

# **DOCUMENTATION**

## ***QUEUES MANAGEMENT APPLICATION USING THREADS AND SYNCHRONIZATION MECHANISMS***

### **ASSIGNMENT 2**

STUDENT NAME: Oltean Darius-Ionuț  
GROUP: 30224

## CONTENTS

1. Assignment Objective .....	3
2. Problem Analysis, Modeling, Scenarios, Use Cases.....	3
3. Design .....	4
4. Implementation .....	8
5. Results.....	10
6. Conclusions.....	13
7. Bibliography .....	14

## 1. ASSIGNMENT OBJECTIVE

The main objective of this project is to design, implement and test a Queues Management Application using threads and synchronization mechanisms. To achieve this, I had to go through the following steps:

- Problem analysis, modeling solutions and scenarios and build use cases will be described in detail in the second section.
- OOP design of the application along with the UML package and class diagrams, in section three.
- In the fourth section it will be described the implementation of each class and important methods contained therein.
- Section five will be dedicated to the result and test cases of the application.
- In the last section I will describe a brief conclusion and possible future developments.

## 2. PROBLEM ANALYSIS, MODELING, SCENARIOS, USE CASES

A queues management application should take a few inputs to simulate the queues such as number of clients, number of queues, simulation time, minimum arrival time, maximum arrival time, minimum service time and maximum arrival time.

These inputs are used to instantiate a simulation manager which accounts the simulation time step by step.

Using a scheduler we can dispatch tasks through a strategy (time strategy). Each task is processed in a server which is implemented as an independent thread. Because of the simultaneous servers we must implement a synchronization mechanism.

The client modeling is implemented by three parameters: ID, Arrival Time and Service Time.

The simulation should have N clients which should be dispatched at a queue, wait, arrive, being served and finally leave the queue.

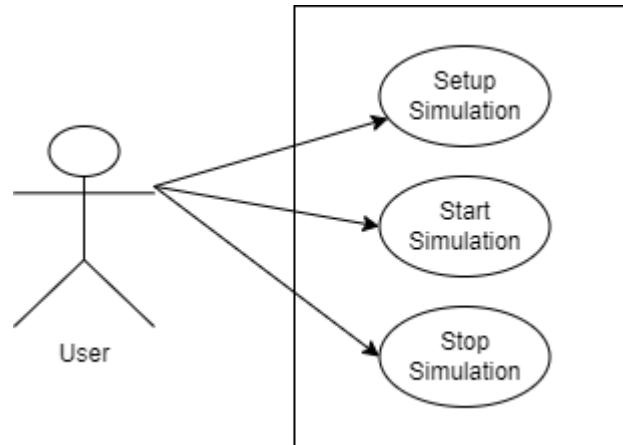
The application tracks the total time spent by every client in the queues and computes the average waiting time. Each client is added to the queue with the minimum waiting time when its *tarrival* time is greater than or equal to the simulation time ( $tarrival \geq tsimulation$ ).

### ➤ Use case description scenario:

- **Use case:** Setup simulation.
- **Primary actor:** User.
- **Success scenario steps:**
  - The user inserts the values for the: number of clients, number of queues, simulation interval, minimum and maximum arrival time, and minimum and maximum service time
  - The user clicks on the validate input data button.
  - The application validates the data and displays a message. informing the user to start the simulation.
- **Alternative scenarios:**
  - **Invalid input:**
    - ✓ The user inserts invalid values for the application's setup parameters.

- ✓ The application displays an error message and requests the user to insert valid values.
- ✓ The scenario returns to step 1.

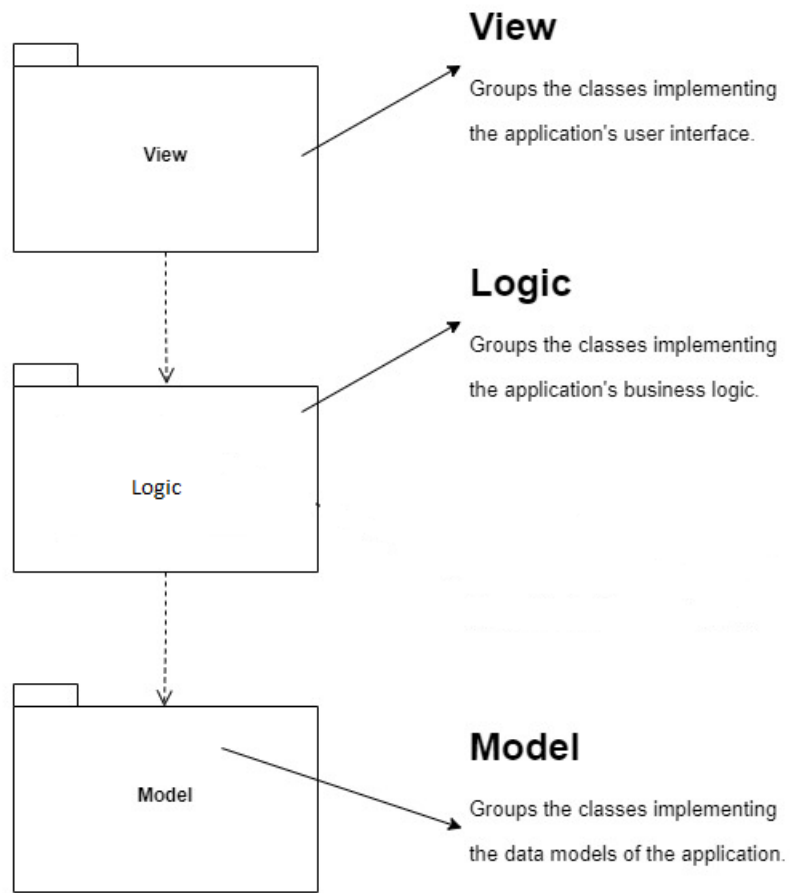
**Use case diagram:**



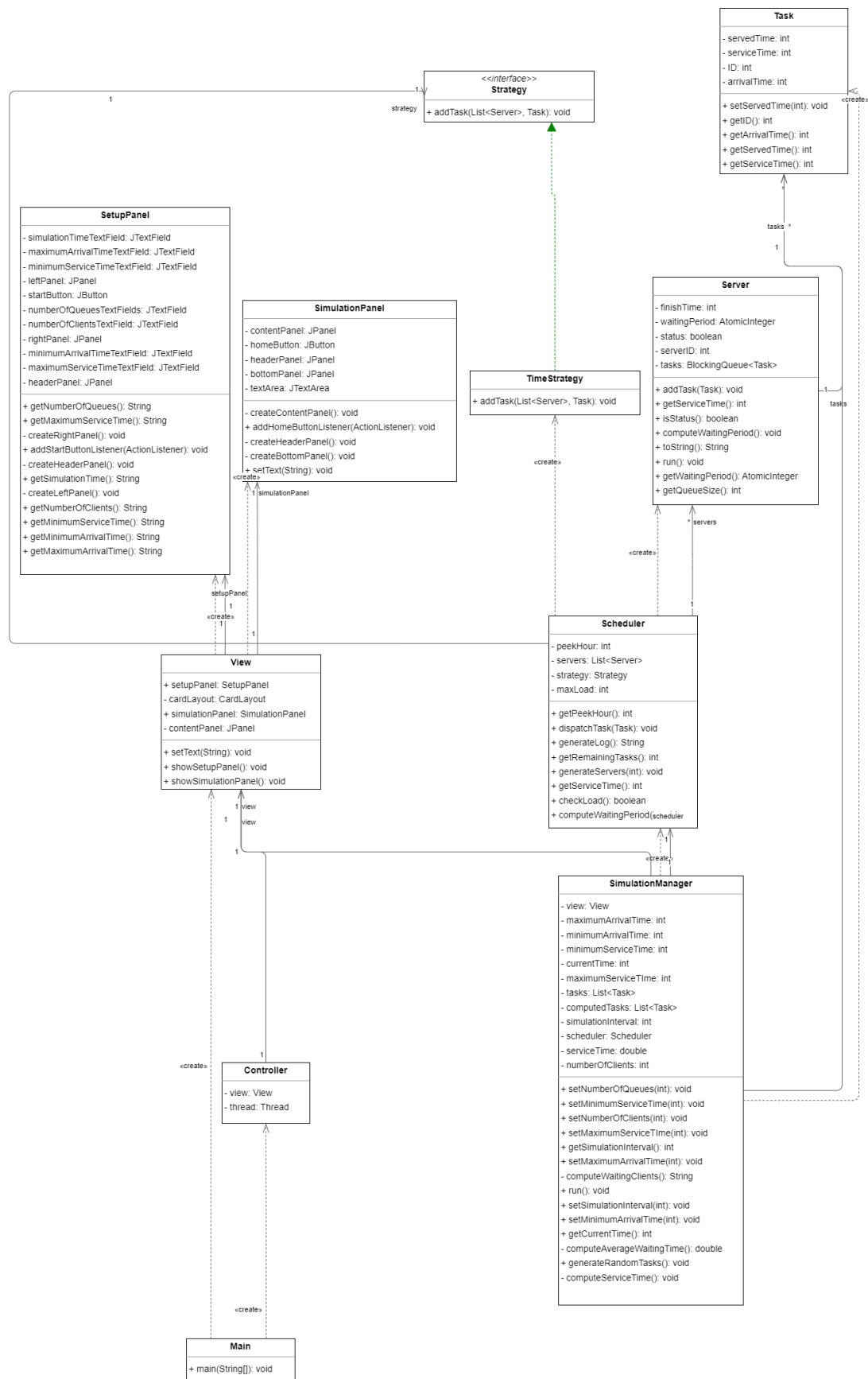
### 3. DESIGN

The OOP design of the application is structured in 3 packages. The graphical user interface is implemented in the View package which contains View class and Controller class. Data models are described in the Model package with the Server Class and Task Class. Logic package implements SimulationManager Class, Scheduler Class, Strategy Interface and TimeStrategy Class.

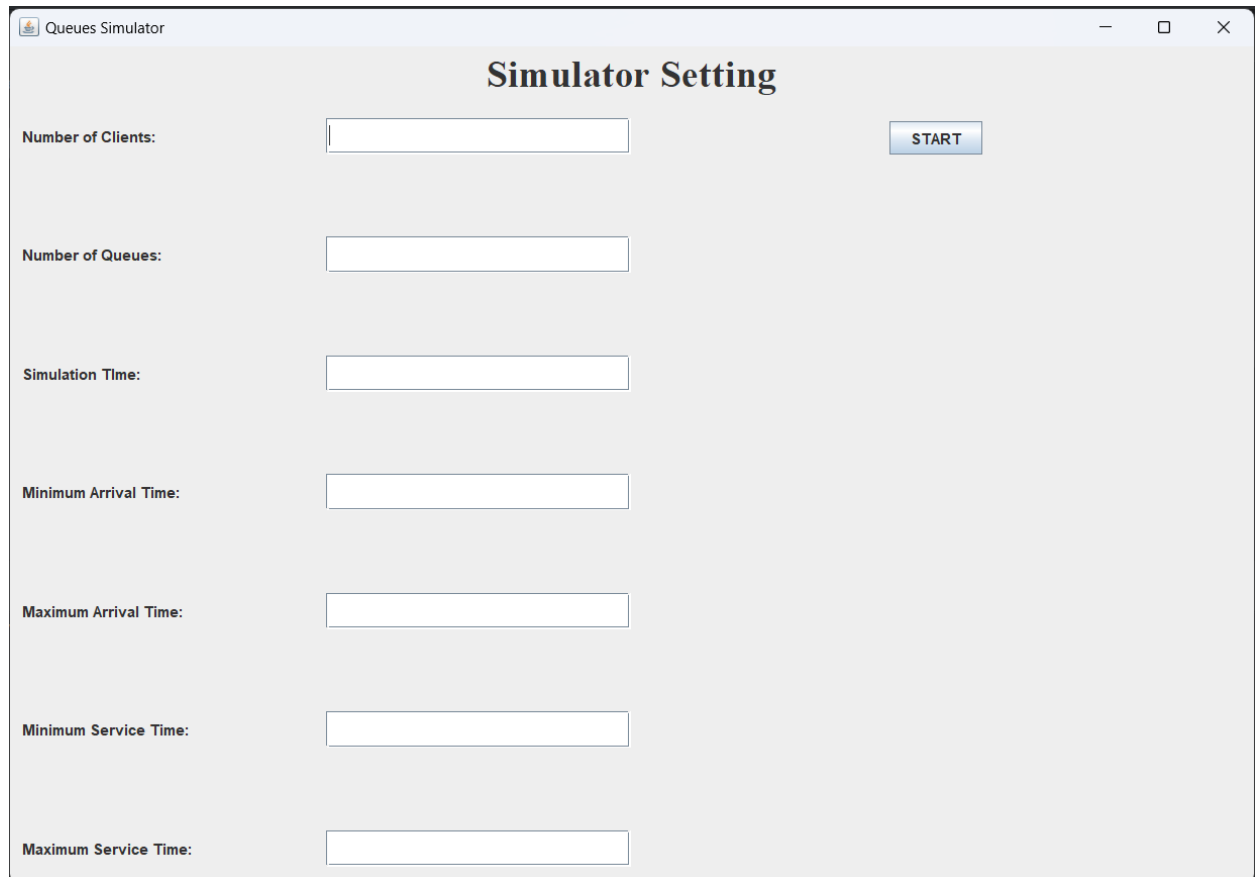
**UML package diagram:**



**UML class diagram:**



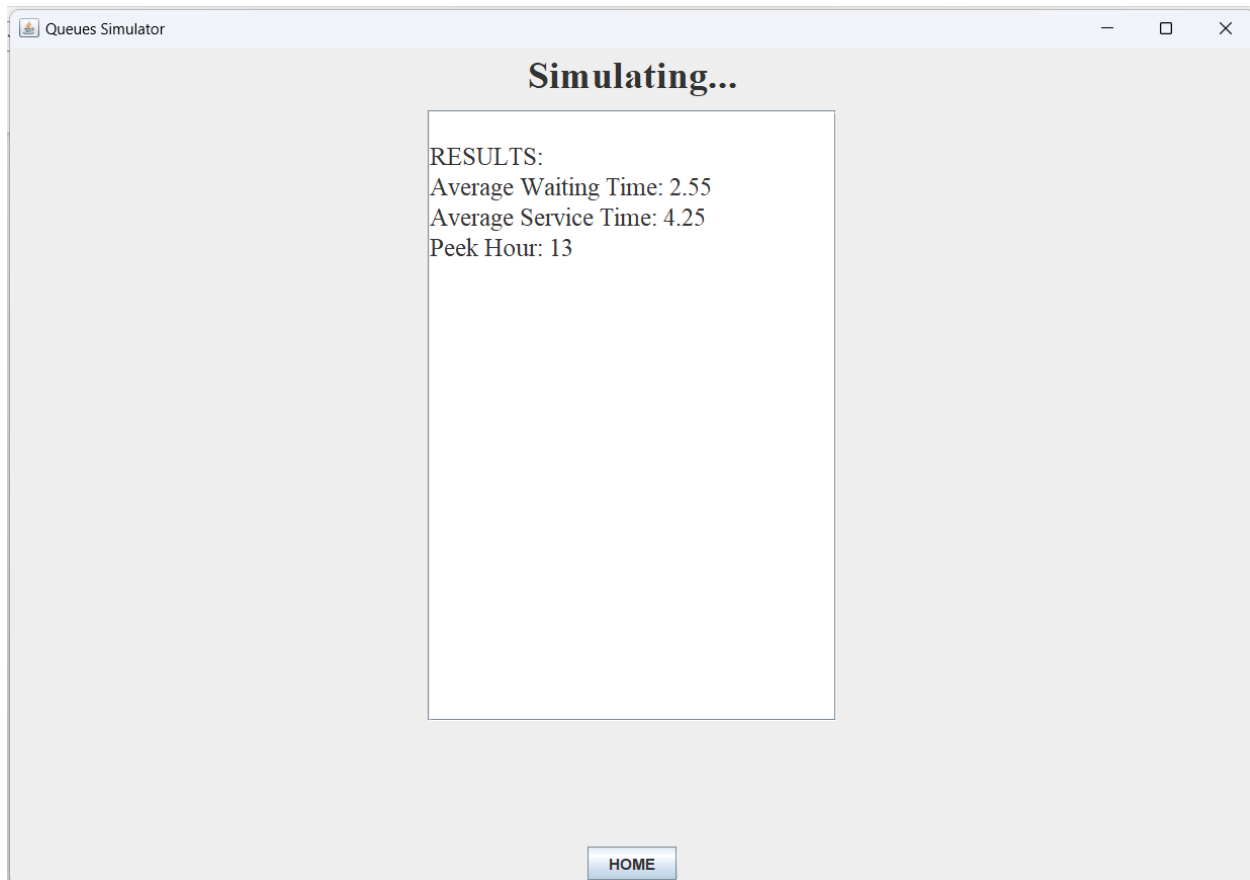
## Graphical user interface:



The screenshot displays a graphical user interface for a 'Queues Simulator'. The window has a title bar with the text 'Queues Simulator' and standard window control buttons (minimize, maximize, close). The main content area is titled 'Simulator Setting' in a large, bold font. Below the title, there are seven input fields for configuring the simulation parameters, each preceded by a label. The labels are: 'Number of Clients:', 'Number of Queues:', 'Simulation Time:', 'Minimum Arrival Time:', 'Maximum Arrival Time:', 'Minimum Service Time:', and 'Maximum Service Time:'. Each label is followed by a white rectangular input box. To the right of the 'Number of Clients' input box, there is a blue button with the text 'START' in white capital letters.

Parameter	Input Field
Number of Clients:	<input type="text"/>
Number of Queues:	<input type="text"/>
Simulation Time:	<input type="text"/>
Minimum Arrival Time:	<input type="text"/>
Maximum Arrival Time:	<input type="text"/>
Minimum Service Time:	<input type="text"/>
Maximum Service Time:	<input type="text"/>

START



## 4. IMPLEMENTATION

### ❖ *Task Class*

Represents the client of the application and contains three main fields: ID, Arrival Time and Service Time, and another Served Time field which helps us to calculate the waiting time of each task.

### ❖ *Server Class*

This is one of the most important parts of the application, it is also a thread type class implementing Runnable interface and contains a BlockingQueue which is a thread safe Java Collection, an ID, a status, finish time to account the finish time of the current running task and an atomic Integer which is a thread safe type in Java (all the operations on this variable will happen instantly). In the run() method checks if there are any tasks to compute and if they are takes out the head of the queue, sets the served time of the tasks and the finish time then sleeps the amount of time contained in the task. At every step check if there are anymore tasks to compute and sets the status of its own.

It also implements a method which computes the waiting period.



```

@Override
public void run() {
    while (SimulationManager.getSimulationInterval() > SimulationManager.getCurrentTime()) {
        try {
            if (!tasks.isEmpty()) {
                Task task = tasks.element();
                task.setServedTime(SimulationManager.getCurrentTime());
                finishTime = task.getServiceTime() + SimulationManager.getCurrentTime();
                computeWaitingPeriod();
                sleep( millis: task.getServiceTime() * 1000L);
                tasks.remove();
                if (tasks.isEmpty()) {
                    status = false;
                }
                finishTime = 0;
            }
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
}

```

### ❖ *SimulationManager Class*

This is the brain of the application, it accounts the simulation time, generates and dispatch tasks, holds the scheduler and also computes the average waiting time. It contains the fields: scheduler, a list of tasks, number of clients, simulation interval, arrival range interval, service range interval and the current time.

In run() method it initializes the logger then proceeds to iterate through all the generated tasks and dispatch them at the right moment and logs at every second then sleeps for 1 second. At the end the manager computes the results sets and logs it.

```

public void generateRandomTasks() {
    Random random = new Random();
    for (int i = 0; i < numberOfClients; i++) {
        int arrivalTime = random.nextInt( bound: (maximumArrivalTime + 1) - minimumArrivalTime) + minimumArrivalTime;
        int serviceTime = random.nextInt( bound: (maximumServiceTime + 1) - minimumServiceTime) + minimumServiceTime;
        tasks.add(new Task( ID: i + 1, arrivalTime, serviceTime));
    }
}

```

### ❖ *Scheduler Class*

As its name says it schedules the dispatched task and through the hold strategy assigns the task to a server. It also generates the servers and starts the threads and generates logs.

The check Load() method checks all the servers queued tasks and accounts the peek hour.

```

public void generateServers(int numberOfQueues) {
    for (int i = 0; i < numberOfQueues; i++) {
        servers.add(new Server( serverID: i + 1));
    }
    for (Server server : servers) {
        (new Thread(server)).start();
    }
}

```

❖ *Strategy Interface*

It defines the strategy addTask() method and helps future implementation of another strategies.

❖ *TimeStrategy Class*

Implements a strategy, as its name says, based on time which iterates through all servers. and checks which has the least waiting time and adds the task there.

## 5. RESULTS

Besides the manual testing I introduced three inputted data sets and saved their logs.

One test has 2 queues which process 4 tasks in 60 seconds with a arrival time range of [2, 30] and service time range of [2, 4].

The second test has 5 queues which process 50 tasks in 60 seconds with a arrival time range of [2, 40] and service time range of [1, 7].

The last test has 20 queues which process 1000 tasks in 200 seconds with a arrival time range of [10, 100] and service time range of [3, 9].

Here is the first test:

Apr 26, 2023 11:39:05 PM logic.SimulationManager run

INFO:

Time: 0

Waiting Clients: (1, 13, 4); (2, 9, 3); (3, 20, 3); (4, 14, 4);

Queue 1: closed.

Queue 2: closed.

Apr 26, 2023 11:39:06 PM logic.SimulationManager run

INFO:

Time: 1

Waiting Clients: (1, 13, 4); (2, 9, 3); (3, 20, 3); (4, 14, 4);

Queue 1: closed

Queue 2: closed

Apr 26, 2023 11:39:07 PM logic.SimulationManager run

INFO:

Time: 2

Waiting Clients: (1, 13, 4); (2, 9, 3); (3, 20, 3); (4, 14, 4);

Queue 1: closed

Queue 2: closed

Apr 26, 2023 11:39:08 PM logic.SimulationManager run  
INFO:  
Time: 3  
Waiting Clients: (1, 13, 4); (2, 9, 3); (3, 20, 3); (4, 14, 4);  
Queue 1: closed  
Queue 2: closed

Apr 26, 2023 11:39:09 PM logic.SimulationManager run  
INFO:  
Time: 4  
Waiting Clients: (1, 13, 4); (2, 9, 3); (3, 20, 3); (4, 14, 4);  
Queue 1: closed  
Queue 2: closed

Apr 26, 2023 11:39:10 PM logic.SimulationManager run  
INFO:  
Time: 5  
Waiting Clients: (1, 13, 4); (2, 9, 3); (3, 20, 3); (4, 14, 4);  
Queue 1: closed  
Queue 2: closed

Apr 26, 2023 11:39:11 PM logic.SimulationManager run  
INFO:  
Time: 6  
Waiting Clients: (1, 13, 4); (2, 9, 3); (3, 20, 3); (4, 14, 4);  
Queue 1: closed  
Queue 2: closed

Apr 26, 2023 11:39:12 PM logic.SimulationManager run  
INFO:  
Time: 7  
Waiting Clients: (1, 13, 4); (2, 9, 3); (3, 20, 3); (4, 14, 4);  
Queue 1: closed  
Queue 2: closed

Apr 26, 2023 11:39:13 PM logic.SimulationManager run  
INFO:  
Time: 8  
Waiting Clients: (1, 13, 4); (2, 9, 3); (3, 20, 3); (4, 14, 4);  
Queue 1: closed  
Queue 2: closed

Apr 26, 2023 11:39:14 PM logic.SimulationManager run  
INFO:  
Time: 9  
Waiting Clients: (1, 13, 4); (3, 20, 3); (4, 14, 4);  
Queue 1: (2, 9, 3);  
Queue 2: closed

Apr 26, 2023 11:39:15 PM logic.SimulationManager run  
INFO:  
Time: 10  
Waiting Clients: (1, 13, 4); (3, 20, 3); (4, 14, 4);  
Queue 1: (2, 9, 3);  
Queue 2: closed

Apr 26, 2023 11:39:16 PM logic.SimulationManager run  
INFO:  
Time: 11  
Waiting Clients: (1, 13, 4); (3, 20, 3); (4, 14, 4);  
Queue 1: (2, 9, 3);  
Queue 2: closed

Apr 26, 2023 11:39:17 PM logic.SimulationManager run  
INFO:  
Time: 12  
Waiting Clients: (1, 13, 4); (3, 20, 3); (4, 14, 4);  
Queue 1:  
Queue 2: closed

Apr 26, 2023 11:39:18 PM logic.SimulationManager run  
INFO:  
Time: 13  
Waiting Clients: (3, 20, 3); (4, 14, 4);  
Queue 1: closed  
Queue 2: (1, 13, 4);

Apr 26, 2023 11:39:19 PM logic.SimulationManager run  
INFO:  
Time: 14  
Waiting Clients: (3, 20, 3);  
Queue 1: (4, 14, 4);  
Queue 2: (1, 13, 4);

Apr 26, 2023 11:39:20 PM logic.SimulationManager run  
INFO:  
Time: 15  
Waiting Clients: (3, 20, 3);  
Queue 1: (4, 14, 4);  
Queue 2: (1, 13, 4);

Apr 26, 2023 11:39:21 PM logic.SimulationManager run  
INFO:  
Time: 16  
Waiting Clients: (3, 20, 3);  
Queue 1: (4, 14, 4);  
Queue 2: (1, 13, 4);

Apr 26, 2023 11:39:22 PM logic.SimulationManager run  
INFO:  
Time: 17  
Waiting Clients: (3, 20, 3);  
Queue 1: (4, 14, 4);  
Queue 2: closed

Apr 26, 2023 11:39:23 PM logic.SimulationManager run  
INFO:  
Time: 18  
Waiting Clients: (3, 20, 3);  
Queue 1: closed  
Queue 2: closed

Apr 26, 2023 11:39:24 PM logic.SimulationManager run  
INFO:  
Time: 19  
Waiting Clients: (3, 20, 3);  
Queue 1: closed  
Queue 2: closed

Apr 26, 2023 11:39:25 PM logic.SimulationManager run  
INFO:  
Time: 20  
Waiting Clients:  
Queue 1: closed  
Queue 2: (3, 20, 3);

Apr 26, 2023 11:39:26 PM logic.SimulationManager run  
INFO:  
Time: 21  
Waiting Clients:  
Queue 1: closed  
Queue 2: (3, 20, 3);

Apr 26, 2023 11:39:27 PM logic.SimulationManager run  
INFO:  
Time: 22  
Waiting Clients:  
Queue 1: closed  
Queue 2: (3, 20, 3);

Apr 26, 2023 11:39:28 PM logic.SimulationManager run  
INFO:  
Time: 23  
Waiting Clients:  
Queue 1: closed  
Queue 2: closed

Apr 26, 2023 11:39:28 PM logic.SimulationManager run  
INFO:  
RESULTS:  
Average Waiting Time: 0.0  
Average Service Time: 3.5  
Peak Hour: 14

## 6. CONCLUSIONS

- Conclusion:
  - The application is functional and can perform any required simulation on any desire input.
- Learned skills:
  - I delved into Java Threads and learned a new type of software architecture. I upskilled my abilities to organize my code.
- Future developments:

- Possible features to implement: A new strategy, a better interface and more interactive.

## 7. BIBLIOGRAPHY

1. *Regex:* [Tutorials List - Javatpoint](#)
2. [Stack Overflow - Where Developers Learn, Share, & Build Careers](#)
3. [diagrams.net](#)