DOCUMENTATION

Assignment 2

STUDENT NAME: SANDOR SERBAN
GROUP: 30422

CONTENTS

1. Assignment objective

The main objective of the assignment is to implement a management system using queues, which assign clients to the queues so that the waiting time is minimized.

The management system should simulate a series of N clients which come for services,waiting at Q queues,being served and finally leaving out of the system.All the clients are generated with random attributes when the simulation starts having the following parameters: ID (a number between 1 and N), arrivalTime (the time when they are ready to be queued), and serviceTime (how long it takes to be served). The application will count the total time spent by each client in the queue and will achieve the average waiting time.

2. Problem Analysis, Modeling, Scenarios, Use Cases

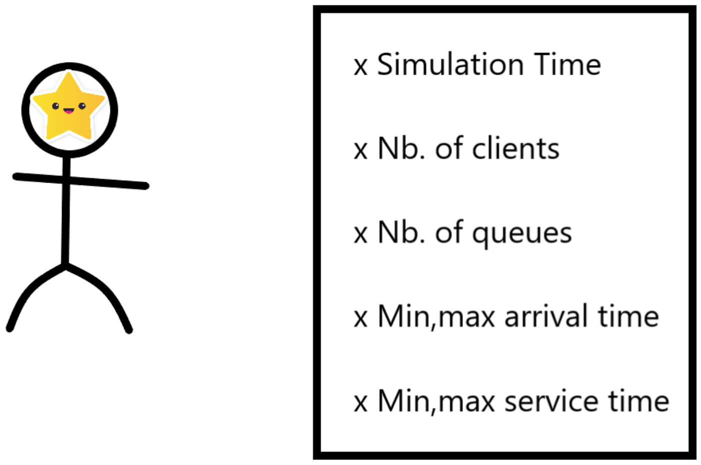
In order to achieve our objectives, we need to meet several functional requirements:

- The management system must be able to receive as input the data desired by the user, such as: the number of customers, the number of queues, the simulation time of the store, and the minimum and maximum time interval for coming / serving the customers
- The system must recognize wrong data received by the client and notify him through a message

We can also meet non-functional requirements:

- The interface must be user-friendly
- The user does not have to do unnecessary operations (e.g. extra buttons)

The system must display the simulation of the store in a clean and easy way to read



Use – Case Diagram :
Success scenario:

1. 1. The user can successfully insert the management system data.
2. The system simulates the store desired by the customer and displays the data in the specific text field, present near the data entry area.

Alternative scenario :

1. The user enters the wrong input
2. The system displays an error message informing the user of the mistake, and resets the input boxes

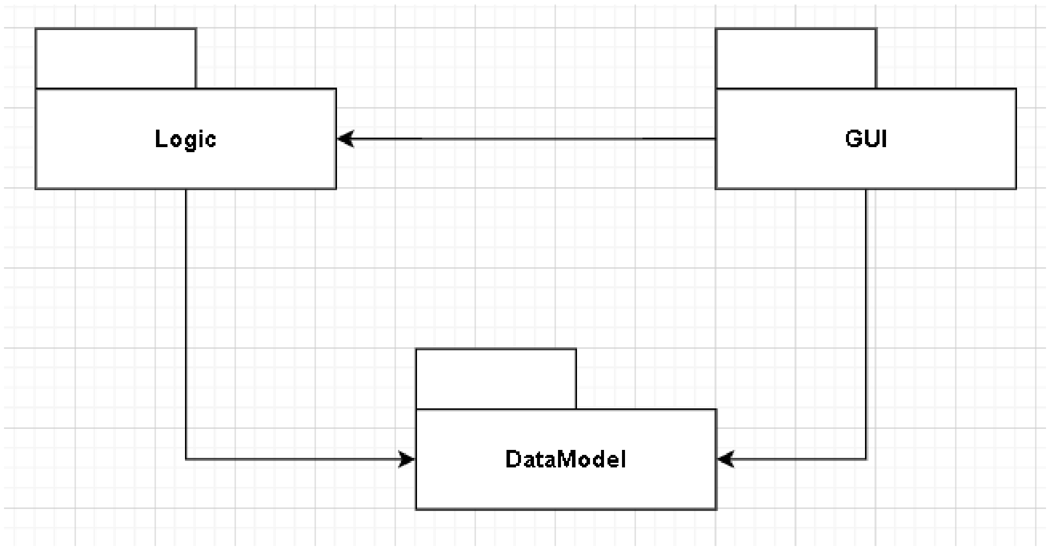
3. Design

1. The OOP design of the application.

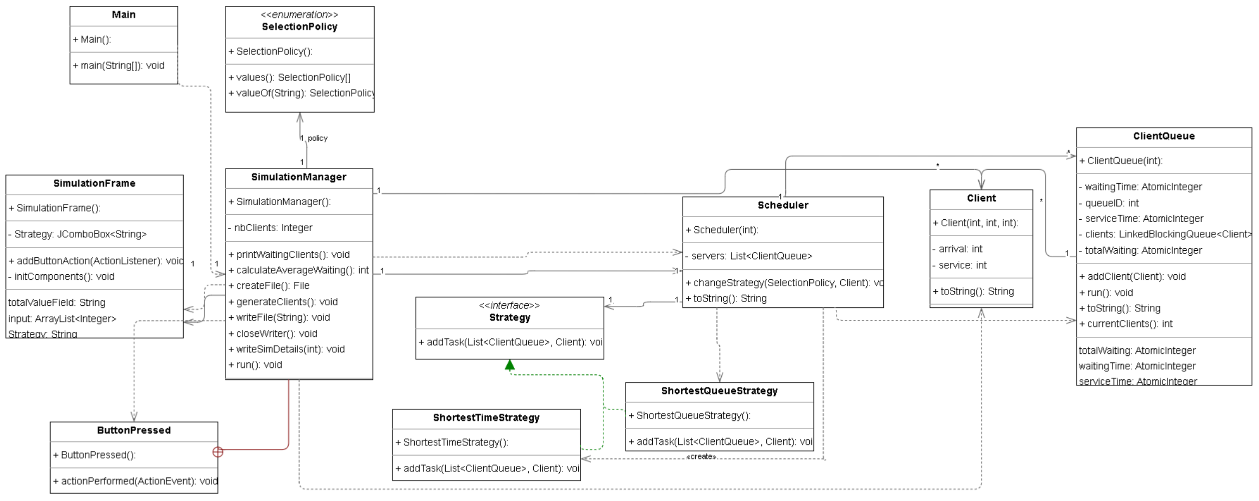
In the OOP design of the application, using the Swing graphical interface, the data provided by the input user will be retrieved, and will display the result after pressing an enter button. The interface is simplified as much as possible, to be easily understood by the user.

2. Presentation of packages and classes through UML diagrams

Thus, we used an architectural interface, having the BusinessLogic – Model – GUI packages. In the DataModel there are the basic classes, necessary to perform the necessary operations on them – Client, ClientQueue. In Logic the algorithmic logic of the program is realized – Scheduler, SimulationManager. In GUI are the classes related to GUI interface – SimulationFrame.



The classes and the relationships between them can be seen in the UML diagram below:


Main
+ Main():
+ main(String[]): void
<<enumeration>>
SelectionPolicy
+ SelectionPolicy():
+ values(): SelectionPolicy[]
+ valueOf(String): SelectionPolicy
1 policy
SimulationFrame
+ SimulationFrame():
- Strategy: JComboBox<String>
+ addButtonAction(ActionListener): void
- initComponents(): void
totalValueField: String
input: ArrayList<Integer>
Strategy: String
SimulationManager
+ SimulationManager():
- nbClients: Integer
+ printWaitingClients(): void
+ calculateAverageWaiting(): int
+ createFile(): File
+ generateClients(): void
+ writeFile(String): void
+ closeWriter(): void
+ writeSimDetails(int): void
+ run(): void
Scheduler
+ Scheduler(int):
- servers: List<ClientQueue>
+ changeStrategy(SelectionPolicy, Client): void
+ toString(): String
Client
+ Client(int, int, int):
- arrival: int
- service: int
+ toString(): String
ClientQueue
+ ClientQueue(int):
- waitingTime: AtomicInteger
- queueID: int
- serviceTime: AtomicInteger
- clients: LinkedBlockingQueue<Client>
- totalWaiting: AtomicInteger
+ addClient(Client): void
+ run(): void
+ toString(): String
+ currentClients(): int
totalWaiting: AtomicInteger
waitingTime: AtomicInteger
serviceTime: AtomicInteger
<<interface>>
Strategy
+ addTask(List<ClientQueue>, Client): void
ShortestQueueStrategy
+ ShortestQueueStrategy():
+ addTask(List<ClientQueue>, Client): void
ShortestTimeStrategy
+ ShortestTimeStrategy():
+ addTask(List<ClientQueue>, Client): void
«create»
ButtonPressed
+ ButtonPressed():
+ actionPerformed(ActionEvent): void

4. Implementation

Next, I will describe each class with the important fields and methods, and I will explain the implementation of the user interface.

1. <u>Client</u>
- - Instance variables: ID, arrivalTime, serviceTime, all int type
- Important methodsL
    - toString : will display in the interface the task in the form (id, arrivalTime, serviceTime);
2. <u>ClientQueue</u>
   - Instance variables: waitingTime , serviceTime, totalWaiting(all AtomicInteger),clients(LinkedBlockingQueue<Client>),queueID(int)

   Important methods :
    - addTask(Task t) : when called, will insert a new client in the list of tasks and will update waitingTime
    - run() : the class expands the Thread class and overwrites the run() method to work with threads, thus simulating the queues. In this case, when calling the method by each thread will find clients with queues that aren't served and decrements the waiting time by 1.
3. <u>Scheduler</u>
- - Instance variables:servers(List<ClientQueue>), threads(List<Thread>).
- Important methods:
    - Scheduler (int maxNoServers) : adds a queue to a list of queues and a thread for that queue to a list of threads and starts it.
    - changeStrategy(SelectionPolicy policy,Client t):we add the clients to a queue based on the strategy that we selected(shortest time or shortest queue).
4. <u>SimulationManager</u>
- Impotant methods:
    - SimulationManager (): creates a text file and a writer for the file.We also start the GUI and the button for beginning the simulation.
    - setValues(ArrayList<Integer> inputs):here we allocate values to our variables based on the inputs given in the application window.

- generateClients():we generate clients with random attributes using the Math.random function and add them to a list of clients.
- calculateAverageWaiting():we calculate the average waiting time based on the total waiting time and the number of queues.
- writeFile(String string):writes in the text file and puts the result the in the result field.
- writeSimDetails: writes the average waiting time,the average service time and the peak hour of the simulation in the text file.
- printWaitingClients(): writes the waiting clients in the text file and also adds a client to a queue based on the policy and if its arrival time is smaller or equal to the time of the queue.
- run() : synchronized method that starts all threads and calls by default the run method of the Server class for each queue. He takes care of organizing the queues according to the simulation time offered by the user, and knows how to call the tasks to the queues when the atomic time variable is equivalent to the arrival time of the task. Also, depending on the serviceTime and arrivalTime of each task, constantly update the waitingTime and serviceTime of the class, which will represent the average values of the store. Meanwhile, use the StringBuilder object to build the message that will be displayed on the screen for the user (and copied to the txt file), where he will be able to constantly see the time, tasks waiting in line (and their information), and the tasks at each queue, being able to observe how their service time decreases.

5. <u>SimulationFram</u>

- Instance variables:
  - MinMaxArr,MinMaxSer,NrClients,NrQueues,Result,SimTime,StartButton – all of the JTextField type: fields where the user enters the data necessary to start the application
  - StartButton – JButton : when pressing it, call the SimulationManager class that will take the information entered and will start the simulation .
  -

  Impotant methods:

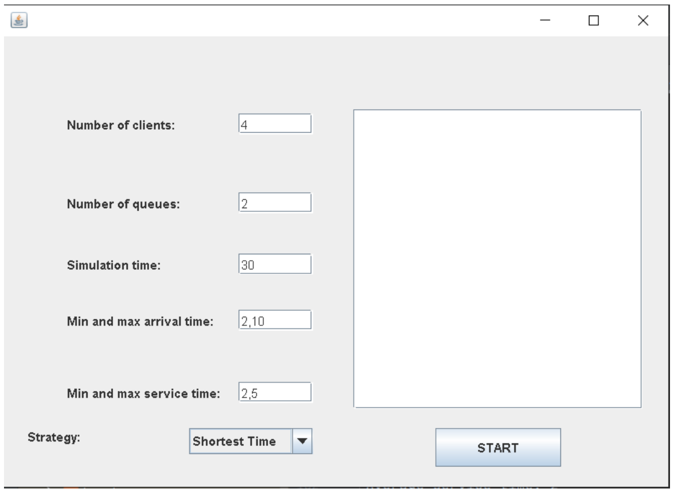
  - showMessage(String message) – if the try-catch block of the Action Listener method in the controller throws an expression, it will display an error message notifying the user that it has entered a wrong input, and will call the refresh function
  - intitComponents():initializes the fields ,labels of the interface.
  - ArrayList<Integer> getInput:gets the the values from the fields and returns them as inputs.
  - getClientsField() => Int – returns the number entered by the user that specifies the number of desired clients in the simulation
  - addButtonAction(ActionListener actionListener) – notify the program when the Start button was pressed
  - setTotalValueField(String resultField1) – will set the field for the result with the string formed above in the SimulationManager class, to give the user the opportunity to view the simulation.

6. <u>Main</u>

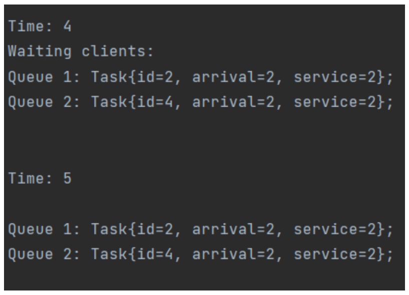
initializes an object from the SimulationManager class, thus starting the GUI interface

5. Results

Next, I will present the scenarios for testing:



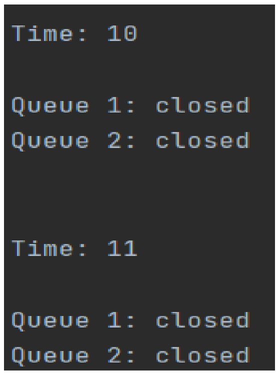
The customer correctly entered the desired data for the simulation of the system. It will press the enter button, and in the application will be displayed real-time the status of the store and queues, depending on the input entered.



```
Time: 4
Waiting clients:
Queue 1: Task{id=2, arrival=2, service=2};
Queue 2: Task{id=4, arrival=2, service=2};


Time: 5

Queue 1: Task{id=2, arrival=2, service=2};
Queue 2: Task{id=4, arrival=2, service=2};
```



```
Time: 10

Queue 1: closed
Queue 2: closed


Time: 11

Queue 1: closed
Queue 2: closed
```

6. Conclusions

In conclusion, the management system using threads was a very interesting project to carry out, helping me to become more familiar with the concept of synchronization in Java and the problems related to it that may arise. The application is not perfect, and in the future I would like to improve it through a cleaner and easier to use interface for the user, and more efficient algorithms for assigning a task to a queue.

7. Bibliography

1. https://app.diagrams.net/

2. FUNDAMENTAL PROGRAMMING TECHNIQUES (dsrl.eu)

3. https://dsrl.eu/courses/pt/

4. http://docs.oracle.com/javase/tutorial/essential/concurrency/index.html

5. https://www.tutorialspoint.com/java/util/timer_schedule_period.htm