DOCUMENTATION

ASSIGNMENT *2*

STUDENT NAME: Săvianu Bogdan-Casian
GROUP: 30223

CONTENTS

1. Assignment Objective

The main objective of this assignment was to implement an application that finds out the shortest average waiting time for the customers in the lines of a supermarket.

The objective was reached by modelling each queue as its own thread working independently from the others and implementing two different strategies for adding the customers to the queues, the shortest time method and the shortest queue one, which is used in case there are multiple queues with the minimum waiting time. Other smaller objectives were logging the events of the simulation in a .txt file and having a dynamic GUI that shows the contents of the queues at all times.

2. Problem Analysis, Modeling, Scenarios, Use Cases

After careful consideration I modeled my problem as a list of clients and a list of servers, each server containing a queue to which clients are added according to the chosen strategy.
The way a use case would take place is:
-the simulation time reaches the arrival time of a client;
-we find the queue with the shortest waiting time;
-if multiple, we then switch the strategy to the shortest queue one among the one found at the previous step;
-we add the client to the suitable queue and add the sum of the service times of the clients in front of him to the average time;
-after the client's service time reaches 0, we remove it from the queue;
-we let the simulation finish and then divide the average time by the total number of clients in the simulation and we have our answer.

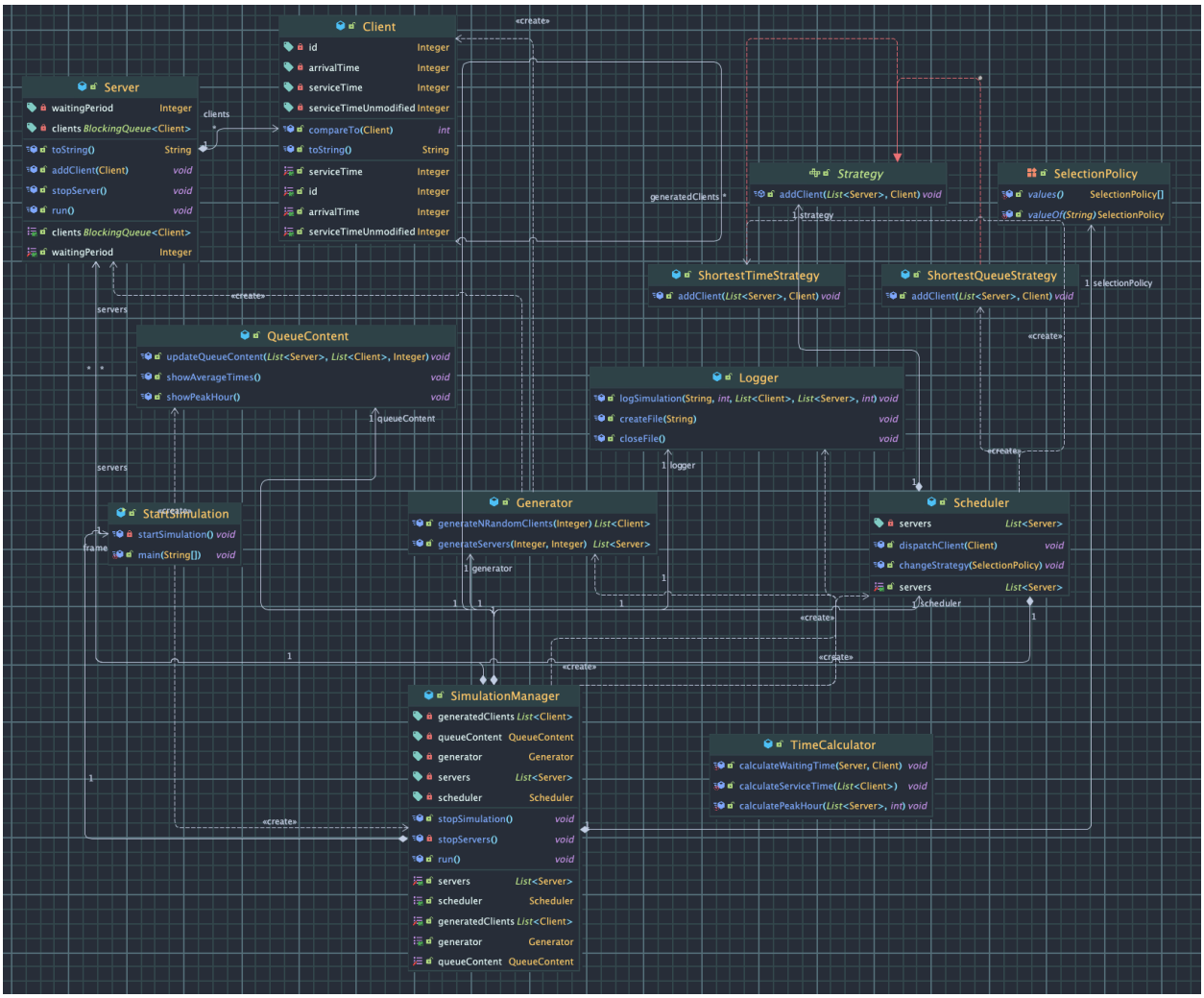
3. Design

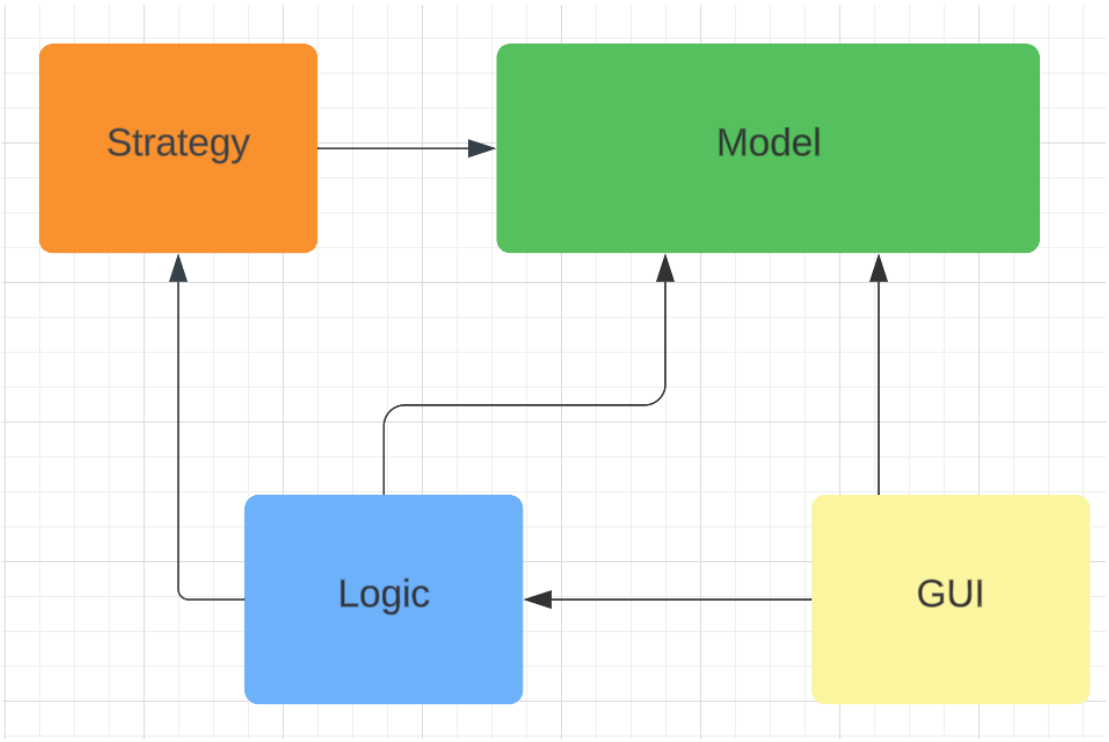
I started the design by modelling the problem into to Client and Server classes, then I created the strategies to add the clients to servers. In order to select the correct strategy I created the Scheduler class that determines which strategy is to be used and dispatches the given client to the correct method of addition to the queue.

After these steps were done I was able to implement the SimulationManager class and test everything up to this point. Then, the next thing I did was implement a way to write the events in a .txt file.

Once all things worked fine it was time for the GUI. I have two different GUI classes, one is the one where I add the details of the simulation, such as the time limit, processing times, etc. The other is for displaying the content of the queues at the current time.

At the end, I created another class responsible for calculating the average waiting/service times and the peak hour and I display those at the end of the simulation.


Client
id Integer
arrivalTime Integer
serviceTime Integer
serviceTimeUnmodified Integer
compareTo(Client) int
toString() String
serviceTime Integer
id Integer
arrivalTime Integer
serviceTimeUnmodified Integer
«create»
Server
waitingPeriod Integer
clients BlockingQueue<Client>
toString() String
addClient(Client) void
stopServer() void
run() void
clients BlockingQueue<Client>
waitingPeriod Integer
clients
generatedClients
Strategy
addClient(List<Server>, Client) void
1 strategy
SelectionPolicy
values() SelectionPolicy[]
valueOf(String) SelectionPolicy
ShortestTimeStrategy
addClient(List<Server>, Client) void
ShortestQueueStrategy
addClient(List<Server>, Client) void
1 selectionPolicy
«create»
servers
QueueContent
updateQueueContent(List<Server>, List<Client>, Integer) void
showAverageTimes() void
showPeakHour() void
1 queueContent
Logger
logSimulation(String, int, List<Client>, List<Server>, int) void
createFile(String) void
closeFile() void
1 logger
StartSimulation
startSimulation() void
main(String[]) void
frame
Generator
generateNRandomClients(Integer) List<Client>
generateServers(Integer, Integer) List<Server>
1 generator
Scheduler
servers List<Server>
dispatchClient(Client) void
changeStrategy(SelectionPolicy) void
servers List<Server>
1 scheduler
SimulationManager
generatedClients List<Client>
queueContent QueueContent
generator Generator
servers List<Server>
scheduler Scheduler
stopSimulation() void
stopServers() void
run() void
servers List<Server>
scheduler Scheduler
generatedClients List<Client>
generator Generator
queueContent QueueContent
TimeCalculator
calculateWaitingTime(Server, Client) void
calculateServiceTime(List<Client>) void
calculatePeakHour(List<Server>, int) void



4. Implementation

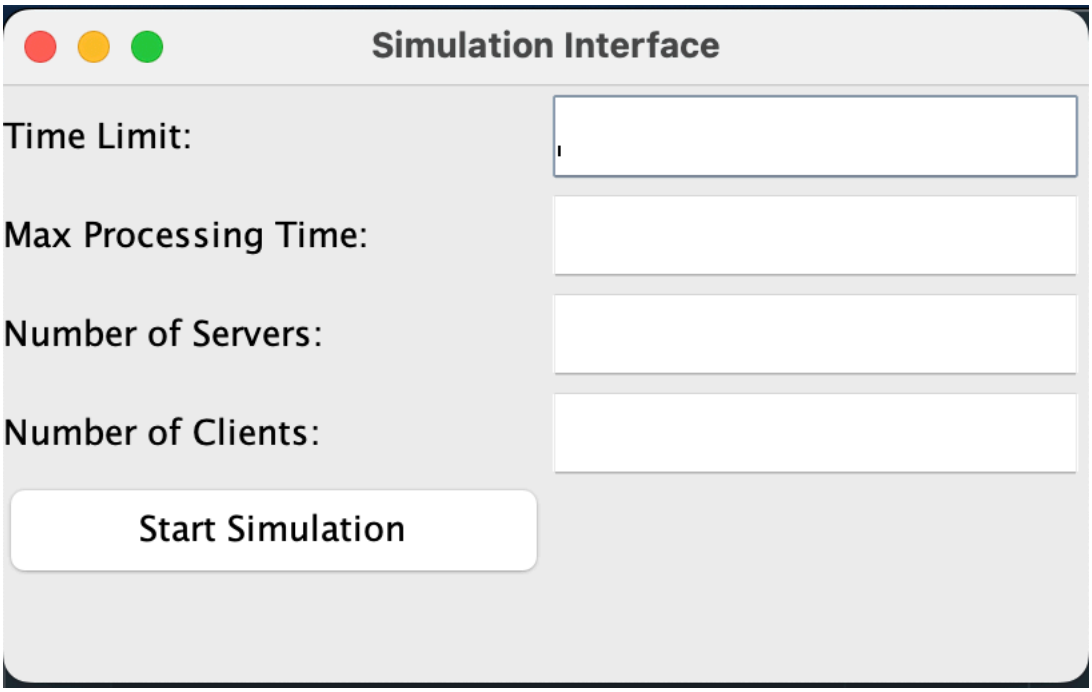
Client - three fields: id, arrivalTime, serviceTime;
Server – BlockingQueue<Client>, Integer waitingPeriod, bool isRunning;

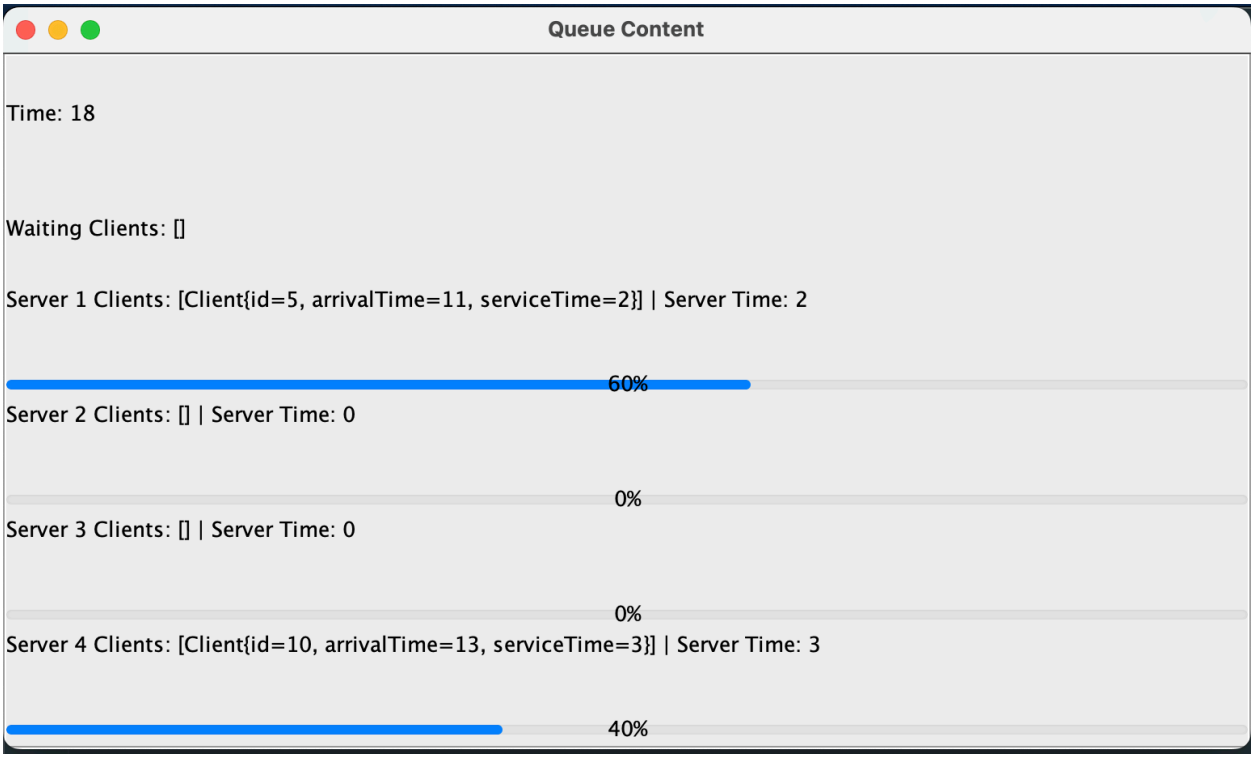
- has a method addClient and the run method where it decreases the service time of the leading client and removes him when the time reaches 0;

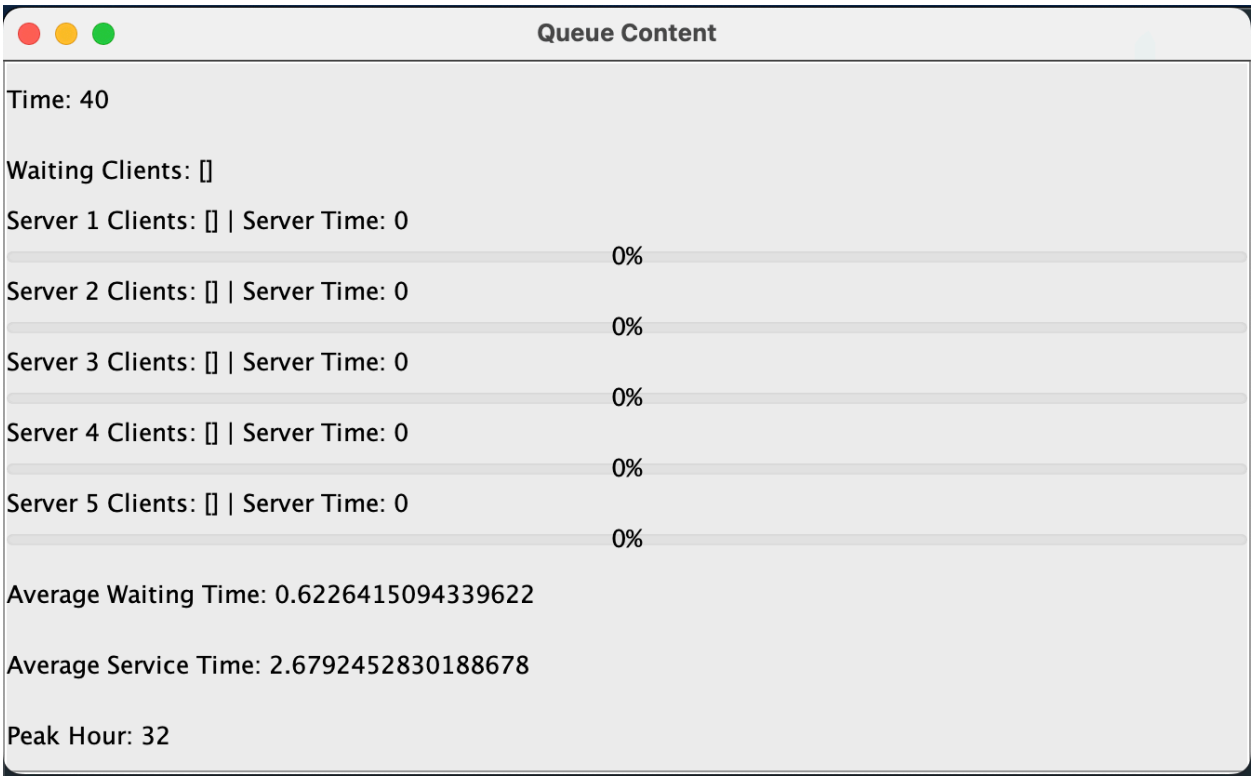
Generator – generates the random clients and the necessary servers;
Strategy interface – abstract method addClient;
ShortestTime/QueueStrategy – implement the addClient method from the interface accordingly
SelectionPolicy – enum used to choose the strategy;
Scheduler – chooses the appropriate strategy and dispatches the clients;
TimeCalculator – calculates the average waiting and service times and the peak hour;
Logger – writes the events to a file
SimulationManager – stores information about the clients and queues at all times

- is used by the logger and the GUI to get their information

StartSimulation – this is the GUI class where the information is entered and then sent to the SimulationManager to do its job.
QueueContent – shows the contents of the queues and updates the interface at each interval of time.


Simulation Interface
Time Limit:
Max Processing Time:
Number of Servers:
Number of Clients:
Start Simulation


Queue Content
Time: 18
Waiting Clients: []
Server 1 Clients: [Client{id=5, arrivalTime=11, serviceTime=2}] | Server Time: 2
60%
Server 2 Clients: [] | Server Time: 0
0%
Server 3 Clients: [] | Server Time: 0
0%
Server 4 Clients: [Client{id=10, arrivalTime=13, serviceTime=3}] | Server Time: 3
40%



This last one is at the end when the simulation is finished. It shows the average waiting/service times and the peak hour.

5. Results

I tested my application with the suggested tests and a lot of tests of my own and it behaves as expected.

5. Results

I have conducted many tests: first, I tested the validation functionality and performance by introducing invalid data. Secondly, I tested various simulation cases, calculating the expected output manually and comparing it with the output generated by the application.

Additionally, I've conducted the three main tests presented in the assignment:

Test 1	Test 2	Test 3
N = 4 Q = 2 $t^{MAX}_{simulation}$ = 60 seconds $[t^{MIN}_{arrival}, t^{MAX}_{arrival}]$ = [2, 30] $[t^{MIN}_{service}, t^{MAX}_{service}]$= [2, 4]	N = 50 Q = 5 $t^{MAX}_{simulation}$ = 60 seconds $[t^{MIN}_{arrival}, t^{MAX}_{arrival}]$ = [2, 40] $[t^{MIN}_{service}, t^{MAX}_{service}]$= [1, 7]	N = 1000 Q = 20 $t^{MAX}_{simulation}$ = 200 seconds $[t^{MIN}_{arrival}, t^{MAX}_{arrival}]$ = [10, 100] $[t^{MIN}_{service}, t^{MAX}_{service}]$= [3, 9]

In the end, all of my tests have passed successfully, indicating that the functionality is working as expected.

6. Conclusions

The most important conclusion I gathered from working on this assignment is that working with threads is harder than a lot of people would think and there are so many things that could go wrong. Also when getting unexpected behavior it is harder to find the cause because of the parallelism.

All in all, this assignment helped me deepen my understanding of threads by a lot. In the future I would like to add a functionality where a server breaks randomly and all its clients have to be dispatched to the other remaining servers.

7. Bibliography

https://dsrl.eu/courses/pt/
https://www.geeksforgeeks.org/scheduledexecutorservice-interface-in-java/
https://docs.oracle.com/javase/8/docs/api/java/lang/Thread.html