DOCUMENTATION

ASSIGNMENT *2- Queue management system*

STUDENT NAME: Robotin George Cătălin

GROUP: 30422

# CONTENTS

[1. Assignment Objective 3](#_Toc128043139)

[2. Problem Analysis, Modeling, Scenarios, Use Cases 3](#_Toc128043140)

[3. Design 4](#_Toc128043141)

[4. Implementation 5](#_Toc128043142)

[5. Results 6](#_Toc128043143)

[6. Conclusions 7](#_Toc128043144)

[7. Bibliography 7](#_Toc128043145)

# Assignment Objective

*The main objective of the assignment is to design a java application to manage the synchronization of a number of queues, like we would have at a supermarket, making use of threads, such that each queue has its own thread, thus providing parallel execution and better performance.*

*Sub objectives include:*

* *Using an OOP design for the application*
* *Having a GUI to display the real time evolution of the queues*
* *Correct implementation of the 2 strategies for adding clients(tasks) to the queue*
* *Using thread safe data structures(collections), i.e. LinkedBlockingQueue()*

# Problem Analysis, Modeling, Scenarios, Use Cases

*In this case, the functional requirements are easy to understand: a user inputs the parameters of the simulation, and when they click the start sim button, the text area updates, showing the evolution of the queues in real time, until the maximum simulation time is reached. The application should provide thread safety.*

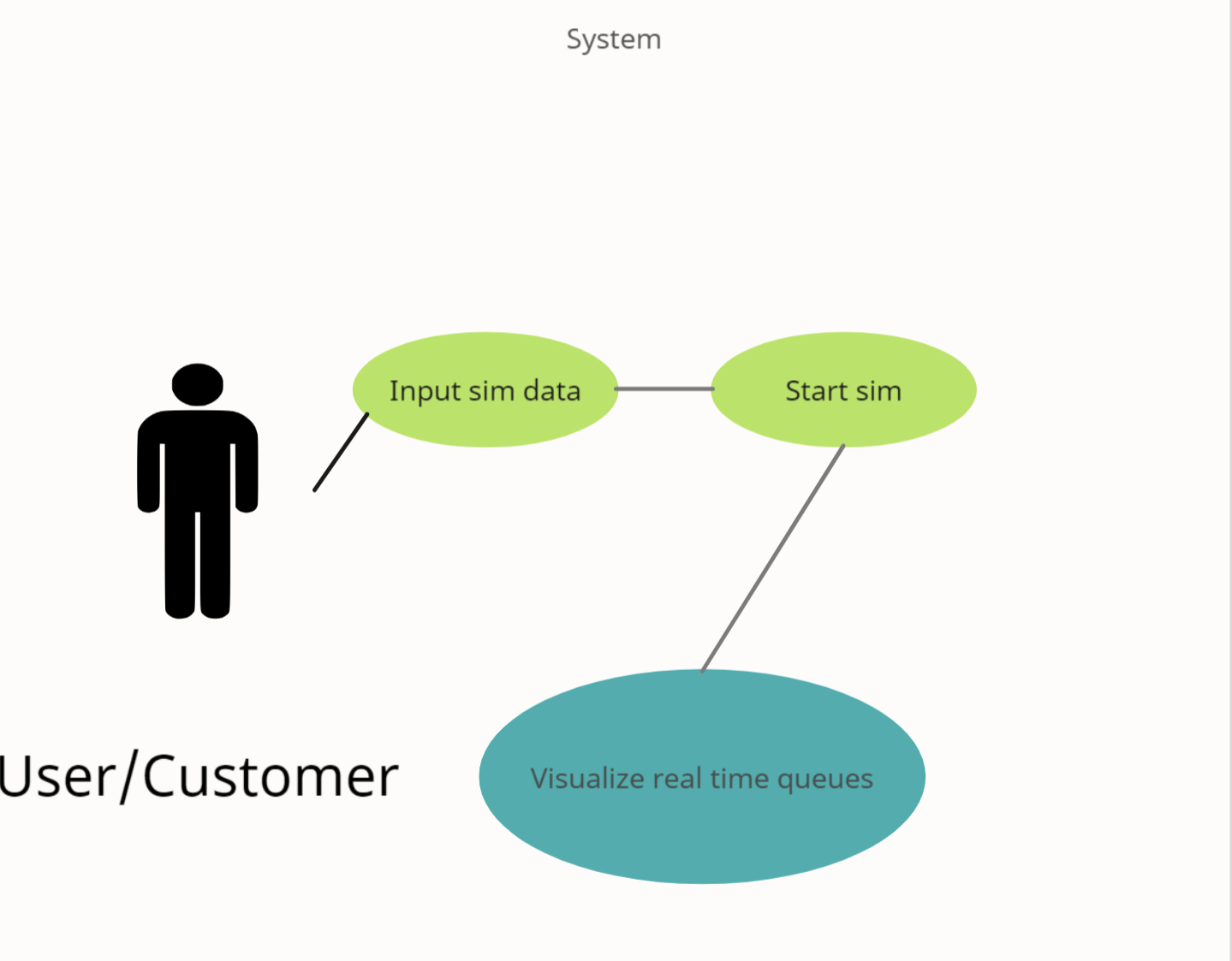


Figure 1:Use case diagram

*There is only one actor, the user who inputs the initial parameters for the simulation, who then presses the start sim button. Then, the app starts(the threads), and the real time evolution of the system can be observed on the text area in the middle of the UI.*

*Successful usage scenario:*

* *The user inputs valid data(positive integer numbers(>0), with minArrivalTime<=maxArrivalTime and minProcessingTime<=maxProcessingTime)*
* *The user presses the Start sim button*
* *Real time evolution is shown on the UI*
* *Statistics, such as the peak hour, average processing time and average waiting time are shown as a popup at the end*
* *Log file is generated, in order to be inspected after execution*

*Unsuccessful usage scenario:*

* *User inputs wrong data, either non numeric or non-logical*
* *Upon pressing the Start sim button, the operation fails and the GUI doesn’t update*

*The non-functional requirements include:*

* *Having a user-friendly GUI*
* *Performance, which is provided by the usage of multiple threads, one per queue.*
* *Safety, necessary when using threads*

# Design

*The design of the app abides the OOP principles, as it is split into multiple classes, which cooperate with each other, which, in turn are grouped into packages, based on their utility. The structure can be observed in the UML class and package diagrams.*

*In terms of the used data structures, they need to be thread safe, such that each queue runs on its own thread, and they are synchronized. For the queues themselves, I used the LinkedBlockingQueue1 data structure, from the BlockingQueue1 interface. For the waiting time, I used an AtomicInteger1. In order to use the concurrent collection, I chose to implement the Runnable interface and override the run() method. In the Server class, the run method makes the thread sleep after each element is taken out, and then decreases the waiting time. For the simulation manager, each moment in time is displayed, with the current state of the queues. They run concurrently (and synchronized), thanks to the BlockingQueue.*

*I defined one interface, namely the Strategy, with its 2 implementations: TimeStrategy and ShortestQueueStrategy, based on the desired way of introducing clients into the queues. The client enters in either the queue with the shortest waiting time, or the one with the least people in it.*

*The main management of the logic in the application is done in the SimulationManager class, which uses the Scheduler to synchronize and dispatch the tasks, such that they comply with the strategy and are executed concurrently.*

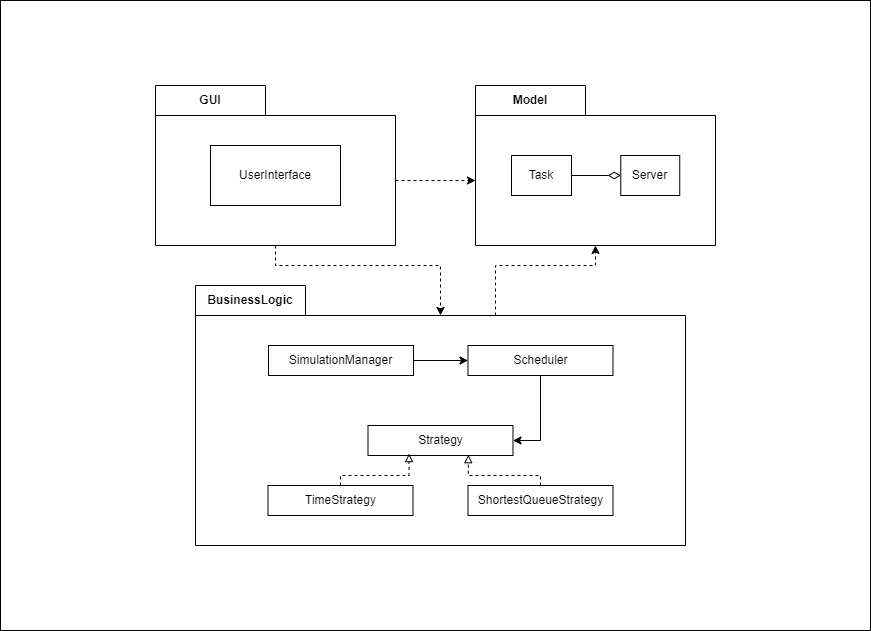


Figure 2:UML Package Diagram

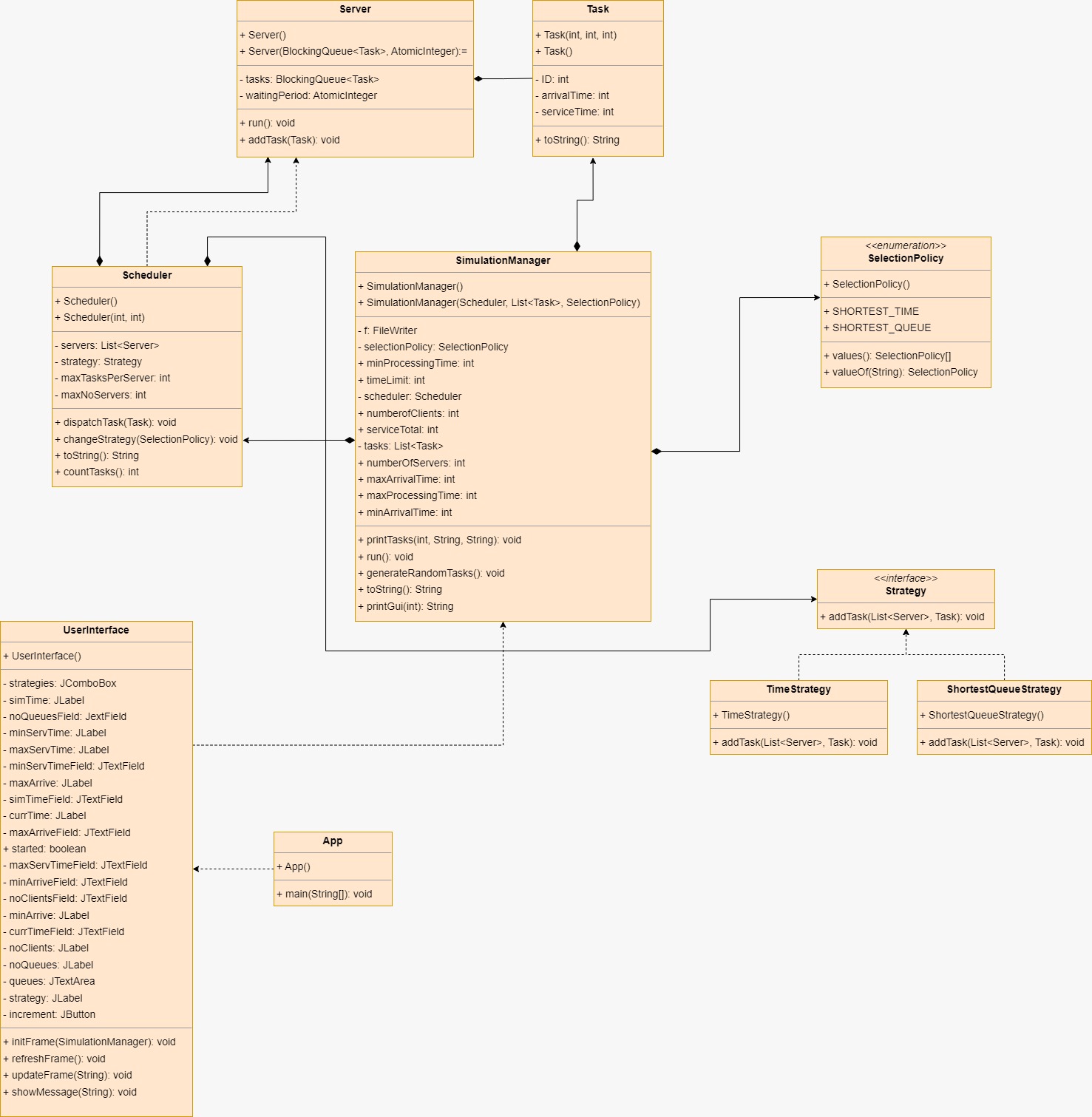


Figure 3:UML Class Diagram

# Implementation

* *Model package:*
  + *Task class:*
    - *ID, arrivalTime, serviceTime: fields used to describe the customers*
    - *Override toString(), for displaying*
  + *Server class:*
    - *tasks: BlockingQueue1, waitingTime: AtomicInteger: thread safe data structures, used to model one queue(Server)*
    - *override run() method, to add delay after each element is taken out of the queue*
* *Business\_logic package:*
  + *Strategy interface:*
    - *addTask() method: 2 implementations, based on the way the people should be added*
      * *ShortestQueueStrategy: uses getSize() method from Server*
      * *TimeStrategy: uses getWaitTime() method from Server*
  + *Scheduler class:*
    - *List<Server> servers: field to store all the queues*
    - *Strategy strategy: add tasks according to the strategy described by the selection policy(enum)*
    - *Initialize threads in constructor, based on the number of queues*
    - *dispatchTask() -to use the correct addTask() implementation, based on the selection policy*
    - *Helper methods, for the statistics at the end*
    - *Override toString(), for displaying each queue*
  + *SimulationManager class:*
    - *Takes the values from the ui and initializes the simulation process(in its constructor)*
    - *Generates n random tasks, sorted by arrival time*
    - *Overrides run and makes all the queues run synchronously*
    - *Helper functions for generating the logs and printing in the ui*
* *Gui package*
  + *UserInterface class:*
    - *Creates the SimulationManager and its first thread when the start sim button is pushed*
    - *Has text fields for all parameters that are input by the user*
    - *Has some static methods to provide the values of the text fields to the simulation manager*

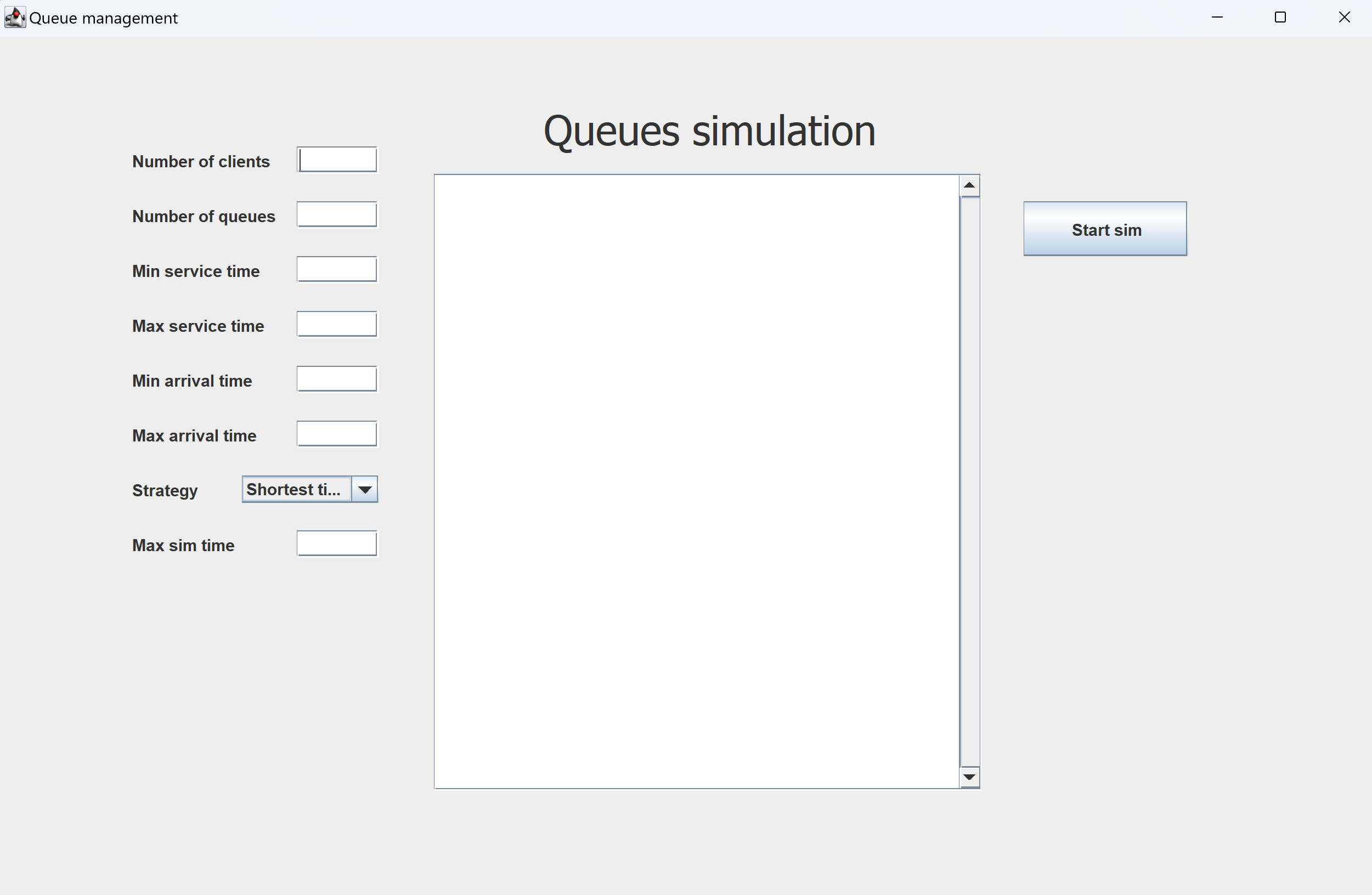


Figure 4:GUI

# Results

*For this assignment, there are 3 input cases provided for testing, for which I have appended the log files to the GitHub repository. The behavior can be observed at each moment in time, showing the state of each queue and the customers which haven’t yet arrived to a queue.*

# Conclusions

*Implementing this app, I learned about java threads5, i.e. how to use them, create them and synchronize them. I also learned about concurrent collections2, what thread safety is and how to implement it.*

*For future developments, some options include:*

* *Having a nicer looking GUI*
* *Asserting the input and showing messages when it is incorrect*
* *Adding more statistics*
* *More thorough testing*

# Bibliography

1. *Blocking queue example:* [*https://www.digitalocean.com/community/tutorials/java-blockingqueue-example*](https://www.digitalocean.com/community/tutorials/java-blockingqueue-example)
2. *For UML Class and Package diagrams:* [*https://drawio-app.com/*](https://drawio-app.com/)
3. *For use case diagram:* [*https://creately.com/diagram-type/use-case/*](https://creately.com/diagram-type/use-case/)
4. *General java language help:* [*https://www.baeldung.com/*](https://www.baeldung.com/)
5. *Java threads:* [*https://www.w3schools.com/java/java\_threads.asp*](https://www.w3schools.com/java/java_threads.asp)[*https://docs.oracle.com/javase/tutorial/essential/concurrency/runthread.html*](https://docs.oracle.com/javase/tutorial/essential/concurrency/runthread.html)