DOCUMENTATION

ASSIGNMENT #*2*

QUEUES MANAGEMENT APPLICATION USING THREADS AND SYNCHRONIZATION MECHANISMS

STUDENT NAME: Iobaj Andrei Sebastian
GROUP: 30423

CONTENTS

1. Assignment Objective 3
2. Problem Analysis, Modeling, Scenarios, Use Cases 3
3. Design 5
4. Implementation 6
5. Results 9
6. Conclusions 9
7. Bibliography 10

1. Assignment Objective

The main objective of the assignment was to propose, design and implement a system that manages the use of threads to simulate a queue system, like those present at supermarkets. Some requirements for completing this task were to properly process the data from the provided fields in the UI, to synchronize all the threads and methods to work correctly in parallel and display the desired results.

2. Problem Analysis, Modeling, Scenarios, Use Cases

1) Problem Analysis

For this problem, I will compare our simulation to the real-life case of a supermarket with multiple cash registers and queues going to them and, of course, waiting clients in queue.

Firstly, let's look at how these queues work. As a client, we do our shopping in the store and we must check-out, meaning, we must get processed. This means, we have a certain "Arrival Time" at the start of the cash registers, and we must wait before we are served.

Based on the number of products we bought, we might have a larger or smaller "Service Time" than other people. This means that a cash register will take a certain amount of time to process us. As a result, queues might form when there are too many people and too few cash registers available. So, where do we sit?

Here, there are two strategies that people could take. The first one is to go to the cash register where there are the fewest people. This is a possible naive approach, because it is unknown how long it will take to get processed, but it is indeed a valid method of choosing a cash register. The second approach is to go to the cash register where the waiting time in queue is the lowest. This is computed by adding the service time of all the other people at the specific cash register. This is the fastest method, but more complicated to achieve than the first.

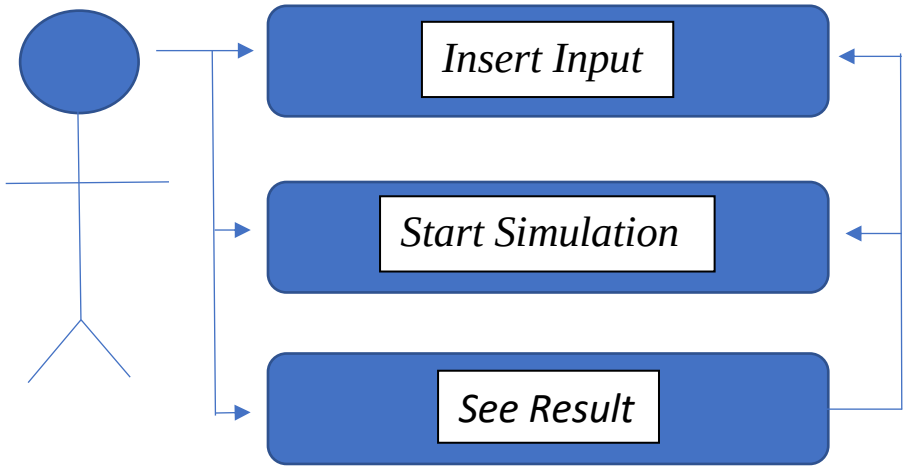
After choosing the cash register that fits our needs, it is time to wait in line before being served. The needed time to wait is, as stated above, the sum of all the service times of the people in front of us. After our turn has come, we shall get processed by the cash register and then we can depart the queue successfully.

2) Problem Modelling

Our program will have to be able to simulate such a queue system described above, with the help of threads. We shall implement an interface that will help us input the necessary information, such as Minimum and Maximum Arrival Time, Minimum and Maximum Service Time, Max Number of Clients, Max Number of Queues, Simulation Time. After that, with a press of a button, we shall begin the simulation that will generate the desired number of queues, clients, will generate randomly clients with the desired data that is in between the inserted intervals and will, obviously, monitor the status of the queues, will send clients to their appropriate queue and then, the queues will process their clients accordingly.

3) Scenarios and Use Cases

Use cases are used to predict all the possible ways a user can manipulate the interface/program in such a way that they get the result they desire. In our case here, the process is very simple. The user writes their input data in the Text Fields provided by the interface and presses the button that will then start the simulation. The process can be run multiple times without the need to exit the program. The output data will then be written in the corresponding fields in the UI and in a LOGS.txt file located in the project folder. This process is shown in the Use Case Diagram below:



a) Insert Input

The user shall write the input data in the marked fields. The fields must contain integers, otherwise the application will not work.

b) Select Operation

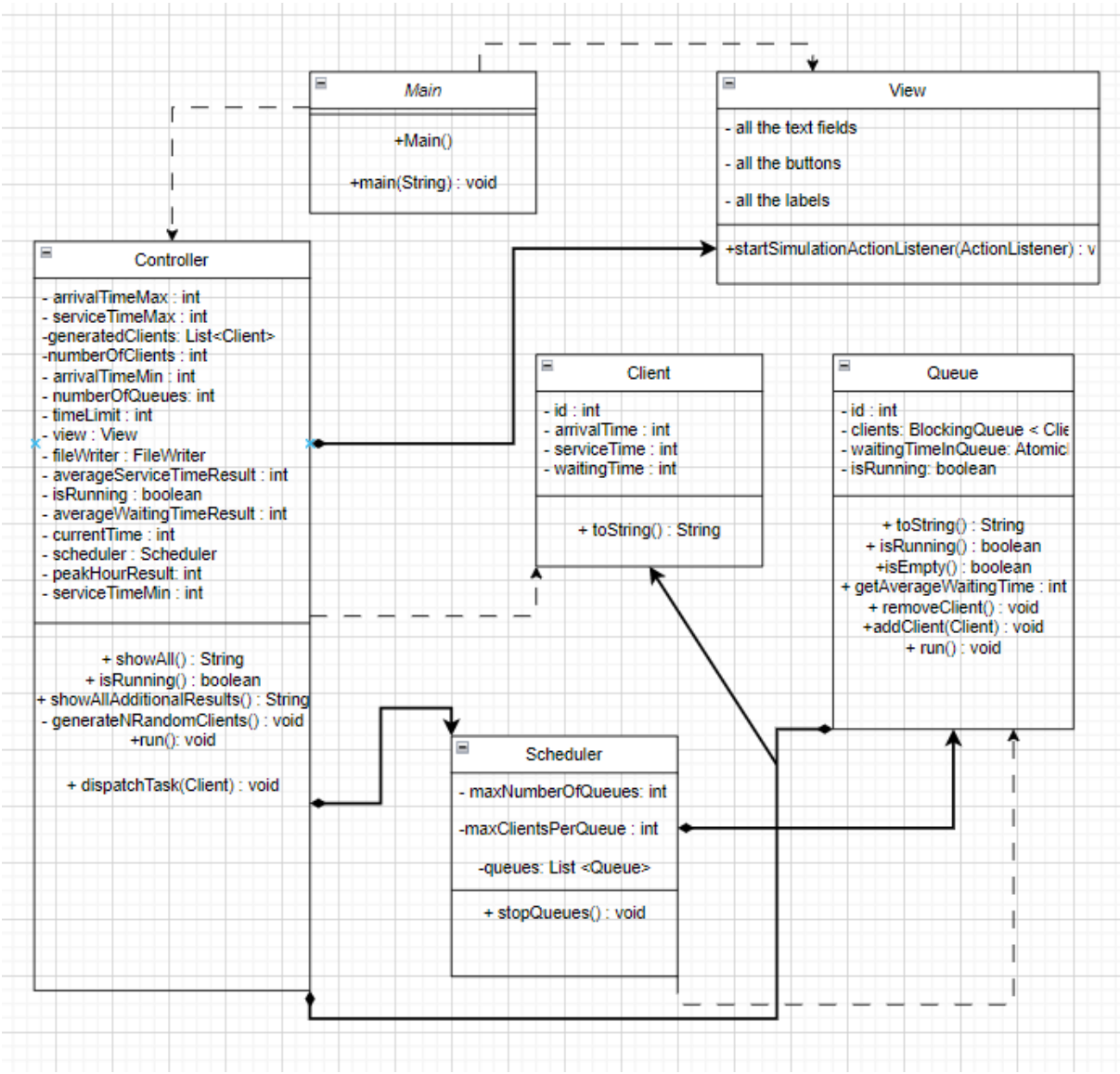
The user will press the button on the screen and then he will be able to track the real time progress of the simulation in the text field located in the UI.

c) See Result

The result of the simulation shall be displayed in the Text Field especially marked for the result. Also, in the Text Field for addition results, there will be some statistics that have been drawn from the previously run simulation. The process can then be repeated as already explained above.

3. Design

1) Class Diagram



2) Data Structures

In this project, I have decided after discussions during the lab and with fellow colleagues that my best approach was to use primitive types, such as integers and Strings. The data types that were not so commonly used by me before were the AtomicIntegers and BlockingQueue. They were used to properly synchronize the same variable among multiple threads. Also, FileWriter was used to write in files.

3) Packages

A well designed and structured project is organized in packages that help similar styles of data structures stay in one place.

The project is based on the MVC principle. That means that the program is structured in three big packages, Models (M), Views (V) and Controller (C).

- Models: It contains only the pure application data and interpretation of it, with no actual logic on how the data is shown to the user.
- Views: It represents multiple classes that form the user interface, that being the link between the user and the program (everything the user could see or interact with on the screen).
- Controller: It is the link between the Models and Views packages. It also contains the methods and logic behind all the elements of the view.

There is also an auto-generated package named "org.example", where the Main class was created to initialize the necessary elements to run the program.

4. Implementation

The project is organized in a multitude of classes briefly mentioned above. Let us take a closer look at what each class is designated to do and what the thinking behind implementing them was.

Note1: The presented classes are presented in a simplified manner. Methods such as Getters and Setters shall not be mentioned.

In models, we have 3 classes:

1) Client:

One of the most basic classes in the whole project is the Client class. This is the backbone of the whole project, storing the key information each Client must have. An ID, a service time and a waiting time. It will be used later in the project as Lists of clients, on more than one occasion.

2) Queue

The Queue class implements Runnable and as such will be treated as a thread. This is extremely important because as such, we would be able to achieve a parallel execution of the program, being able to run multiple queues in the same program to handle the clients. They will store them and have methods to handle the clients, such as adding or removing them and to monitor the time it takes to handle all the clients in the queue.

3) Scheduler

The Scheduler class is responsible for sending clients to the proper queue based on the information it collects from the queues. It will create the desired number of queues, based on the input and the method of Dispatching Clients that will check the waiting time in each of the queues and choose which queue has the lowest waiting time and will send a client to it.

Here, we could be implementing also the other strategy described above, the one with sending the client to the queue with the lowest number of clients.

```
public void dispatchTask(Client client){
    synchronized (queues) {
        Queue chosenQueue = queues.get(0);
        AtomicInteger minWaitingTime = chosenQueue.getWaitingTimeInQueue();
        AtomicInteger waitingTime;

        Iterator iterator = queues.iterator();

        while(iterator.hasNext()){
            Queue queue = (Queue) iterator.next();
            waitingTime = queue.getWaitingTimeInQueue();
            if (waitingTime.get() < minWaitingTime.get()) {
                chosenQueue = queue;
                //minWaitingTime.set(waitingTime.get());
                minWaitingTime = waitingTime;
            }
        }
        chosenQueue.addClient(client);
    }
}
```

Normally, no code should be inserted in the documentation, however the strategy of dispatching clients is the core of the program, so I will insert it.

A quick look over the code, we can observe that the described data structures from above are present here. Here we could observe how the implementation of the iteration over a list of clients in a queue is done. Some of these techniques were new to me, although the principle was the same.

We used an Iterator that takes the .next element in he queues list and checks the waiting time. If it is lower than that the minimum time identified until that specific moment, it is chose as the queue to send the client to .

Also we can notice that, because of the needed thread implementation, we had to make this part of the code synchronized, to be able to be run by all the queues that are in reality threads.

In the second package, views, we only have one class, because there is only need for one simple display window with a multitude of fields and a button, each serving a purpose.

1) View

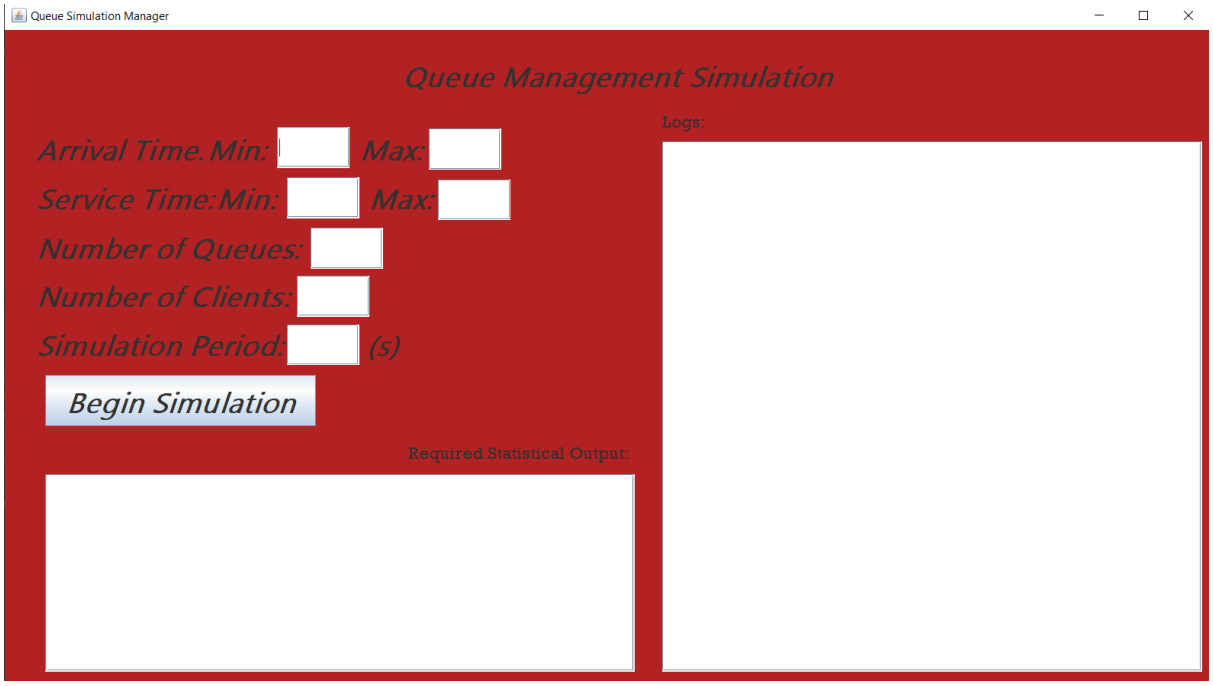
Our view extends the JFrame class that offers us the perfect tools to properly lay out an interface that best suits our needs. We first started with declaration of buttons and text fields, with the names reflecting what we want to implement. The constructor does not require any parameter.

In the constructor, with the help of Eclipse, I was able to properly construct the interface and customize it to provide a nice user experience.

Something to note is that for the button, a method was created that takes an ActionListener linked to that button which shall take its functionality from the controller (more on that below)

```
public void startSimulationCreateActionListener(ActionListener actionListener){
    btnNewButton.addActionListener(actionListener);
}
```

This is the one and only example of such a method that takes an ActionListener and links it with the button in the current project.



Above, I have inserted a screen capture of the Interface. As can be clearly seen, the GUI is user friendly, clearly stating all the instructions, either on the button or near the text fields that need to be filled for simulation to be able to be performed correctly.

The package Controller holds two classes, one that implements the controller described above and one where the operations were implemented.

1) Controller

The class Controller takes the View as an argument and links the ActionListeners created previously to methods created in this class. With this, we can control (as the class name suggests) the functionality of the elements present in the GUI described above.

The problems I encountered during implementation were to properly synchronize the interface with the controller methods. It had to run as a thread when the button was pressed and all the preparation work had to be done beforehand, such as generation of the clients with the random times of service or arrival.

As such, a few moments of clever thinking were necessary to properly analyze the needed order of doing things, such as assigning the input data to the variables, or where certain parts of the code had to be placed for it to work with the UI. Everything that has been done is explained in the code.

And the last class, in a package generated by the Maven Architecture when the project was first created is the Main class.

1) Main

In the case at hand, Main has one simple and extremely important purpose, to initialize the controller. In this implementation, the controller does all the work for us. It has linked the GUI with the logic of the classes in the models package, doing all the work behind the scenes, but nothing initialized the controller. That is solved by the Main class, having a view and the controller taking that view as a parameter declared. As such, the GUI is initialized and displayed to the user, making it possible to properly use the program as desired.

5. Results

In order to test the program, some practical tests were provided in the assignment PDF document to run. There were run on the application, with the results being more than satisfactory. Being asked to write the logs of the execution in a .txt file, to properly see the practical results of such tasks, the logs will pe uploaded to the Git Repository, as asked. For the purpose of this documentation, however, do know that testing has been done for this project.

6. Conclusions

From a personal point of view, I found this assignment challenging but fun. Some concepts were pretty much foreign to me, such as thread utilization. Learning these new concepts proved difficult at first, especially, as stated above, during the final stages of linkage between the controller and the interface.

Besides the programming challenges, the best lesson learnt is time management. Working with a deadline in front of you, having to write a documentation for a project you did and needed to upload the code to git is a nice experience that will be useful in the future.

7. Bibliography

1. OOP Lectures and Laboratory presentations – prof. M. Joldos
2. PT Lectures and Laboratory presentations – prof. C. Pop
3. https://www.tutorialspoint.com/
4. https://www.wikipedia.org/
5. https://stackoverflow.com/
6. https://app.diagrams.net/