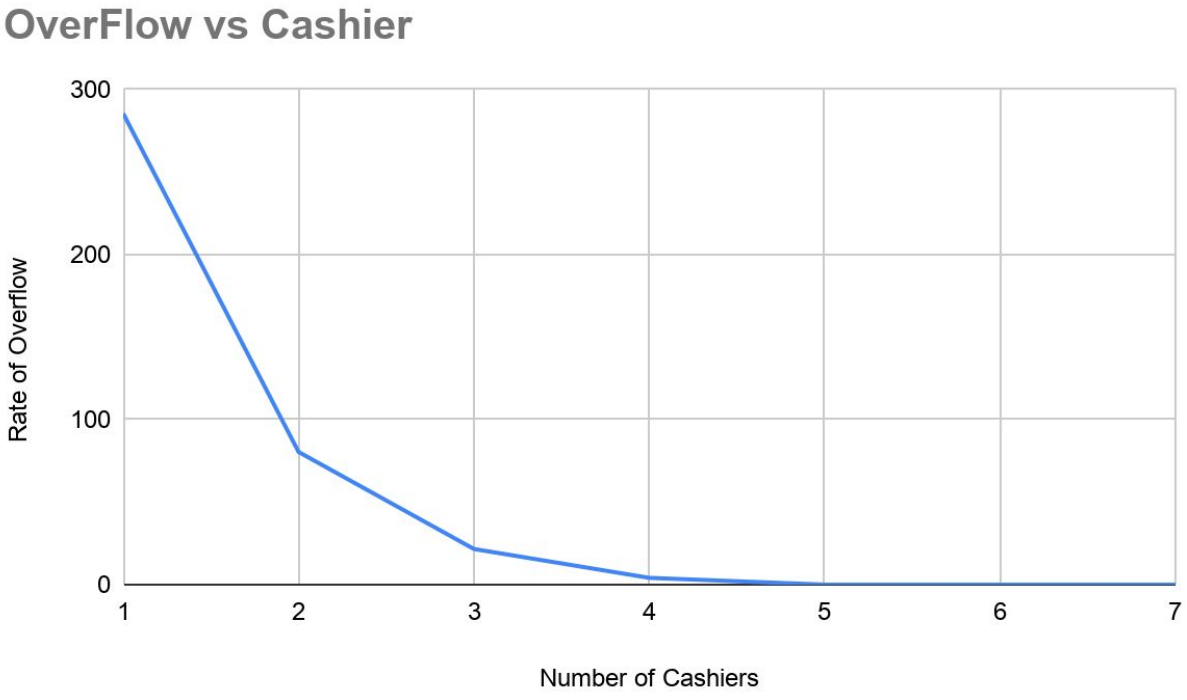
The final class corresponds to testing the earlier mentioned classes. This is the RunCheck() class. RunCheck holds methods regular(int n), overflowCheck(), allCheckLink(), allCheckDeque(), allCheck(), fourCheck(), and fiveCheck(). The regular method takes an integer that represents the number of cashiers and generates the full statistics. These statistics include the time it took, the amount of overflow, the amount of customers served, the number of attempted customers, the rate of overflow, the profit, the cashier cost, the number of cashiers, cost per cashier, the net profit, average wait time, and maximum wait time. The allCheckLink() and allCheckDeque return run time values for ten trials for one through seven cashiers. The allCheck() does this again using LinkedList but returns the profit values. fourCheck() and fiveCheck() returns results for 100 trials for 4 and 5 cashiers. This was necessary because 4 and 5 cashiers had the most efficient results of all numbers of cashiers.

Data and Analysis

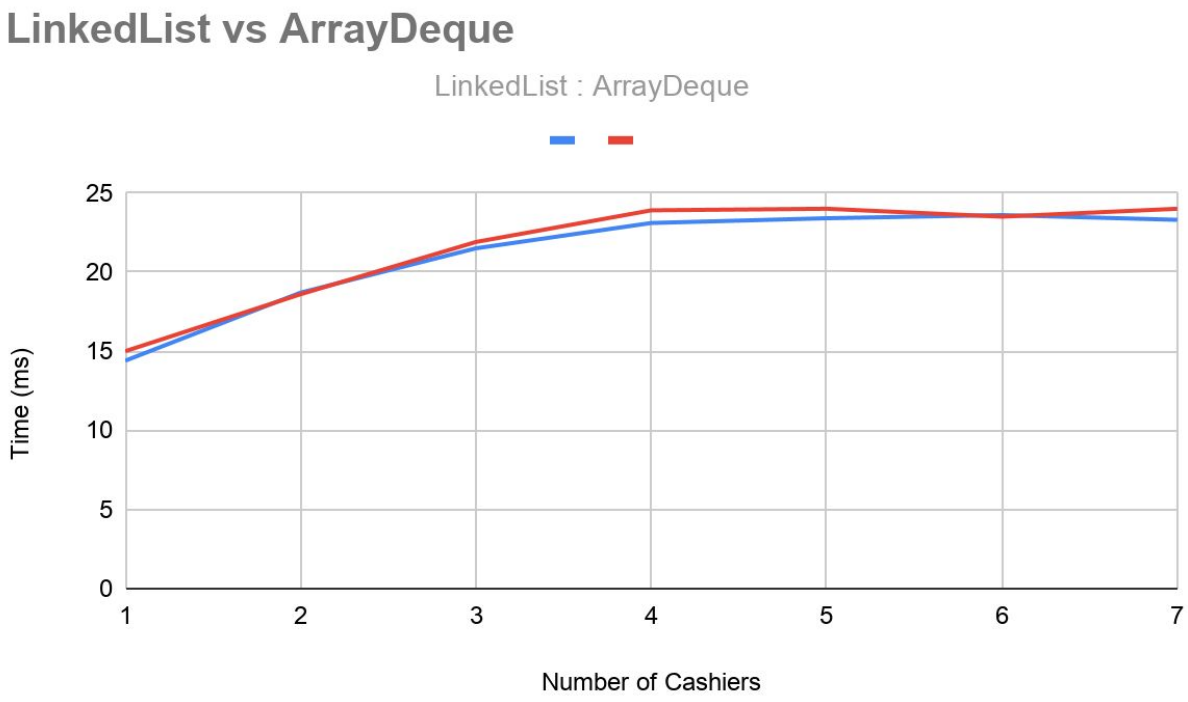
The line chart below represents the profit as the number of cashiers are increased. This was done by calculating the average profit over twenty trials for 1-7 cashiers. The line chart hits a maximum profit at 4 cashiers at around $2700.



The next line chart represents the rate of overflow as the number of cashiers is increased. At cashier 1, the rate of overflow is almost 300%. This rapidly decreases until it reaches 5 cashiers. If there are 5 or more cashiers, there will not be overflow.



The final line chart below represents the runtimes for LinkedList and ArrayDeque Queues. The runtime slowly increases from 1 to 4 cashiers for both. At 4 cashiers, the runtimes begin to plateau. LinkedList is slightly more efficient for this problem because it takes less or the same amount of time to run in comparison to the ArrayDeque.



Conclusion

In conclusion, 4 cashiers is the most efficient choice for the coffee shop to maximize profit. There was an average profit of $2743.35. At 3 and 4, the profit was $2474.35 and $2612.7. This was lower in both cases for twenty trial averages. The overflow and wait time decrease with an increase of the number of cashiers. LinkedList Queue runs slightly faster than ArrayDeque regardless of the amount of cashiers. The runtime for both increases for cashiers 1 through 4. The runtime plateaus at 4 cashiers.

References

Scanner(Java Platform SE 7). (2020, June 24). Retrieved November 10, 2020, from https://docs.oracle.com/javase/7/docs/api/java/io/Scanner.html

Oracle. (2020, June 24). Comparable. Retrieved November 10, 2020, from https://docs.oracle.com/javase/7/docs/api/java/lang/Comparable.html

Oracle. (n.d.). Generic Types. Retrieved November 10, 2020, from https://docs.oracle.com/javase/tutorial/java/generics/types.html

(2020, June 24). Retrieved November 10, 2020, from https://docs.oracle.com/javase/7/docs/api/java/util/PriorityQueue.html

Queue. (2020, June 24). Retrieved November 18, 2020, from https://docs.oracle.com/javase/7/docs/api/java/util/Queue.html

Date. (2020, July 09). Retrieved November 18, 2020, from https://docs.oracle.com/javase/8/docs/api/java/util/Date.html