# DOCUMENTATION

Assignment nr 2

Dora Cristian Adrian
30422

Contents

# 1. Objectives

## 1.1. Main Objectives

Build an application that can simulate a real-world queue evolution. The application should be able to receive input from the user. The input will determine the total number of clients that would arrive in a time interval. The interval is also given from the user input alongside the maximum and minimum service time for each client. Other inputs are the maximum simulation time and the number of queues. After data validation, the input is manipulated by the app and in return, a graphic representation of the queues evolution in real time should be displayed. A log of events is also given in a text file. Clients should be generated randomly.

## 1.2. Secondary Objectives

- Analysing the given problem and finding the requirements for solving it
- Designing and the implementation of the queue management system
- Determine the peak hour, average waiting time and the average service time
- Designing a user-friendly graphical interface

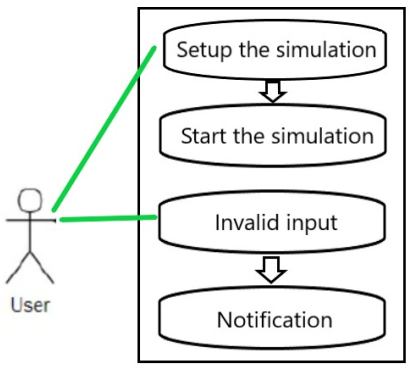
# 2. Functional and non-functional requirements

## 2.1. Functional requirements

### 2.1.1. MoSCoW classification

- Must have: Simulation of a queue in real time with a given number of clients, queues, and maximum simulation time. Saving the results in a log file. Queues working as separate threads.
- Should have: GUI that receives data from the user and uses it to process the simulation. Data should be validated. Real time simulation of the queue evolution. Calculation of peak hour, average waiting time and the average service time.
- Could have: Selection of desired strategy: shortest waiting time, lowest number of clients, etc.
- Won't have: Visualisation of clients that left the queues

### 2.1.2. Use case diagram



### 2.1.3. Use case scenarios

The user enters the data in the text areas. The values are then validated by the application. If the validation passed, the simulation starts with the given parameters.

The user enters non – numerical values in the text areas. A notification appears on the screen letting the user know which was the first wrong input.

The user enters numerical values, but the logic tests are not passed (e.g., the total simulation time is smaller than the maximum arrival time). A similar notification message appears and informs the user about the invalidity of the provided data.

## 2.2. Non-functional requirements

- Ease of use: the user interface should be simple and intuitive, with labels that indicate what the application is expecting from the user
- Compact design: the application's graphical interface should not have a greater size than necessary but should be big enough to display all the information that has to be shown

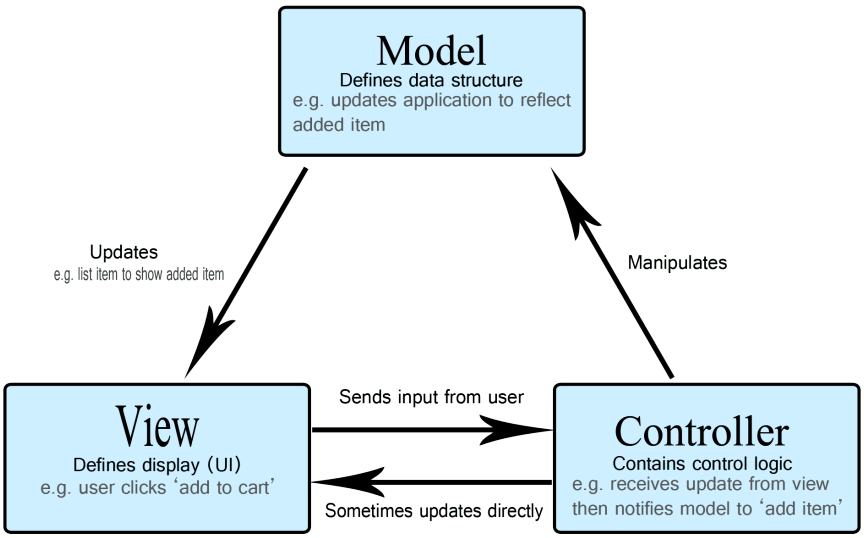
# 3. Conceptual architecture

The code is written in OOP style, following the 4 pillars of Object Oriented Programming paradigm: Encapsulation, Abstraction, Polymorphism, Inheritance. It also follows MVC design pattern.

Queues are represented as threads in the code. They run independently of each other, but they receive their clients from one common thread. This thread is responsible for keeping the current simulation time, updates and writing in the file.

# 4. Design and Implementation

## 4.1. Package diagram



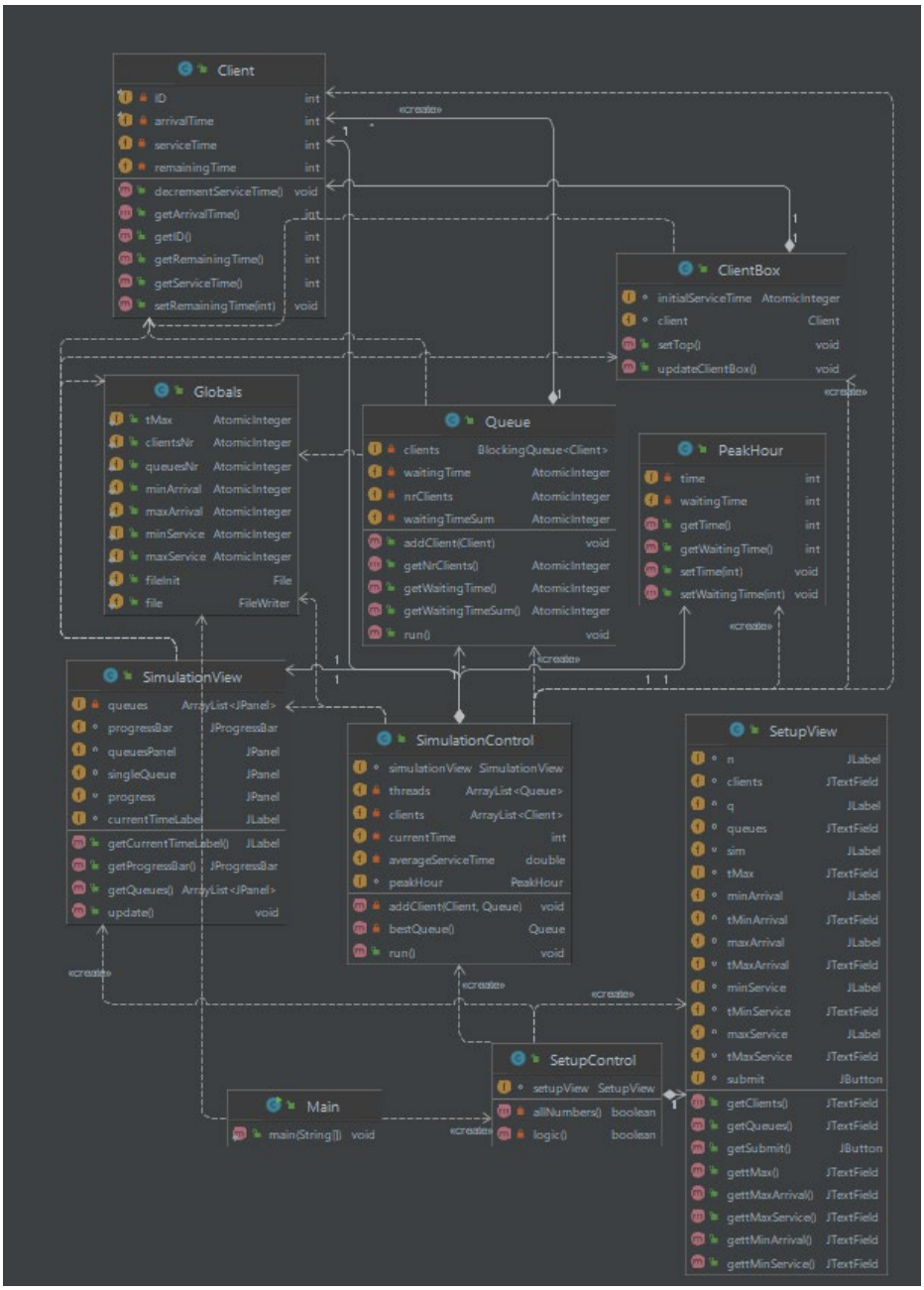

MVC (Model-View-Controller) is a design pattern that is very commonly used in the implementation of graphical user interfaces that include besides the graphical part also some data and controlling logic. Its emphasis is on separation between the view (display) and the business logic of the software. This "separation of concerns" provides a better labour division and makes maintenance easier.

The Model contains the classes that define data structures that will be used alongside some of their methods. Here are implemented the Queues, Clients, Peak hour, and some global variables used. The only logic in this part is the addition of clients in a queue.

The View contains classes that define what is displayed: Setup menu and Simulation window. The SimulationView class contains a method for updating data displayed. Also, the representation for the clients is implemented in this package

The Controller is the “glue” between the Model and the View. It contains most of the logic and controls when what and if changes should be displayed in the GUI and performs other business in order to keep the simulation correct

## 4.2. Class Diagram



## 4.3. Classes implementation

### 4.3.1. Model

Client – contains data about each individual client and methods to obtain the private attributes of the class such as ID, arrival time, service time, and remaining time.

Queue – implements the Runnable interface so it has a run() method. In this method, operations are made on the first client in the queue. Clients are stored in an ArrayBlockingQueue to provide thread safety. The first client is removed if its task is completed. In the other case, it will decrement its remaining time in the queue. Also from here, these actions are written in the log file.

Globals – this class contains some attributes that are used in more than one class and are not passed as parameters. They are Atomic Integers, so they are thread safe.

Peak hour – is a class that has as attributes the time the peak hour of the simulation was and also the waiting time that determined that “hour” to be the highest demanding “hour” of the simulation.

### 4.3.2. View

SetupView – this class displays the first frame of the application, the first view in other words. Here the user must introduce the data needed to start the simulation. This data consists of: number of clients, number of queues, maximum simulation time, minimum arrival time, maximum arrival time, minimum service time, and maximum service time. This data is passed to the controller for validation.

SimulationView – the second frame (view) of the app. It displays the real time evolution of all the queues. The main frame has a BorderLayout and in the North side there is a so called “progress” panel that contains a label that represents the current time and is continuously updated and a progress bar that is also updated as one unit of time passes. In the Centre part another panel is placed. It has a GridLayout with 1 column and as many rows as the number of queues. Each of these rows is a panel on its own. Each panel has also a GridLayout that splits the ClientBoxes between them.

ClientBox – is a custom JTextArea. It contains information about each client such as ID, arrival time, service time, and remaining time until he or she can leave the queue having finished his or her task. It has a method that changes the background colour if the client arrives at the top of the queue.

### 4.3.3. Controller

SetupControl – has the implementation of the ActionListener of the submit button. If the received from the SetupView is validated, it starts the “main” thread that takes care of everything. The validation consists in checking if the data introduced by the user, is all numbers. If it’s not the case, the app will display a message that tells the user which field of the following: number of clients, number of queues, maximum simulation time, minimum arrival time, maximum arrival time, minimum service time, or maximum service time was not a number. If all the inputs are valid numbers, a logic test is performed. This checks for: minimum service time can't be bigger than maximum service time, minimum arrival time can't be bigger than maximum arrival time, values should be positive, maximum arrival time can't be greater than maximum simulation time. If all those conditions are passed, the simulation can begin. In the other case, the user needs to introduce valid data

SimulationControl – works as a thread and is the only one that has access to the current time of the simulation. Here clients are initialised with random arrival time and service time but between the bounds set by the user for the arrival and service respectively. Also, two of the most important methods of the app are implemented here. The first one is the bestQueue that, as the name suggests, finds the best queue at the current simulation time. More details will be given in the following section of the documentation. The other important method is addClient which receives the client and the queue the client should be placed into. It will be also described in great detail later int the documentation. The run method for this class as previously noted, modifies the current time and dictates what should be performed. It sleeps for 1 second (1000 milliseconds) to make the simulation human perceptible. It also writes in the log file the current time and the values calculated for the peak hour, average waiting time and average service time. At each iteration, the current time increases by 1 until it reaches maximum simulation time. At each iteration, it goes through the clients ArrayList and checks if any client has the arrival time equal to the current time. If it does, it finds out which is the best queue in terms of waiting time using the bestQueue method. The client is added in that queue with the help of addClient method. At each current time, the peak hour is selected as the maximum between the current peak hour and the longest waiting

time at the current moment. Also, while checking the peak hour, all the threads are started. Before the next iteration, the method updates the view such that it displays the current situation. At the end of the method, after the current time becomes equal to the maximum simulation time, the peak hour, average waiting time and average service time are written in the file log.txt. A message is displayed at the end of the simulation informing the user that the simulation ended successfully.

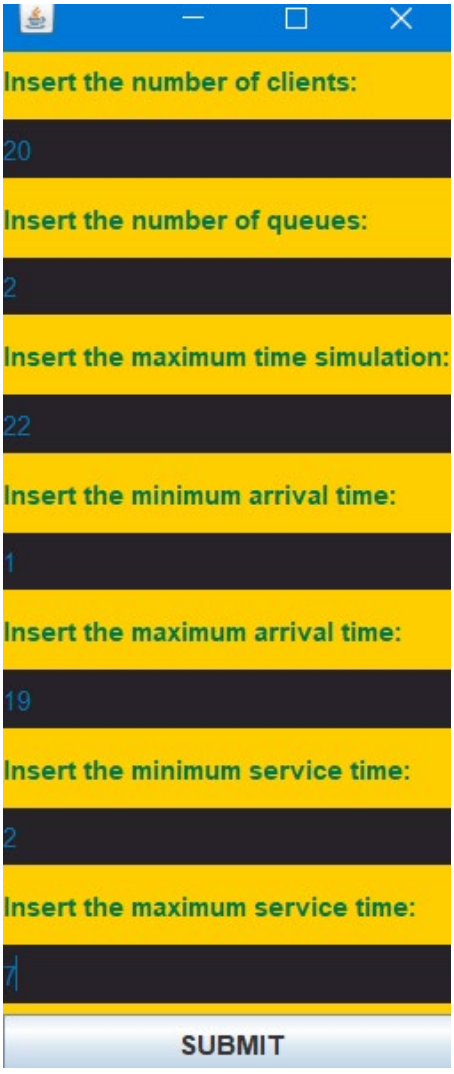
### 4.4. Important methods

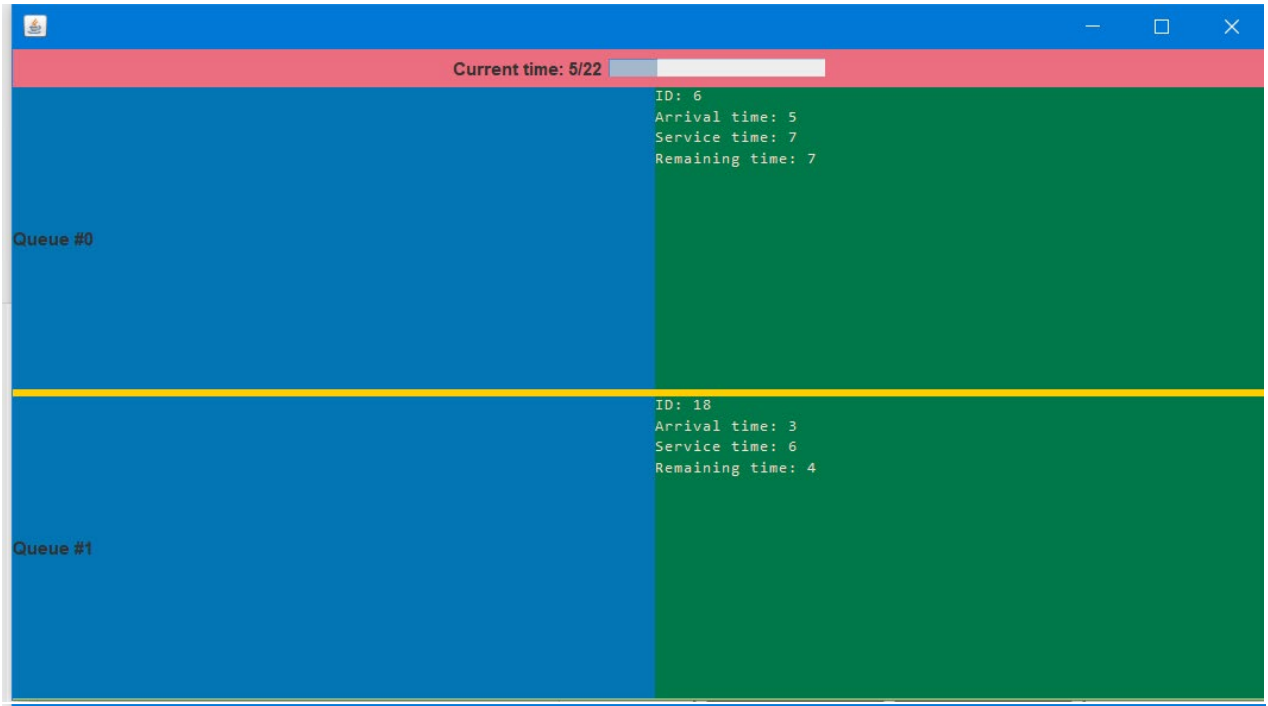
bestQueue – find the best queue based on the lowest waiting time. Go through all threads (queues) to find the best one. If the waiting time of the queue is smaller than the current lowest, set the current lowest to be the current queue (finding minimum).

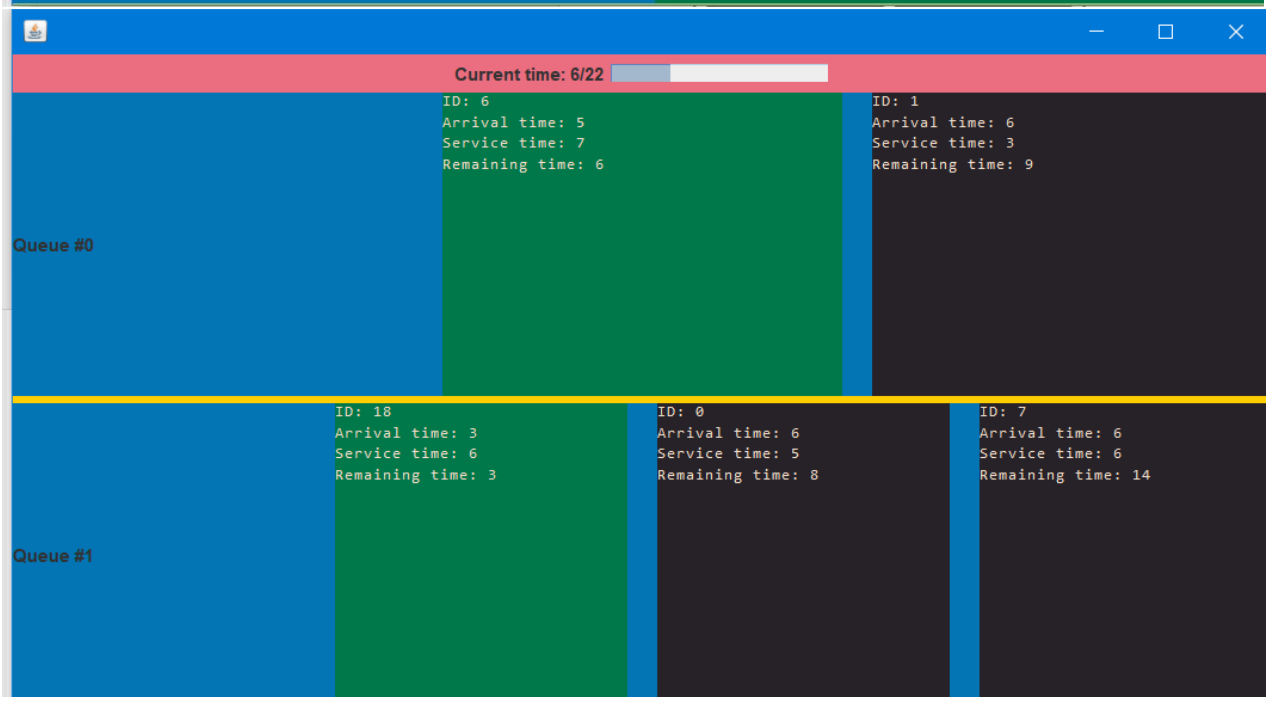
addClient – performs several operations based on a condition: check if there is enough time to serve the client in any of the queues. If it can't, print (write in file) a suggestive message. If it can, adds the client to the selected queue (thread backend), and adds it to the frontend also. There is an additional variable to store the total waiting times to compute the average waiting time. Increments the total number of clients that entered the queue for each client that entered. This is used for average calculations. It also sets the individual waiting time – to be displayed on the GUI. Finally, it writes a suggestive message in the log file about the client entering the queue.

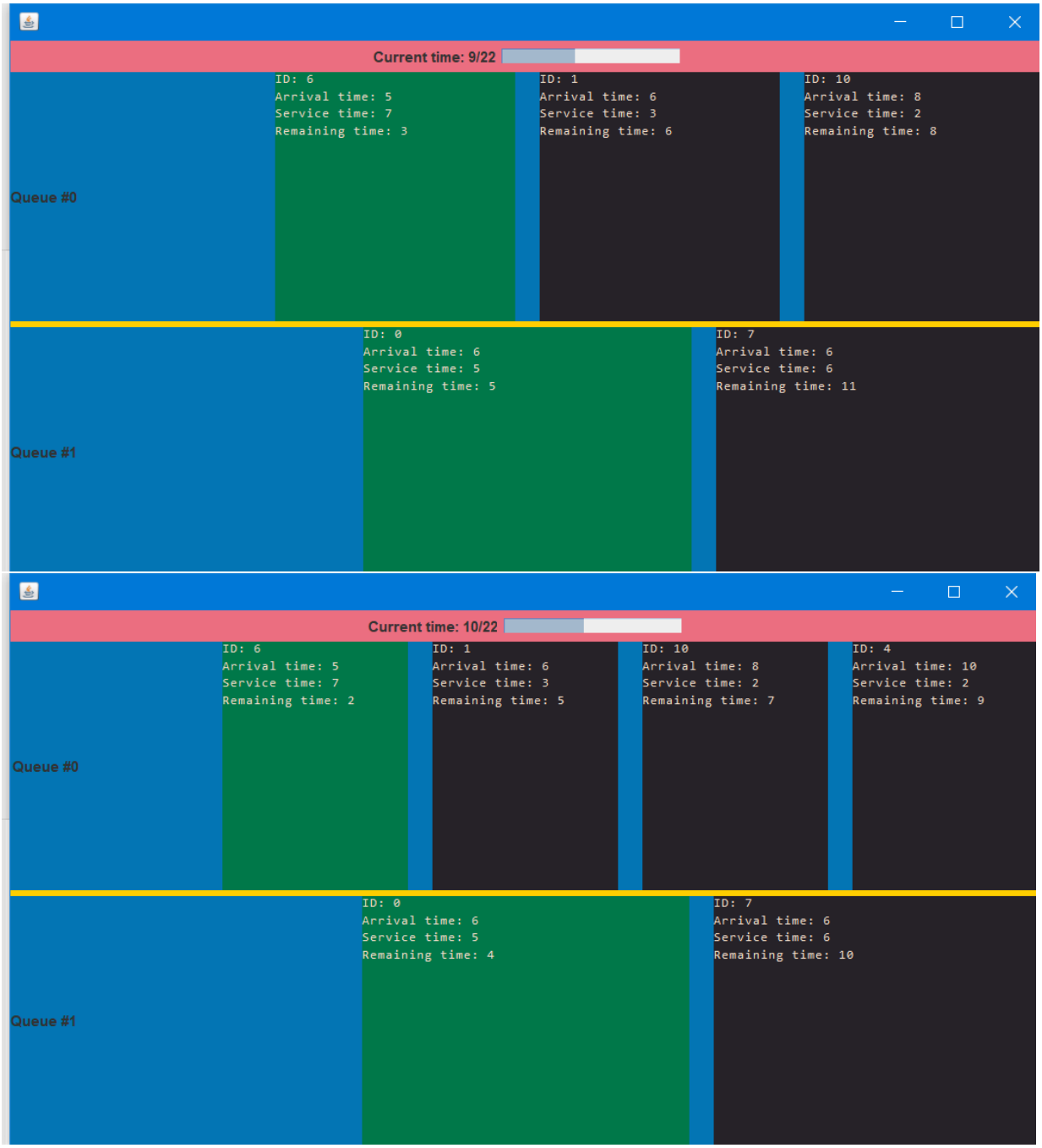
## 5. GUI presentation

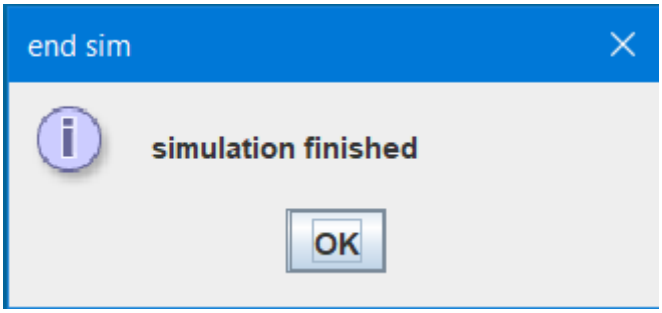
### 5.1. Successful setup




Current time: 5/22
ID: 6
Arrival time: 5
Service time: 7
Remaining time: 7
Queue #0
ID: 18
Arrival time: 3
Service time: 6
Remaining time: 4
Queue #1

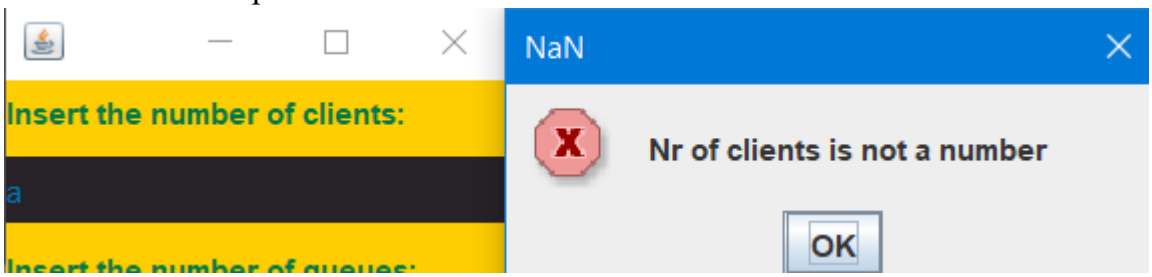

Current time: 6/22
ID: 6
Arrival time: 5
Service time: 7
Remaining time: 6
ID: 1
Arrival time: 6
Service time: 3
Remaining time: 9
Queue #0
ID: 18
Arrival time: 3
Service time: 6
Remaining time: 3
ID: 0
Arrival time: 6
Service time: 5
Remaining time: 8
ID: 7
Arrival time: 6
Service time: 6
Remaining time: 14
Queue #1



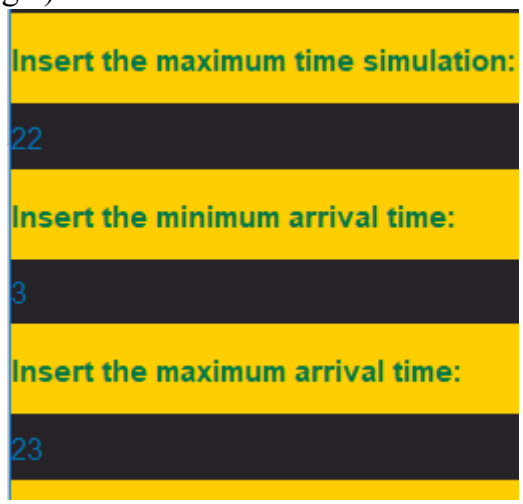

This is how a successful simulation will behave. I did not include all the steps of the simulation as there would be too many pictures, but a complete summary of the simulation can be seen in the log file.

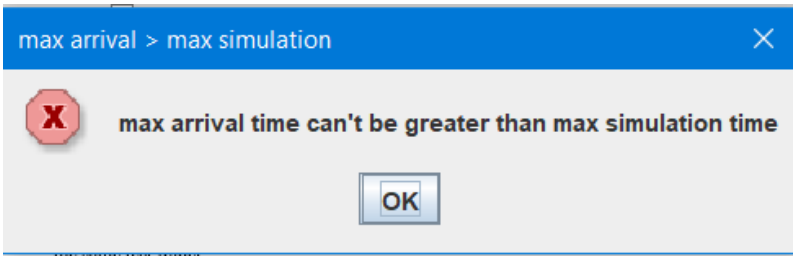
### 5.2. Bad input




If the user tries (accidently or not) to insert a string that is not a number, a message will be displayed, and the simulation will not start until the user provides correct input. For the rest of the fields, a similar message will pop up.

### 5.3. Bad input (logic)




The logic conditions will check for data logic and will display an error message letting the user know about the mistake. Again, the simulation will not start until a correct input will be given

## 6. Conclusion

This project helped me discover new functionalities in Java programming language, for instance, how to read from a file and to use threads, which could be extremely useful for performing complicated tasks in the background without interrupting the main program and are fundamental in improving the performance of the application. All the main and secondary requirements have been accomplished. At the end of the assignment, I consider that my Java programming skills evolved since the last project.

# 7. Bibliography


- the lectures from the course
- support presentation
- Wikipedia
- Geeks for Geeks
- W3schools
- Stack Overflow
- Quora
- irisys
- freeCodeCamp
- Computer Hope
- javaTpoint
- Tutorials point
- IBM