

# **DOCUMENTATION**

## **QUEUE MANAGEMENT APPLICATION**

NAME: Mihai Anca

# CONTENTS

1.	Assignment Objective .....	3
2.	Problem Analysis, Modeling, Scenarios, Use Cases.....	4
3.	Design .....	5
4.	Implementation .....	8
5.	Results.....	13
6.	Conclusions.....	15
7.	Bibliography .....	15

# 1. Assignment Objective

**Main objective:** Design and implement a queues management application which assigns clients to queues such that the waiting time is minimized.

(1) The queues management application should simulate (by defining a simulation time *tsimulation*) a series of N clients arriving for service, entering Q queues, waiting, being served and finally leaving the queues.

- All clients are generated when the simulation is started, and are characterized by three parameters:

- > *ID* (a number between 1 and N)

- > *tarrival* (simulation time when they are ready to enter the queue)

- > *tservice* (time interval or duration needed to serve the client; i.e. waiting time when the client is in front of the queue).

- Each client is added to a queue when its *tarrival* time is greater than or equal to the simulation time ( $tarrival \geq tsimulation$ ).

- The following data should be considered as *input data* for the application that should be inserted by the user in the application's user interface:

- > Number of clients (N)

- > Number of queues (Q)

- > Simulation interval (*tsimulation MAX*)

- > Minimum and maximum arrival time ( $tarrival MIN \leq tarrival \leq tarrival MAX$ )

- > Minimum and maximum service time ( $tservice MIN \leq tservice \leq tservice MAX$ )

(2) The average waiting time, average service time and peak hour of the simulation should also be computed.

## **Sub-objectives:**

1. understand the problem and its domain

2. analyze the problem and identify the requirements

3. find solutions to individual problems, from which the overall solution is synthesized, sub-objective which can be divided into another two:

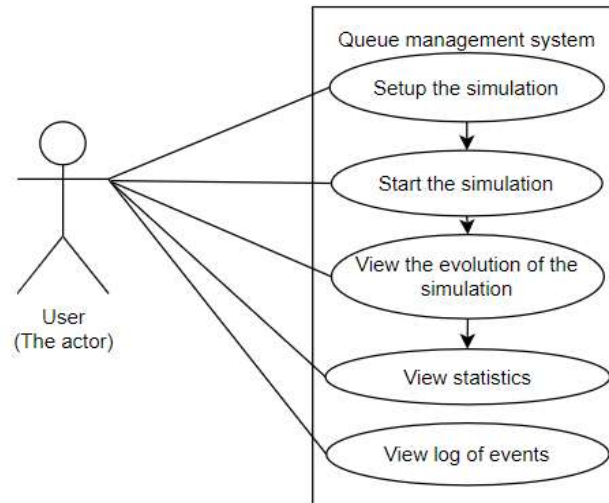
- 3.1 design the queue management system

- 3.2 implement the queue management system

4. test the simulation application

## 2. Problem Analysis, Modeling, Scenarios, Use Cases

Use case diagram:



### Functional requirements:

1. The simulation application should allow users to setup the simulation
2. The simulation application should allow users to start the simulation
3. The simulation application should display the real-time queues evolution
4. At the end of the simulation, the application should display the following elements:
  - *average waiting time*: at each period (second), a temporary average = average of the waiting periods of all queues at that moment is calculated; at the end of the simulation, average waiting time = average of those temporary averages is computed
  - *average service time*: the average of the service times of the clients which have been processed until the end of the simulation
  - *peakhour*: the time at which there were the most clients in the queues

### Non-functional requirements:

1. The queue management system should be intuitive and easy to use by the user
2. The queue management system should perform the operations fast
3. The queue management system should be reliable and consistent (should be able to run the corresponding simulation correctly for ANY data introduced)

### Use case:

**Name:** *start simulation*

**Primary Actor:** *user*

**Main Success Scenario:**

- I. The user inserts the values for the: number of clients, number of queues, simulation interval, minimum and maximum arrival time, and minimum and maximum service time
- II. The user clicks on Start simulation button
- III. The application validates the data and starts running the simulation
- IV. The application displays the real-time evolution of the simulation
- V. At the end of the simulation, the average waiting time, average service time and peak hour are also displayed on the screen
- VI. Moreover, at the end of the simulation, a log of the events is generated

#### **Alternative Sequence:**

*Invalid values for the setup parameters*

- The user inserts invalid values for the application's setup parameters
- The application displays an error message and requests the user to insert valid values
- The scenario returns to step I

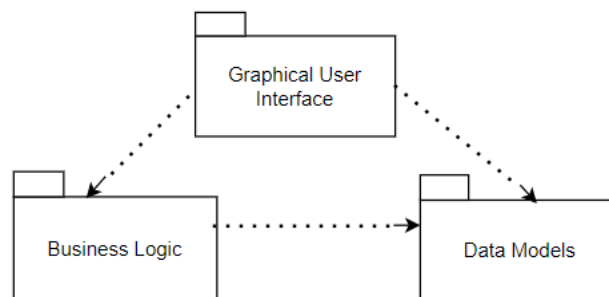
## **3. Design**

### **OOP design of the application:**

The project is based on a layered architecture, which has the following advantages:

1. **Code Maintenance is easy:** we can easily determine any kind of change in the code
2. **Security:** the data-providing package isn't affected by the other packages
3. **Ease of development:** building time taken by the application will be small as all the layers can work together at the same time

### **UML package diagram:**

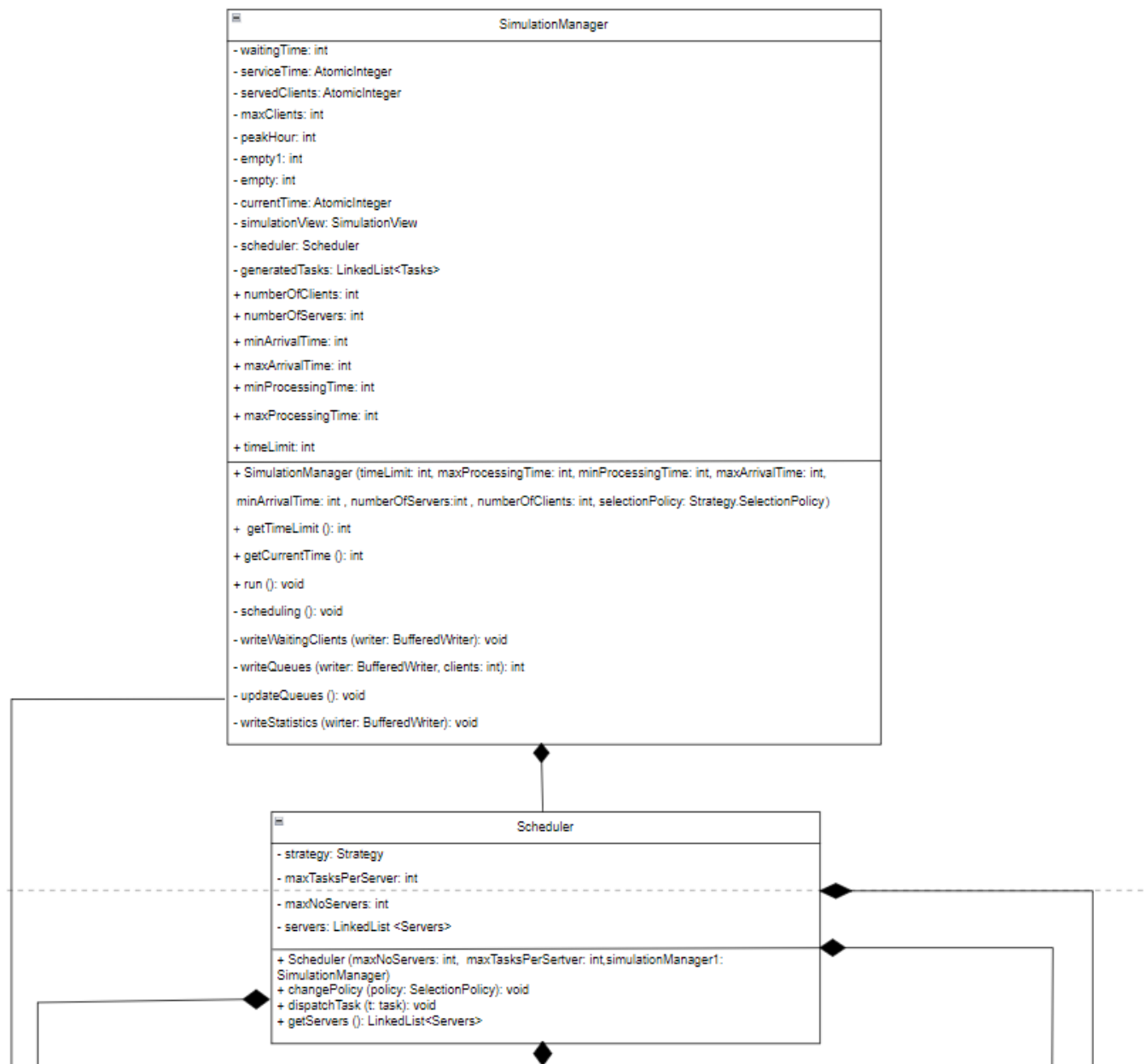


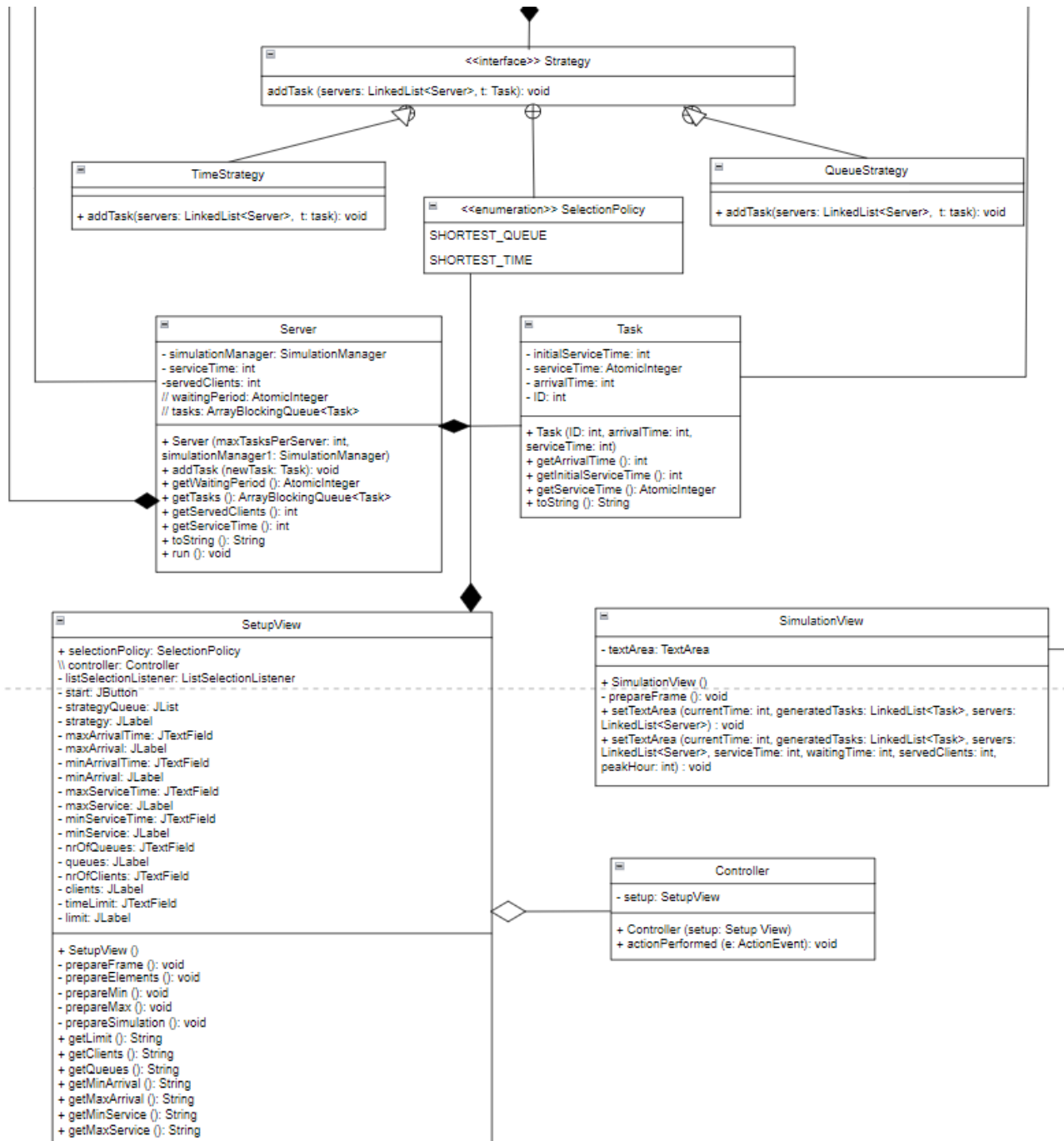
The project has 3 packages + the *App* class (its purpose is to run the application):

1. *Data Models* – contains the classes modeling the application data
  - in this app, it contains 2 classes: the *Server* and *Task* classes
2. *Graphical User Interface* – contains the classes implementing the graphical user interface

**3. Business Logic** – contains the class and the interface which determine each queue each client goes to (*Scheduler, Strategy*) and also the class getting the inputs of the graphical user interface, generating the simulation based on them and, at a given period, giving updates about the state of the simulation (*Simulation Manager*)

### Class Diagram:





### Used data structures:

- *primitive data types* (ex: int)
- *List* (LinkedList): I used *Java Collections Framework* because it reduces programming effort and increases program speed, quality respectively; I used the *List* interface in order to store the servers and the tasks generated by the Random Client Generator
- data structures for *writing in files* (FileWriter, BufferedWriter)
- appropriate *synchronized* data structures to assure *thread safety*: volatile variables, AtomicInteger, BlockingQueue interface (implemented by ArrayBlockingQueue class)
- *newly created objects* (ex: Task)

## 4. Implementation

### Implementation of classes:

#### I. data\_models Package:

##### *Server Class*

###### - important fields:

- *ArrayBlockingQueue* <Task> tasks : the current clients which should be served by the server in the future
- *AtomicInteger* waitingPeriod: the sum of the remaining service times of the clients which wait at that server
- *private int* servedClients: the number of clients which have been served by that server until the current time
- *private int* serviceTime: the sum of service times of clients which have been served by that server until the current time

###### - important methods:

- *public Server (int maxTasksPerServer, SimulationManager simulationManager1)*: its constructor
- *public void addTask (Task newTask)*: this method simulates the arriving of a new client at that server
- *public String toString()*: puts into a string all the necessary information about each client waiting at that queue(server)
- *public run()*: a method specific to threads which executes the code described in it

##### *Task Class*

###### - important fields:

- *private int* ID : the ID of the client
- *private int* arrivalTime: the time the client gets into one of the queues
- *private AtomicInteger* serviceTime: the remaining time (out of the initial service time) for the client to be completely processed by the server
- *private int* initialServiceTime: the initial service time of the client, before being processed by a server



- important methods:
- *public Task (int ID, int arrivalTime, int serviceTime):* its constructor
- *public String toString():* puts into a string all the necessary information about that client

## II. business\_logic Package:

### *Scheduler Class*

- important fields:
- *private LinkedList <Server> server:* the considered servers which a new client could be sent to
- *private Strategy strategy:* the strategy based on which a new client is sent to one of the existing queues
- important methods:
- *public Scheduler(int maxNoServers, int maxTasksPerServer, SimulationManager simulationManager1):* its constructor
- *public void changeStrategy (Strategy.SelectionPolicy policy):* this methods establish the strategy based on which the new clients are sent to the servers
- *public void dispatchTask (Task t):* sends a new client to one of the servers. According to the established strategy

### *SimulationManager Class*

- important fields:
- *public int timeLimit:* the maximum number of seconds for which the simulation runs
- *public int maxProcessingTime, minProcessingTime:* the maximum, minimum respectively necessary time for a client to be serviced by a server
- *public int maxArrivalTime, minArrivalTime:* the maximum, minimum respectively time at which a client could eneter a queue
- *public int numberOfServers, numberOfClients:* the number of queues, clients respectively which will be generated by the simulation
- *private LinkedList<task> generatedTasks:* the clients generated by Random Client Generator
- *private SimulationView simulationView:* gui which will display the data of the simulation in real time
- *private AtomicInteger currentTime:* for how seconds has the simulation run
- *private int volatile empty, empty1:* volatile variables used to check whether the list of clients which haven't been assigned to a queue yet and the queues, respectively are empty
- *private int peakHour:* the variable which keeps the current peak hour of the simulation
- *private AtomicInteger servedClients:* sum of the numbers of served clients of each queue (at the end of simulation)
- *private AtomicInteger serviceTime:* sum of sums of initial service times of served clients of each queue (at the end of simulation)

- *private int waitingTime*: current sum of average of waiting periods of all queues at each second
- important methods:
  - *public SimulationManager (int timeLimit, int maxProcessingTime, int minProcessingTime, int maxArrivalTime, int minArrivalTime, int numberOfServers, int numberOfClients, Strategy.SelectionPolicy selectionPolicy)*: its constructor
  - *private void generateRandomTasks()*: Random Client Generator
  - *public run()*: a method specific to threads which executes the code described in it
  - *private void scheduling()*: the clients which have the arrival time=current time of the simulation are sent to a queue
  - *private void writeWaitingClients (BufferedWriter writer),*  
*private int writeQueues (BufferedWriter writer, int clients),*  
*private void writeStatistics(BufferedWriter writer)*: methods used for writing into files/gui data about the (current) state of the simulation
  - *private void updateQueues()*: decrements the current service time of the client currently processed by each queue by 1

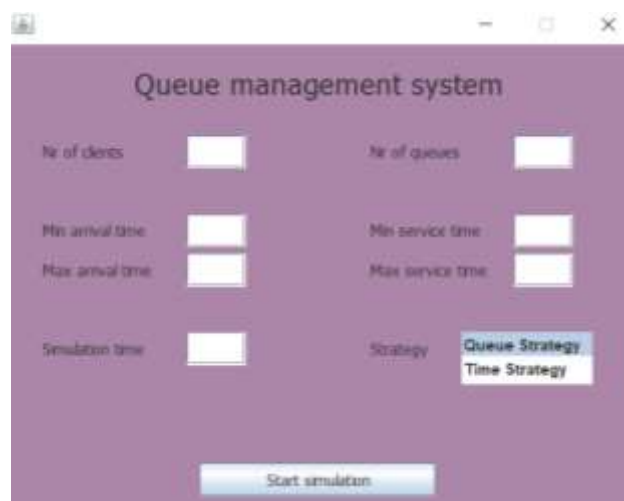
#### Strategy Interface

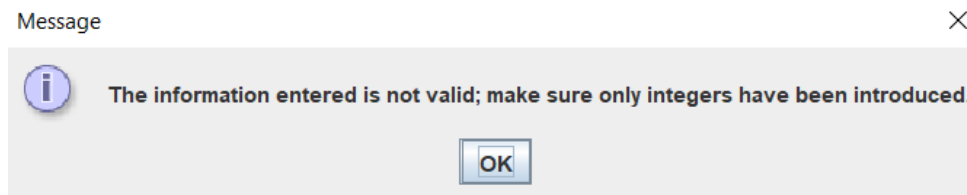
-method: *public void addTask (LinkedList<Server> servers, Task t)* – adds a new client to the queue decided by the strategy

- implemented by 2 inner classes:

- *QueueStrategy*: adds the new client to the queue with the least clients waiting to be processed by that server
- *TimeStrategy*: adds the new client to the queue with the smallest waiting period
- contains enum *SelectionPolicy*, which contains 2 options: SHORTEST QUEUE, SHORTEST TIME respectively

### III. gui Package (Implementation of the Graphical User Interface)





*SetupView class* – this class presents the graphic which allows the user to enter the necessary information for the simulation (number of clients, number of queues etc. )

- extends JFrame because, in this way, I can freely choose its size, visibility and layout
- I chose the AbsoluteLayout because I can put my elements (labels, TextFields, buttons) however I want, by specifying the bounds of each element
- important fields:
  - *private* JLabels and JTextFields
  - *private* JList strategyQueue: used for creating a list with the 2 options for the strategy of assigning new clients to the existing queues
  - *private* JButton start: button which, when clicked, starts the simulation with the data introduced by the user
  - Controller controller
- important methods:
  - *public* SetupView(): its constructor, which uses the following methods:
    1. *private* void prepareFrame ();
    2. *private* void prepareElements ();
    3. *private* void prepareMin ();

4. *private void prepareMax ();*
5. *private void prepareSimulation ();*
  - getters: used by the controller to take the information introduced by the user in the gui

#### *Controller class*

- this class takes the information entered by the user in the setup and, whenever the *Start simulation* button is pressed, starts a simulation based on the data introduced (that's why it's only field is *private Setup View setupView*)
- if *Start simulation* button is pressed and not all of the data introduced by the user isn't composed of integers, then an error message is displayed
- important methods
  - *public Controller (Setup View setupView)*: its constructor
  - *public void actionPerformed (ActionEvent e)* (for detecting whenever *Start simulation* button is pressed)

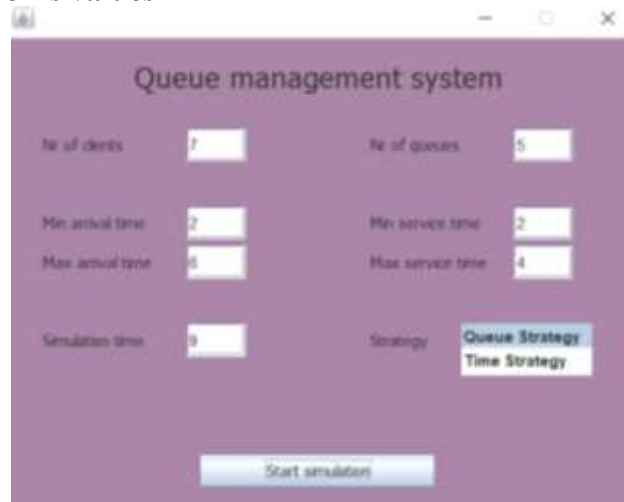
*SimulationView Class*– this class presents the graphic which displays the real-time evolution of the simulation

- extends JFrame because, in this way, I can freely choose its size, visibility and layout
- I chose the AbsoluteLayout because I can put my elements (labels, TextFields, buttons) however I want, by specifying the bounds of each element
- important fields:
  - *private JTextArea*: used to display the contents of the list of clients which haven't been assigned to a queue yet, current clients of each queue respectively
- important methods:
  - *public SimulationView()*: its constructor, which uses the followin method: *private void prepareFrame ();*
  - *public void setTextArea(...)* methods: displays the data of the simulation with regard to the current time; in addition, at the end of simulation, the statistics are also displayed on the interface

## 5. Results

Displayed by the Graphical User Interface

### 1. Setting the simulation's values



The screenshot shows a window titled "Queue management system" with a purple background. It contains several input fields and a "Start simulation" button. The fields are arranged in two columns:

Parameter	Value
No. of clients	7
No. of queues	5
Min. arrival time	2
Min. service time	2
Max. arrival time	6
Max. service time	4
Simulation time	9
Strategy	Queue Strategy Time Strategy

A "Start simulation" button is located at the bottom center.

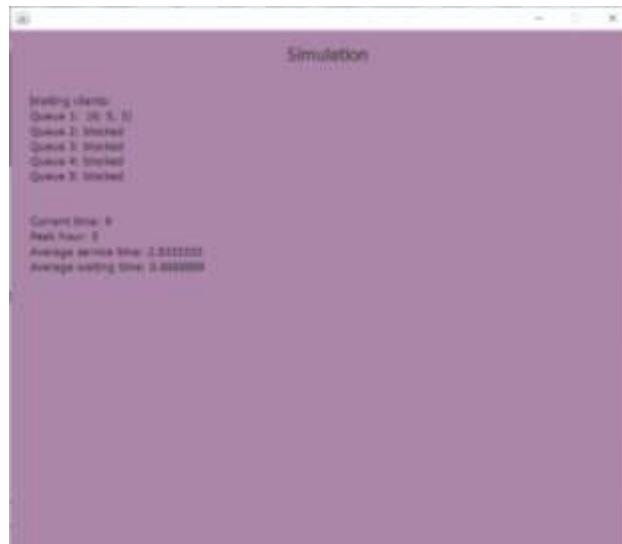
### 2. Simulation while running



The screenshot shows a window titled "Simulation" with a purple background. It displays the state of the simulation during execution. The text is as follows:

```
Waiting clients: (1) 5 (4) (6) 6 (2) (5) 3 (3) (6) 3 (2)
Queue 1: blocked
Queue 2: (3) 3 (3)
Queue 3: (7) 6 (3)
Queue 4: blocked
Queue 5: blocked
Current time: 4
```

### 3. End of simulation



### Generated logs of events

Test 1	Test 2	Test 3
N=4 Q=2 Time limit=60 seconds [t_min_arrival, t_max_arrival] = [2,30] [t_min_service, t_max_service] = [2,4]	N=50 Q=5 Time limit=60 seconds [t_min_arrival, t_max_arrival] = [2,40] [t_min_service, t_max_service] = [1,7]	N=1000 Q=20 Time limit=200 seconds [t_min_arrival, t_max_arrival] = [10,100] [t_min_service, t_max_service] = [3,9]

For the 3 tests presented in the table above, the results of the simulations are kept in the *Test1.txt*, *Test2.txt* and *Test3.txt* files, found in the git repository. (*Queue Strategy* was used)  
Else, each time a simulation is run, the results are kept in *outputFile.txt* file.

## 6. Conclusions

### Conclusions

Firstly, with the help of this assignment, I have remembered the OOP notions learned last semester and also improved them. In addition, I have “relearned” the basics of polynomial arithmetic.

Secondly, I had to manage my time properly in order to be able to complete until deadline this assignment which represents quite a complex project.

Lastly, I faced and learned from multiple scenarios when the code I wrote wasn’t doing exactly what I was expecting it to do and, therefore, I had to debug it, tool which can be seen as crucial when programming.

**What I have learned from this assignment:** working with threads and how to ensure thread safety

**Future developments:** queue management system where there exists the possibility to have *closed* queues (queues which clients can’t go to) -> like in real life

## 7. Bibliography

1. Lectures of Fundamental Programming Techniques - S.L. dr. ing. Cristina POP
2. <https://www.geeksforgeeks.org/>
3. <https://stackoverflow.com/questions>
4. <https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/atomic/AtomicInteger.html>
5. <https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/BlockingQueue.html>
6. <https://www.java67.com/2017/07/how-to-sort-arraylist-of-objects-using.html>
7. <https://www.codespeedy.com/how-to-change-jframe-background-color-in-java/>