Technical University of

Cluj-Napoca

Faculty of Automation

and

Computer Science

Queues-Java Concurrency

Discipline: Programming Techniques

Date: 25.05.2018

Gavril Luca

Group:30422

1.Objective:

Propose, design and implement an application whose scope is to simulate systems based on queues in order to determine and optimize the client's waiting time.

- The main objective of this project is to create a Java application, using the multithreading programming concept, that simulates the random generation of a number of clients, sitting each client in a queue, and processing the waiting time for each client, eliminating him from the system.
- As a secondary objective, the designer has implemented a Graphical User Interface, in order to allow the user to easy access and use the application.
- The application also has a real time view of the simulation, showing the evolution of the queues during the simulation interval.

2.Analysis:

The user works with the Graphical User Interface. He has to choose a simulation time duration, a number of randomly generated clients, a number of queues, and two intervals: one for arrival times of the clients, and another for the service time for each client. Regarding the application, the analysis can be viewed as such:

Input data:

-Minimum and maximum interval of arriving time between customers;

-Minimum and maximum service time;

-Number of queues;

-Simulation interval;

-Other information you may consider necessary;

Minimal output:

-The 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 (other useful information may be also considered);

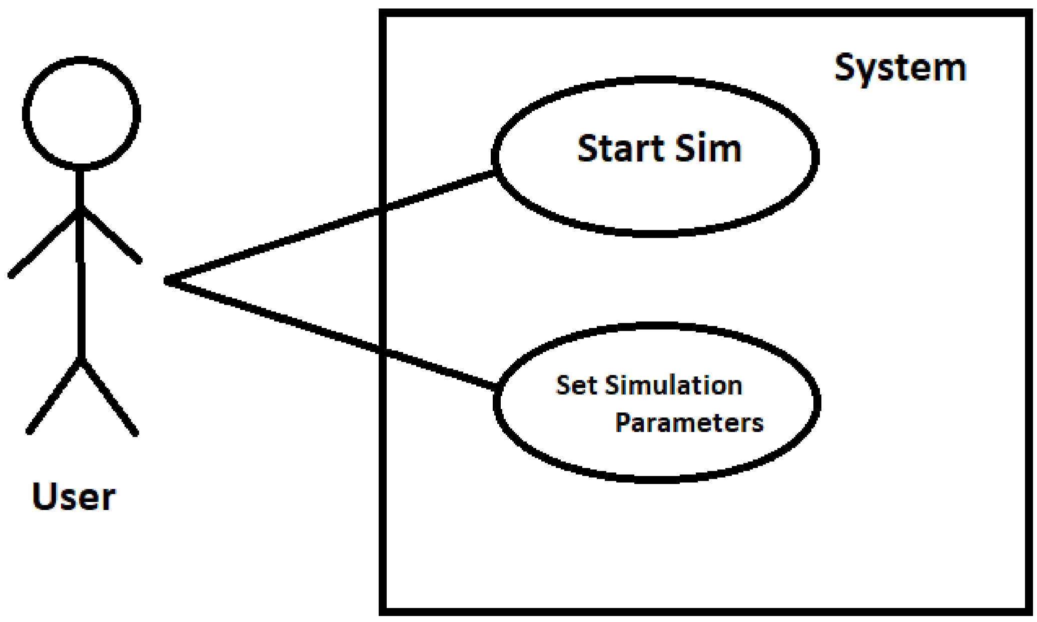
-Log of events and main system data;

-Queue evolution;
-Peak hour for the simulation interval;

Queues are commonly used to model real world domains. 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 their "clients" are waiting in queues before they are served. This can be done by adding more queues, or implementing the program in such a way that a new client enters a minimal queue.

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 customers 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 clients in the queue and their service needs.

2.1. Use case diagram:



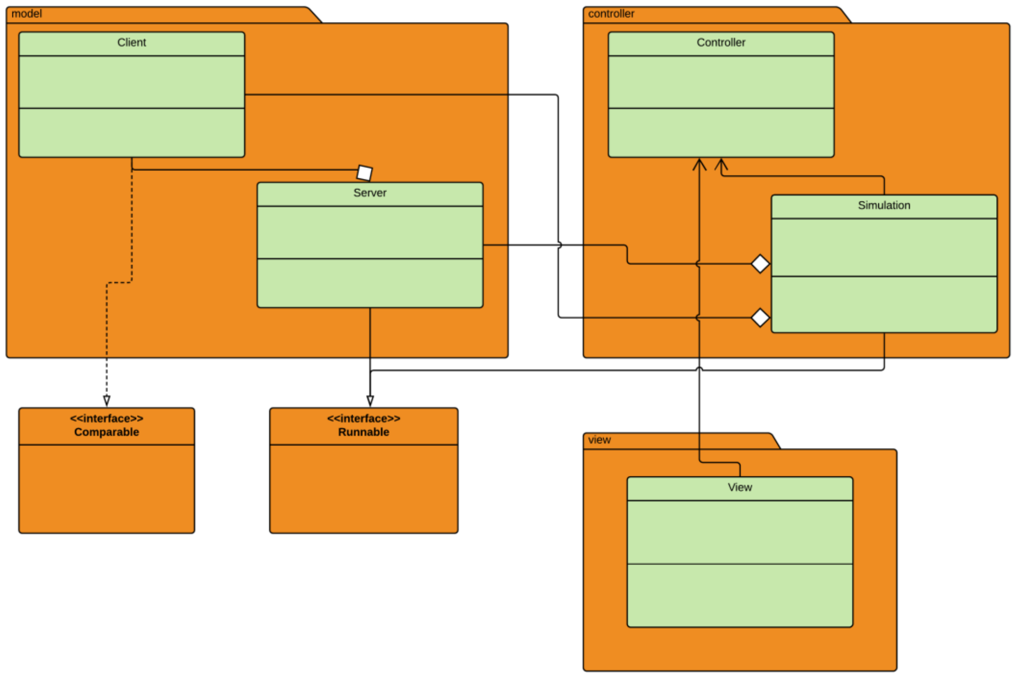
The user has two main possibilities to interact with the application, dependent to one another : he has to set the parameters of the simulation: simulation duration time, a number of randomly generated clients, and two intervals for the arrival times of the clients, respectively for the service times of the clients and he has to start the simulation.

2.2. Assumptions:

The main assumption regarding this application is that the user is capable of inserting right values in the Graphical User Interface Text Fields. He has to insert lower values in the minArrival and minService boxes, compared to the values in the maxArrival and maxService boxes. Also, there cannot be more queues than clients, because it does not make sense. The maximum number of queues has to be less or equal to 10, for a nice view of the simulation. However, the program works with a N number of queues. The user must obviously only introduce numbers in the text fields, otherwise the simulation will not start, or it will act badly.

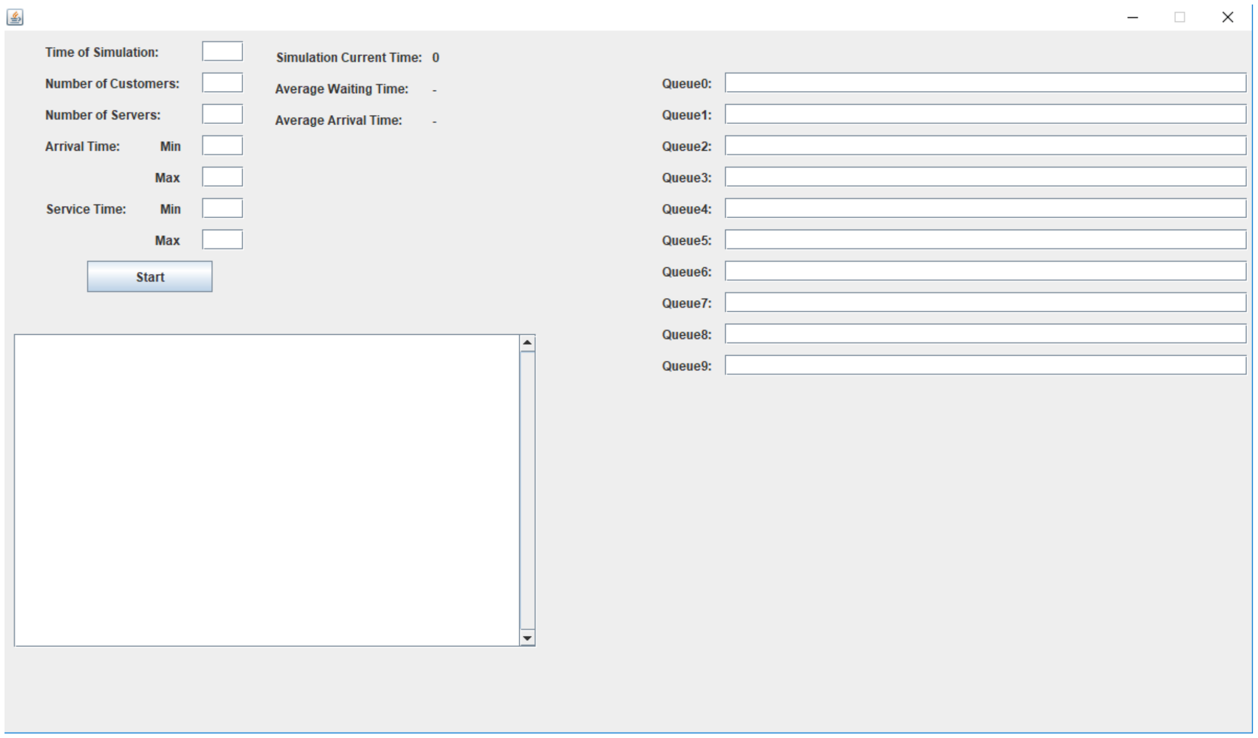
2.3. Design:

2.3.1. UML Diagram:



As it can be seen from the UML Diagram , the system is split in multiple packages, each holding a specific purpose in the implementation of the program. The system is built under the MVC model, with classes defined for specific roles each. The view package contains the GUI class, the model package contains 2 classes that handle the clients and the servers (queues) , while the controller package contains 2 classes: Controller and Simulation. The Controller class logically connects the View class and the Simulation class, while the Simulation class holds methods and attributes needed for creating the simulation.

2.3.2. Graphical User Interface:



The Graphical User Interface is easy to understand and use, and it is the main way for the user to connect to the simulated system. The Graphical User Interface is quite complex, holding numerous labels (for simulation duration time, number of customers, arrival and service time intervals as well as for showing the current stage of the simulation, for the average arrival time and for the average waiting time) . The GUI only has one JButton, defined as “Start” that will start the simulation of the system, after the JTextFields of the formerly explained JLabels will be written in. Below the button, a JPanel holding a JTextArea was created for the user, showing the results of each task. In case the data overflows the JPanel size, a JScrollPane was implemented to move through the text area. The JTextArea is updated in real simulation time, showing what happens as each second passes. On the right hand side of the frame, 10 labels joined by 10 textfields, counting from 0 to 9, are shown for the queues, also updated in real time.

3. Implementation:

Regarding the structure of the project, we start from the definition of a thread, as well as its way of working. Besides the principle thread that is created by the main function, a new thread will be created at the push of a button. This thread has the role to generate and distribute a number of clients to a number of queues, creating a thread for each different queue. These smaller, server thread, will take care of dequeuing each server, as well as calculating specific times for the servers and for the clients. The Graphical User Interface was implemented for "non experienced" users, meaning that while complex, it is easily readable and usable. Regarding relationships between classes, inside the model package, the relationship between Client and Server is one of aggregation. As the Simulation class HAS-A Client list but it also HAS-A Server list, again, the relationship is one of aggregation. The children can independently exist without the parent. Between the classes, the Server and Simulation classes implement the Runnable interface, for the implementation of the overridden method run(), while the Client class implements the Comparable interface, because of the overridden method compareTo().

As previously mentioned, the system is split in packages, each holding one or more classes.

- controller package:

  The controller package holds two different classes, Controller and Simulation. The Controller class is a lot more basic compared to other projects, holding 2 attributes: s of type Simulation and view of type View.

  Only one method is found inside the class, the constructor, in which we initialize the view attribute, and use the actionListener method for the start button to start a new simulation, based on the information found in the JTextFields in the frame. The Simulation class has a number of attributes : two specific ArrayLists of clients and queues, a view attribute of type View, and a number of int attributes with the purpose of defining the simulation. Inside the constructor of the Simulation class, each int value and the view are attributed a specific value, while the clients and queues being attributed the results of of two methods: generateClients(), which generates a given number of clients with random arrival and service times and generateQueues(), which creates a given number of new servers. For the generateClients() method, we use the Collections.sort method to arrange the clients in the ArrayList after their arrival times. The next method in the class is the getMinimalQueue() method, that always returns the index of the least populated queue in the system. For working with the view attribute, there are a number of methods : showTime(), showQueues(), averageWaiting() and averageArrival(), all of which add results to the view. The startQueues() method uses the multithreading concept, starting a new thread for each server in the ArrayList. Finally, the overridden run() method, for which we make use of the Runnable interface, executes the simulation of the system, while using the threads created in the startQueues() method and the list of randomly generated client list.

- model package:

The model package contains 2 classes : Client and Server. The Client class implements the Comparable interface, as it is needed to later sort a list of clients based on the arrival times. It has 3 attributes: serviceTime, arrivalTime and an id, each with their own getter and setter. It also contains an overridden method, toString(). The Server class also implements the Runnable interface and it has one attribute : an ArrayList of clients. Besides a getter and a setter for this attribute, the class has enqueue and dequeue methods, a toString() method and the run() overridden method, that dequeues each server while it is not empty. In order for the simulation to be accurate, the thread must sleep equal to the number of seconds of the serviceTime of each client.

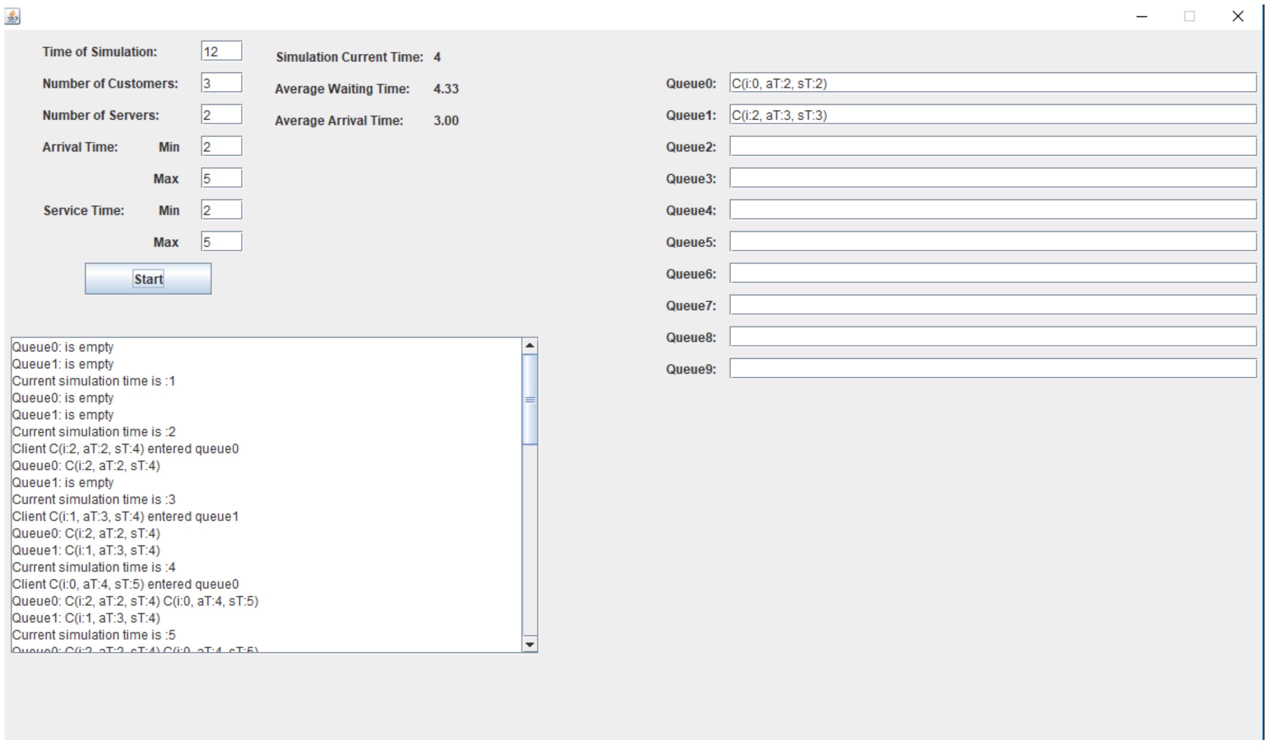
- view package:

As previously said, the Graphical User Interface is quite complex. Inside the view package, there only exists a View class, that has a constructor where the main frame is created. The button is then added to the frame, alongside the JPanel that holds the JTextArea, showing the results of the simulation. Next to the constructor, the class holds getters and setters for its attributes, a init() method for the main frame and an actionListener method for the start button.

- default package:

The default package holds the Main class, where we create a new Controller instance, in order for the system to start working (in order for the user to access and use the Graphical User Interface).

4.Results:



Time of Simulation:	12
Number of Customers:	3
Number of Servers:	2
Arrival Time: Min	2
Max	5
Service Time: Min	2
Max	5

Simulation Current Time: 4
Average Waiting Time: 4.33
Average Arrival Time: 3.00

Queue0:	C(i:0, aT:2, sT:2)
Queue1:	C(i:2, aT:3, sT:3)
Queue2:	
Queue3:	
Queue4:	
Queue5:	
Queue6:	
Queue7:	
Queue8:	
Queue9:	

```
Queue0: is empty
Queue1: is empty
Current simulation time is :1
Queue0: is empty
Queue1: is empty
Current simulation time is :2
Client C(i:2, aT:2, sT:4) entered queue0
Queue0: C(i:2, aT:2, sT:4)
Queue1: is empty
Current simulation time is :3
Client C(i:1, aT:3, sT:4) entered queue1
Queue0: C(i:2, aT:2, sT:4)
Queue1: C(i:1, aT:3, sT:4)
Current simulation time is :4
Client C(i:0, aT:4, sT:5) entered queue0
Queue0: C(i:2, aT:2, sT:4) C(i:0, aT:4, sT:5)
Queue1: C(i:1, aT:3, sT:4)
Current simulation time is :5
Queue0: C(i:2, aT:2, sT:4) C(i:0, aT:4, sT:5)
```

The result of this project is a complete showing of a client/queues simulation system. Because the designer has used threads in the implementation of the program, the simulations become very real. As each second passes, the user is announced of this through multiple ways : the

Current Time label, the Text Area information , the way the queues change their content etc. . In the Text Area, the user can see at each second the status of the queues involved in the simulation, whether they are empty, or if they hold clients. Each client arrives at his specific arrival time, and leaves the queue he was put in only after he has been serviced. At the end of the simulation, the user is announced that the simulation has ended through the Text Area. He will also find information regarding the average waiting time and average service time of the simulation. The right side of the text area presents the queues, at a maximum number of 10 (starting with 0 because of implementation reasons). The queues change their content with each second that passes (if the system says so).

5.Further developments:

Because the designer is working with threads, multithreading or Java Concurrency to be more exact, there is always room for improvement. The synchronization of the threads involved in the program can be a lot more strict, as there still is room for bugs or glitches with the current way they are synchronized. Regarding the simulation in itself, more averages can be implemented, as well as a better viewing of the clients' movements through and out of the queues. Also, the simulation could be implemented for a N number of servers, not having a limit (in the case of this project, the limit was 10). Furthermore, the program could be further modularized, with the introduction of separate threads to work with the processing of information and with the graphical user interface display.

6.Conclusion:

With this project, the student (the designer) is introduced to Java Concurrency, to the ability to run programs / parts of a program in parallel. He learns about the concept of multithreading, about executing processes or threads concurrently. It is also one of the first "simulation system" designs that the student has come in contact with, starting from a relatively simple task. The student keeps on developing his MVC model type designing skills, as well as his use of Graphical User Interfaces, as the project requires the usage of many JFrame / javax.swing components. Designing the project has proved to give the student an abundance of information, further developing his general JAVA code writing capabilities.

7.Bibliography:

https://docs.oracle.com/javase/tutorial/essential/concurrency/index.html

http://www.tutorialspoint.com/java/util/timer_schedule_period.htm

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

https://stackoverflow.com/