**ASSIGNMENT 2 DOCUMENTATION**

**QUEUES SIMULATOR**

**Name: Zarnescu Radu**

**Group: 30423**

**Laboratory Assistant: Antal Marcel**

Contents

[1. Assignment Objective 3](#_Toc476131445)

[2. Problem Analysis 3](#_Toc476131446)

[3. Design 6](#_Toc476131449)

[3.1. Data Structures 6](#_Toc476131451)

[3.2. UML Diagram 7](#_Toc476131452)

[4. Implementation 8](#_Toc476131454)

[4.1. Class Description 8](#_Toc476131453)

[4.2. Algorithms 10](#_Toc476131453)

[5. Testing 10](#_Toc476131454)

[6. Conclusions and Further Improvements 22](#_Toc476131455)

[7. Bibliography 23](#_Toc476131456)

1. **Assignment Objective**

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

The application should simulate (by defining a simulation time 𝑡𝑠𝑖𝑚𝑢𝑙𝑎𝑡𝑖𝑜𝑛) a series of N clients arriving for service, entering Q queues, waiting, being served and finally leaving the queues. All clients are generated when the simulation is started, and are characterized by three parameters: ID (a number between 1 and N), 𝑡𝑎𝑟𝑟𝑖𝑣𝑎𝑙 (simulation time when they are ready to go to the queue; i.e. time when the client finished shopping) and 𝑡𝑠𝑒𝑟𝑣𝑖𝑐𝑒 (time interval or duration needed to serve the client; i.e. waiting time when the client is in front of the queue). The application tracks the total time spent by every client in the queues and computes the average waiting time. Each client is added to the queue with minimum waiting time when its 𝑡𝑎𝑟𝑟𝑖𝑣𝑎𝑙 time is greater than or equal to the simulation time (𝑡𝑎𝑟𝑟𝑖𝑣𝑎𝑙 ≥ 𝑡𝑠𝑖𝑚𝑢𝑙𝑎𝑡𝑖𝑜𝑛).

1. **Problem Analysis**

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. 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 service supplier.

In computer science, a thread of execution is the smallest sequence of programmed instructions that can be managed independently by a scheduler, which is typically a part of the operating system. The implementation of threads and processes differs between operating systems, but in most cases a thread is a component of a process. Multiple threads can exist within one process, executing concurrently and sharing resources such as memory, while different processes do not share these resources. In particular, the threads of a process share its executable code and the values of its dynamically allocated variables and non-thread-local global variables at any given time.

A use case is a written description of how users will perform tasks on your website.  It outlines, from a user’s point of view, a system’s behavior as it responds to a request. Each use case is represented as a sequence of simple steps, beginning with a user's goal and ending when that goal is fulfilled.

Basic Flow: Simulating the queue progression:

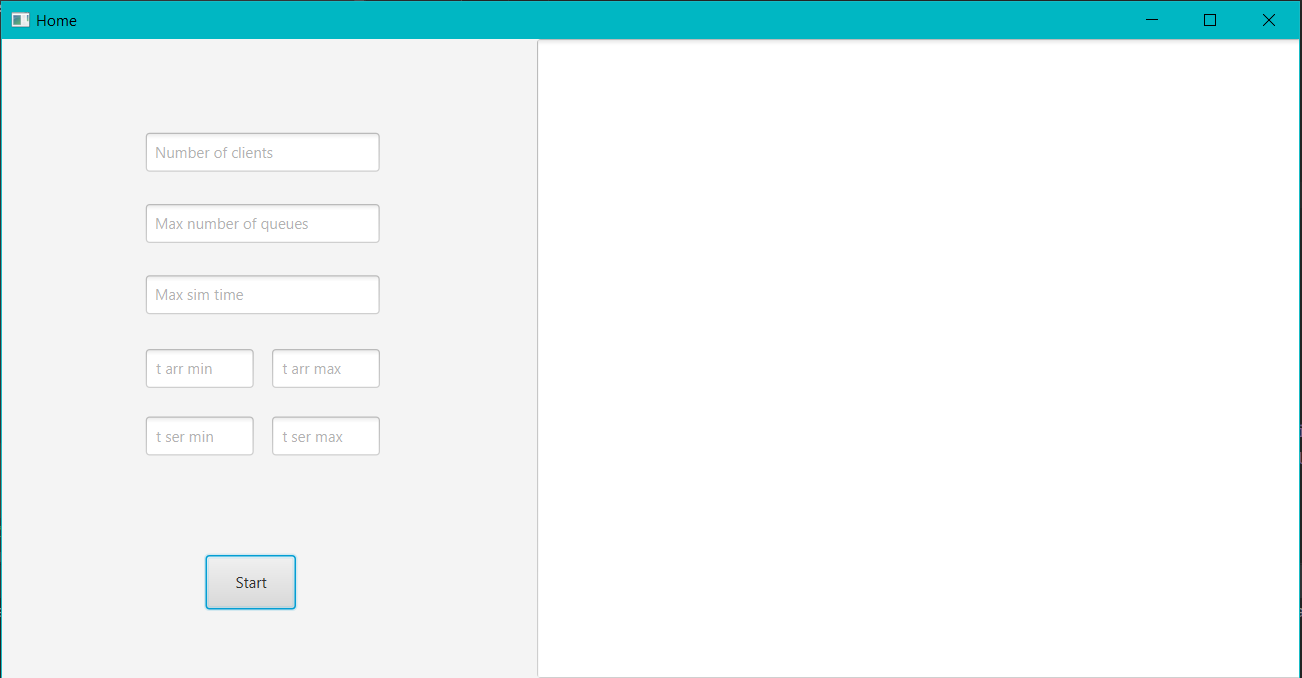
* Actor: User
* Description: The main success scenario is the situation where the user gives the program a number of clients, a number of queues, a maximum simulation time and two intervals: arrival time and service time

1. User inserts the numbers mentioned above
2. User presses the “Start” button

* Termination outcome: the queue progression is displayed on the right side of the GUI and in the “output.txt” file.

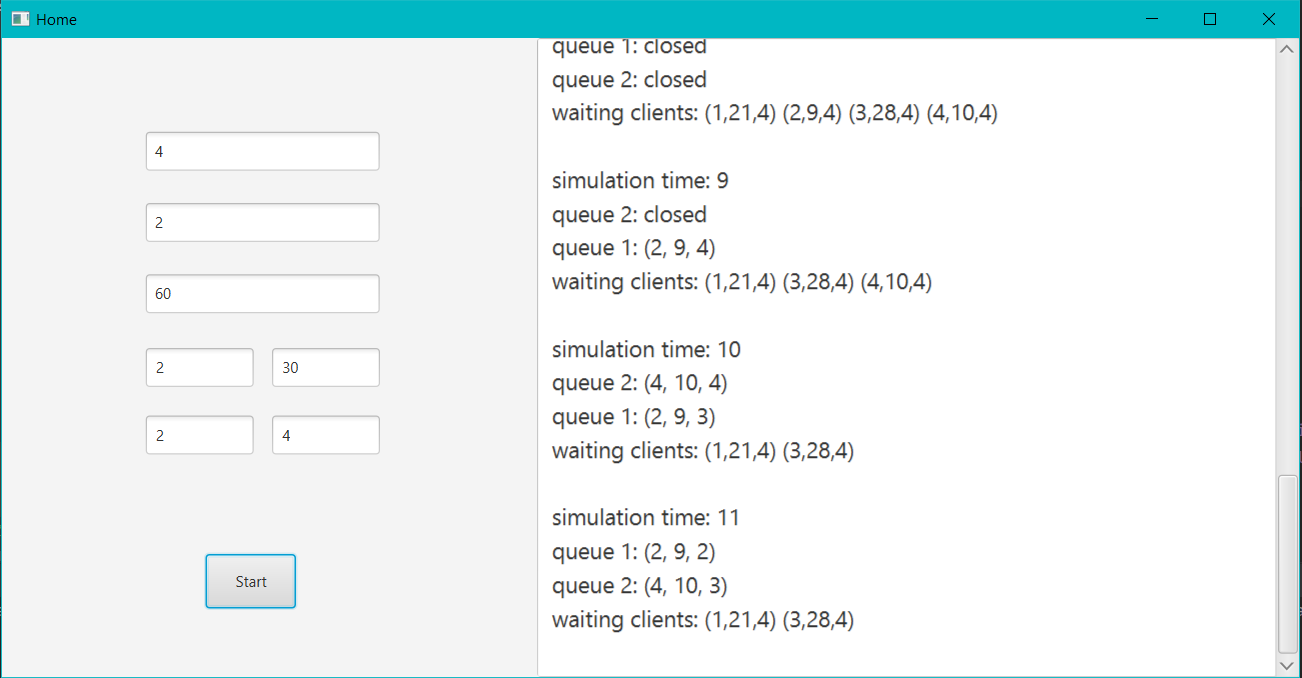
Illustration of the steps a user has to make in order to correctly use the application:

The user interacts with the program through a dedicated UI found in the screenshot below. It is very simple to use and it indicates the user what every field represents.



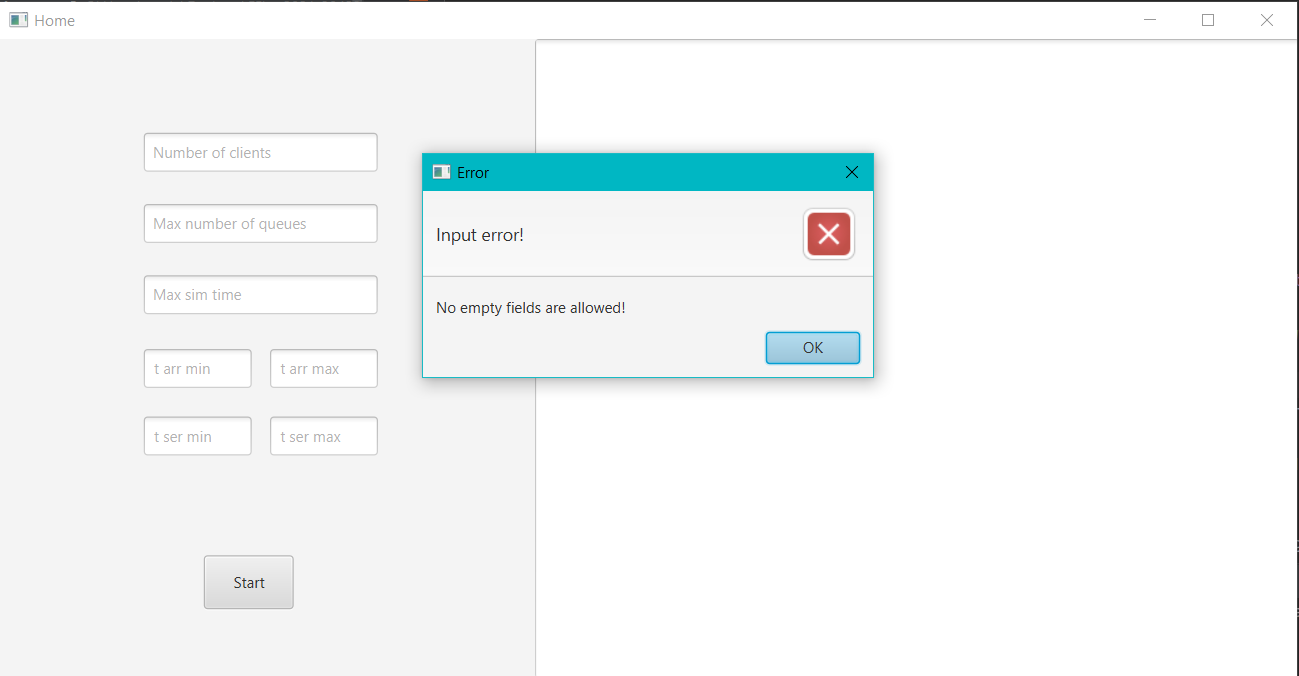
The only steps a user has to take is to insert the required parameters and press “Start”. Afterwards the queue progression will be displayed and updated every second until the maximum simulation time is reached.

A moment in the simulation is illustrated below:

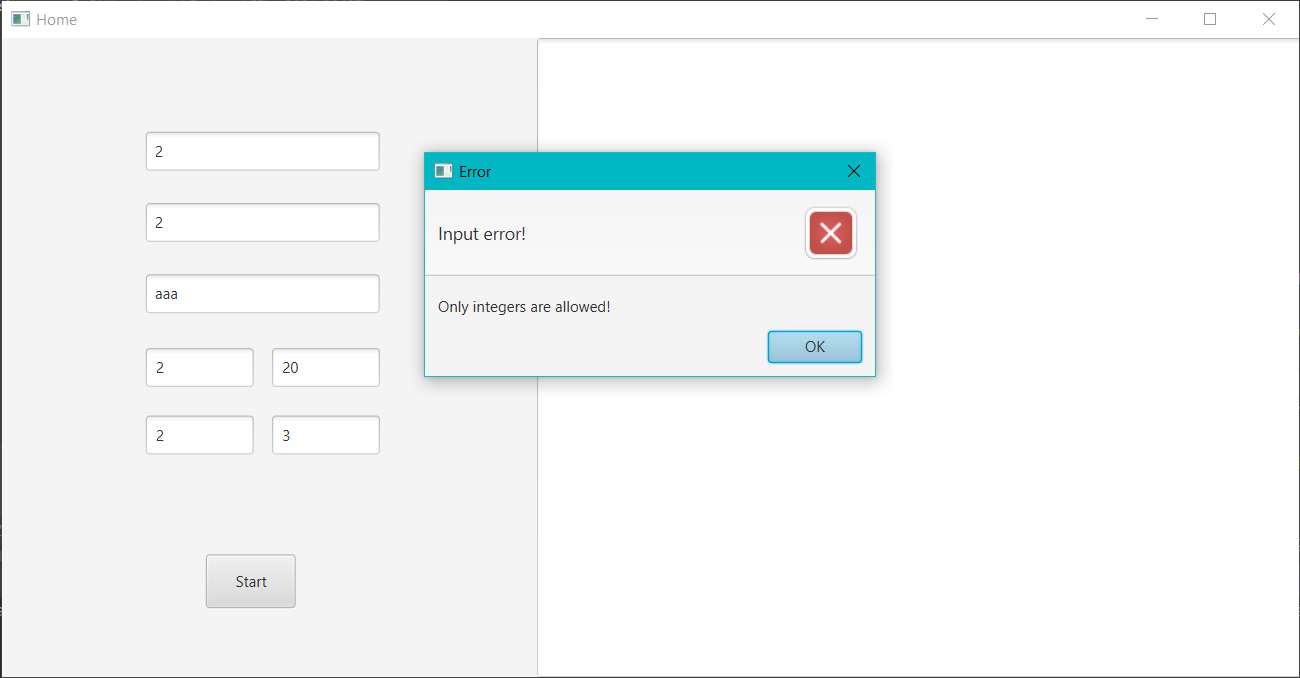


Cases when the program cannot function properly (error cases):

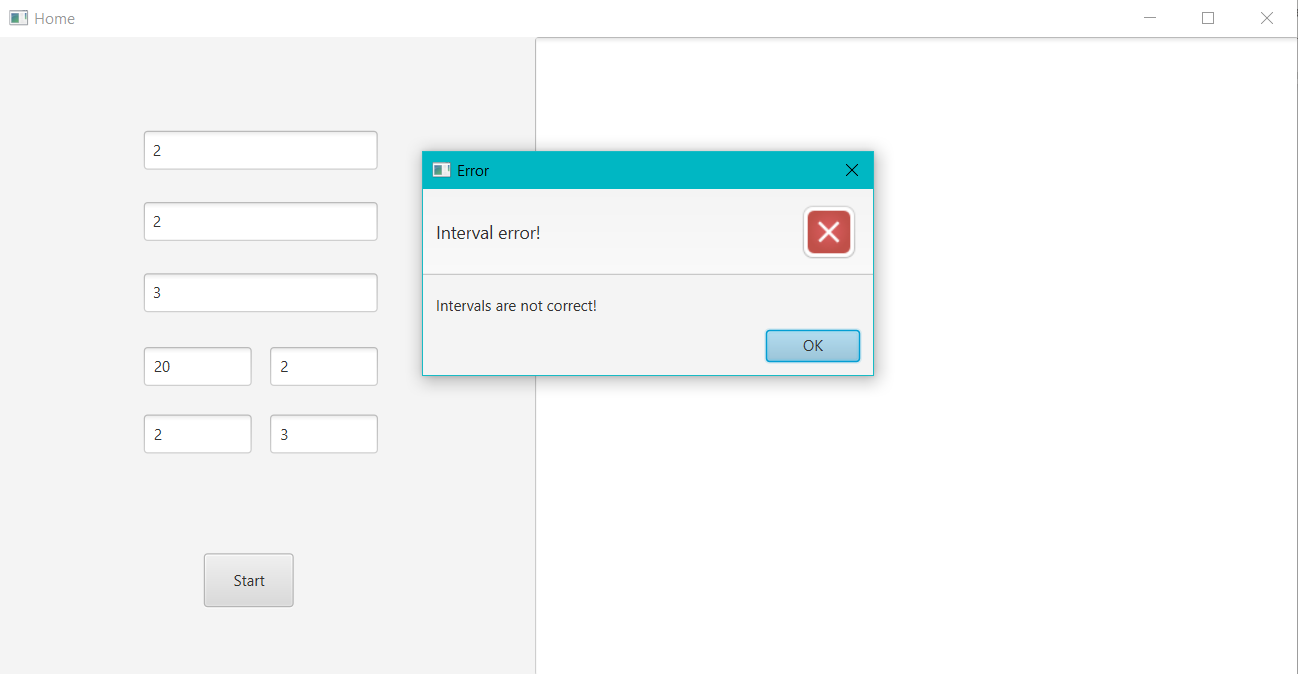
If any of the seven fields is empty the following error will pop up:



If any one of the fields have a value other than an integer the error below will occur:



The last possible error is triggered when the lower bound for any of the two intervals (arrival and service) is larger than the upper bound.

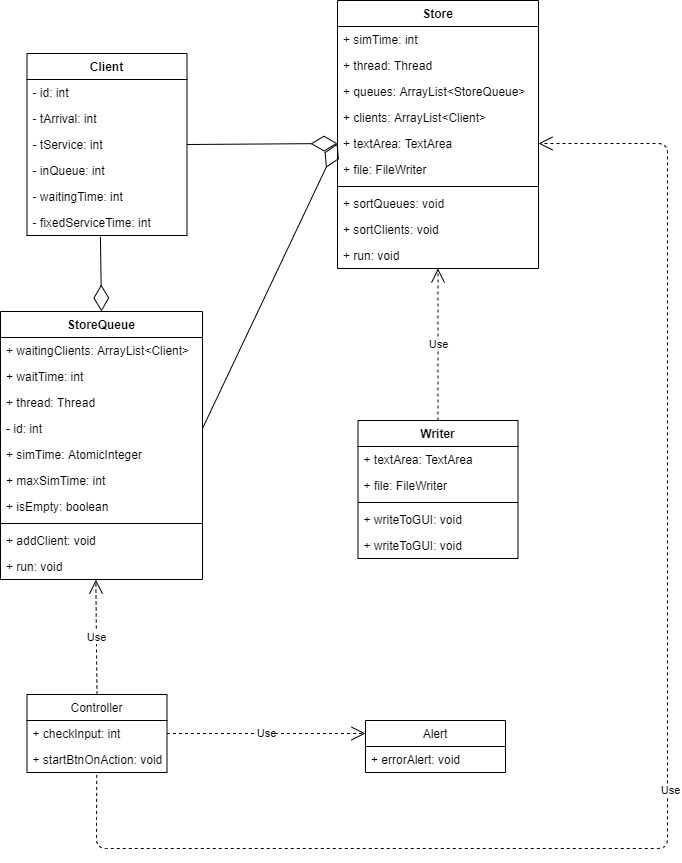


1. **Design**
   1. **Data Structures**

In order to implement the required problem, I have chosen to use three data structures: client, which has multiple integer attributes (input data for the problem); StoreQueue which represents a queue in the simulation as a thread; Store, the data structure which runs the entire simulation.

I have chosen to use ArrayLists instead of simple Arrays because they are more efficient for memory management and also, they do not have a fixed number of elements.

* 1. **UML Diagram (Class Diagram)**



1. **Implementation**
   1. **Class Description**

I based my class structure on the Model – View – Controller methodology. It is a pattern for modularizing the classes based on what each of them do.

1. Model includes the classes which mostly deal with the back end of the project, the algorithms and the operations which allow the program to function. It is the part which the user does not see at all when using the app. Classes: Client, Store, StoreQueue
2. View represents the classes that form the user interface, the part with which the user interacts. Classes: Alert, Main
3. Finally, controller is the collection of classes which make the connection between back end and front end, model and view. For example, when a button is pressed, the controller is responsible for calling the appropriate methods and algorithms that apply to that action. Classes: Controller, Writer.

* Client Class

The client class is the data structure used to store all the information about the simulation’s clients.

Methods:

* public Client(int id, int tArrivalMin, int tArrivalMax, int tServiceMin, int tServiceMax): the constructor which overloads the default one. It creates a new object of the Client class, for which it generates two random values, between tArrivalMin – tArrivalMax and, respectively, tServiceMin – tServiceMax.
* Setters and Getters which are needed to access private attributes.
* StoreQueue Class

The StoreQueue represents the class which contains a thread in order to be a independently running queue.

Methods:

* public StoreQueue(int id): the constructor which creates new instances of Polynomial and sets the id to the given value and waitTime to 0.
* public void addCLient(Client c): method which adds a new client to the waitingCLients of the queue and updates the wait time for that queue.
* public void run(): the method needed to start the thread; it is used to process all the clients from waitingClients and update the simulation every second.
* Getter and Setters
* Store Class

This is the main class that makes the simulation work. It is the application’s “brain”.

Methods:

* public void sortQueues(): this method sorts the queues from the store in increasing order of their wait time using Collections.sort.
* public void sortClients(): this method sorts the clients from the store in increasing order of their arrival time using Collections.sort.
* public void run(): the method needed to start the thread; it is used to process all the clients from waitingClients and update the simulation every second.
* public Store(TextArea textArea): the constructor used to create the store; the text area is given as a parameter to display the output on that text area in the graphical user interface.
* Writer Class

This is a class used to write the application log to the GUI and to a file.

Methods:

* public void writeToGUI(String s): this method writes the string s to the text area in the GUI.
* public void writeToFile(String s): this method writes the string s to the output file.
* Alert Class

A class used to display the errors illustrated previously.

Methods:

* public void errorAlert(String header, String context): this method prints an error message on the screen and displays the parameter messages.
* Controller Class

This class is used to control the interactions between the user and the user interface.

Methods:

* public int checkInput(): this method check if the input is not empty, all integers and checks if the lower bounds of the intervals are not greater than the higher bounds. It returns -1 if something is wrong with the input.
* Public void startBtnOnAction(ActionEvent event): this method sets up and starts the simulation when the “Start” button is pressed.
* Main Class

The main class extends the Application class and is used to start and shape the User Interface.

* 1. **Algorithms**

In this section I will briefly describe the algorithms I used to implement the required task.

Firstly, after all the input data has been stored, n random clients and m queues will be created and inserted into the store.

When the Start button is pressed, the simulation starts. The clients are sorted in increasing order of their arrival time. Then, the program checks if there are clients with the arrival time equal to the current simulation time and sends them to the queue with the lowest waiting time. If the client is the first client to enter that queue, the thread will start and will continue to run until the end of the simulation.

If a queue is not empty, it will process the clients who are in it, meaning that it decreases their service time until it is 0. At this moment the client is removed from the queue. Each queue stops when the simulation time reaches the maximum limit.

Each second the program will display: the simulation time, the status of each queue (closed or the client which occupies it) and the waiting clients.

At the end of the program there will be displayed on the GUI the average waiting time, average service time and peak hour.

1. **Testing**

The most important part of the project is the testing. If the results are not correct, then the program is not doing it’s job properly. In order to test whether the app is working well, we can just run the program, follow the steps presented in section 2, and observe the output.

Below you will find the log displayed when running the first test for the program.

Input:

N = 4 clients

Q = 2 queues

𝑡𝑠𝑖𝑚𝑢𝑙𝑎𝑡𝑖𝑜𝑛 𝑀𝐴𝑋 = 60, a 60 seconds simulation interval

[2, 30] - the bounds for the client parameters, respectively a minimum and maximum arrival time, meaning that clients will go to the queues from second 2 up to second 30.

[2, 4] - the bounds for the service time, meaning that a client has a minimum time to wait in front of the queue of 2 seconds and a maximum time of 4 seconds.

Log:

simulation time: 0

queue 1: closed

queue 2: closed

waiting clients: (1,24,4) (2,11,4) (3,27,4) (4,9,4)

simulation time: 1

queue 1: closed

queue 2: closed

waiting clients: (1,24,4) (2,11,4) (3,27,4) (4,9,4)

simulation time: 2

queue 1: closed

queue 2: closed

waiting clients: (1,24,4) (2,11,4) (3,27,4) (4,9,4)

simulation time: 3

queue 1: closed

queue 2: closed

waiting clients: (1,24,4) (2,11,4) (3,27,4) (4,9,4)

simulation time: 4

queue 1: closed

queue 2: closed

waiting clients: (1,24,4) (2,11,4) (3,27,4) (4,9,4)

simulation time: 5

queue 1: closed

queue 2: closed

waiting clients: (1,24,4) (2,11,4) (3,27,4) (4,9,4)

simulation time: 6

queue 1: closed

queue 2: closed

waiting clients: (1,24,4) (2,11,4) (3,27,4) (4,9,4)

simulation time: 7

queue 1: closed

queue 2: closed

waiting clients: (1,24,4) (2,11,4) (3,27,4) (4,9,4)

simulation time: 8

queue 1: closed

queue 2: closed

waiting clients: (1,24,4) (2,11,4) (3,27,4) (4,9,4)

simulation time: 9

queue 1: (4, 9, 4)

queue 2: closed

waiting clients: (1,24,4) (2,11,4) (3,27,4)

simulation time: 10

queue 2: closed

queue 1: (4, 9, 3)

waiting clients: (1,24,4) (2,11,4) (3,27,4)

simulation time: 11

queue 1: (4, 9, 2)

queue 2: (2, 11, 4)

waiting clients: (1,24,4) (3,27,4)

simulation time: 12

queue 1: (4, 9, 1)

queue 2: (2, 11, 3)

waiting clients: (1,24,4) (3,27,4)

simulation time: 13

queue 1: closed

queue 2: (2, 11, 2)

waiting clients: (1,24,4) (3,27,4)

simulation time: 14

queue 1: closed

queue 2: (2, 11, 1)

waiting clients: (1,24,4) (3,27,4)

simulation time: 15

queue 1: closed

queue 2: closed

waiting clients: (1,24,4) (3,27,4)

simulation time: 16

queue 1: closed

queue 2: closed

waiting clients: (1,24,4) (3,27,4)

simulation time: 17

queue 1: closed

queue 2: closed

waiting clients: (1,24,4) (3,27,4)

simulation time: 18

queue 1: closed

queue 2: closed

waiting clients: (1,24,4) (3,27,4)

simulation time: 19

queue 1: closed

queue 2: closed

waiting clients: (1,24,4) (3,27,4)

simulation time: 20

queue 1: closed

queue 2: closed

waiting clients: (1,24,4) (3,27,4)

simulation time: 21

queue 1: closed

queue 2: closed

waiting clients: (1,24,4) (3,27,4)

simulation time: 22

queue 1: closed

queue 2: closed

waiting clients: (1,24,4) (3,27,4)

simulation time: 23

queue 1: closed

queue 2: closed

waiting clients: (1,24,4) (3,27,4)

simulation time: 24

queue 2: closed

queue 1: (1, 24, 4)

waiting clients: (3,27,4)

simulation time: 25

queue 2: closed

queue 1: (1, 24, 3)

waiting clients: (3,27,4)

simulation time: 26

queue 2: closed

queue 1: (1, 24, 2)

waiting clients: (3,27,4)

simulation time: 27

queue 1: (1, 24, 1)

queue 2: (3, 27, 4)

waiting clients:

simulation time: 28

queue 1: closed

queue 2: (3, 27, 3)

waiting clients:

simulation time: 29

queue 1: closed

queue 2: (3, 27, 2)

waiting clients:

simulation time: 30

queue 1: closed

queue 2: (3, 27, 1)

waiting clients:

simulation time: 31

queue 1: closed

queue 2: closed

waiting clients:

simulation time: 32

queue 1: closed

queue 2: closed

waiting clients:

simulation time: 33

queue 1: closed

queue 2: closed

waiting clients:

simulation time: 34

queue 1: closed

queue 2: closed

waiting clients:

simulation time: 35

queue 1: closed

queue 2: closed

waiting clients:

simulation time: 36

queue 1: closed

queue 2: closed

waiting clients:

simulation time: 37

queue 1: closed

queue 2: closed

waiting clients:

simulation time: 38

queue 1: closed

queue 2: closed

waiting clients:

simulation time: 39

queue 1: closed

queue 2: closed

waiting clients:

simulation time: 40

queue 1: closed

queue 2: closed

waiting clients:

simulation time: 41

queue 1: closed

queue 2: closed

waiting clients:

simulation time: 42

queue 1: closed

queue 2: closed

waiting clients:

simulation time: 43

queue 1: closed

queue 2: closed

waiting clients:

simulation time: 44

queue 1: closed

queue 2: closed

waiting clients:

simulation time: 45

queue 1: closed

queue 2: closed

waiting clients:

simulation time: 46

queue 1: closed

queue 2: closed

waiting clients:

simulation time: 47

queue 1: closed

queue 2: closed

waiting clients:

simulation time: 48

queue 1: closed

queue 2: closed

waiting clients:

simulation time: 49

queue 1: closed

queue 2: closed

waiting clients:

simulation time: 50

queue 1: closed

queue 2: closed

waiting clients:

simulation time: 51

queue 1: closed

queue 2: closed

waiting clients:

simulation time: 52

queue 1: closed

queue 2: closed

waiting clients:

simulation time: 53

queue 1: closed

queue 2: closed

waiting clients:

simulation time: 54

queue 1: closed

queue 2: closed

waiting clients:

simulation time: 55

queue 1: closed

queue 2: closed

waiting clients:

simulation time: 56

queue 1: closed

queue 2: closed

waiting clients:

simulation time: 57

queue 1: closed

queue 2: closed

waiting clients:

simulation time: 58

queue 1: closed

queue 2: closed

waiting clients:

simulation time: 59

queue 1: closed

queue 2: closed

waiting clients:

simulation time: 60

queue 1: closed

queue 2: closed

waiting clients:

The averages and peak hour are only displayed in the GUI.

It can be observed that every client enters a queue when their arrival time reaches the required simulation time and that they remain in the queues as long as their service time is greater than 0.

1. **Conclusions and Further Improvements**

To conclude, this assignment was a very good exercise in learning and understanding threads. Now, I would consider myself more experienced in the techniques of programming.

The most important thing I learned was how to use and create threads in java. It was a little complicated to understand how to synchronize all the threads, but in the end, I managed to understand and implement the problem properly.

One problem I encountered was when I ran the third test proposed in the assignment, with 1000 clients. The problem was that a few seconds after the simulation started, the application froze and wouldn’t recover. I think this issue was due to my efficiency of writing code, not the logic behind it.

As for further improvements, it would be a good idea to make the simulation stop when there are no more clients left because then the simulation wouldn’t work in vain. Another improvement would be to improve the efficiency of the app and to use a scheduler.

1. **Bibliography**

* wikipedia.org
* [www.stackoverflow.com](http://www.stackoverflow.com)
* geeksforgeeks.org
* lecture courses and laboratory sessions