DOCUMENTATION

QUEUE MANAGEMENT SYTEM

2ND ASSIGNMENT

STUDENT NAME: IAZ ANIA MEDEEA

GROUP:30424

Content

1.Theme objective

Main 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.

Some sub-objectives of the project are: analyzing the problem and identifying the requirements, designing the simulation application, implementing the simulation application and testing the simulation application.

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 $t_{\text{simulation}}$) 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), $t_{arrival}$ (simulation time when they are ready to enter the queue) and t_{service}(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 $t_{arrival}$ time is greater than or equal to the simulation time ($t_{arrival} \geq t_{\text{simulation}}$).

2.Problem analysis, modeling, scenarios, use cases

2.1 Problem analysis

In this problem a certain number of clients are generated randomly and placed in a queue where their service will be processed, after which they will leave.

At first sight the problem may seem pretty easy to implement, but the truth is the problem is more complex than one may think. Because we want the client to wait as little as possible when they are placed in a queue we need to consider some strategies to help us. There would be two: either we place the client to the queue with the least clients, either we place him or her to the queue with the smallest waiting time. I have chosen the latest to implement the management system. Besides this strategy, we need to consider multithreading so that all the queues can work concomitant. This will help with the time performance and will resemble more how queues work in real life.

Another thing to consider is how the user will interact with the graphical user interface. He or she needs to introduce some details about the system like simulation time, number of clients and others. To make sure that everything works as it should I validated the data with the help of a regex. If a mistake is found an error message is displayed on the screen and the simulation will not start until the data is valid.

2.2 Modeling

The system design can be seen as information introduced by the user: number of clients, number of queues, simulation time, minimum arrival time, maximum arrival time, minimum service time and maximum service time. The program will generate N random clients and Q servers representing the queues. At each second the evolution of the program will be seen on the graphical user interface, until the time is equal with the simulation time introduced.

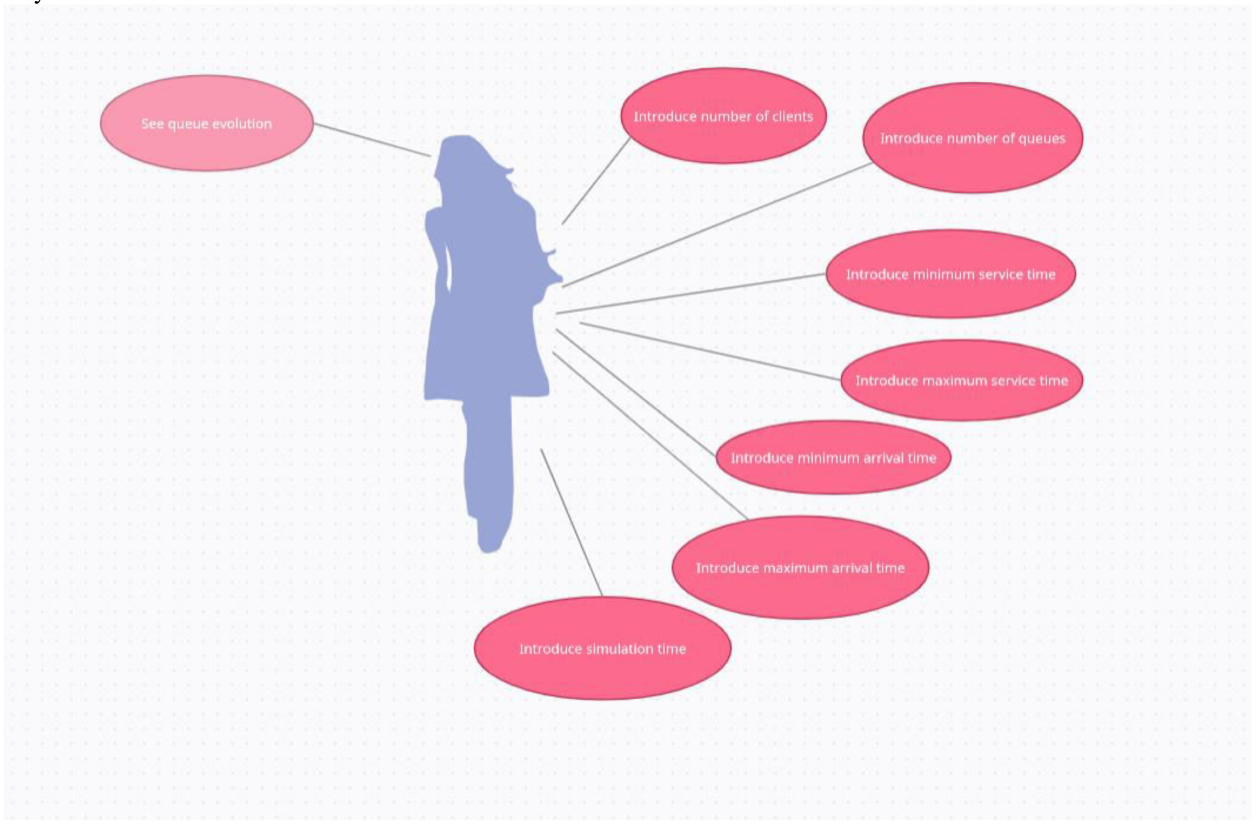
This program is based on the concept of queues which has a FIFO policy- first in, first out. Classes Task and Server will hold information about clients and queues. The class Task corresponds to the clients where a unique ID, a random arrival time and service time, which represents how long the client needs assistance, will be given to each of them. Every queue has a thread linked so that all of them to work at the same time. The main thread is associated with the simulationManager that starts all the process.

2.3 Scenarios

Since it may be possible that the user will introduce wrong data, some error messages will appear on the screen. Even if in my simulation I will use 3 exact tests to show that my application is working I made it functional for the cases in which the user will introduce any numbers he or she pleases.

2.4 Use Cases

The use case diagram can be a pleasant way to visualize the interaction between the user and the system that he or she may use. It also shows what the user needs to do in order to get the right results, giving a general description of the system.


See queue evolution
Introduce number of clients
Introduce number of queues
Introduce minimum service time
Introduce maximum service time
Introduce minimum arrival time
Introduce maximum arrival time
Introduce simulation time

As it can be seen in the picture above the actions will take place in the following order:

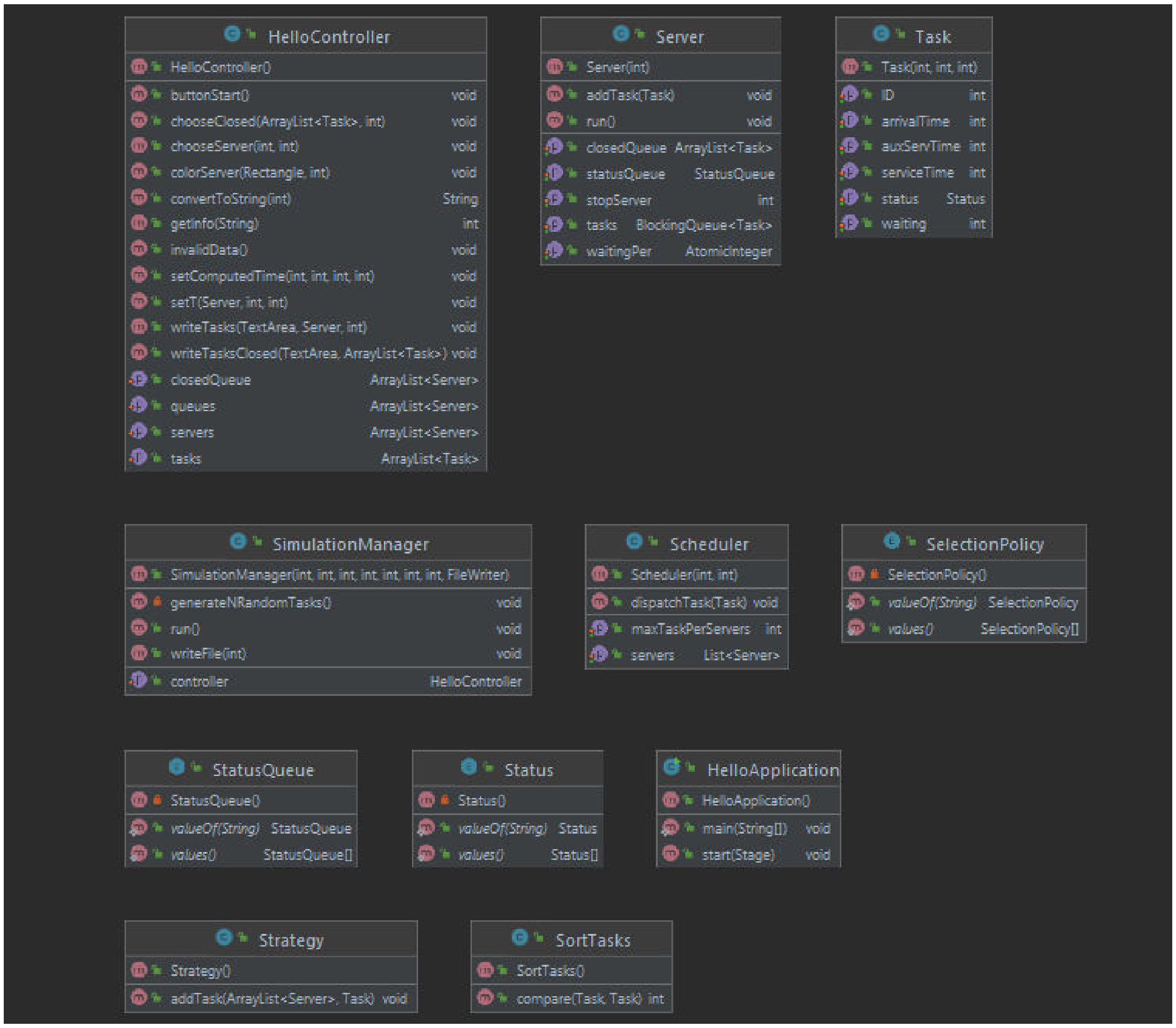
- Introduce all the information that is needed for the application
  - Simulation time
  - Number of clients
  - Number of queues
  - Minimum arrival time
  - Maximum arrival time
  - Minimum service time
  - Maximum service time
- Press the start simulation button
- Visualize the evolution of each queue

3. Design

For the design I chose to use the MVC pattern for a better modularization of the project. This design specifies that an application consists of data model, presentation information, and control information. The pattern requires that each of these be separated into different objects.

For the model part of the pattern I have encapsulated the classes Task and Server which contain information about the clients, respectively about the queues. The controller, named HelloController, is situated in the package BusinessLogic. This is the part that controls the view and sends the information to the model, being a connection between the model and the view. If the correct input is provided, after the user presses the button “Start Simulation” the controller will create an object of type SimulationManager together with the associated thread which will launch the threads for the queues. At each second of the simulation, the result is displayed on the graphical interface and can be easily seen how the process goes. The other classes that belong to the BusinessLogic package are: Scheduler, SortTask and Strategy. The view is represented by a .fxml file that can be found in the resources. The role of the view is to communicate with the user.

Below we have the UML diagram of the classes.



HelloController	
HelloController()	
buttonStart()	void
chooseClosed(ArrayList<Task>, int)	void
chooseServer(int, int)	void
colorServer(Rectangle, int)	void
convertToString(int)	String
getInfo(String)	int
invalidData()	void
setComputedTime(int, int, int, int)	void
setT(Server, int, int)	void
writeTasks(TextArea, Server, int)	void
writeTasksClosed(TextArea, ArrayList<Task>)	void
closedQueue	ArrayList<Server>
queues	ArrayList<Server>
servers	ArrayList<Server>
tasks	ArrayList<Task>

Server	
Server(int)	
addTask(Task)	void
run()	void
closedQueue	ArrayList<Task>
statusQueue	StatusQueue
stopServer	int
tasks	BlockingQueue<Task>
waitingPer	AtomicInteger

Task	
Task(int, int, int)	
ID	int
arrivalTime	int
auxServTime	int
serviceTime	int
status	Status
waiting	int

SimulationManager	
SimulationManager(int, int, int, int, int, int, int, FileWriter)	
generateNRandomTasks()	void
run()	void
writeFile(int)	void
controller	HelloController

Scheduler	
Scheduler(int, int)	
dispatchTask(Task)	void
maxTaskPerServers	int
servers	List<Server>

SelectionPolicy	
SelectionPolicy()	
valueOf(String)	SelectionPolicy
values()	SelectionPolicy[]

StatusQueue	
StatusQueue()	
valueOf(String)	StatusQueue
values()	StatusQueue[]

Status	
Status()	
valueOf(String)	Status
values()	Status[]

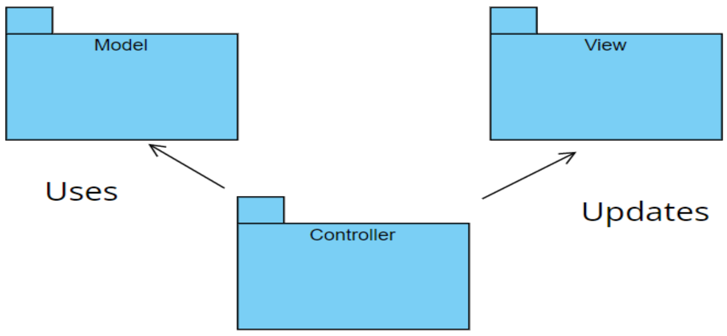
HelloApplication	
HelloApplication()	
main(String[])	void
start(Stage)	void

Strategy	
Strategy()	
addTask(ArrayList<Server>, Task)	void

SortTasks	
SortTasks()	
compare(Task, Task)	int

It cand be easily seen that the view is missing due tot the fact that it is a .fxml file and not a java class. Also the class that contains the main is missing, its only purpose being to launch the application and set the title, width and length of the screen that will be displayed.

The general package diagram it is shown below:



This diagram shows how the model, view and controller communicate with each other. The view displays the information obtained from the model to the user. The model encapsulates data and functionality and the controller receives input, events that denote a mouse click. These events generate a request that is sent to the model or to the view.

In my case the Controller is name BusinessLogic and the view is not an actual class, but a .fxml file.

- **Data structures**

A new type of data structure that I used for my project is a Blocking Queue. This kind of data structure is thread safe, the reason why I chose to have it in my application. I mostly used the functions add, take and peek in order to get the information needed or to erase it after the service is done.

4. Implementation

o Model Classes

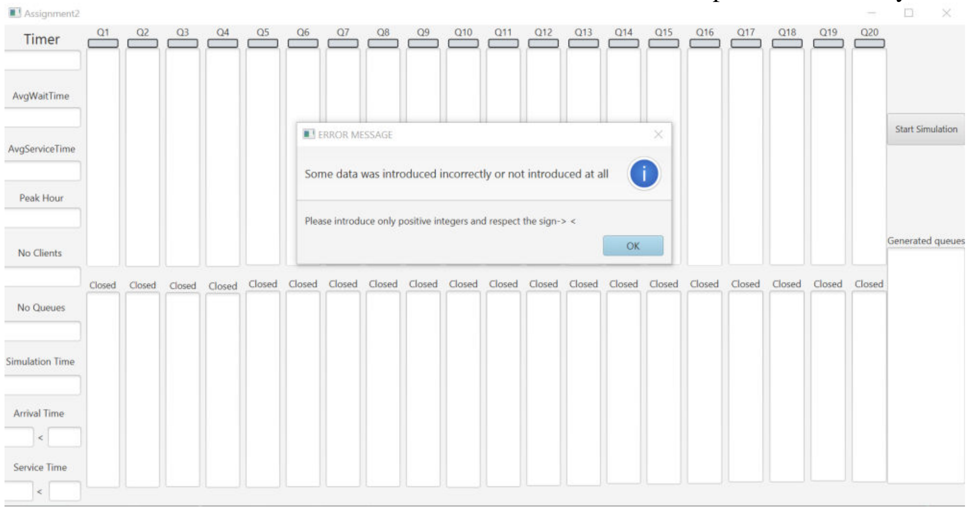
The main classes from the Model package are the Task class and Server class. In the Task class I have six private attributes: the ID, the arrivalTime, the waitingTime, the status, the serviceTime and the auxServTime. The latest I use for displaying the task that ended on the interface since I decrement at each second the service time. The status refers to the status of the client. The clients can be active, which means he is being served, pending, which means he is waiting in the line to be served, or inactive when his or her serviceTime ended. I have one constructor to which I pass all the argument needed to initialize the attributes. Besides the constructor, this class only contains getters and setters for the private fields.

The Sever class is more complex since it has a thread associated to it. In the run method of the thread some decisions are being made: from the queue the first task is taken in order to decrement the service time until it is zero and add the task to the closedQueue ArrayList. This process repeats until all the tasks are added to the closedQueue attribute of each server. This class contains as attributes an atomic Integer named waitingPer, a BlockingQueue tasks, a statusQueue and a stopServer, which refers to the condition of the while loop for the run method. The statusQueue cand be active, stand by or inactive. This attribute is set from the beginning as inactive, until a task is added. How the statusQueue behave it is decided in the run method.

o BussinesLogic Classes

In this package I have five classes. Besides the HelloController and SimulationManager classes which can be considered as main classes, I have some secondary classes which mostly help me to initialize the queues and their threads, sort the clients by the arrival time or choose to which queue to add the new client, depending on the waiting period of each queue.

The class HelloController is the one that links the model with the view. Here I have declared every component that I need to modify or from which I need to get the information. Some notable components are the rectangles that represent the front desk that will color to green when the queue is active, to blue when is in the stand by phase or will remain gray if the queue is never activated. Mostly the code consists of the declaration of the components and which one to choose when I need to write something on the graphical interface. In addition to these methods I have one that will create an error screen if the information is not provided correctly.


Assignment2
Timer
AvgWaitTime
AvgServiceTime
Peak Hour
No Clients
No Queues
Simulation Time
Arrival Time
<
Service Time
<
Q1 Q2 Q3 Q4 Q5 Q6 Q7 Q8 Q9 Q10 Q11 Q12 Q13 Q14 Q15 Q16 Q17 Q18 Q19 Q20
Closed Closed Closed Closed Closed Closed Closed Closed Closed Closed Closed Closed Closed Closed Closed Closed Closed Closed Closed Closed
Start Simulation
Generated queues
ERROR MESSAGE
OK

Some data was introduced incorrectly or not introduced at all

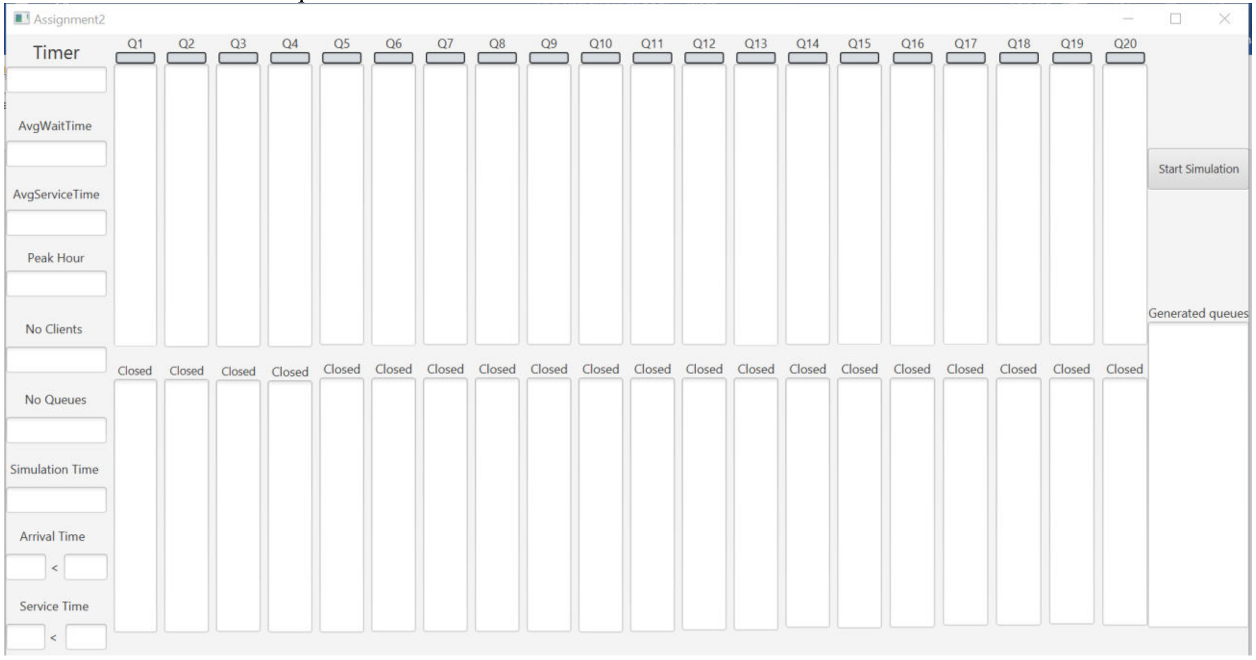
Please introduce only positive integers and respect the sign-> <

The class SimulationManager contains the main thread that launches the others. Here I have declared all the attributes that I need to get from the interface or write on it. Apart from them I have a list of tasks that I randomly generate with the help of the function Math.random, a scheduler and a variable of type HelloController in order to access the methods for writing on the interface. I have two constructors, one that I use to initialize all the attributes declared and one that has in plus a file as arguments used to write the results of the simulation in it.

```
for (int i = 0; i < numberOfClients; i++) {
    aux1 = (int) (Math.random() * (maxArrivalTime - minArrivalTime + 1) +
minArrivalTime);
    aux2 = (int) (Math.random() * (maxServiceTime - minServiceTime + 1) +
minServiceTime);
    Task t = new Task(i + 1, aux1, aux2);
    task.add(t);
}
Collections.sort(task, new SortTasks());
```

- **Graphical User Interface**

As it can be seen in the picture below, I have one button to start the simulation, eleven text fields that will either display an information or either needs to be completed by the user. For the queues I have twenty text areas that have an automatic scroll bar. In addition, I have added a text area that contains all the generated tasks to be clear which client will be added next in the queues.



5. Results

The result can be seen in the .txt files under the name of three different tests- tes1, test2, test3. All three scenarios were tested and can be followed for each second that passes in the simulation.

6. Conclusions

From this project I learned how to work with threads and the specific data structures for them in order to have everything synchronized.

As limitation of the project or further improvements a reset button could be added, as well as new queues added dynamically for depending on the number the user introduced. Another feature that could be added is an abord button in order to stop the simulation, but not exit from it.

Having to work with this kind of application will surely help with a future job, where most likely for the implementation of such system threads will be needed.

7. Bibliography

- https://regexr.com/
- https://stackoverflow.com/
- https://app.creately.com/d/pfulq18TWJw/edit
- https://online.visual-paradigm.com/diagrams/features/package-diagram-software/