DOCUMENTATION

ASSIGNMENT 2

NAME: Muntean Cristian

GROUP: 30424

Contents

[1. Objective 3](#_Toc99652566)

[2. Analysis, modelling, scenarios and use cases 3](#_Toc99652567)

[3. Design 5](#_Toc99652568)

[4. Implementation 6](#_Toc99652569)

[5. Results 9](#_Toc99652570)

[6. Conclusions 11](#_Toc99652571)

[7. Bibliography 11](#_Toc99652572)

# Objective

Design and implement an application aiming to analyze queuing-based systems by simulating a series of N clients arriving for service, entering Q queues, waiting, being served and finally leaving the queues, and computing the average waiting time, average service time and peak hour. Finally, create a file that acts as a log of the events that happened.

# Analysis, modelling, scenarios and use cases

1. Analysis

In the application, a queue is represented by the “Server” class and its aim will be to eliminate the clients (in the application being named “Task”) that it contains. Each queue will work in parallel, independent of each other and the existence of a manager (named “Scheduler”) that will put each client on the best queue at the current moment in time.

The queue manager has two selection options (strategies). The first one is based on the shortest waiting time. So, for example if we have 3 queues, one with a waiting time equal to 3, one with a waiting time equal to 6 and the third one with the waiting time equal to 4, the next client will go to the first queue. The second selection strategy is based on the shortest queue length. So, if we use the same example as for the first strategy but with number of clients in stead of waiting time, the next client will still go to the first queue, since that is the one which has the least number of clients.

Each client will stay in a queue for a “service time” and an “arriving time’ that receive a random value bounded by two inputs each. When the simulation is performed, a “current time” will be incremented up until an input value that is represents the time limit. During each time i, the clients that have the arriving time equal to that i will be placed in a queue by the manager, where i=0, 1, …, time limit.

1. Modelling

The user will have to introduce all of the following values with the mentioned restrictions in order for a simulation to be started:

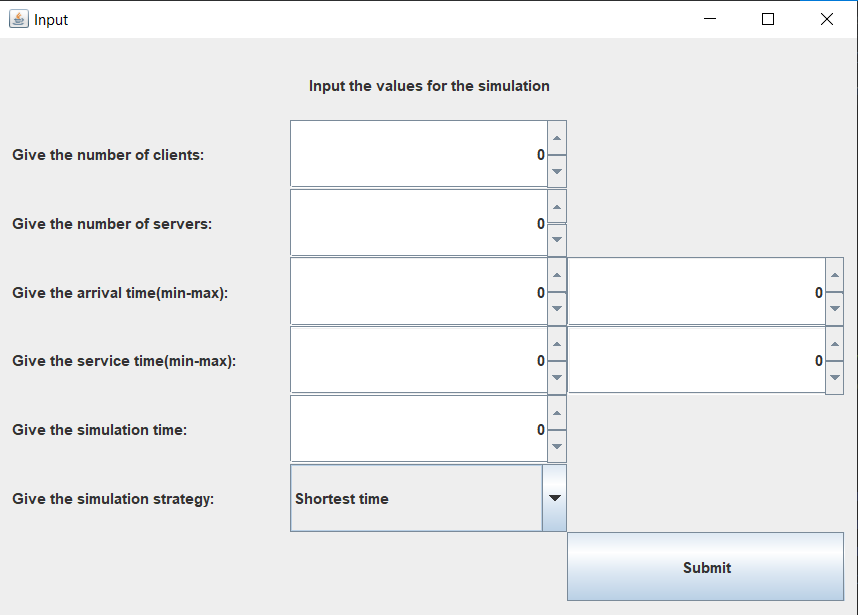
* Number of clients, value must be greater than 0.
* Number of servers, value must be greater than 0.
* Minimum and maximum arriving time, values must be greater than 0 and the value of the minimum input must be smaller or equal than the value of the maximum input.
* Minimum and maximum service time, values must be greater than 0 and the value of the minimum input must be smaller or equal than the value of the maximum input.
* Time limit, value must be greater than 0.
* Selection policy, the user is only able to choose between two options: shortest time, shortest queue.

If the user has introduced all these inputs correctly, the simulation will begin and on each second the application will output all the clients that have not entered a queue yet and all the queues with information about them (waiting time for the queue, length of the queue, all the clients that are currently in the queue).

1. Scenarios and use cases

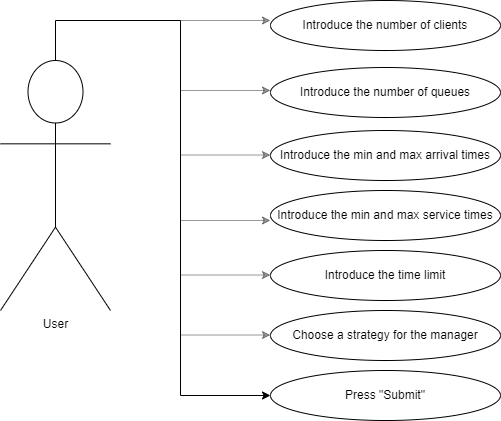
A use case is a methodology used in system analysis to identify, clarify, and organize system requirements. The use case is made up of a set of possible sequences of interactions between systems and users in a particular environment and related to a particular goal.

The use cases are strongly connected with the steps the user has to make, so I created the user interface with this in mind, resulting in a straight forward design.



The user will have to introduce the number of clients, the number of servers, the minimum and maximum arrival time, the minimum and maximum service time, the simulation time and choose from the two options available in the simulation strategy and then click the “Submit” button.

The use case diagram for this process is presented below:

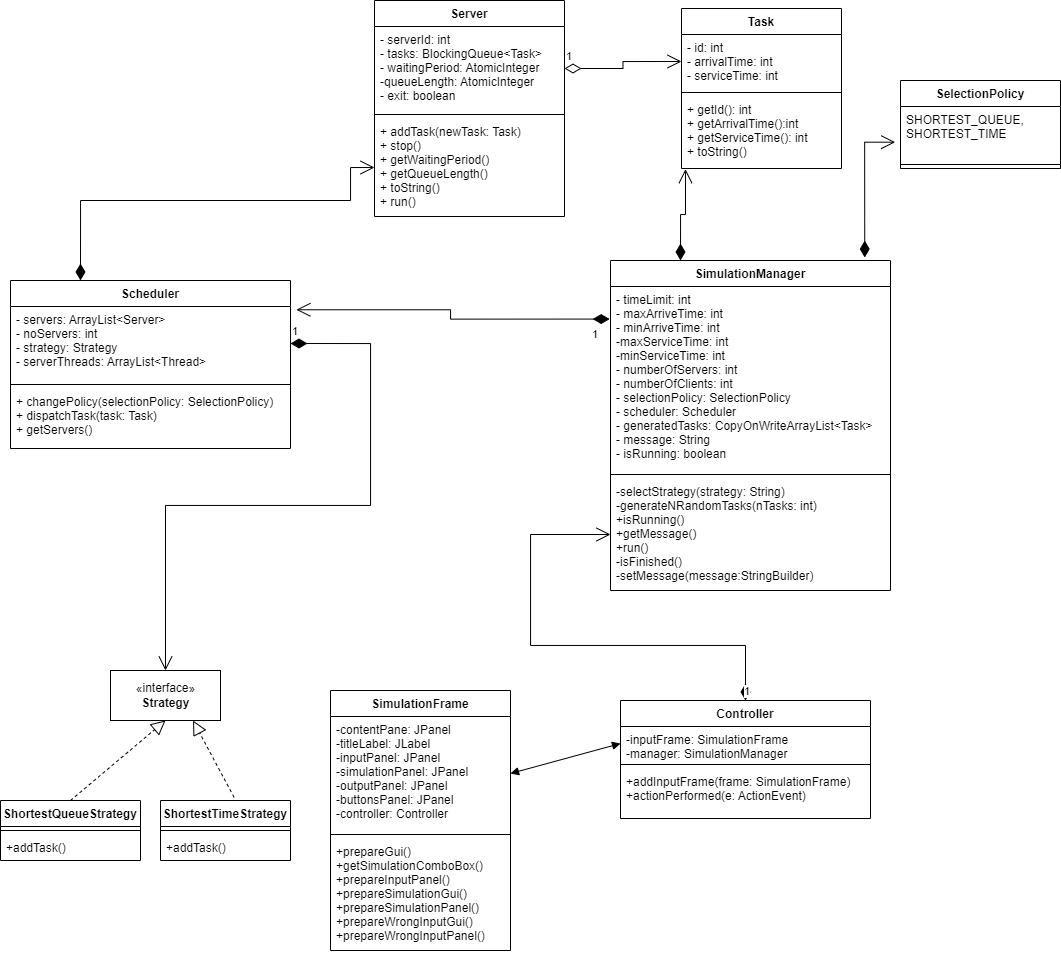


# Design

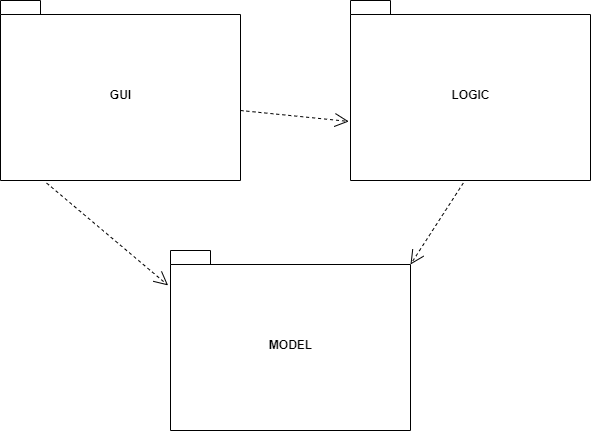
In the design of the application I used the model-view-controller (MVC) architectural pattern, which separates the logic package (that contains the classes which implement the mathematical operations) from the model (that contains the classes which model the application data) and from the view (that contains the classes that are related to the graphical user interface). The MVC pattern has been heralded by many developers as a useful pattern for the reuse of object code and a pattern that allows them to significantly reduce the time it takes to develop applications with user interfaces.

The model-view-controller pattern proposes three main components to be used in the development process of the application:

* The *logic* package contains the “Scheduler” class, the “SimulationManager” class, the “Strategy” interface, the “TimeStrategy” and the “ShortestQueueStrategy” classes (that implement the “Strategy” interface).
* The *model* package contains the “SelectionPolicy” enumeration, the “Server” and “Task” classes, which represent the queue and client respectively.
* The *gui* package contains the “SimulationFrame” and “Controller” classes, which implement and manage the graphical user interface. The “Controller” class is responsible of connecting the logic and model part of the application with the user interface. The “SimulationFrame” class implements the user interface.
* UML diagram



* Package diagram



* Data structures

The data structures that I have used in this project are either primitive types (such as int or boolean), or more complex data structures such as ArrayLists, BlockingQueue, CopyOnWriteArrayList or AtomicIntegers.

I used the BlockingQueue, AtomicIntegers and the CopyOnWriteArrayList because all of then are thread-safe. The CopyOnWriteArrayList has been used to keep track of the generated clients that have not yet entered a queue. The BlockingQueue and the AtomicIntegers have been used to keep track on the clients that are currently in the queues and to keep track of the size and the waiting time of each queue.

# Implementation

Because I used the MVC pattern design architecture, my application has 4 main parts:

1. The Model part

This part cotains the classes which model the application. The model has 2 classed and 1 enumeration:

1. The “Task” class

This class represents the clients that will enter the queues. It contains 3 fields: id, arrivalTime and serviceTime. All of them are integers.

The only methods that this class has are the Constructor, getters for all of the fields and the override of the “toString” method which will be used in printing every client.

1. The “Server” class

This class represents the queues in which the clients will enter. It contains 5 fields: serverId – integer, tasks – BlockingQueue of Tasks, waitingPeriod and queueLength – AtomicIntegers and exit – Boolean.

It implements the runnable interface because each queue will be on a separate thread and all of them will work in parallel.

The most important methods of this class are:

* The Constructor. It takes one parameter: the id of the queue and assigns it to the id field. It also initializes the BlockingQueue, the AtomicIntegers (which get an initial value of 0) and the exit field receives the value “false”.
* The “addTask” method. It takes as a parameter an object of type Task and adds it to the BlockingQueue. After the addition in the queue, the fields “waitingPeriod” and “queueLength” are updated (the waitingPeriod is incremented by the “serviceTime” field of the Task object and the queueLength is incremented by one).
* The “run” method. This method will keep the thread active until the exit field will be switched to “true”. During the time the thread is active, the first element of the queue is removed, the thread is blocked for the first’s element service time and the waitingPeriod and queueLength fields of the Server are updated.
* The “toString” method. It is overridden and used in order to print each queue with the clients that are currently in it.

1. The “SelectionPolicy” enumeration

Contains the two strategies the queues can work on: shortest time and shortest queue.

1. The Logic part

This part contains the classes that use the Model classes in order to create the functionality of the application. It contains 4 classes and 1 interface:

1. The “Scheduler” class

This class is the one that is responsible of dispatching clients to queues and it does it by using one of the two selected strategies. Its fields are: an ArrayList containing the queues (named servers), the number of queues that will be used, the selected strategy and an ArrayList containing the threads that the queues will run on.

The most important methods of this class are:

* The Constructor. It takes as parameters the number of queues and the selected strategy and gives these values to the respective fields. It also instantiates the ArrayLists fields and creates all the necessary queues and the necessary threads. Finally, it starts all the threads that contain the queues.
* The “dispatchTask” method. It uses the selected strategy field in order to dispatch a task that it receives as a parameter.

1. The “Strategy” interface

This interface is the blueprint of the concrete strategies and contains a single abstract method: “addTask”.

1. The “ShortestQueueStrategy” class

This class implements the “Strategy” interface and the “addTask” method. The “addTask” method has as parameters the task it needs to add and all of the queues. Since it has to find the queue with the shortest waiting period, it searches for it and adds the task to the respective queue using the queue’s “addTask” method that has been presented above.

1. The “TimeStrategy” class

This class also implements the “Strategy” interface and the “addTask” method. The difference between this class and the previous one is that this one searches for the queue with the shortest length and doesn’t take into consideration the waiting period.

1. The “SimulationManager” class

This class is responsible with the simulation itself. It has as fields the time limit of the simulation, the minimum and maximum arrival time and service time for the clients, the number of queues, the number of clients, the strategy that will be used, a Scheduler object, the randomly generated tasks, a String named “message” that represents the message that will be displayed on each second on the user interface and a Boolean field that will specify whether the simulation is running or has finished.

It also implements the “Runnable” interface, meaning it will be run on a separate thread.

The most important methods of this class are:

* The Constructor. It receives as parameters all of the user inputs mentioned above and it creates an instance of the Scheduler
* The “generateNRandomTasks”. It receives as a parameter how many clients it needs and generates that amount of clients with random fields that are withing the intervals introduced by the user.
* The “isFinished” method. It checks whether there are anymore clients waiting to be placed in a queue and whether there are anymore clients in the queues. If both of these conditions are false, it means that the simulation has ended.
* The “run” method. It runs when the thread starts and it is the method that implements the simulation. It starts from the current time 0 and goes up to the time limit field. On each time, the clients that have the arrival time equal to the current time will be assigned to a queue. During the simulation, the average waiting time, the peak hour and the average service time are calculated.

1. The View part

Contains 2 classes: ”Controller” and “SimulationFrame”.

1. The “SimulationFrame” class

This class implements the graphical user interface. It has three main frames: the input frame, the simulation frame and the wrong input frame. When the user first starts the application, the input frame will be the one that is visible. After the user has introduced all of the required inputs and if those inputs are valid the simulation frame will be the one that appears next. In this frame, the user will be able to see the simulation happening in real time. If the user has introduced at least one field wrong, the wrong input frame will appear, which will let the user know that he has introduced one field wrong.

1. The “Controller” class

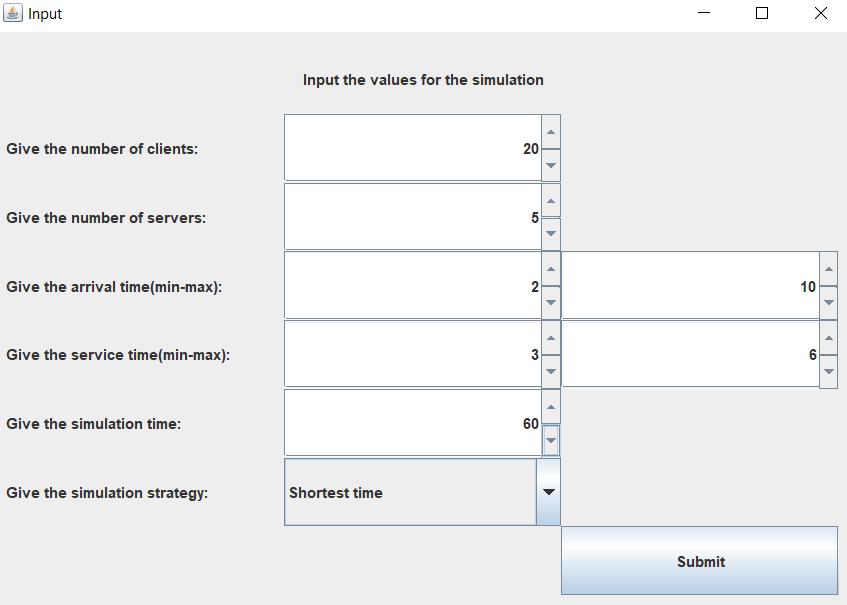
This class makes the connection between the graphical user interface and the logic and model part. Its most important method is the “actionPerformed” one, which recognizes wether the user has pressed the “Submit” button that is available in the input frame and starts the simulation.

1. The App part

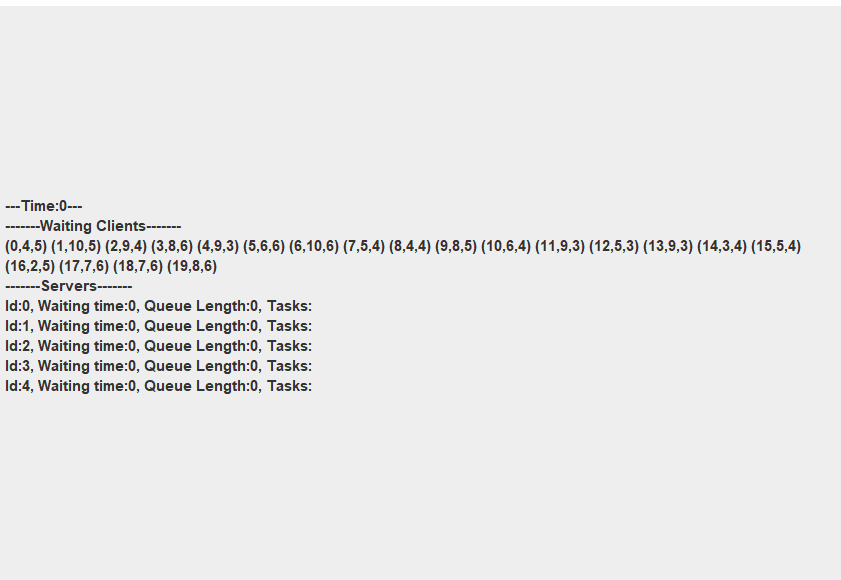
The App part contains only the “App” class which creates a new JFrame of type “SimulationFrame”, sets the frame visibility to true and sets the default close operation of the frame to exit the whole application on the click of the “X”.

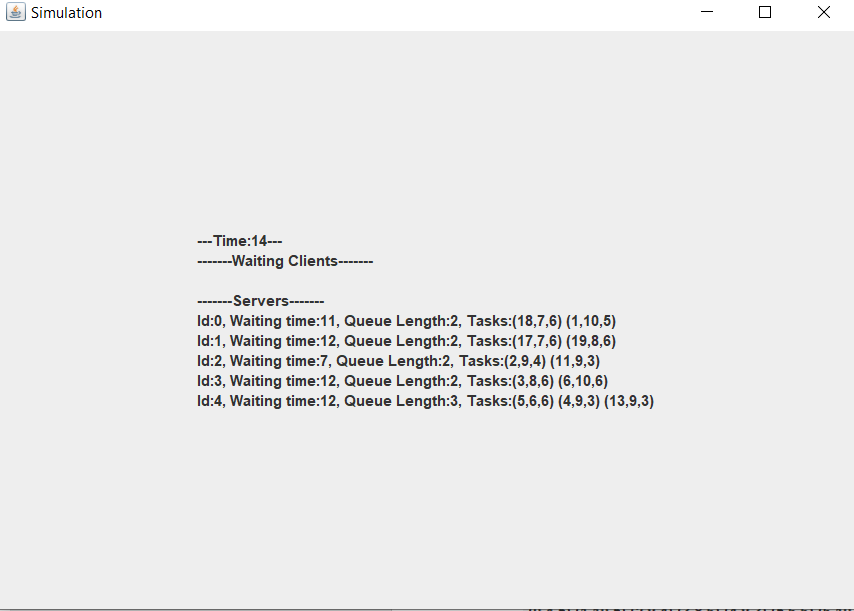
# Results

For the testing part of the application I introduced some cases manually. Firstly, when all of the inputs are correct,

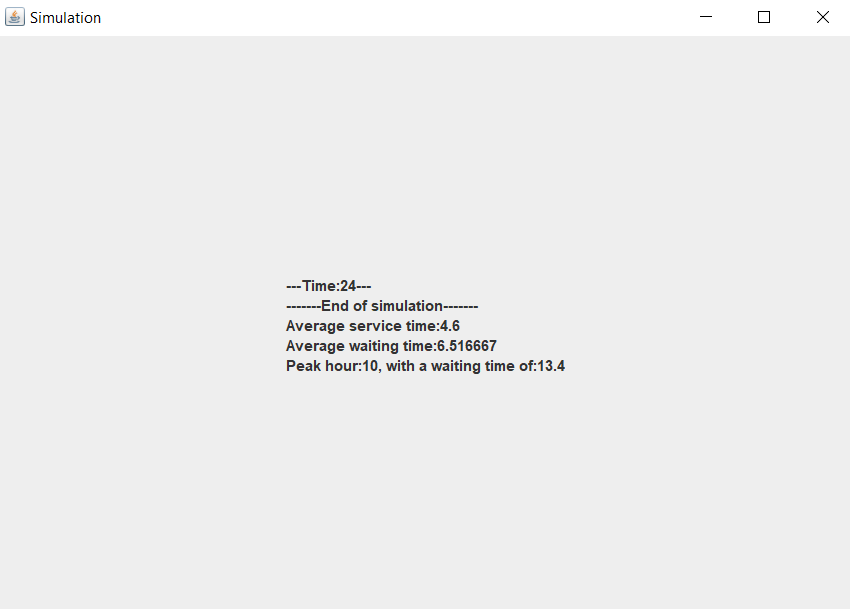


After the press of the “Submit” button, the simulation starts and it will show the development of the clients in the queues.

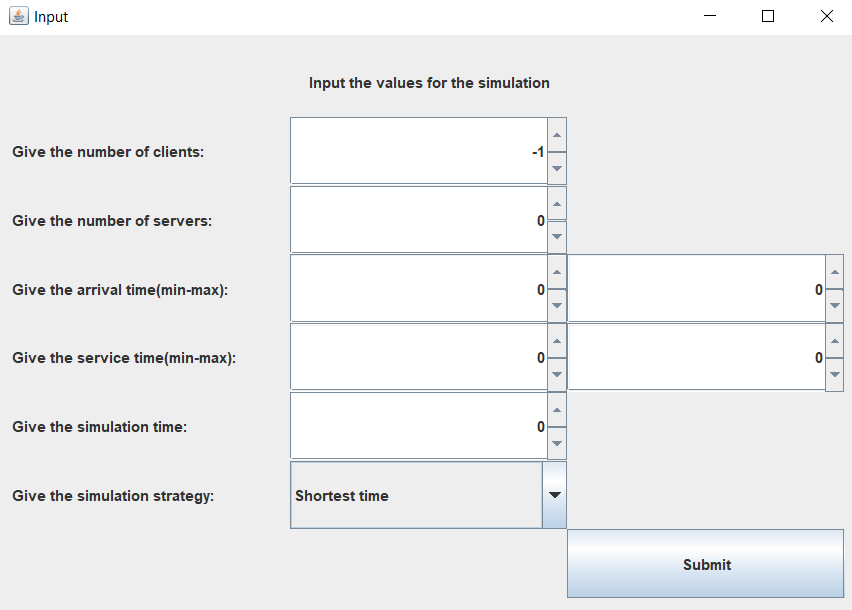


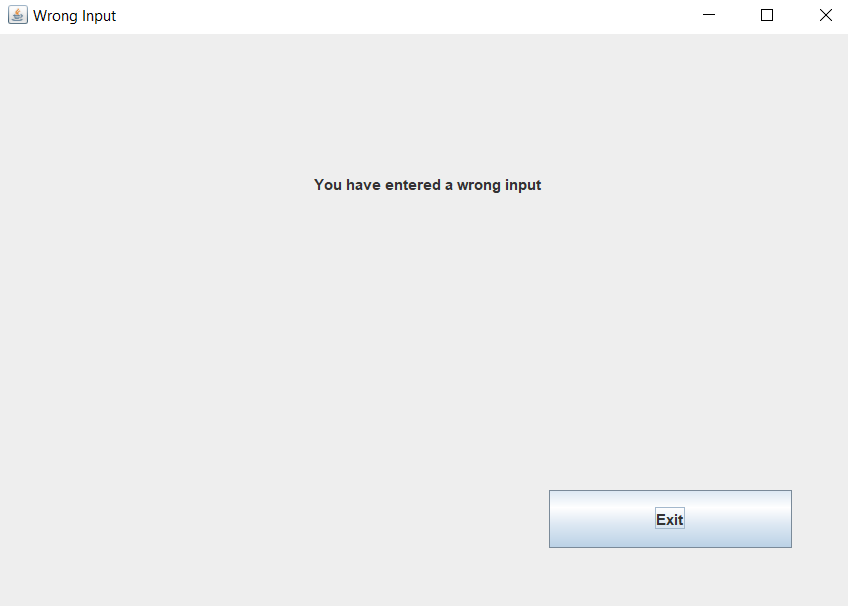


At the end of the simulation, the average service time, the average waiting time and the peak hour are displayed.



If the user enters a wrong input, such as a negative value or if one of the minimum times are greater that one of the maximum times, the wrong input frame will be shown.





# Conclusions

This project has given me the opportunity of working with threads and to learn how to manage the synchronization between them. As an example, I had to synchronize the SimulationManager thread with the graphical user interface in order to be able to display the progression of the simulation in real time.

# Bibliography

* <https://draw.io/> - for diagrams
* <https://stackoverflow.com/> - - for random bugs I encountered during the development
* Programming Techniques in Java – Lectures of prof Cristina Bianca Pop