DOCUMENTATION

ASSIGNMENT 2

STUDENT NAME: Persea Oana - Mihaela
GROUP: 30424

CONTENTS

DOCUMENTATION ……………………………………………………………………………………1

ASSIGNMENT 2 ……………………………………………………………………………………1

CONTENTS ……………………………………………………………………………………………2

1. Assignment Objective ……………………………………………………………………3
2. Problem Analysis, Modeling, Scenarios, Use Cases ………………………………4
3. Design ………………………………………………………………………………………7
4. Implementation …………………………………………………………………………9
5. Conclusions ………………………………………………………………………………10
6. Bibliography ………………………………………………………………………………13

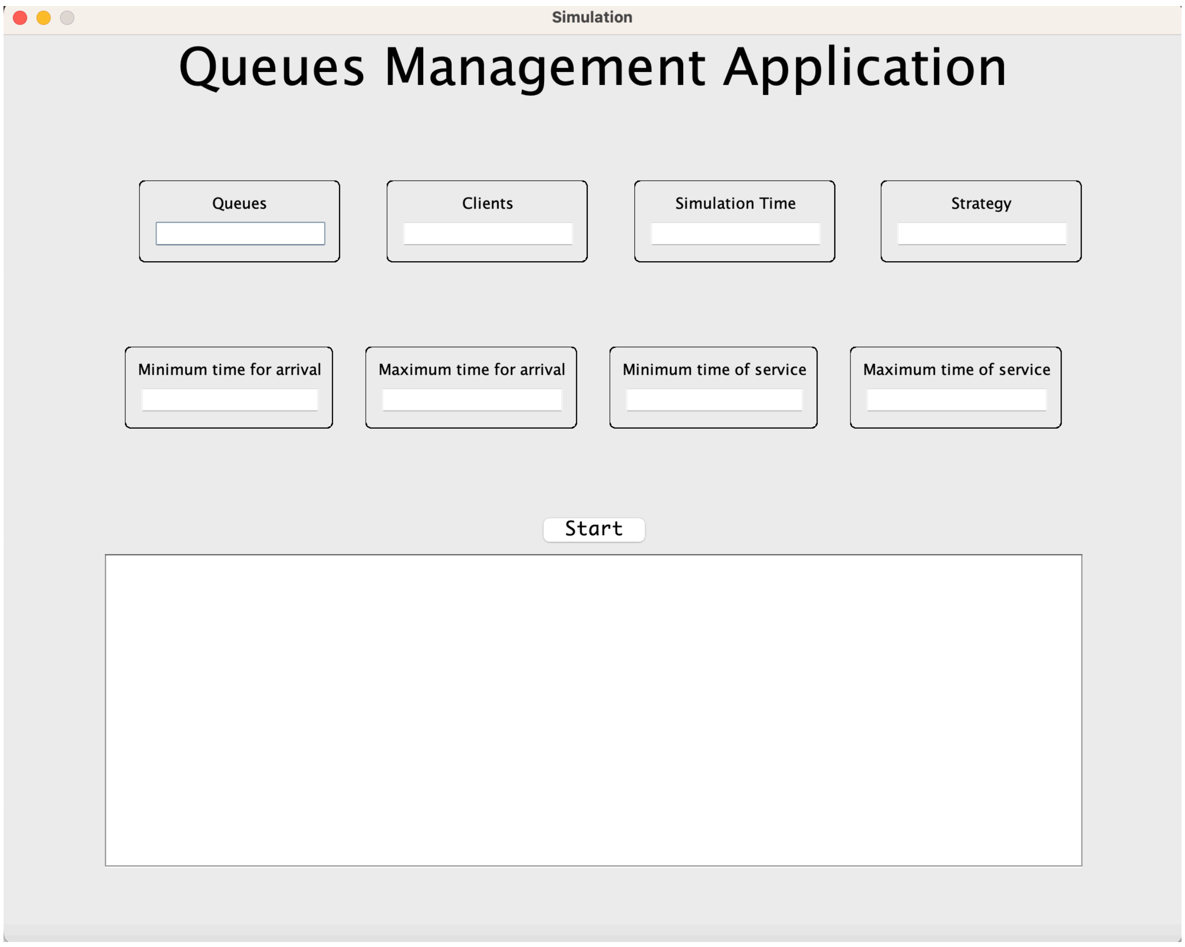
1. Assignment Objective

The main objective of the assignment is to efficiently manage and maintain queues of incoming requests from multiple clients, ensuring that all requests are handled in a fair and timely manner. The sub-objectives include:

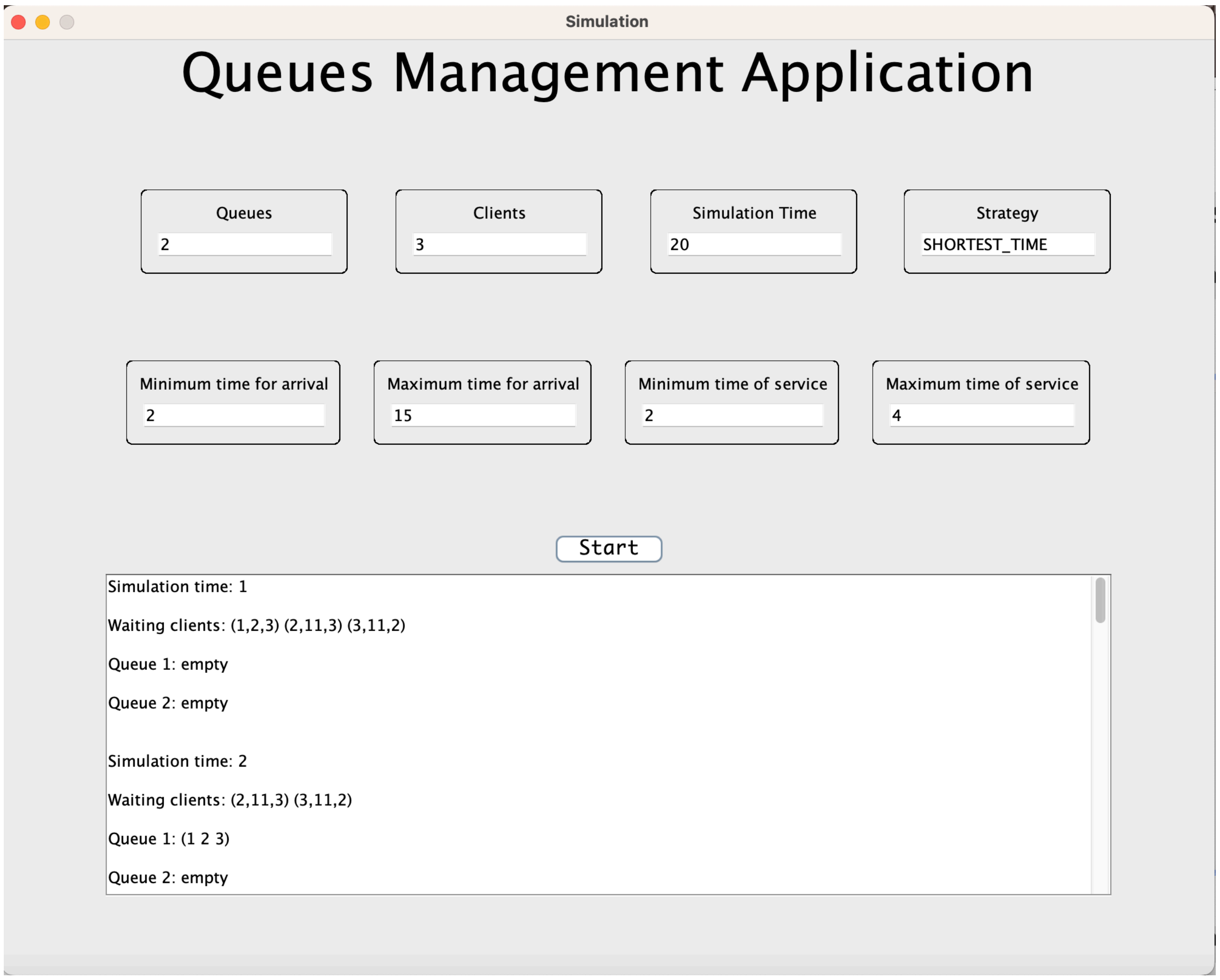
- understanding the requirements thoroughly
- making sure the application is able to receive requests from clients and add them to a queue
- handling requests in a timely manner
- prioritizing requests based on arrival time
- resolving conflicts between clients that arrive at the same time
- tracking the status of each client in the queue
- providing analytics: average waiting time, average service time and peak hour
- developing the user interface

2) Problem Analysis, Modeling, Scenarios, Use Cases

Firstly , the user is required to input the necessary data. In the strategy selection box, the user can opt for either the SHORTEST_TIME strategy, which prioritizes sending the client to the queue that will complete their tasks in the shortest amount of time, or the SHORTEST_QUEUE strategy, which directs the client to the queue with the fewest number of clients.



The application begins by creating clients. The user can monitor the real-time evolution of the queues. Every second, the application displays information on the clients still waiting and those who have been assigned to a queue

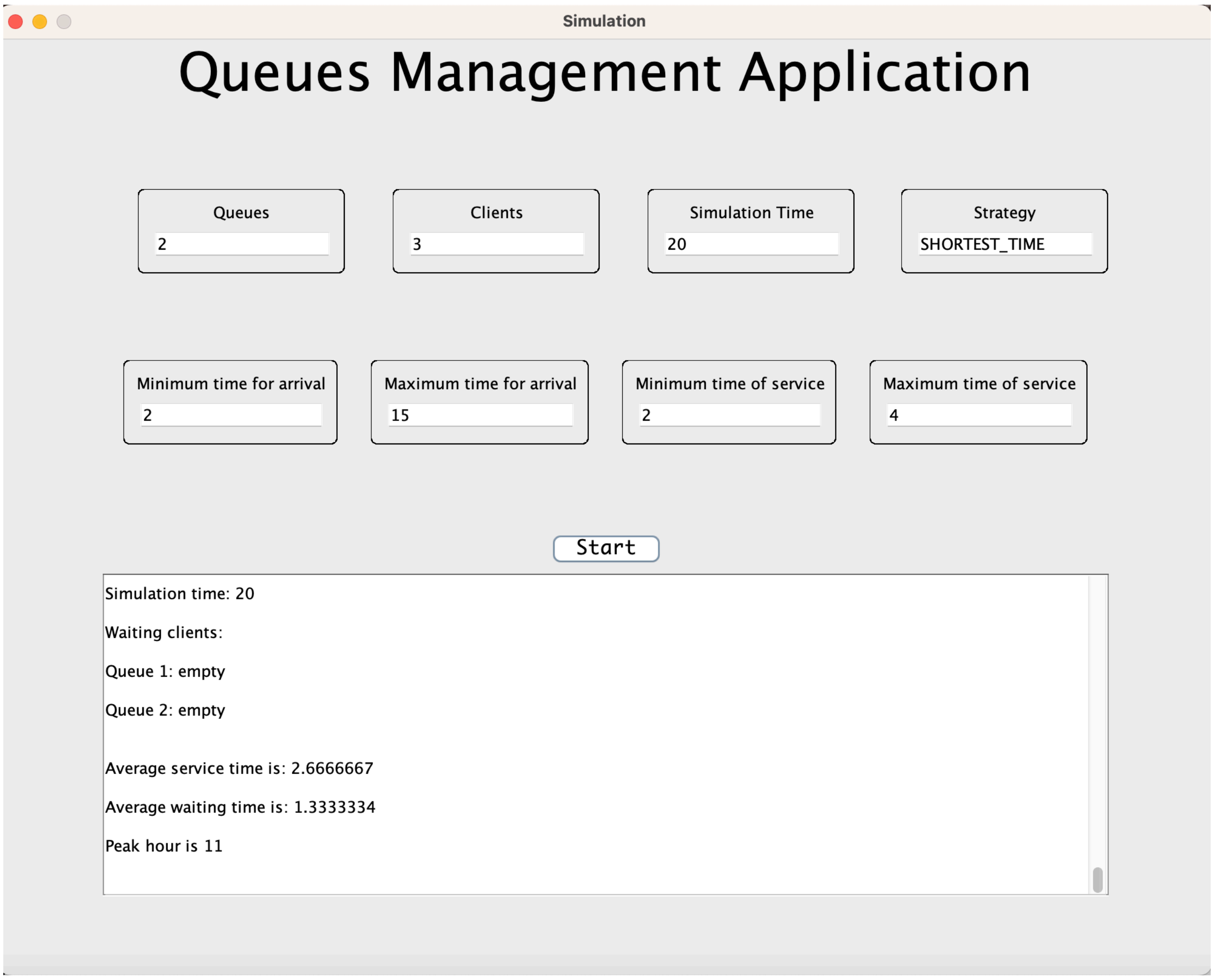


Queues Management Application

Queues: 2
Clients: 3
Simulation Time: 20
Strategy: SHORTEST_TIME
Minimum time for arrival: 2
Maximum time for arrival: 15
Minimum time of service: 2
Maximum time of service: 4

```
Simulation time: 1

Waiting clients: (1,2,3) (2,11,3) (3,11,2)

Queue 1: empty

Queue 2: empty


Simulation time: 2

Waiting clients: (2,11,3) (3,11,2)

Queue 1: (1 2 3)

Queue 2: empty
```

In the end, the application computes and displays the average waiting time, average service time, and peak hour. This information provides valuable insights into the performance of the queue system and can be used to optimize the system's efficiency in the future.

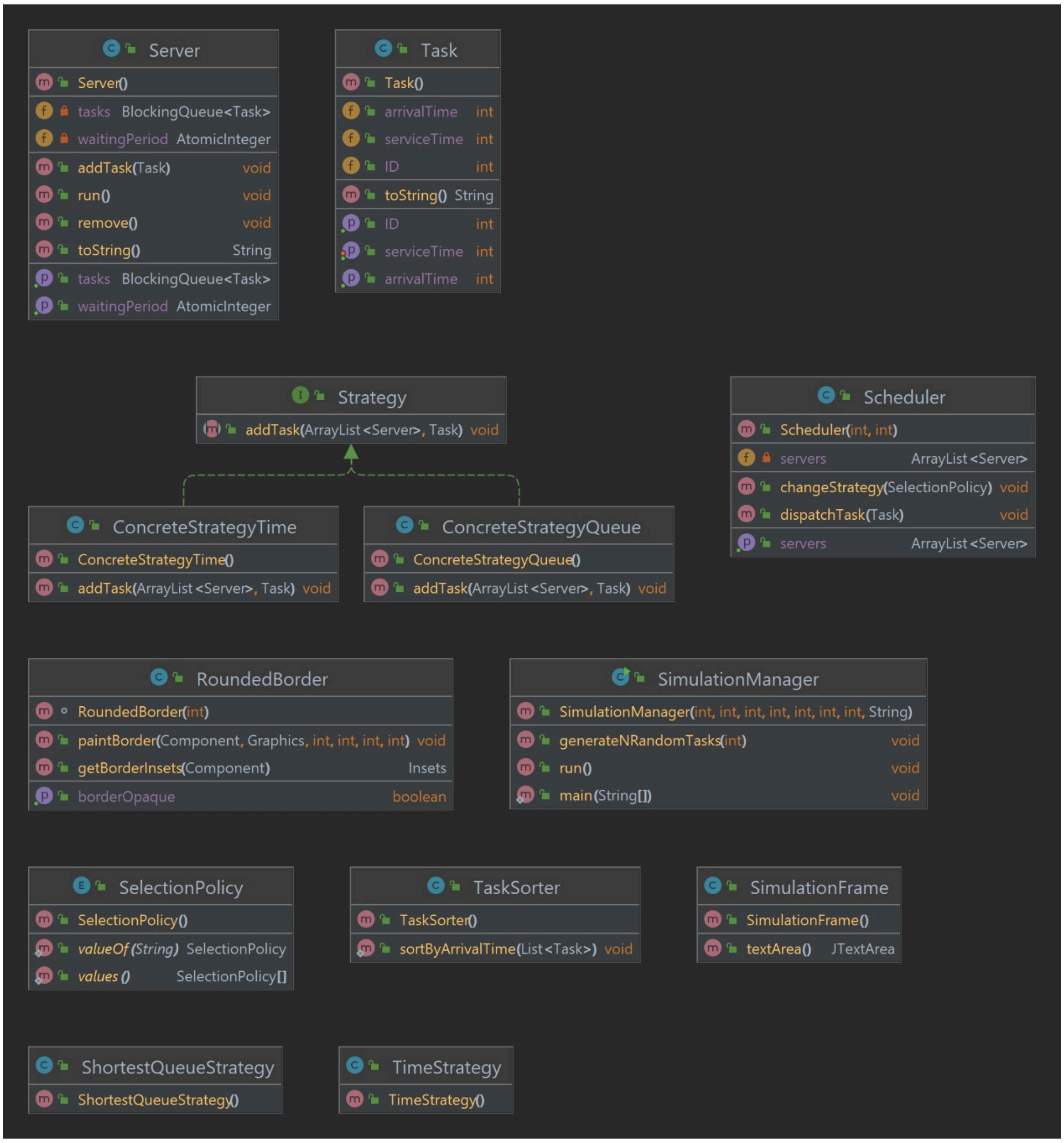


Queues: 2
Clients: 3
Simulation Time: 20
Strategy: SHORTEST_TIME
Minimum time for arrival: 2
Maximum time for arrival: 15
Minimum time of service: 2
Maximum time of service: 4

```
Simulation time: 20
Waiting clients:
Queue 1: empty
Queue 2: empty

Average service time is: 2.6666667
Average waiting time is: 1.3333334
Peak hour is 11
```

3. Design

—UML Diagram—



—Data Structures—

This application uses both primitive data types like integers or double type, strings and more complex ones, like ArrayList. I chose to use these types of data structures because they are more efficient in the case of memory management. We also got rid of the problem of overflow because we do not have to specify the length (like in the case of simple arrays). I also use thread-safe data structures, like BlockingQueue.

4. Implementation

- The **Concrete Strategy Queue** and **Concrete Strategy Time** classes define the prioritization rules for client insertion into the queue, using a defined method for each.
- The **Scheduler** class is responsible for adding clients and queues, and implements necessary methods for the same.
- The **SimulationFrame** class is responsible for the graphical user interface, which serves as the primary way for users to interact with the system. This class is mostly composed of action listeners, which implement the desired behavior of objects in the interface.
- The **SimulationManager** class is the core of the system and is responsible for managing the simulation from start to finish. It is in charge of manipulating objects in memory, such as client objects in queues, and presenting real-time representations that can be easily understood by the user.
- The **Server** class represents the queues in the system, and implements all necessary methods for queue manipulation.
- The **Task** class represents the clients in the system and implements all necessary methods for client manipulation.

5. Conclusions

In the end, this project made me improve my knowledge regarding Object Oriented Programming and the ability to handle different data structures. After developing this application I came to the conclusion that it was harder to design than I was expecting, but this only made me search solutions to my problems and challenged me to get to the final result. I also learned how to implement a Graphical User Interface using Swing and how to deal with threads.

This program could be improved by incorporating additional features such as prioritization of clients based on various criteria, handling of queue overflow scenarios, and computation of more queue performance metrics. Additionally, it could support graphical visualization of queue status and integration with external systems such as databases and analytics tools. To ensure a seamless user experience, robust error handling mechanisms must be implemented to handle exceptions and prevent application crashes due to user errors. Also, the graphic user interface could be improved by making it more interactive.

6. Bibliography

1. Programming Techniques – Lectures of prof. Cristina Bianca Pop
2. stackoverflow.com
3. geeksforgeeks.org