**Technical University of Cluj-Napoca**

**Faculty of Automation and Computer Science**

**Computer Science Department**

**Programming Techniques**

**Third Homework: Queue Simulation**

**Mihoc Nicolae**

**Group: 30425**

Table of Contents

*1. Problem Specification . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3*

*1.1 Problem Analysis . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .4*

*2. Diagram Implementation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4*

*2.1 Use Case Diagram . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4*

*2.2 Class Diagram . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5*

*2.3 Sequence Diagram . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7*

*2.4 Activity Diagram . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7*

*3. Design . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8*

*3.1 Packages . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8*

*3.2 Classes . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8*

*4. Implementation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9*

*5. Improvements . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12*

*6. Conclusions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 13*

*7. Bibliography . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 13*

**1.** **Problem Specification**

**Objective**

Design and implement a simulation application aiming to analyze queuing based systems for determining and minimizing clients’ waiting time.

**Description**

Queues are commonly seen both in real world and in the models. The main objective of a queue is to provide a place for a "client" to wait before receiving a "service". The management of queue based systems is interested in minimizing the time amount its "clients" are waiting in queues. One way to minimize the waiting time is to add more servers, i.e. more queues in the system (each queue is considered as having an associated processor) but this approach increases the costs of the supplier. When a new server is added the waiting clients will be evenly distributed to all current available queues. The application should simulate a series of clients arriving for service, entering queues, waiting, being served and finally leaving the queue. It tracks the time the clients spend waiting in queues and outputs the average waiting time. To calculate waiting time we need to know the arrival time, finish time and service time. The arrival time and the service time depend on the individual clients – when they show up and how much service they need. The finish time depends on the number of queues, the number of other clients in the queue and their service needs.

**Input data:**

- Minimum and maximum interval of arriving time between clients;

- Minimum and maximum service time;

- Number of queues;

- Simulation interval;

- Other information you may consider necessary;

Minimal output:

- Average of waiting time, service time and empty queue time for 1, 2 and 3 queues for the simulation interval and for a specified interval;

- Log of events and main system data; - Queue evolution; - Peak hour for the simulation interval;

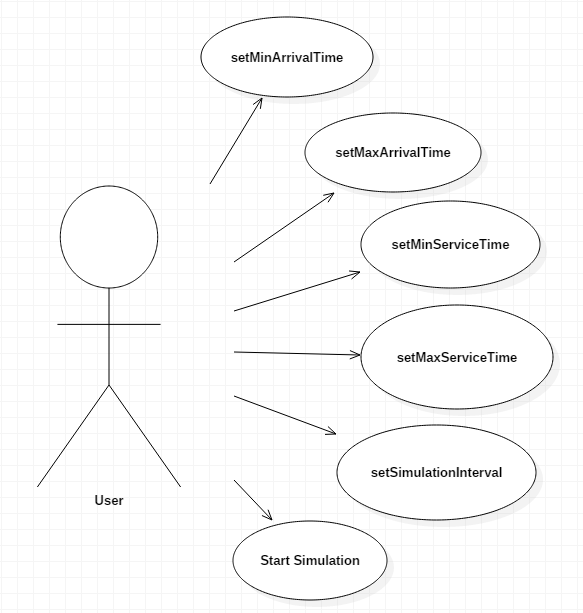
**1.1** **Problem Analysis**

This type of problem is a problem that once implemented “acts on its own” meaning that the user sets some input data and based on that input the program executes until reaches a finish condition which in our case is the end of the simulation interval. This can be done using Java Threads or Executors that based on some conditions like waiting time or service time can execute the same operation of adding and removing a customer from the queue until the specified condition is fulfilled. The queue is a FIFO type because the first customer arrived is the first one served. To resemble more the real world queues a random timer for arrival time and service time must be set.

**2. Diagram Implementation**

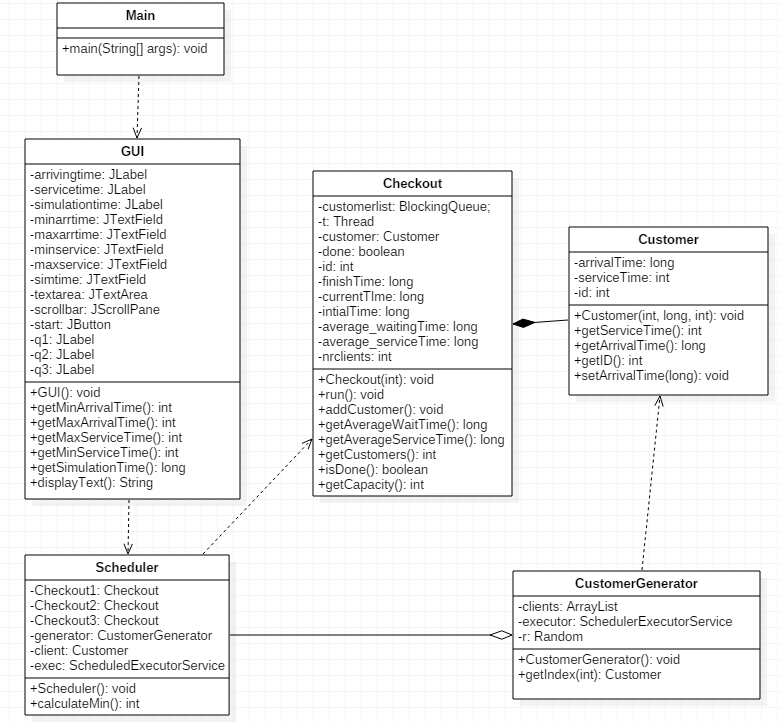
**2.1 Use Case Diagram**

The use case diagram as always shows us all available fields where the user can enter the simulation data and the options he has. In this case he can control every aspect of the queue starting from the arrival time and up to the simulation interval.



**2.2 Class Diagram**

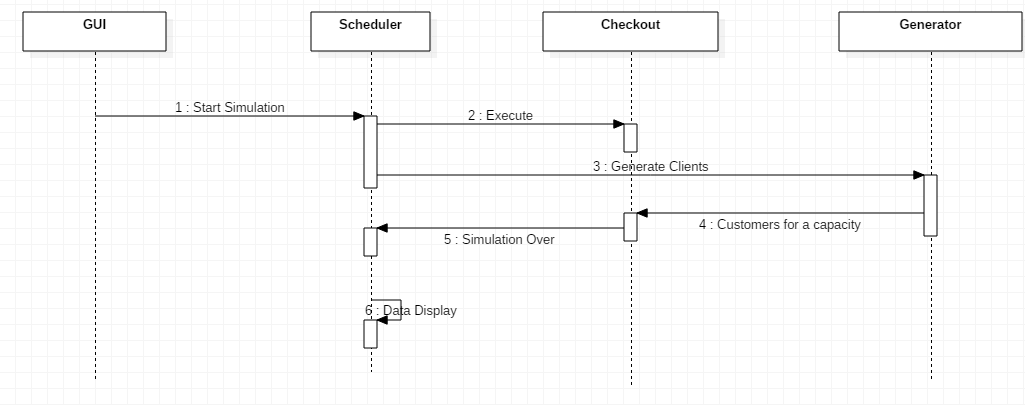
The class diagram is the best diagram because it shows in detail all the attributes and all the operations that each class performs and also the relations between the classes.



From this diagram it can be seen that only 6 classes are enough for the implementation of the problem. The relations between them are pretty self explanatory. In order to display something the Main class needs the GUI class. In the GUI class a new Scheduler class is called every time so there shows up a dependency relationship. After that the Scheduler class “prepares” the check-out class and clients are generated that enter the queue at one of the 3 available check-outs. The Customer and Checkout class have a composition between them because the customers are created specially for the check-outs and the aggregation relation between Scheduler class and CustomerGenerator class exists because there will be nothing to schedule without the generated customers. There are not many classes but each class is linked with another and are heavy dependent of each other.

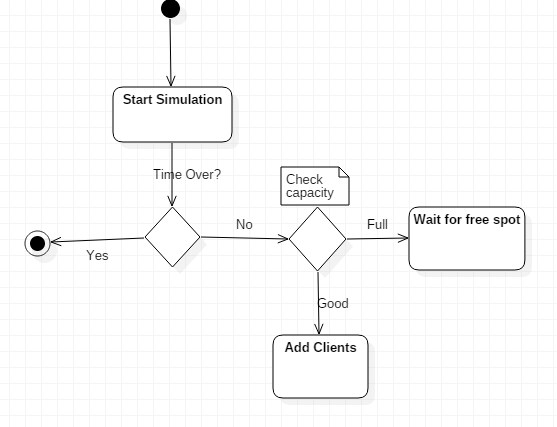
**2.3 Sequence Diagram**

The sequence diagram shows the flow of the application from the moment the start button is pressed and the simulation starts. It ends with the data display on the text area showing the average service time and waiting time for each checkout.



**2.4 Activity Diagram**

The activity diagram helps us visualize the decisions that are made in the execution of the application. There are 2 major decisions to be made in this process. The first one is checking if the queue is full and waiting for it to clear and the second one is checking the time of the simulation to not be over.



**3. Design**

**3.1 Packages**

In my opinion only 3 packages are enough for this kind of application. I chose to divide the classes in the following packages:

-a package named View which contains the Graphical User Interface

-a package named Controller which contains the class Main

-a package named Queues which contains the classes: Customer, CustomerGenerator, Scheduler, Checkout

**3.2 Classes**

I think this is the minimum number of classes needed to solve such a problem. The logic behind this decision is this: for visualizing the queue and for data introduction we need a graphical user interface so the a class GUI is absolutely necessary; for controlling this class we need a Main class to display it; for the logic of the problem we need 4 classes which I used to solve different parts of the problem.

In order for this application to work certain tasks must be completed. First things first, a Customer class which is needed to define an object with the properties of a real world customer. Second, a Checkout class to place the customers and where the queue is formed dealing with the time problems of each customer and the functioning time of each checkout. Third, a Scheduler class to manage the placement of the clients in the queue choosing each time the smallest queue to minimize the time spent. Finally because is like a live queue we need more than just a client so we need a Customer Generator class to create timing constraints and execute them for each client. All of these are powered by Java using Threads for continuous execution over a period of time.

**4. Implementation**

After this breef introduction it is time to discuss about the actual code of the problem for each class.

Customer Class:

Each customer has an arrival time and a service time and for uniqueness also an ID. The constructor will create an object with these characteristics. In this class will be also getters and setters for these.

CustomerGenerator class:

For this application to resemble the real world where each client has an arrival time and a proccessing time for it’s shopping cart different from the person before it is required from us to do the same. To implement this application in this way a random number must be chose for each customer. Because I wanted to introduce the values myself and not to be hardcoded the minimum arrival time, minimum service time, maximum arrival time and maximum service time are taken from the graphical user interface. The service time and the arrival time are computed with values that I will enter at the simulation screen in this class using a formula like: minArrival/ServiceTime + random(maxArrival/ServiceTime- minArrival/ServiceTime). To save these data an array of Customers must be created and each object of type Customer having a different arrival time and service time than the one before must be added in this list and these values set. I chose execute these for a max of 100 clients but this value can be modified at any time.

Checkout class:

Now that we created the Customers and we set them different values for arriving time and service time we can place them at the check-out. For this task I chose to have only 3 check-outs but there can be any number. Like in the real world a queue will form at these check-outs because the processing time is not instant. This class must implement the Runnable interface because the logic of this will be ran using threads. This class constructor will create a check-out with an id and at each creation the system’s time in miliseconds will be saved in a variable and the finish time will also be set as the sum of the current time and the simulation time that we enter in the graphical user interface.

Besides the constructor some other methods are necessary. First, because we implemented the Runnable interface a method called “run” will be automatically be created where the code that we want to repeat will be placed. In this case the checkout must serve the customers. This simulation must work as long as the current time does not succed the finish time. In this method customers will be taken from an arraylist of customers that were added with a method called addCustomer from the CustomerGenerator and printed in the graphical user interface . Each client that passes will be counted and an initial time saved and this will be used to calculate the average service time and average wait time. With the help of threads we synchronize the clients such that each customer must wait for the one that is served in front of him. Even if the simulation finished if there are at most one client in the queue it will be served and the average times for wait and service saved.

Scheduler class:

In this class the customers are scheduled at check-outs. In the constructor of this class is the logic of how to customers will be placed at check-outs. Starts by initializing 3 check-outs, saves the start time of the system and the end time which is calculated as the sum of the start time and simulation time. After that with help of 3 ThreadPool each check-out is generated and a client is added at each one. In a while which loops if the execution or the start time are not over clients are generated and with help from a function which calculates the minimum number of clients at each check-out the new customers are placed at the check-out with the minimum number of clients to have an even distribution. A customers is not added if the capacity of the queue will not allow it.

Because is the class that is called from the graphical user interface also the texts with the average service time and average arrival time will be displayed at the end of the execution in the text area of the graphical user interface for each check-out.

GUI class:

After all that logic that I explained from the previous classes is time to see exactly how the application is working in a visual environment.

One of the requirements for this problem is to not have values hardcoded in the program so for that in the graphical user interface we need some boxes where to place the values and also a button to decide when the simulation to start.

We need 5 boxes: a minimum arrival time box, a maximum arrival time box, a minimum service time box, a maximum service time box and a simulation interval time box. In these boxes the values I will enter will be in miliseconds because I used everywhere the system’s milisecond function(1 sec= 1000 miliseconds).

Also to display the queue evolution a text area will be required. Because the simulation could perform for a long time a JscrollPane will be attached to this text area to be able to scroll to the whole information diplayed in that text area.

In the constructor of this class all the elements that we will see in the simulation window are dealt with. I chose to not have a layout for this window because I wanted to place all those things where I wanted . All the elements like labels , text fields , text area and button are initialized here and are given certain bounds of the x axis, y axis and certain width and height using setBounds command and after that are added to the screen. The only button in this class is the start button which will start the simulation and an action listener will be added to it

In order for this application to work properly when the start is pressed the simulation must start and stop only when the finishing conditions are met. For that to happen I created outside the constructor of this class an item of type runnable with the job to create a new Scheduler class. I did this because the Scheduler class does not implement the runnable interface. When the start is pressed using an executor defined in the attributes section this runnable element will be executed.

At the end of this constructor the usual commands of size, visibility, close operation and location are placed.

Besides the constructor because in the previous classes I used input directily from the boxes some addition methods are implemented. These are the getter methods which will take the string information from the boxes and convert it into integers that I work with for each minimum and maximum arrival and service times. The final method is a method for displaying the text as string in the textarea and console.

Main class:

The final class is the Main class whose only job is to create a new object of type GUI. Basically this is the start point in our application.

**5. Improvements**

As always this is a minimum implementation, a “demo” of a real world problem which can be improved in many ways. Things to improve like:

-To display the customers at each check-out with live addition and removal

of customers;

-To have an option to transfer the remaining clients from a check-out to the

next ones if this one closes;

-To have a button which adds more check-outs when we want to or when

they are needed for a large number of customers;

-To remove a customer if it changed its mind and to decrease the time for

others.

**6. Conclusions**

This Queue Simulation problem was a new challenge for me because I never worked with threads before and with an application that works “ on its own” from start to end. I was quite difficult but I learned new things. The hard part was to synchronize the threads to work as they should.

**7. Bibliography**

**1. https://www.javacodegeeks.com/2013/01/java-thread-pool-example-using-executors-and-threadpoolexecutor.html**

**2. http://www.tutorialspoint.com/java/java\_multithreading.html**