DOCUMENTATION

ASSIGNMENT 2

NUME STUDENT: UTIU MONICA-IULIA
GRUPA: 30423

CONTENTS

1. Assignment objective

Design and implement a queues management application which assigns clients to queues such that the waiting time is minimized.
Queues are commonly used to model real world domains. The main objective of a queue is to provide a place for a "client" to wait before receiving a "service". The management of queue-based systems is interested in minimizing the time amount their "clients" are waiting in queues before they are served. One way to minimize the waiting time is to add more servers, i.e., more queues in the system (each queue is considered as having an associated processor) but this approach increases the costs of the service supplier.

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) and $tservice$ (time interval or duration needed to serve the client; i.e. waiting time when the client is in front of the queue). The application tracks the total time spent by every client in the queues and computes the average waiting time. Each client is added to the queue with the minimum waiting time 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 ($tsimulationMAX$);
- Minimum and maximum arrival time ($tarrivalMIN \leq tarrival \leq tarrivalMAX$);
- Minimum and maximum service time ($tserviceMIN \leq tservice \leq tserviceMAX$);

To achieve this objective, our secondary objectives are:

- Model and implement our primary structures queues and tasks/clients
- Model and implement the business logic and assign their tasks/ responsibilities (scheduler – in charge of scheduling tasks on queues according to criterion for example)
- Properly synchronize executions for correct and deterministic output
- Provide an user interface that should allow user to setup and start simulation and give results and real-time evolution of queues
- Provide a log of events on the queues
- Provide a clean, intuitive and easy to use application for the user

2. Problem analysis, modelling, scenarios, use-cases

General overview

The application simulates customers waiting to receive a service (e.g. supermarket, bank, etc.). Like in the real world, they have to wait in queues, each queue processing clients simultaneously and in parallel. The idea is to analyze how many clients can be served in a certain simulation interval, by entering parameters in an intuitive, user-friendly, application graphical interface.

Input and Output

The customers are generated randomly, each having it's own id, service time and arrival time. How many clients are generated, depends on the input values.

The user can set:

- The maximum number of queues available to process customers.
- Minimum and maximum arrival interval: the delay between customers arriving to receive a service. When generating clients, the arrival time will be chosen randomly based on these values.
- Minimum and maximum service time: the number of units of time needed for a client to be processed, a value is chosen randomly.
- Simulation duration: the finishing time of the simulation.

The user can read:

- How many customers were served during the simulation interval.
- The average service time of the served customers.
- The average waiting time of the served customers (in minutes): how much the customers have waited in queue to receive the service.
- The peak hour, when the most clients were served amongst the maximum clients at a time

Additional details and statistics can be read in the detailed log, in the graphical interface and in the file "out.txt".

Data Structures

To assure synchronization, the blocking queue is used for the tasks which are tied to the servers (queues) which represent a thread each. Atomic Integer is used for the same reason. Other data structures used are Lists, ArrayLists for other common data that doesn't need synchronization.

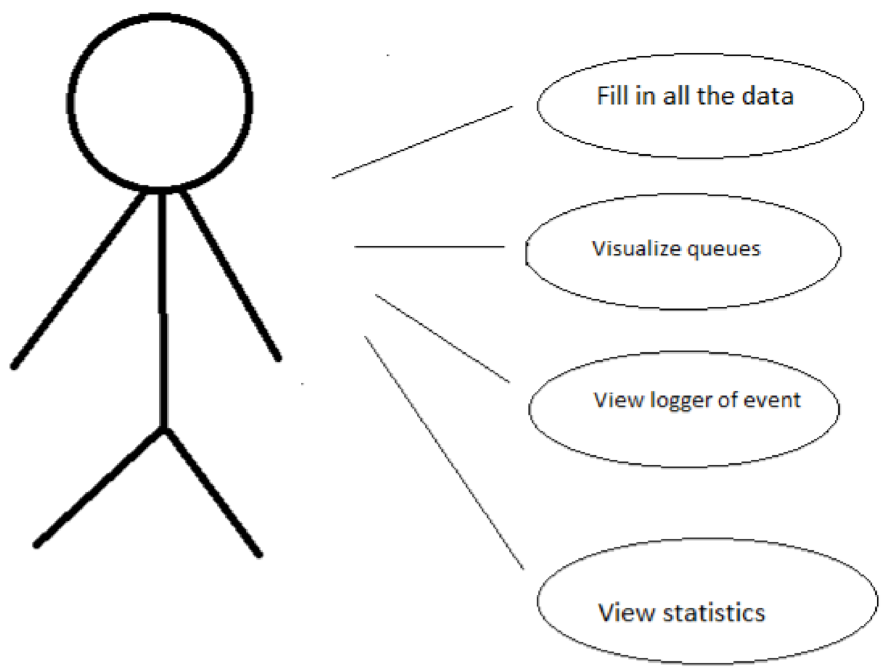
Algorithm

Clients, that represent tasks, will arrive at the 'shop' in a random manner and will have random needs for service. Clients will be generated randomly and having random needs. Each client will be assigned a client id, which will serve for tracking the evolution of the queues.

Each task will be distributed to the queue which corresponds to its needs by a Simulation Manager. This means we will need a connection between the available queues(servers) and the corresponding tasks chosen taking in account the Selection Policy – which could be Shortest Time or Shortest Queue. The manager activates the servers which solve different tasks and puts the tasks in them.

Example: Client arrives with different needs that sum up to a certain amount of time. He will be put in a queue that matches our criterion. If on one else has arrived in this queue, he will be served, otherwise he must wait for those in front of him to finish.

Use Cases



Title: Queue simulation execution
Resume: The first thing that a user has to do is to insert in the graphical user interface all the needed parameters for the application to work as it should, the ones are the minimal and the maximum arriving time and serving time, the simulation interval, the number of queues.
Actors: The user
Scenarios:
Preconditions:
The user must introduce all the data mentioned before, I assume that all the introduced data is correct (meaning I'm not using regex to check if there are only digits), but they are checked to follow certain conditions(for example arrivalMin>0, arrivalMin<arrivalMax) . If the data is not correct the user will be notified by a pop-up that the input is not correct.
Normal Scenario**:**
The user has perfectly introduced all the required data inputs and it presses the "Start Simulation" button, after this the application is displaying the log and the real time evolution of

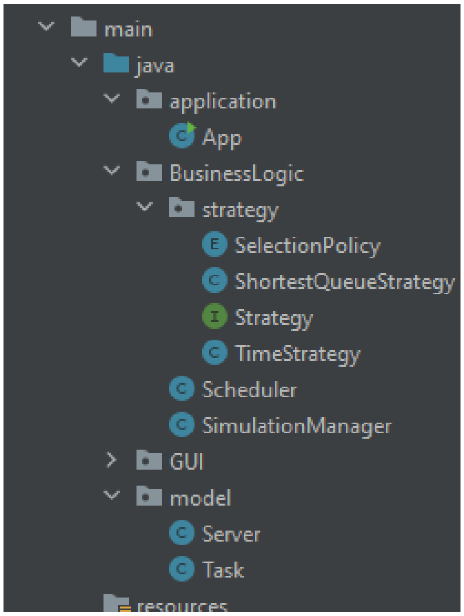
the queues. After this they can look in the minimal output area to see the data which has been calculated during the serving process

Alternative Scenario:

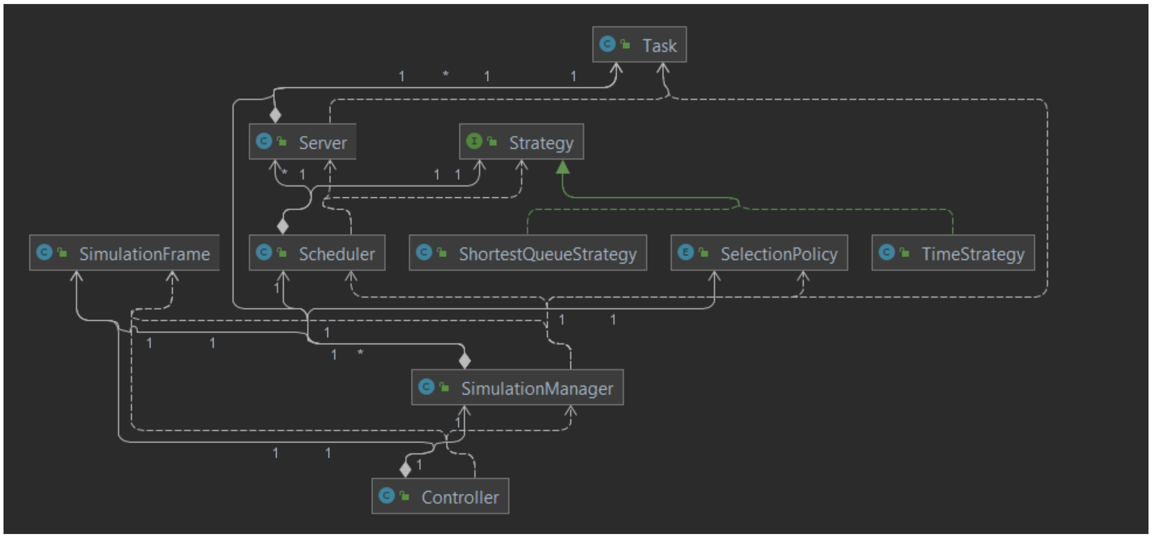
If the user doesn't fill in all the input boxes an error message will be displayed in order to announce the user and not let the simulation start mind him to do so.

3. Design

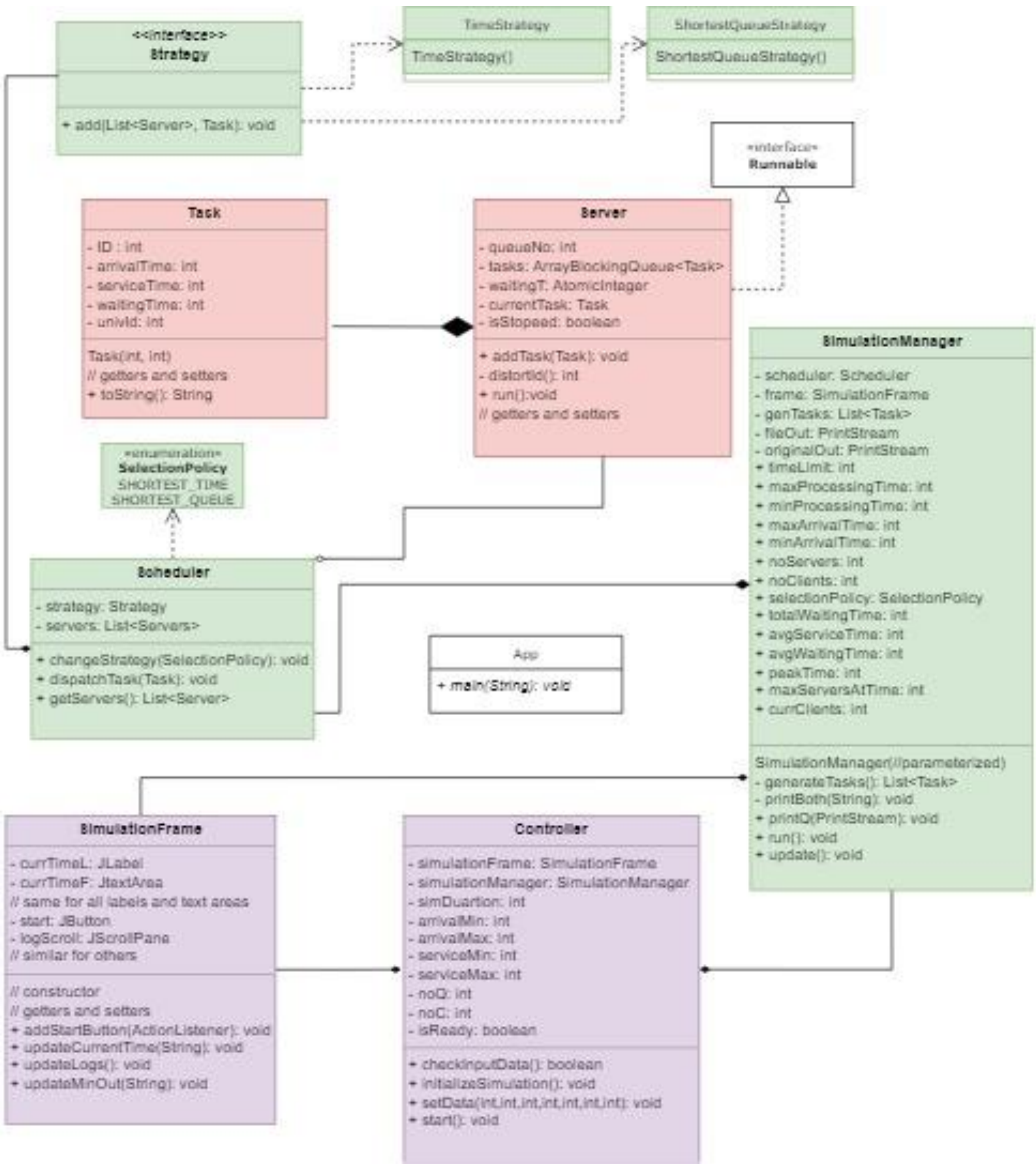
Package structure



UML Diagram – dependencies



- **Class diagram**



4. Implementation

My project is based on Model – View – Controller Pattern, so I split my classes into four packages:

- application – contains a single class, which contains the customary main() method
- BussinessLogic – contains the packages and the classes preoccupied with the strategy, schedule and simulation manager
    - strategy

        - Strategy Interface
        - ShortestQueueStrategy + ShortestTimeStrategy – extend Strategy
        - SelectionPolicy – an enumeration for defining the strategy
    - Scheduler – sends the tasks to the queues according to the strategy
    - SimulationManager – generates the random tasks, contains the simulation which is in charge of current time, calls the scheduler to dispatch tasks, updates UI
- model – contains the "brain" of the application, the classes which model the problem
        - Task – the model of the client
        - Server – implements the Runnable interface
- GUI
    - view – contains a single class which represents the GUI (SimulationFrame)
    - control – it interconnects the model and the view

The model – contains the logic of the application

- Task
    - contains data about the customer, such as id, arrival time and service time;
    - it has a constructor with parameters;
    - it also contains getters and setters

- Server
    - this class implements the Runnable interface and it contains, among other attributes, a list of clients that arrived at that queue in the form of an ArrayBlockingQueue for synchronization.
    - it has a method to add the task to the queue and takes care of the waitingTime
    - it overrides the run method which takes the tasks in queue to be processed

The BusinessLogic – contains all the business logic

- The Strategy package – has an interface extended by the two Strategies, which add the Task in a certain order and an enumeration
- Scheduler
    - Contains a list of servers and a strategy
    - Has a constructor that initializes all the needed queues (servers) and sets strategy
    - Has a method that dispatches tasks
- SimulationManager
    - Contains the list of parameters for the simulation
    - Contains the scheduler, simulation frame, generated tasks and a print stream for my txt log
    - Has a parameterized constructor initializing all of those, initializes scheduler, output file and saves the original output and calls to the generate tasks method
    - Has a method that randomly generates tasks (according to parameters)
    - Overrides the run method to get the tasks and update the queues from the GUI and the logs

The view – it contains the code for the graphical user interface of the application and extends JFrame

- SimulationFrame

- Contains a constructor which "builds" the GUI, it initializes all the labels, text fields, text areas and buttons
- To make my interface very user friendly I used 5 queues and a logging area which display the evolution of the simulation, the queues are updated for each increment of current time (the atomic integer), the same information can be seen in the log area and text file, but in written format
- It also contains methods that update the queues, the logs and the output statistics and numerous getters for the initial simulation parameters
- It contains a button, "Start Simulation" which starts the application.

The controller – this contains the linking between the model and the view

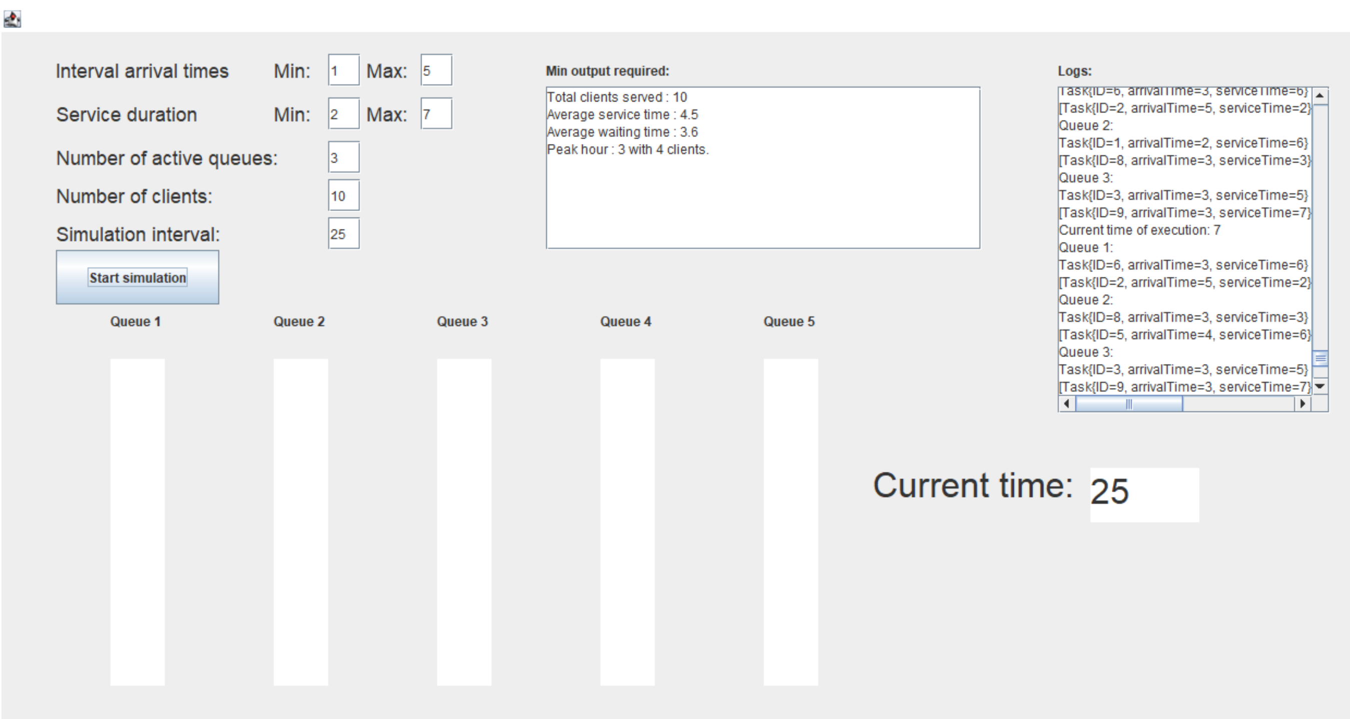
- Controller
  - This is a very important class because it acts on both model and view. It controls the data flow into model object and updates the view whenever data changes.
  - Some methods present here are the ones that read the initial parameters of the simulation from the user interface and fetch them to our controller.
  - Is my main class containing the simulation manager itself which holds and interacts with the others
  - Validates data, sets it and initializes simulation when button is actioned

The application – this is used to start the application

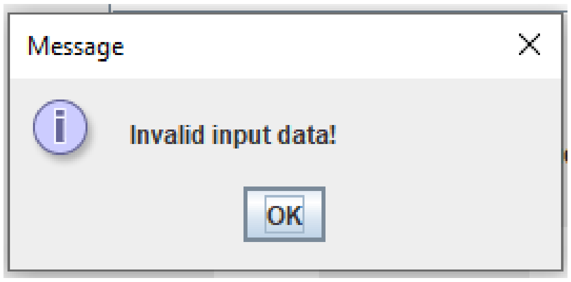
- App
  - Contains the main() method which starts the controller and thus the application.

5. Results

- Example of GUI look after a simulation



- Example of wrong input



- Example of the log file (out.txt) – portion of it

```
Current time of execution: 0
Queue 1: []
Queue 2: []
Queue 3: []
Current time of execution: 1
Queue 1: Task{ID=4, arrivalTime=1, serviceTime=6}[]
Queue 2: []
Queue 3: []
Current time of execution: 2
Queue 1: Task{ID=4, arrivalTime=1, serviceTime=6}[]
Queue 2: Task{ID=3, arrivalTime=2, serviceTime=7}[]
Queue 3: []
Current time of execution: 3
Queue 1: Task{ID=5, arrivalTime=3, serviceTime=7}[]
Queue 2: []
Queue 3: Task{ID=8, arrivalTime=3, serviceTime=2}[]
Current time of execution: 4
Queue 1: Task{ID=5, arrivalTime=3, serviceTime=7}[]
Queue 2: Task{ID=1, arrivalTime=4, serviceTime=3}[]
Queue 3: Task{ID=8, arrivalTime=3, serviceTime=2}[]
Current time of execution: 5
Queue 1: Task{ID=5, arrivalTime=3, serviceTime=7}[]
Queue 2: Task{ID=1, arrivalTime=4, serviceTime=3}[Task{ID=7, arrivalTime=5, serviceTime=2}, Task{ID=9, arrivalTime=5, serv
Queue 3: Task{ID=6, arrivalTime=5, serviceTime=6}[Task{ID=10, arrivalTime=5, serviceTime=4}]
```

Test Input

- N = 4 Q = 2 $tsimulation$
  MAX = 60 seconds
  [$tarrival\ MIN$,$tarrival\ MAX$] = [2, 30]
  [$tservice\ MIN$,$tservice\ MAX$]= [2, 4]

```
Current time of execution: 0
Queue 1: []
Queue 2: []
Current time of execution: 1
Queue 1: []
Queue 2: []
Current time of execution: 2
Queue 1: Task{ID=4, arrivalTime=2, serviceTime=3}[]
Queue 2: []
Current time of execution: 3
```

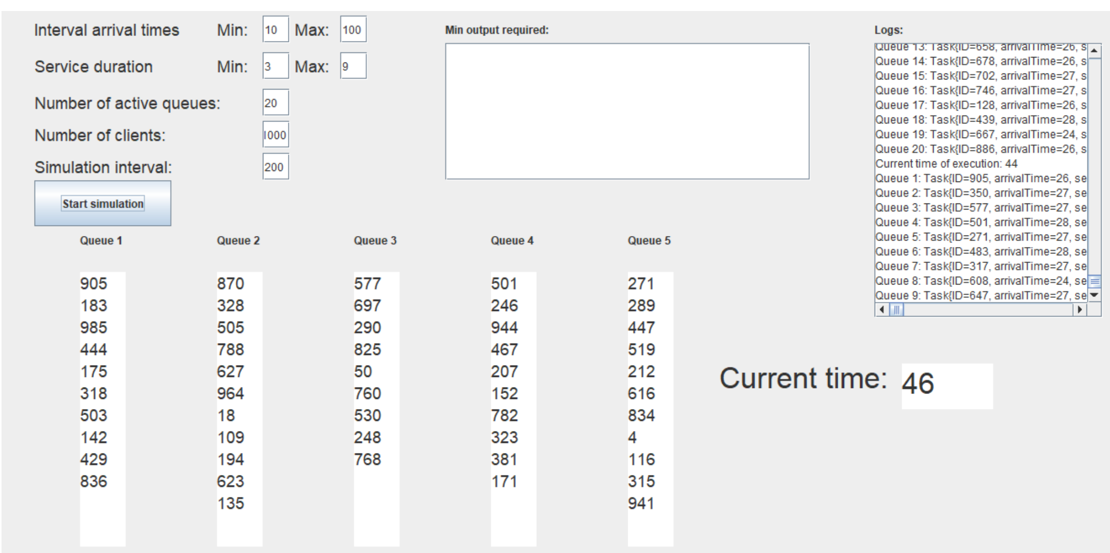
```
Queue 1: Task{ID=4, arrivalTime=2, serviceTime=3}[]
Queue 2: Task{ID=2, arrivalTime=3, serviceTime=2}[Task{ID=3, arrivalTime=3,
serviceTime=2}]
Current time of execution: 4
Queue 1: Task{ID=4, arrivalTime=2, serviceTime=3}[Task{ID=1, arrivalTime=4,
serviceTime=3}]
Queue 2: Task{ID=2, arrivalTime=3, serviceTime=2}[Task{ID=3, arrivalTime=3,
serviceTime=2}]
Current time of execution: 5
Queue 1: Task{ID=1, arrivalTime=4, serviceTime=3}[]
Queue 2: Task{ID=3, arrivalTime=3, serviceTime=2}[]
Current time of execution: 6
Queue 1: Task{ID=1, arrivalTime=4, serviceTime=3}[]
Queue 2: Task{ID=3, arrivalTime=3, serviceTime=2}[]
Current time of execution: 7
Queue 1: Task{ID=1, arrivalTime=4, serviceTime=3}[]
Queue 2: []
Current time of execution: 8
Queue 1: []
Queue 2: []
```

- After this its just empty

- N = 50 Q = 5
  $tsimulation\ MAX$ = 60 seconds
  [$tarrival\ MIN$,$tarrival\ MAX$] = [2, 40]
  [$tservice\ MIN$,$tservice\ MAX$]= [1, 7]

  Total clients served : 50
  Average service time : 3.82
  Average waiting time : 3.14
  Peak hour : 26 with 10 clients.

- N = 1000 Q = 20
  $tsimulation\ MAX$ = 200 seconds - blocks at 46(txts on computer)
  [$tarrival\ MIN$,$tarrival\ MAX$] = [10, 100]
  [$tservice\ MIN$,$tservice\ MAX$]= [3, 9]



For the output "out.txt" file I used a PrintStream to print to the file, whilst also using a PrintStream to keep the original output stream. To transmit to the Logs on the interface I used a FileReader which references the output file and has a LineNumberReader to get and append to the Logs text field area.

6. Conclusions

This project was a good exercise in remembering the OOP concepts learned in the first semester and this one, but also learning about threads and thread management, which is a rather difficult thing.

Further development:

- show an animation of customers waiting in queues, moving to other queues, and leaving (after being processed)
- develop an algorithm the chooses when to open and close queues based on the input parameters, to allow a better distribution of the customers
- implement functionality to pause and resume the simulation before the simulation time reached the simulation end value.
- Fix the bugs that appear in some cases

7. Bibliography

https://www.draw.io
https://www.youtube.com/watch?v=O_Ojfq-OIpM
https://zetcode.com/javaswing/swingmodels/
https://lostechies.com/gabrielschenker/2009/01/23/synchronizing-calls-to-the-ui-in-a-multi-threaded-application/
http://stackoverflow.com/