# CS 150 Project 2 Report

Tafita Rakotozandry

November 8th, 2020

# 1 Introduction

When launching a business, getting the maximum profit at the lowest cost possible is the primary goal. Therefore, it is important to forecast what it would cost to create a new one. In this project, an entrepreneur is planning to create a new coffee shop and wants to know the amount of profit he makes in function of the clients and the number of cashiers.

The coffee shop is expected to open at 06:00 am and closes at 09:00 pm. Each customer arrives at a random time and their serving time is random as well. Depending on the number of cashiers available, the choice is then made to serve the customers upon arrival or have them wait in line until there is an available cashier. The length of the line cannot exceed eight times the number of cashiers hired by the store; if the former is true the customers arriving will be turned away. With these constraints in mind, the goal was to develop a simulation that would emulate the functioning of this coffee shop while keeping track of the profits as the number of cashiers that the store hires increases. Ultimately the number of cashiers that yields the highest net profit, would be the number of cashiers the coffee shop hires. One hypothesis that we can make is that as the number of cashiers is increased, the number of customers served will increase hence elevating profits of the Coffeeshop. However, that will no longer be the case when the number of cashiers exceed a certain number.

# 2 Approach

The project required the use of the following data structures: an Array List, LinkedList, Queue and Priority Queue. It also requires the use of event driven simulation programming. An event-driven simulation consists of processing events (Weiss, 1998). As a result, there were six different classes that were created in this project.Those classes are : Customer, Event, Calculation, Simulation, Main , TestSim and MainSim.

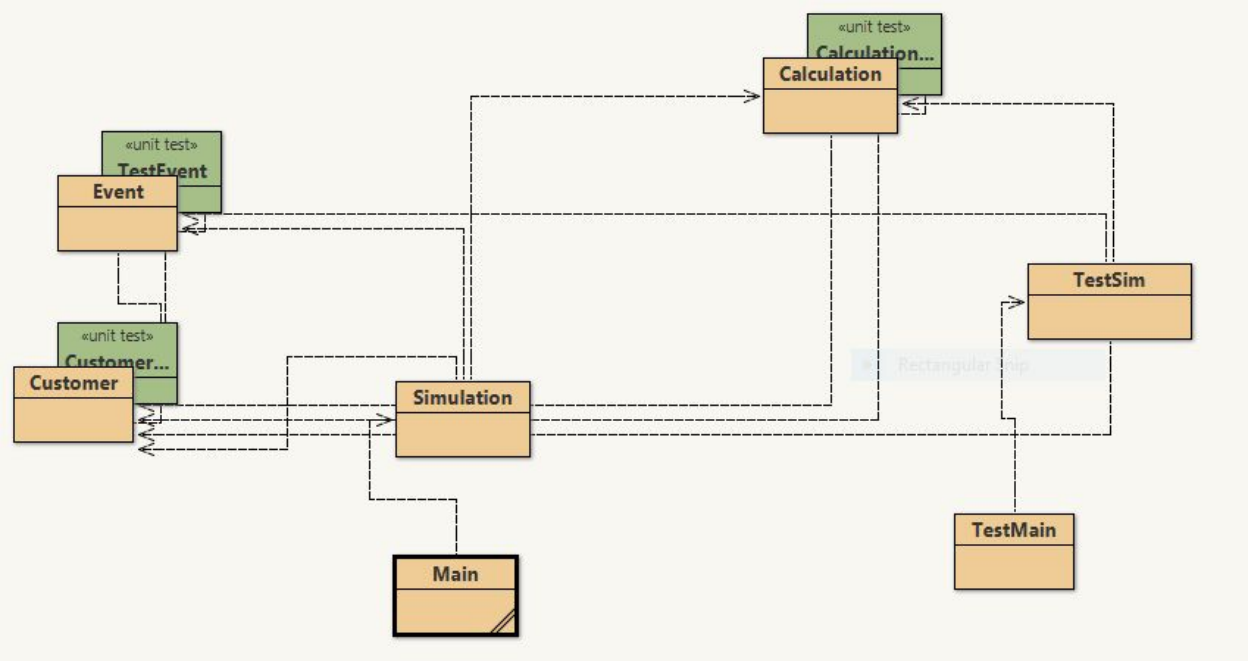


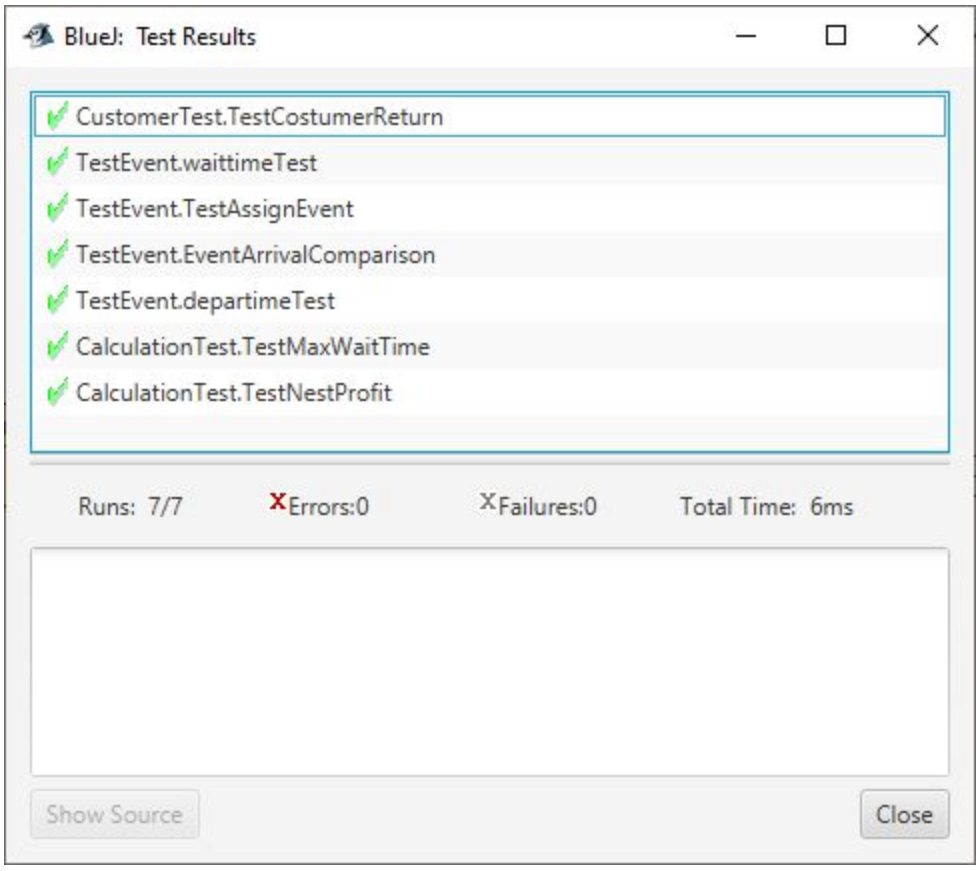

Figure 1: Classes Architecture

- **Customer** class holds the metrics of each customer that is served in the Coffee Shop. Each customer has an arrival time, a wait time, departure time, serving time, profit and name as defined as instance variables at the top of the class. This class includes getter and setter methods: setDepartureTime, getDepartureTime setWaitTime, getWaitTime.
- **Event** class is responsible for dealing with the metrics associated with each event. There are two types of events : **ARRIVAL** and a **DEPARTURE**. They are represented as int type variables ARRIVAL (which is set to 0) and DEPARTURE (set to 1) in the program. Every event has an arrival time, departure time, wait time, and name. These are taken care of by creating instance variables that will hold these values as their computed per event. Further, an instance of the customer is created that is passed the event arrival time and the name of the event. The idea was to have every instance of the customer associated with the same instance of the event, hence why the instance of the customer is passed into the constructor of the Event class, along with the arrival time, the type, and name of the event. Moreover, the Event class includes a compareTo method, getters and setters (setTime, getTime, getName, getType, setType), departTime and waitTime method.
- **Simulation** class is where the event driven simulation is implemented. There are two main methods in this class: fileReading() and run().

- fileReading is in charge of processing the text raw data. It includes parsing the input file and assigning the data extracted to the appropriate data.
- The method run() is in charge of implementing the event driven simulation step by step.

- **Calculation** class is a class that was designed to make a different operation with the result. It allows us to calculate the net profit, wait time and overflow percentage. These methods were set apart to maintain the clarity of the overall program.
- **SimTest** is a class that has the exact same implementation as the Simulation class. It was made for a testing purpose with a smaller data sample.

The Event , Customer and Calculation Classes were all unit tested. Simulation was not unit tested because it has testSim that contains the exact same implementation to test it with different data.



# 3 Method

## 3.1 Simulation

As mentioned previously, the simulation methods implement the event driven simulation. The program starts by reading the input file using the Scanner class. The profits lower and upper bound, the cost of hiring a cashier, the serving time upper and lower bound and the arrival time of the customer are extracted from the file and are assigned to different variables in the program . For each arrival time extracted is equivalent to a new customer object. An event type ARRIVAL will be created for each customer arrival. All of these events are appended to the waitingEvents priority queue. The goal is to empty the waitingEvents priority queue. In order to achieve that, there will be a while loop which includes different conditions.

At the beginning of the loop discussed earlier, the events that have the highest priority are removed from the waitingEvents queue. That removed event will be assigned to an event variable. That event will be verified if it belongs to the range of the opening of the coffeeshop. In case of yes, this event will be verified if its type is ARRIVAL or DEPARTURE.

- In case of an ARRIVAL type, another condition is verified if the number of cashiers available is more than zero.
    - If yes, the customer stored in this event will be served and the profit we get from that customer is added to the total profit . The departure time of this customer will also be set as well as the waiting time. The type of this event will then be changed into DEPARTURE type . At last, this event will be re-appended in the WaitingEvents priority queue.
    - If no, this event will just be added into a Queue called line. That event can only enter the queue if and only if the queue is not full which is determined by the number of cashiers multiplied by the 8.
- In case of an DEPARTURE type, the line is checked if there is an event stored in the line.
    - If yes, the customer saved in that event will be served and the profit will be added to the total profit . The departure time and waiting time will be set. The type of the Event will be set to DEPARTURE type. At last, this event is re appended to the waitingEventsPriority queue

- If the line is empty, it means there is no one to serve in the line and the available cashier is incremented.

When all the events in the WaitingEvents priority queue are all removed, the simulation stops and the result can be displayed using the different variables used.

The figure below shows how the code is executed when the overall program is launched:

```
Number of cashiers 4
Time : 21600 ==> Customer1 Arrives
Time: 21600 =>Customer: 1 is served Number Of Cashier Available: 3
Time : 21600 ==> Customer2 Arrives
Time: 21600 =>Customer: 2 is served Number Of Cashier Available: 2
Time : 21603 ==> Customer3 Arrives
Time: 21603 =>Customer: 3 is served Number Of Cashier Available: 1
Time : 21605 ==> Customer4 Arrives
Time: 21605 =>Customer: 4 is served Number Of Cashier Available: 0
Time : 21687 ==> Customer4 Leaves
New cashier available.Current number of Cashier Available 1
Time : 21723 ==> Customer1 Leaves
New cashier available.Current number of Cashier Available 2
Time : 21765 ==> Customer3 Leaves
New cashier available.Current number of Cashier Available 3
Time : 21766 ==> Customer2 Leaves
New cashier available.Current number of Cashier Available 4
Time : 25210 ==> Customer5 Arrives
Time: 25210 =>Customer: 5 is served Number Of Cashier Available: 3
Time : 25326 ==> Customer5 Leaves
New cashier available.Current number of Cashier Available 4
```

Figure 2: Serial Output of Running Program

# 4 Data and Analysis

The professor provided an input model to analyze in this project. Therefore, the result of this experiment is based on the professor's input. After running the series of test for the different number of cashiers, the following output graph is represented as a result of the experiment.

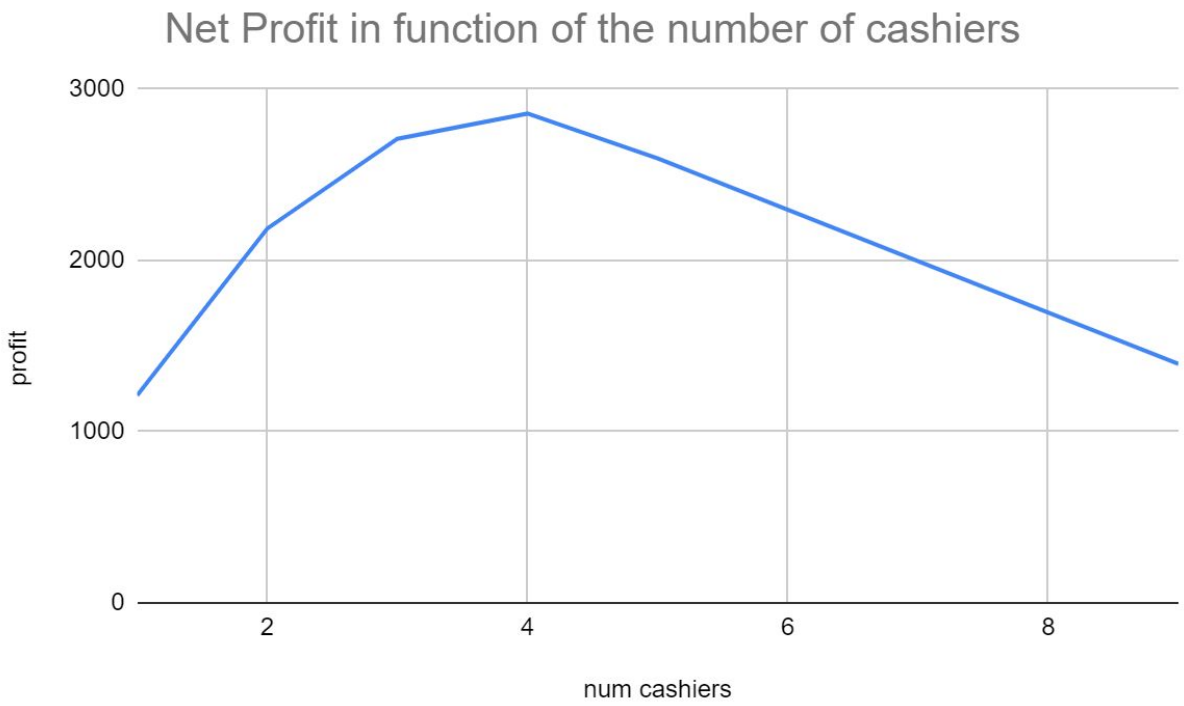



Figure 3:Net Profit vs number of cashiers

The profit vs cashiers graph is close to a bell shaped curve. It increases from 1300 profit for 1 cashier. It arrives in the maximum profit at 4 cashiers with a value of $2850. After that, it decreases slowly. It means that to get the most rentable profit, having 4 cashier is the most ideal.

An analysis of the overflow rate was also realized to determine how many clients the coffee shop had to reject because of the line. The following representation shows how the number of cashiers affects the result.

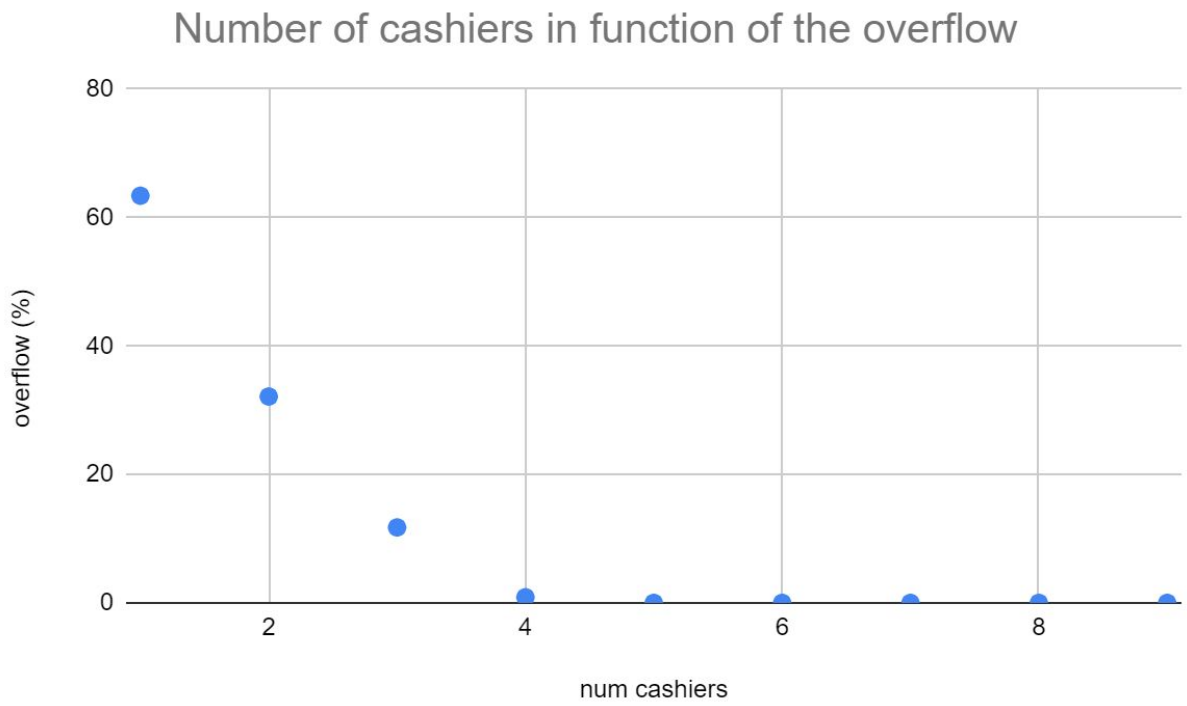



Figure4: Net profit vs number of cashiers

Initially, the overflow percentage with one cashier is 63 % . It decreases to 32% for the two cashiers. and 11% for 3 cashiers. The result starts to become promising using 4 cashiers because the overflow percentage is about 0.87%. It is totally 0% for more cashiers. We can then conclude that in order to obtain the least overflow, having 4 cashiers is the minimum requirement.

The waiting of the customers were also analyzed in order to verify the satisfaction of customer assuming that the longer they stay, the less satisfied they are.

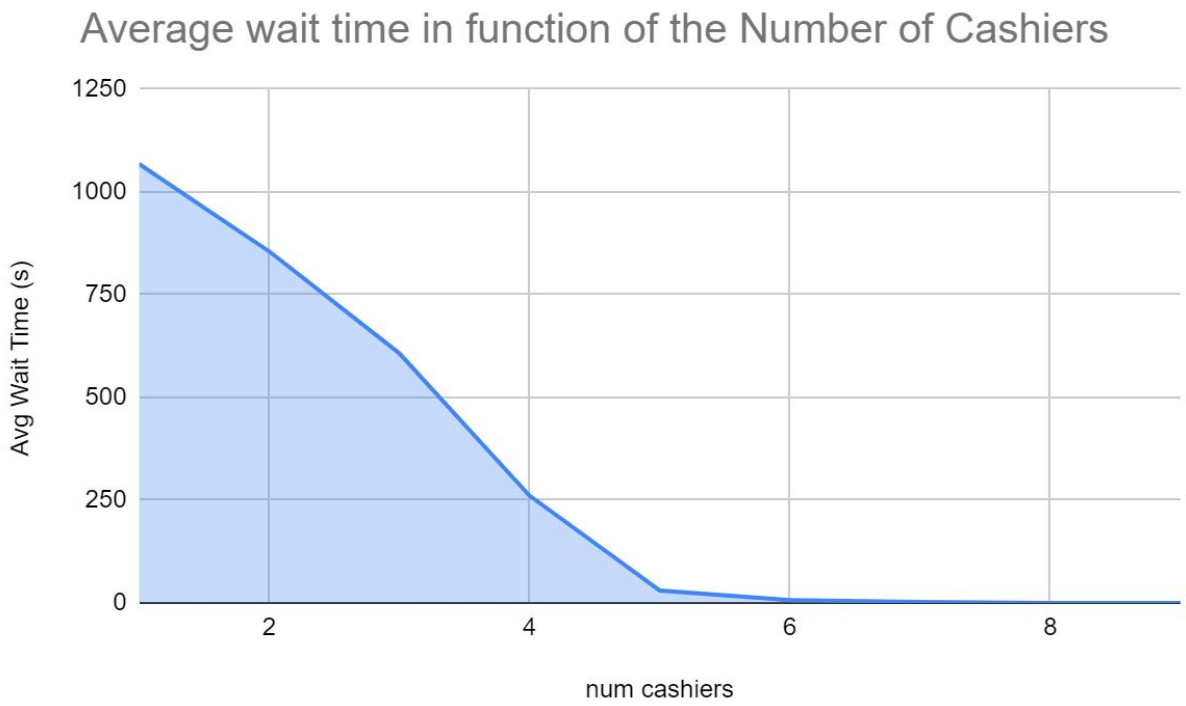



Figure 5: Average wait time vs number of cashiers

This result shows an average waiting time that decreases highly with the number of cashiers. The maximum average waiting time occurs with one cashier having 1050 seconds ( around 18 min). It is only with 6th cashiers that the average waiting time starts to be close to zero. Therefore, we can conclude that for more than 6 cashiers, the customer will be quite satisfied with the service time. It is important information to know because it may affect the chance to get that customer back to the coffee shop in the future. However , it is not a discussion not covered in this project

In order to understand more about the worst case scenario of waiting time , the maximum waiting time of the customer was collected.

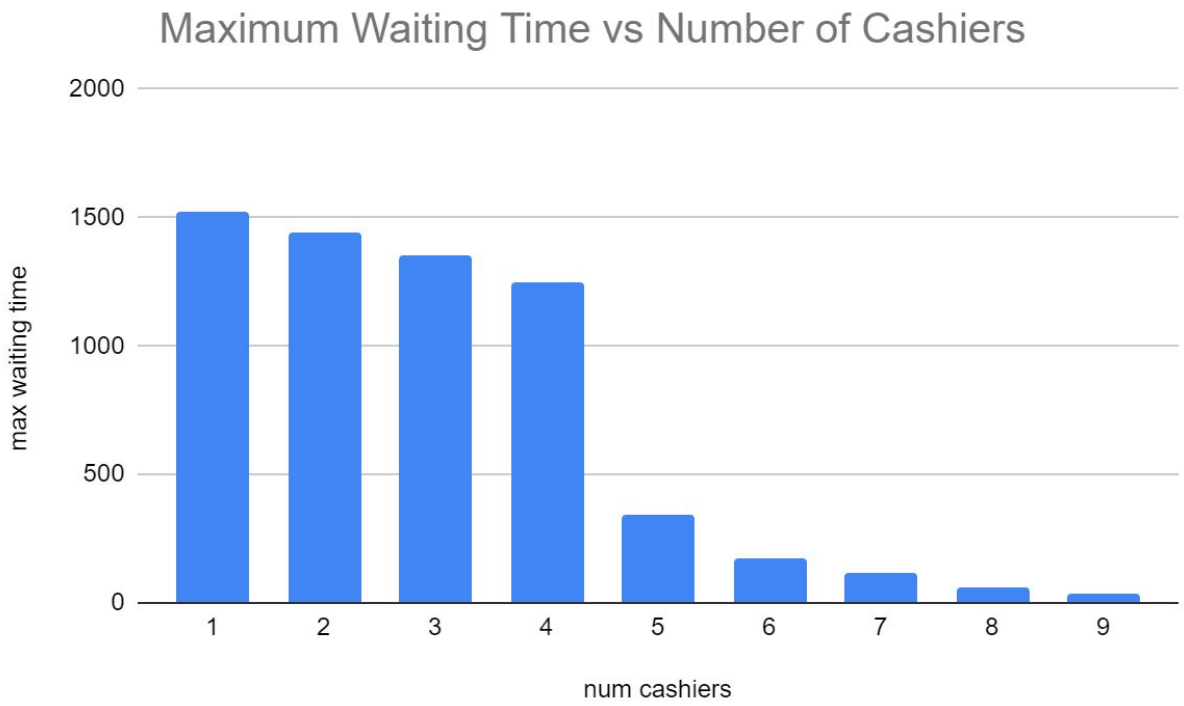



Figure 3: Maximum wait time vs number of cashiers

The maximum waiting that we can get is around 1520seconds (approximately 20 min). From this data, we also learned that between using 4 and 5 cashiers the maximum waiting time difference is quite big. It implies that having 5 cashiers or more would be ideal if we want to have a good customer experience.

Based on all these data results, the perfect number of cashiers that the entrepreneur should hire is 4. It is the most rentable and at the same guarantees that less customers will be turned away. The only problem is that in some cases, some clients may need to wait up to 20 min (1250sec) to get their coffee which is not really pleasant. However , in this specific project, we ignore the client's satisfaction because we only proceed with an analysis for one single day. But in practice, having 5 cashiers would be good for the customers.

# 5 Conclusion

This project allowed us to make use of the event driven simulation programming to simulate a coffee shop. Different data structures such as ArrayList, Queue, PriorityQueue and

LinkedList were used to achieve the implementation. The main goal of the simulation is to determine the number of cashiers required to make the maximum profit. From the experiment, we learned that having 4 cashiers is the most rentable choice for the coffee shop. It gives the maximum net profit. However, if the manager prioritizes the client's satisfaction over profit, having 5 cashiers is appropriate. Caring about the client's satisfaction is very essential because it increases the chance of the return of the customers. In the long term, that would guarantee a stable clients consumption.

The projects took me two weeks in total. Half of the time was spent on how to implement the simulation and the other half of the time was on the debug. Setting the unit test was helpful to keep track of the working codes.

# Reference


- Weiss, M. A. (1998). Data structures and problem solving using Java. ACM SIGACT News, 29(2), 42-49.

- Oracle(2019), Priority Queue: Java API
https://docs.oracle.com/javase/7/docs/api/java/util/PriorityQueue.html