DOCUMENTATION

ASSIGNMENT *2*

STUDENT NAME: Dincă Diana
GROUP: 30223

CONTENTS

1. Assignment Objective

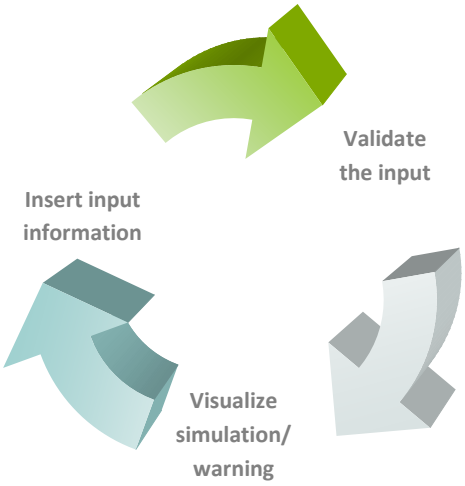
The main objective of the assignment is to design and implement a queues management application which assigns clients to queues such that the waiting time is minimized.

To develop this project, I pursued the following sub-objectives:

- Analyze the problem and identify requirements:
  The application allows users to select the input values for the number of clients, number of queues, the simulation interval, the arrival interval and the service time. For the result to be valid, the input must be valid. (detailed at point "2. Problem Analysis, Modeling, Scenarios, Use Cases")
- Design the simulation application:
  The simulation design must be intuitive and aesthetically pleasing. (detailed at point "3. Design")
- Implement the simulation application:
  The development of the code must be well-organized into packages and classes. (detailed at point "4. Implementation")
- Test the simulation application:
  For proper functionality, every function of the calculator must be verified with examples. (detailed at point "5. Results")

2. Problem Analysis, Modeling, Scenarios, Use Cases

To model the system, we can identify the main entities: Clients and Queues. Clients arrive at the system, enter queues, wait, are served, and then leave. In this system, the main objective is to minimize the waiting time for clients before they are served.

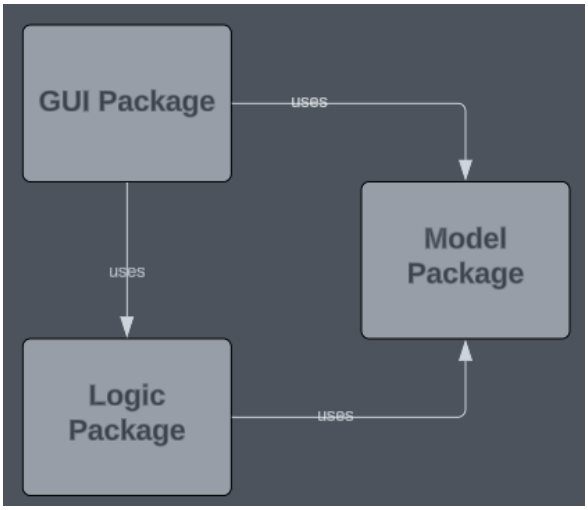


Therefore, efficient management of queues is crucial to balancing service quality and cost-effectiveness. This can be achieved by efficiently assigning clients to queues based on their arrival times and service durations.

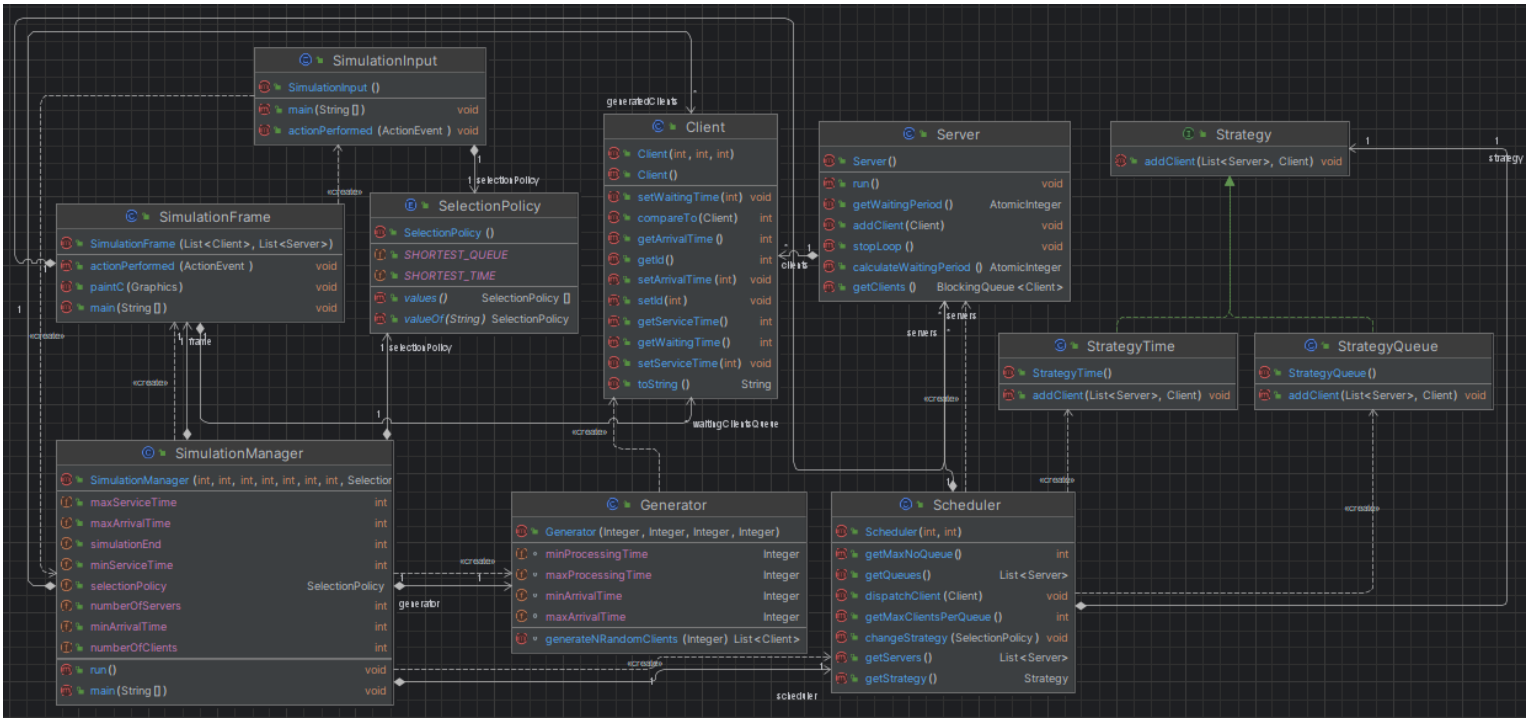
To address the problem, the application input must be validated and then the simulation can begin. In the end, the user will be provided with additional data such as: average waiting time, average service time and peek hour for the clients simulated.

3. Design

Package Diagram:



Classes Diagram:



The application's implementation is organized into four distinct packages:

- *gui package*: this package houses the *SimulationInput* class, which manages the graphical user interface for inputting the data, and *SimulationFrame* class, which displays the dynamic simulation;
- *model package*: within this package, you'll find the *Client* and *Server* classes, essential for representing and manipulating the clients and queues;
- *logic package*: this package includes the *Strategy* interface, *StrategyQueue* and *StrategyTime* classes, which implements Strategy, *SelectionPolicy* enumeration, *Generator* class, *Scheduler* class and *SimulationManager* class, providing the necessary

methods for generating clients and dispatching them into the right queues, based on the strategy selected;

For an improved functionality, I used a BlockingQueue<Client> for storing the list of clients on each queue. The clients contain an id, their arrival time at the queue, their service time and the waiting time.

4. Implementation

Clients Class- stores the data for each Client.

- *compareTo* method helps comparing two objects based on their arrival time;
- *toString* method helps printing the Client information;

Server Class- uses Client Class to store a list of clients waiting to be served in a specific.

- *addClient* method adds a client to a queue;
- *calculateWaitingPeriod* method calculates the waiting period after each client was added in the queue;
- *run* method manages the thread that allows the Client to stay in its queue exactly how long its service time is;

Generator Class- creates a list of random clients with their service and arrival time generated in the right interval, inputted at the beginning of the application.

StrategyQueue Class- implements Strategy and overrides the method *addClient*, which assigns the Client to a queue based on the number of clients that are positioned at that queue (how long the queue is).

StrategyTime Class- implements Strategy and overrides the method *addClient*, which assigns the Client to a queue based on the waiting period (how much time the queue takes).

Scheduler Class- contains a list of servers, the number of queues, the strategy chosen and is able to create and start a thread for each queue.

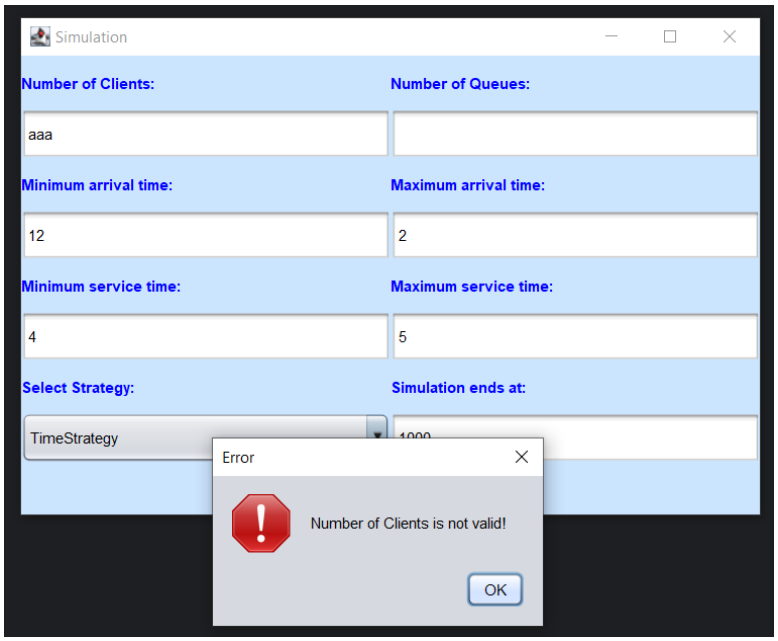
SimulationManager Class- holds the valid data inputted by the user and runs the main thread

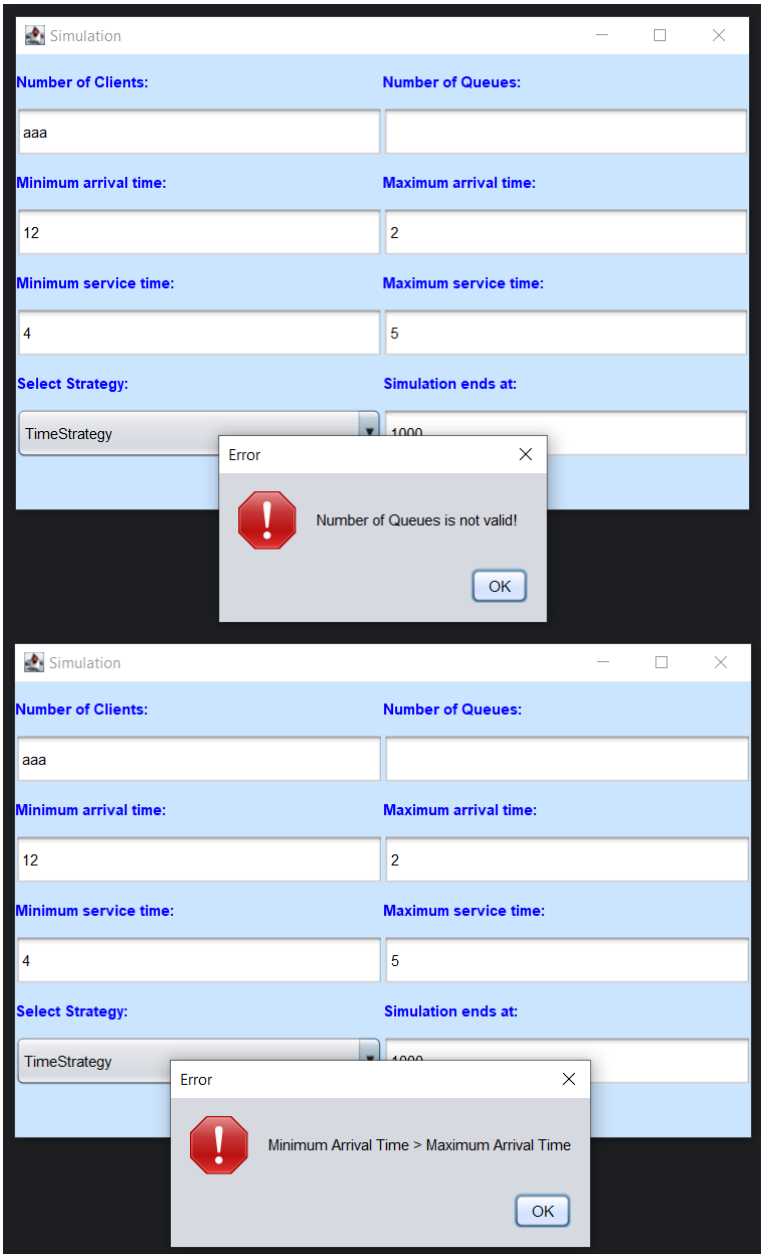
- *SimulationManager* method is the constructor of the class and initializes all the variables with the input provided, also it creates a new *SimulationFrame* and it calls the *Generator* class to get a list of random generated clients;
- *Run* method dispatches the clients to the queues, based on the chosen strategy, it updates the simulation frame and it writes in the log file, the progress made while simulating;

SimulationInput- represents the graphical user interface that helps the user to interact with the represents the application.

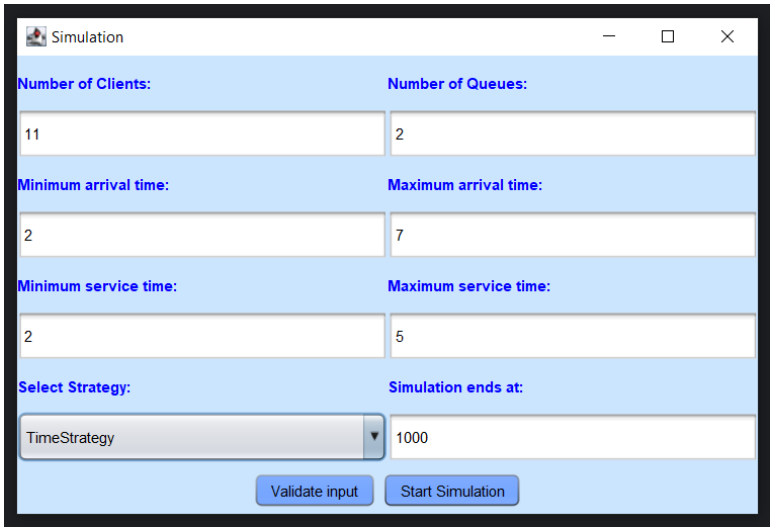
- It shows seven text fields where the users must input the numbers he desires, a combo-box which allows the user to pick the strategy and a button to validate the input.

- If any input field is empty, contains other symbols other than numbers, negative numbers or the minimum time is bigger or equal to the maximum time, there will be warnings.

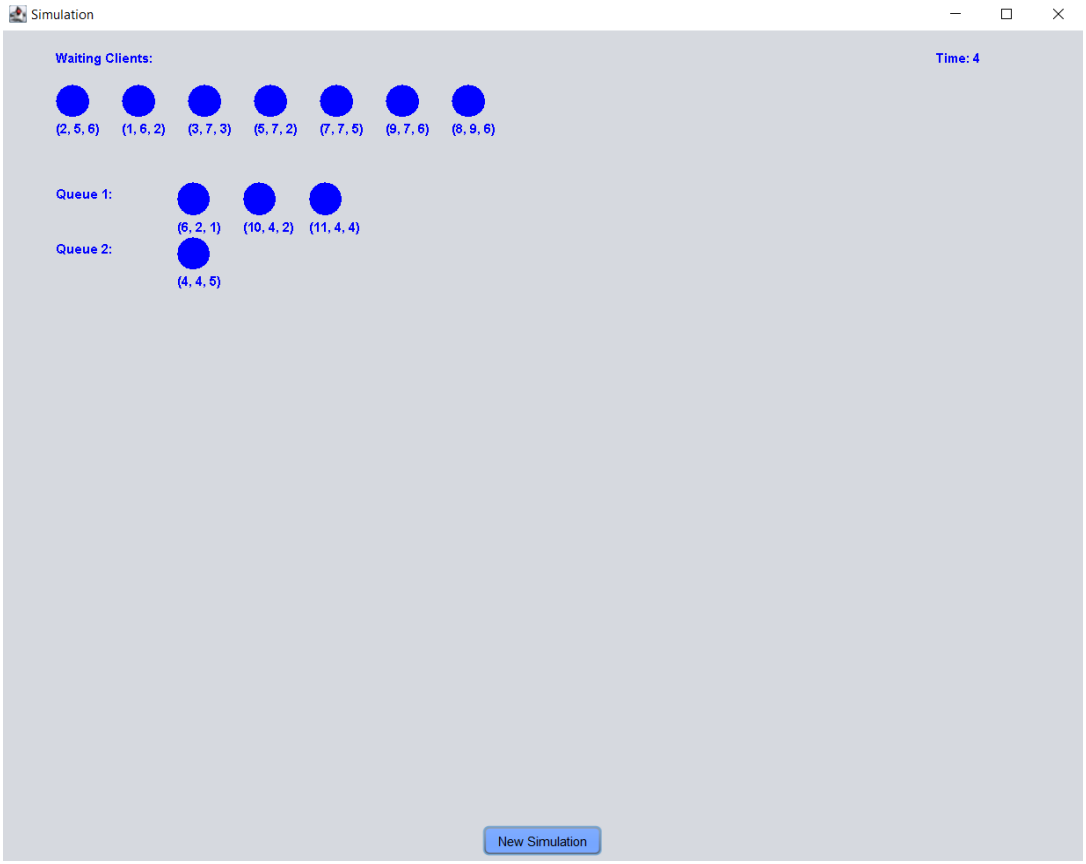




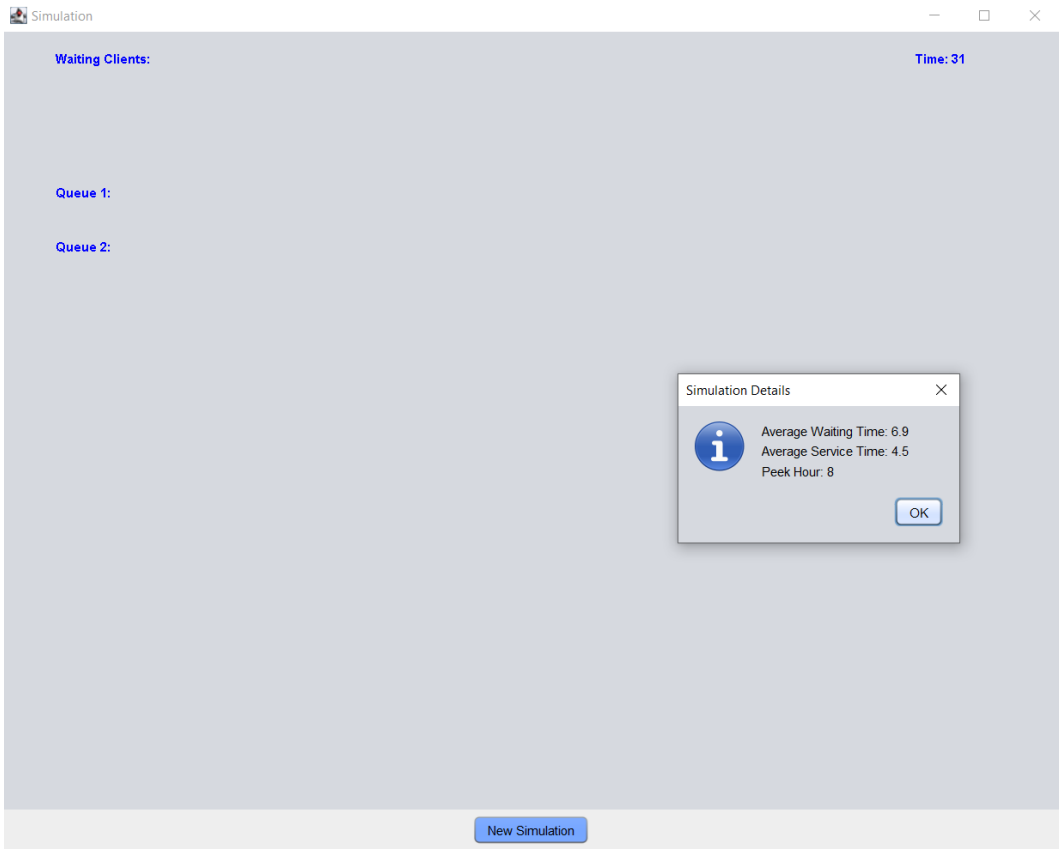
- After a valid set of data was introduced, another button appears, which allows the user to Start the Simulation.



SimulationFrame- represents the graphical user interface that shows a dynamic simulation, where blue circles represent the Clients waiting in line.



- After the simulation has ended, an additional page will appear and show the user the Average Waiting Time, the Average Service Time and the Peek hour.



5. Results

I have conducted many tests: first, I tested the validation functionality and performance by introducing invalid data. Secondly, I tested various simulation cases, calculating the expected output manually and comparing it with the output generated by the application.

Additionally, I've conducted the three main tests presented in the assignment and added them in *logOfEvents.txt* files:

Test 1	Test 2	Test 3
N = 4 Q = 2 $t^{MAX}_{simulation}$ = 60 seconds $[t^{MIN}_{arrival}, t^{MAX}_{arrival}] = [2, 30]$ $[t^{MIN}_{service}, t^{MAX}_{service}] = [2, 4]$	N = 50 Q = 5 $t^{MAX}_{simulation}$ = 60 seconds $[t^{MIN}_{arrival}, t^{MAX}_{arrival}] = [2, 40]$ $[t^{MIN}_{service}, t^{MAX}_{service}] = [1, 7]$	N = 1000 Q = 20 $t^{MAX}_{simulation}$ = 200 seconds $[t^{MIN}_{arrival}, t^{MAX}_{arrival}] = [10, 100]$ $[t^{MIN}_{service}, t^{MAX}_{service}] = [3, 9]$

logOfEventsTest1.txt logOfEventsTest2.txt logOfEventsTest3.txt

logOfEventsTest1.txt

```
Queue 2: Closed

Time: 25
Waiting clients: (4, 27, 2);
Queue 1: Closed
Queue 2: Closed

Time: 26
Waiting clients: (4, 27, 2);
Queue 1: Closed
Queue 2: Closed

Time: 27
Waiting clients:
Queue 1: (4, 27, 2);
Queue 2: Closed

Time: 28
Waiting clients:
Queue 1: (4, 27, 1);
Queue 2: Closed

Time: 29
Waiting clients:
Queue 1: Closed
Queue 2: Closed

Average Waiting Time: 0.0
Average Service Time: 2.0
Peek Hour: 16
```

logOfEventsTest2.txt

```
Queue 2: (13, 38, 2);
Queue 3: Closed
Queue 4: Closed
Queue 5: (29, 39, 3);

Time: 46
Waiting clients:
Queue 1: Closed
Queue 2: (13, 38, 1);
Queue 3: Closed
Queue 4: Closed
Queue 5: (29, 39, 2);

Time: 47
Waiting clients:
Queue 1: Closed
Queue 2: Closed
Queue 3: Closed
Queue 4: Closed
Queue 5: (29, 39, 1);

Time: 48
Waiting clients:
Queue 1: Closed
Queue 2: Closed
Queue 3: Closed
Queue 4: Closed
Queue 5: Closed

Average Waiting Time: 2.5
Average Service Time: 3.66
Peek Hour: 32
```

logOfEventsTest3.txt

```
Queue 17: (877, 76, 5); (742, 78
Queue 18: (879, 76, 5); (753, 78
Queue 19: (350, 75, 2); (66, 78,
Queue 20: (30, 76, 4); (467, 78,

Time: 200
Waiting clients:
Queue 1: (1, 77, 8); (704, 79, 8
Queue 2: (139, 75, 2); (231, 78,
Queue 3: (329, 76, 4); (573, 78,
Queue 4: (19, 77, 7); (93, 79, 6
Queue 5: (71, 77, 3); (262, 78,
Queue 6: (466, 76, 3); (272, 78,
Queue 7: (121, 77, 7); (99, 79,
Queue 8: (472, 76, 1); (643, 77,
Queue 9: (278, 77, 4); (648, 78,
Queue 10: (480, 74, 1); (43, 78,
Queue 11: (320, 77, 7); (387, 79
Queue 12: (709, 76, 5); (811, 78
Queue 13: (913, 76, 6); (965, 78
Queue 14: (372, 77, 5); (823, 78
Queue 15: (495, 77, 3); (275, 78
Queue 16: (768, 76, 3); (389, 78
Queue 17: (877, 76, 4); (742, 78
Queue 18: (879, 76, 4); (753, 78
Queue 19: (350, 75, 1); (66, 78,
Queue 20: (30, 76, 3); (467, 78,

Average Waiting Time: 83.637
Average Service Time: 5.34
Peek Hour: 99
```

In the end, all of my tests have passed successfully, indicating that the functionality is working as expected.

6. Conclusions

The development of a queue management application has highlighted the importance of efficient allocation of clients to queues to minimize waiting times.

I've gained insights into utilizing simulation techniques to model client arrivals and queue assignments, while learning to simulate and understand the dynamics of queue-based systems.

Future developments could focus on implementing priority queuing to serve clients based on urgency or importance, for example, VIP clients or those with urgent needs could be given priority access to queues. Additionally, exploring the integration of digital channels such as mobile apps or websites to provide clients with remote queue access and updates on their status would further elevate the customer experience.

7. Bibliography

- https://docs.oracle.com/javase/tutorial/essential/concurrency/index.html
- https://www.tutorialspoint.com/java/util/timer_schedule_period.htm
- https://www.javacodegeeks.com/2013/01/java-thread-pool-example-using-executors-and-threadpoolexecutor.html