Elene Karangozishvili– Oct 10 2018 – CS 150 Project 01

Lab Notes

Introduction

In this project we have to write a program that will calculate the net profit of the coffee shop. (Xia, 2018). As an input file we have the data of estimated profit of serving each customer, the cost of hiring a cashier per day, the time it takes to serve one customer and the times the customers arrive. The coffee shop opens at 6am and works until serving the customer who arrived at 10 pm or earlier. Based in this data we should write a program that will simulate one working day of the coffee shop for some numbers, the number of cashiers. We want to find the optimal number of cashiers, when the net profit is the largest.

Notice, that when the number of cashiers is 0, the net profit should be 0 since nobody got served. We expect that as the number of cashiers increase the net profit increases, until it reaches its maximum value. After this point, as the number of cashiers increase, the net profit should decrease. Therefore if we plot the graph of net profit as the function of number of cashiers, we expect to have rough shape of parabola with maximum point.

Approach

We decided to use event-driven simulation to address the problem (Weiss, 2010). Using this method our program will be more optimal by processing the simulation event by event instead of every second separately.

We implemented 4 separate classes. The first class is Event. Every instance of the class Event is defined by 3 things: type of the event: arrival or departure of the customer; customer number: the number that identifies the customer; and time: the time the event takes place. By the input file all the events with the type arrival get defined. Departure events depend on simulation, to which we will get later.

The second class that we implemented is Customer. Each instance of the class Customer is defined by 2 integers, the time the customer arrived and the time they got to the cashier.

The third class we implemented is CoffeeShop. Here we initiate priority queue (Oracle, 2017) of Java API of events. This priority queue defines the order of events.

The last class we implemented is Louncher. This class combines all the other classes and runs the simulation. Here we use several data structures. We have ArrayList (Oracle, 2017) of Java API of customers and the index of the customer is used as the defining number for each customer. We also have ArrayDeque (Oracle, 2017) of Java API, which we use for the actual queue in the coffee shop. All these data structures help simulation to run smoothly.

Method

In Event class we have several getters to help use this class and compareTo method. This method defines how we compare instances of Event. compareTo compares events based only on time, so that the events with smaller time happen earlier.

In Customer class we have setter method that sets departure according to the parameter it received, and more importantly waitingTime method, which calculates the amount of time the customer was waiting in the line.

In CoffeeShop class we have add, which adds event to the priority queue; poll, which retrieves the top element and removes it from the priority queue; isEmpty, that checks if the priority queue is empty.

In the Louncher class we have the main method that reads the input file, processes it, calls other functions for simulation and other calculations and prints the results. In the constructor of Louncher we convert String of the form HH:MM:SS AM or PM into amount of seconds from midnight. This helps us use time more easily. We also have a method called preliminaries. Here we create events based on when the customers arrive and add these events to already initiated csh variable of the class CoffeeShop. Next we have simulation method. Here we simulate one working day of the coffee shop. This method returns total profit calculated during the simulation. Since after simulation we have all the information, other methods just retrieve these information from different data structures. The method overflow calculates the rate of overflow based on number of customer who were not served, maxWaitTime calculates maximum waiting time

based on the waiting time of each customer, avarageWaitTime calculates the average waiting time of the customers who got served.

Unit tests

We conducted unit testing for each class separately. In Figure 1 you can see the results of unit testing Louncher. For this class we tested almost every method twice.

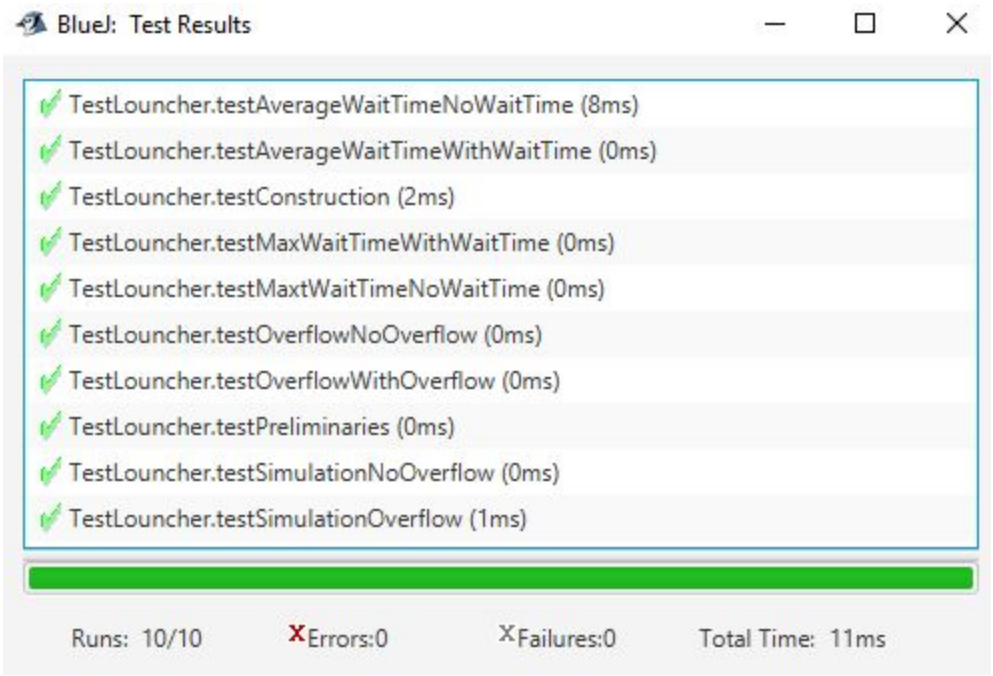

Figure 1
Result of unit testing Louncher

In Figures 2,3 and 4 you can see the results of unit testing CoffeeShop, Customer and Event classes accordingly.

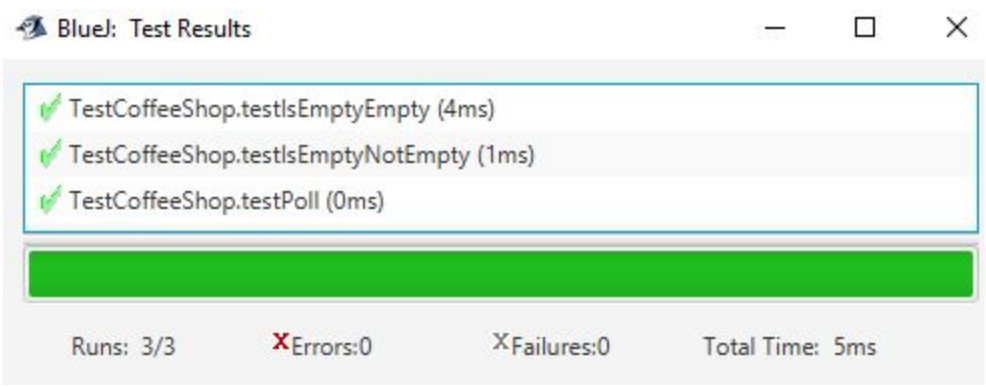

Figure 2
Results of unit testing CoffeeShop

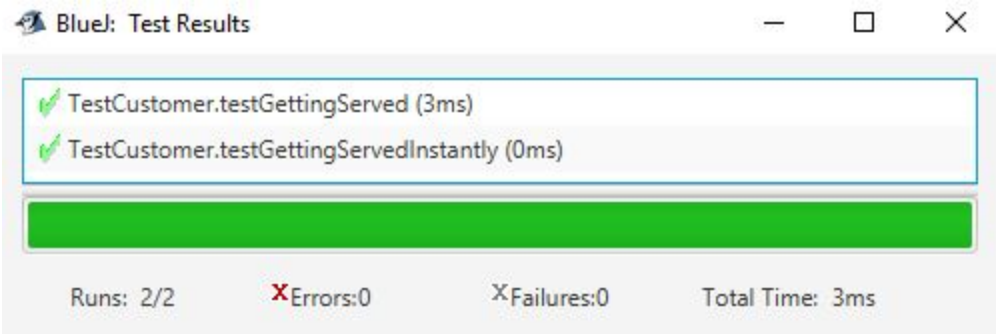

Figure 3
Results of unit testing Customer

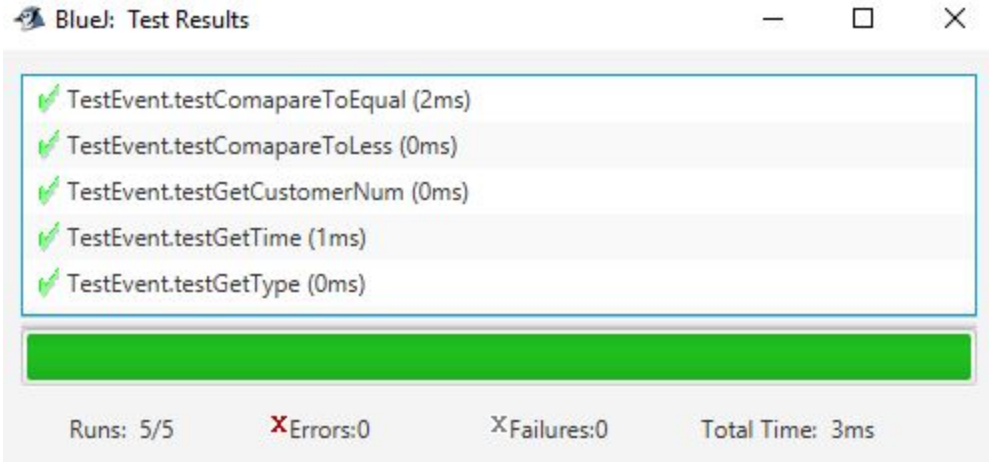

Figure 4
Results of unit testing Event

As you can see, all tests went through successfully.

Data and Analysis

We used the provided input file and varied the number of cashiers to build the graph of net profit as the function of number of cashiers. Than we changed the cost of each cashier and the average profit from each customer to have different graphs to compare to our hypothesis. For the standard input, with number of cashiers varying, we get the results on Figure 5 and Figure 6.

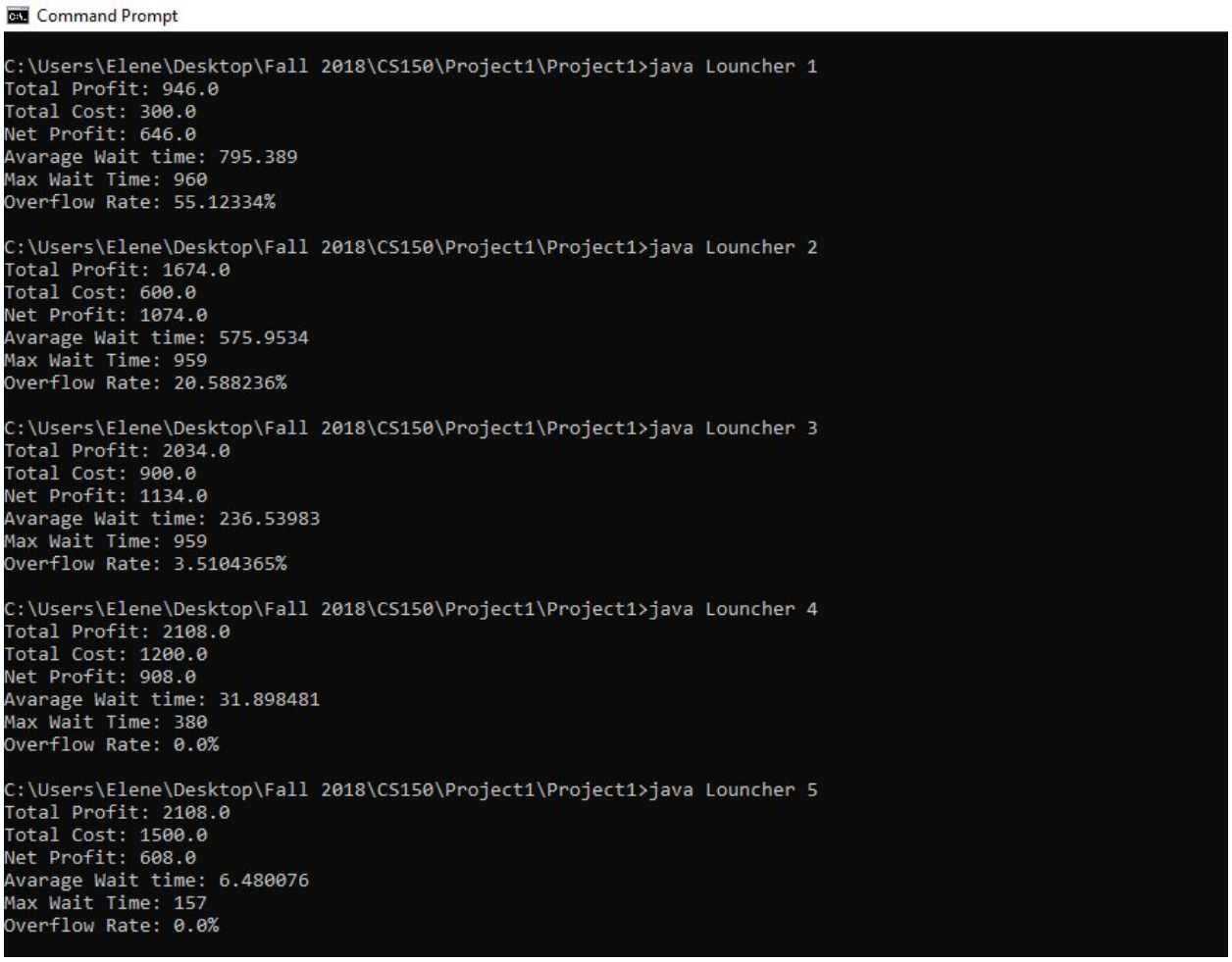

Figure 5
Launching the program from command line with standard input and varying number of cashiers

```
Command Prompt

C:\Users\Elene\Desktop\Fall 2018\CS150\Project1\Project1>java Louncher 6
Total Profit: 2108.0
Total Cost: 1800.0
Net Profit: 308.0
Avarage Wait time: 1.8671727
Max Wait Time: 77
Overflow Rate: 0.0%

C:\Users\Elene\Desktop\Fall 2018\CS150\Project1\Project1>java Louncher 7
Total Profit: 2108.0
Total Cost: 2100.0
Net Profit: 8.0
Avarage Wait time: 0.56641364
Max Wait Time: 65
Overflow Rate: 0.0%

C:\Users\Elene\Desktop\Fall 2018\CS150\Project1\Project1>java Louncher 8
Total Profit: 2108.0
Total Cost: 2400.0
Net Profit: -292.0
Avarage Wait time: 0.20018975
Max Wait Time: 56
Overflow Rate: 0.0%
```

Figure 6
Launching the program from command line with standard input and varying number of cashiers

From the outputs shown above we get the graph shown on Figure 7.

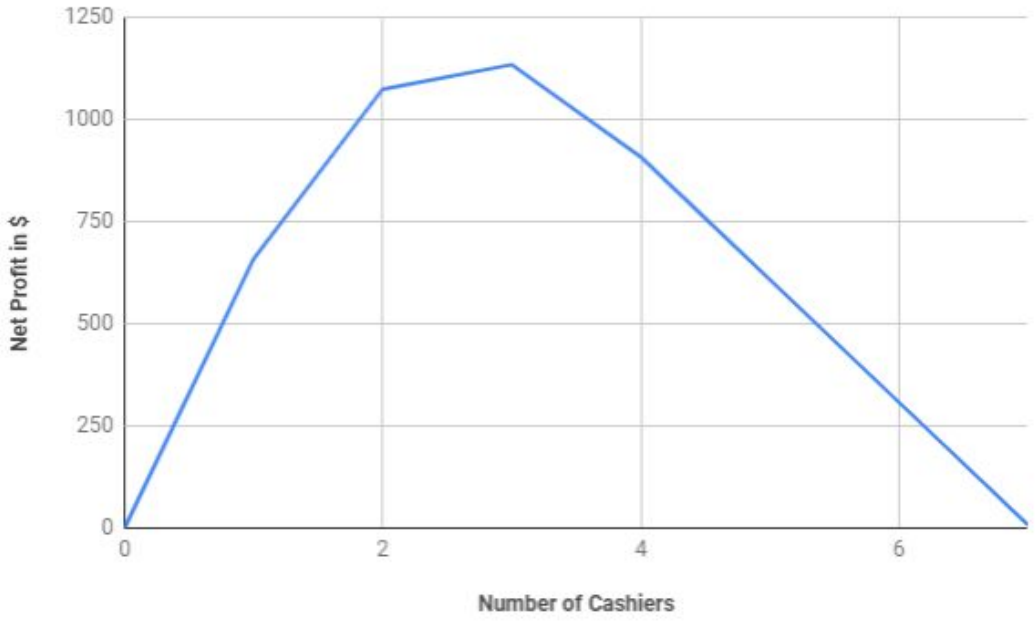

Figure 7
Graph of net profit as the function of number of cashiers using standard input

When we changed the cost of one cashier per day from 300 to 500 the resulting graph was Figure 8. As you can see, in this case optimal number of cashiers is 2.

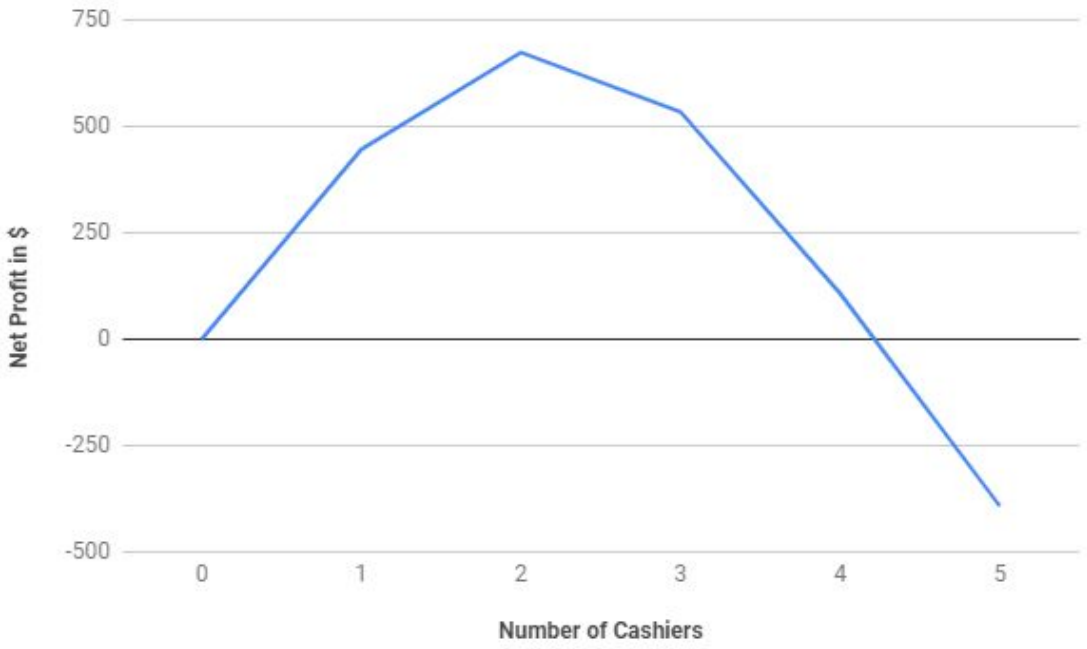

Figure 8
Graph of net profit as the function of number of cashiers using standard input with 500 as the cost of each cashier

When we changed the estimated profit from one customer from 2 to 10, the result was the graph in Figure 9. In this case the optimal number of cashiers is 4.

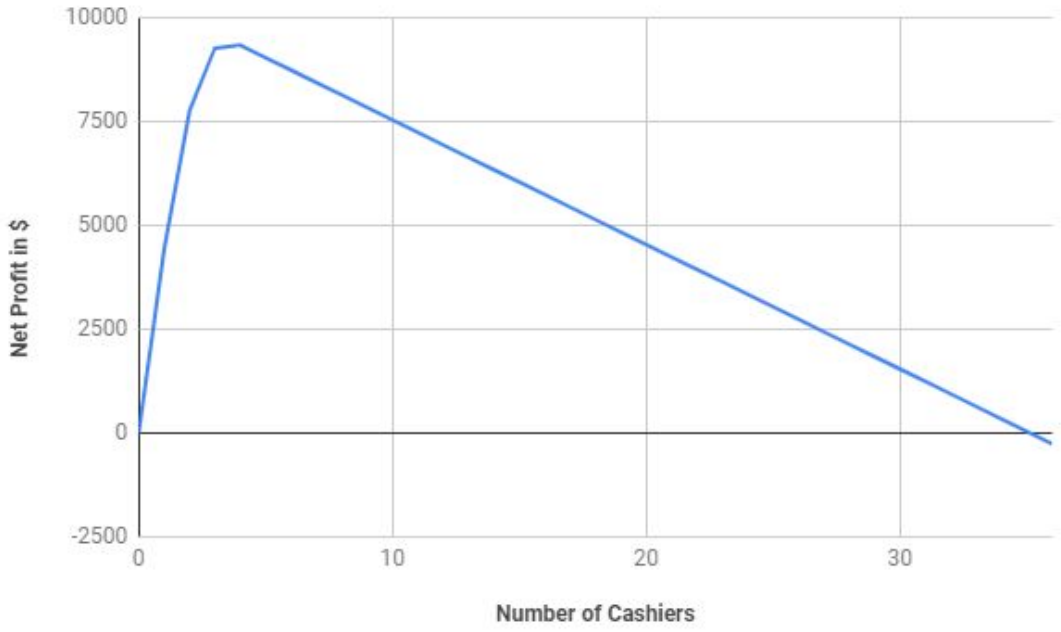

Figure 9
Graph of net profit as the function of number of cashiers using standard input with 10 as the estimated profit from each customer

Conclusion

As we mentioned in the introduction, we wrote a program using event-driven simulation, that simulates one working day of coffee shop. Based on the input file and the number of cashiers hired, we estimate total profit, total cost, net profit, average waiting time, maximum waiting time and overflow rate. Our assumption was that the graph of net profit as the function of number of cashiers should have a rough parabola shape with maximum. From the graphs provided in Results section, we can conclude that our assumptions were right. The net profit is increasing in the beginning as the number of cashiers is increasing and once it hits its maximum the net profit

is decreasing as the number of cashiers are increasing. This gives us clear result of how much cashiers should be hired for the optimal net profit.

References

Ge Xia . CS 150: Project I. (2018). Lafayette College.

Weiss, M. A. (2010). *Data structures & problem solving using Java*. Boston: Pearson Education.

Class ArrayList, Java™ Platform, Standard Edition 8 API Specification. (2017). Retrieved March 12, 2018, from https://docs.oracle.com/javase/7/docs/api/java/util/ArrayList.html Oracle.

Class PriorityQueue, Java™ Platform, Standard Edition 8 API Specification. (2017). Retrieved October 5, 2018, from https://docs.oracle.com/javase/7/docs/api/java/util/PriorityQueue.html Oracle.

Class ArrayDeque, Java™ Platform, Standard Edition 8 API Specification. (2017). Retrieved October 5, 2018, from https://docs.oracle.com/javase/8/docs/api/java/util/ArrayDeque.html Oracle.