DOCUMENTATION

Assignment number 2:

Queue Management System

Student name: Lungoci Toma

Group: 30423

Table of contents

1. Objectives

The main objective is to design and implement a queue managements system implementing efficient queue allocation mechanism. It simulates a series of N clients arriving for service, entering Q queues, waiting, being served and finally leaving the queues, computing the average waiting time, average service time and peak hour. The GUI has been implemented with the JavaFX framework.

Along with the main objective, there are several sub-objectives that must be taken into account. Analysis of the problem along with identifying the requirements, exhibited with the help of use case diagrams and the classification between functional and non-functional requirements. Designing the system into multiple levels of abstraction which resulted in the division in several subsystems such as: Graphical User Interface, Business Logic, and Data Models, and also in structuring the project with the Model View Controller Architectural Pattern in mind. The packages compose the corresponding to provide the necessary functionality. The design of the system is also presented with package and class diagrams. The implementation is done in Java code following the OOP principles. For working with threads in Java, the implementation must use thread-safe data structures and synchronization methods. Some implementation details and code snippets are presented. Finally, the testing of the queue management system is exemplified to several test cases, providing the simulation logs, to ensure the optimal functionality.

2. Problem analysis, modeling, use cases

Functional requirements:

-The simulation application should allow users to setup the simulation

-The simulation application should allow users to start the simulation

-The simulation application should display the real-time queues evolution

-The simulation application should display the real-time waiting clients list evolution

-The simulation application should provide the simulation log in an output text file

-The simulation application should allow the user to stop the simulation, terminating the threads

-The simulation application should display the current time in the simulation thread

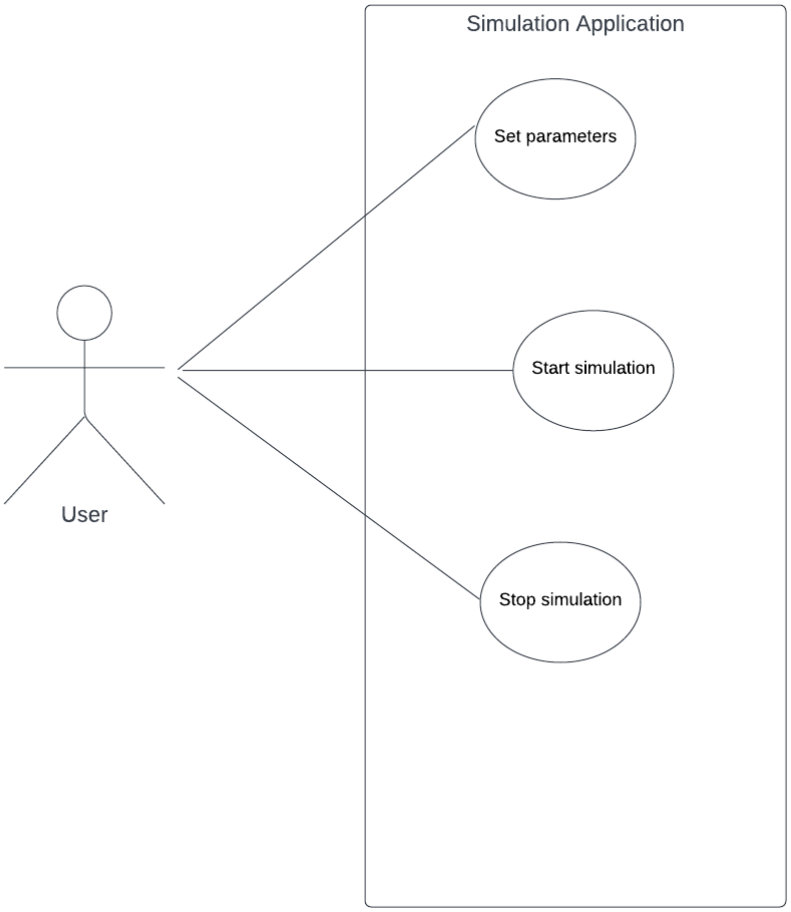
Non-Functional requirements:

-The simulation application should be intuitive and easy to use by the user

-The simulation application should display error messages when the input is invalid

-The simulation application should enable the user to scroll through the text area to see all the values

The following page illustrates the Use Case Diagram of the system.



1. Use case:

Primary Actor: user

Main success scenario:

1) The user sets up the parameters for the simulation
2) The user clicks in the start button from the Simulation Configure part
3) The simulation application starts displaying real-time results (the queues, the time and the waiting clients)
4) The simulation application loads the results in an output text file.

Alternative Sequence:

- Invalid parameters: the values of the parameters do not represent integer values, the simulation cannot start until the parameters are valid
- The user stops the simulation using the Stop button, the simulation stops at the current time, the threads are terminated

2. Use case:

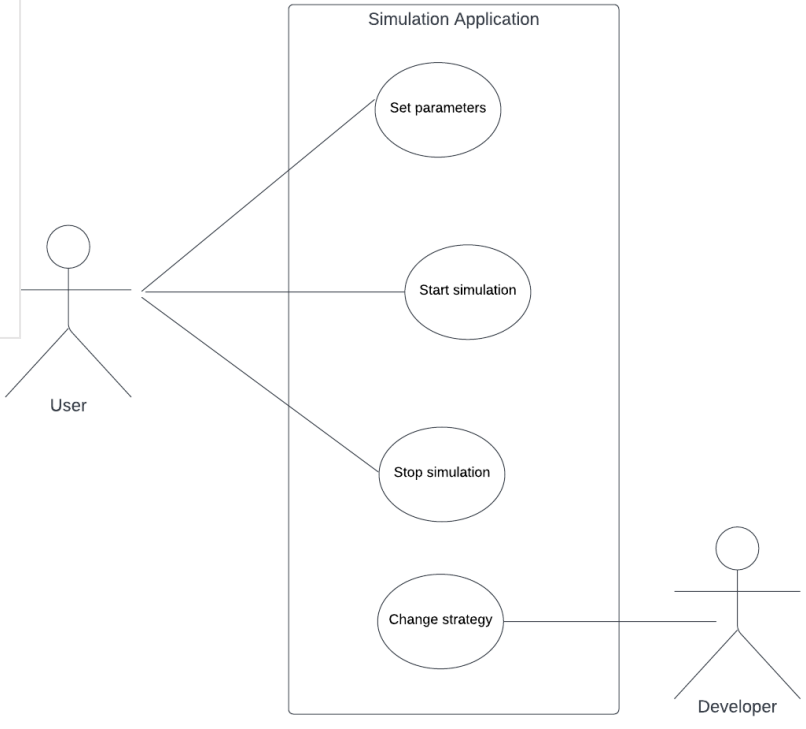
Primary Actor: user

Main success scenario:

1) The user sets up the parameters for the simulation
2) The user clicks in the start button from the Simulation Configure part
3) The simulation application starts displaying real-time results (the queues, the time and the waiting clients)
4) The user stops the simulation at a specific time, blocking the execution to examine the results
5) The simulation application displays the result up to the moment of stopping

Alternative Sequence:

- Invalid parameters: the values of the parameters do not represent integer values, the simulation cannot start until the parameters are valid

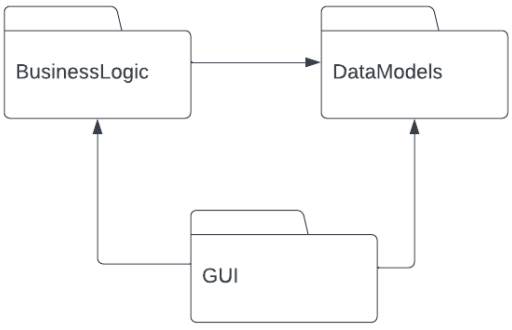


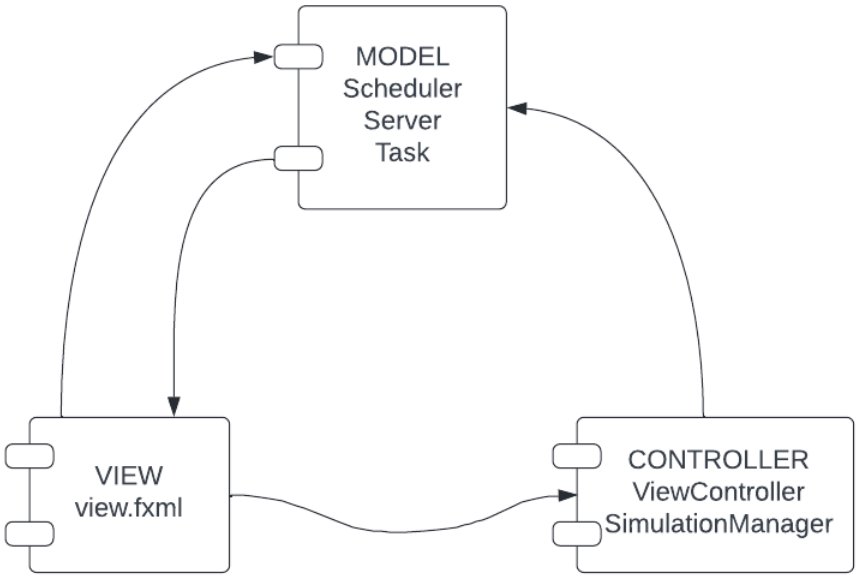
3. Design

- Overall system design

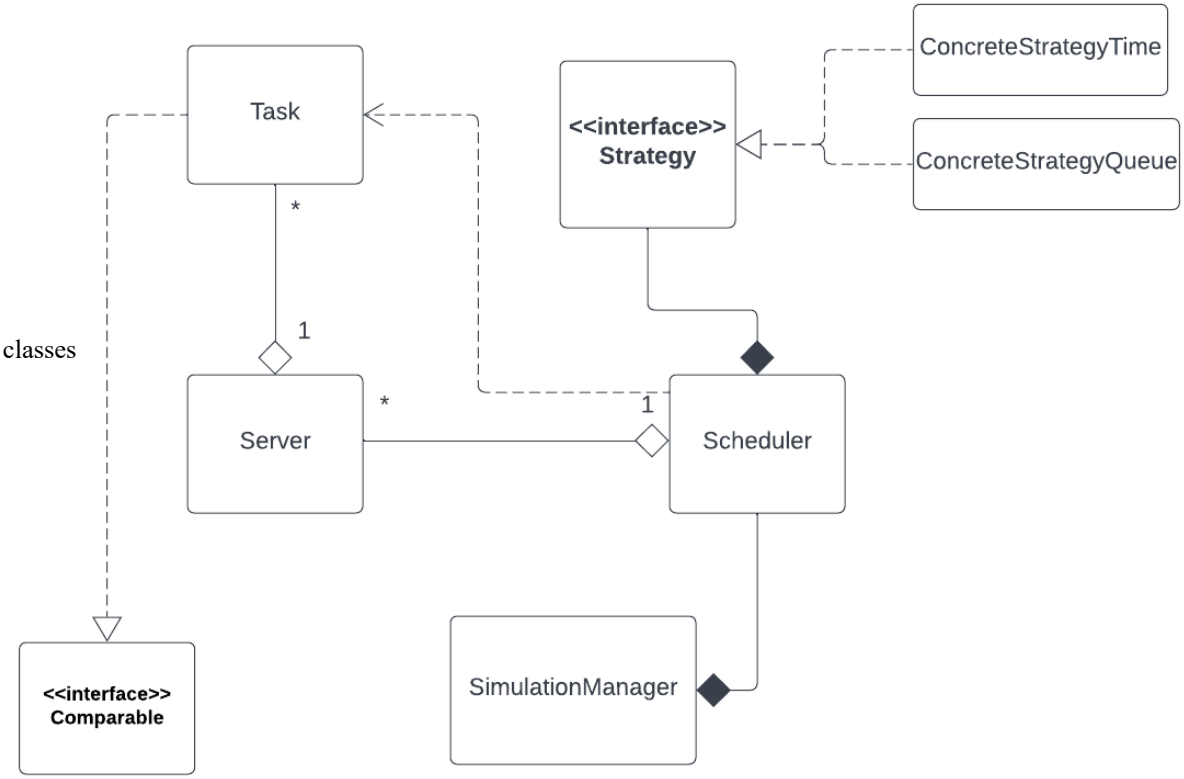
- Division into sub-systems/packages

The DataModel represents the “Scheduler” , “Server” and “Task” classes, which store all the necessary data for manipulating the clients and scheduling them efficiently to different queues. There are some classes which provide the resources for implementing the application’s BussinesLogic: “ConcreteStrategyTime” and “ConcreteStrategyQueue”, both implementing the “Strategy” interface. Along with the strategy, the “SelectionpPolicy” is an enumeration containing the constant identifiers for the two strategies. The GUI is described using the FXML markup language. Following the MVC architecture, along with the View(.fxml) we have the Controller(class “Controller”) which maps the View events to changes in the Model, calling the functions associated with the buttons, labels, etc.. The relationships between the Model and the View represent the request of data.





- Division into classes



The UML Class Diagram illustrated the relations between the classes. The SimulationManager is the main class responsible with starting and setting up the simulation. It is the one creating the Schduler, so there is a strong aggregation between the two. The Scheduler is the only class that uses the SelectionPolicy, again a strong aggregation between the two. A Scheduler creates the Q queues, and stores them in a data structure so a one to many relationship is imposed between Server and Scheduler. Also it contains the method "dispatchTask" which allocates a task to a queue, so it depends on the Task's addTask methos and state. Similarly, a server contains a BlockingQueue of Tasks, imposing the one to many relationship. Weak aggregation has been exemplified since the Tasks and Servers would make sense to exist on their own, without a scheduler object, even though in this application the scheduler is the only one to create servers. The interfaces used are: Comparable for comparing tasks, Strategy with the "addTask"' methos signature, and Runnable for the objects representing threads of execution, that are Servers and SimulationManager.

4. Implementation

The GUI is described in the .fxml file and it contains a AnchorPane, along with buttons and Labels. The input TextFields will provide the strings from which the parameters for the simulation are extracted.

The Controller is the one responsible with mapping events in the view changes in the model and logic. Before that, it checks if all the input parameters are correct. It displays error messages in the contrary case. For example, if the values typed in the text fields do not represent integer numbers, or the minimum arrival time is greater than the maximum arrival time, than the simulation will not proceed. Provided that the parameters are valid and that the user clicks on the Start button, a new instance of a SimulationManager will be created. There is also a Reset button which will erase all your parameter for the previous simulation, to be easier to enter new input data.

```
public int validateInput(){
    try{
        Integer.parseInt(numberOfClientsText.getText());
        Integer.parseInt(numberOfQueuesText.getText());
        Integer.parseInt(simulationIntervalText.getText());
        Integer.parseInt(arrivalTimeMinText.getText());
        Integer.parseInt(arrivalTimeMinText.getText());
        Integer.parseInt(serviceTimeMaxText.getText());
        Integer.parseInt(serviceTimeMinText.getText());
    }catch(NumberFormatException e){
        return -1;
    }
    return 0;
}
```

The SimulationManager will take the values from the UI and set the selection policy. After that, N tasks/clients will be generated randomly, using the generateNRandomTasks. A scheduler object will be created, which has great importance in dispatching the tasks efficiently to servers and also retaining all the valuable information about the system. While the simulation thread is running, the scheduler will provide the communication path between the simulation and the clients, queues. The two selection policies that the scheduler uses are: adding the client to the queue which has the minimum waiting time or adding the client to the queue which is the shortest(least clients). The connection between the simulation manager and the controller is essential, since the UI must be updated each second with info about the current simulation manager. That is the reason for which the simulation manager has a controller instance and calls in the run method the setViewText method, which displays on the screen the real time results,

through the TextArea UI objects, which also pops up a slider so the user is able to scroll through and view all the simulation data. The application is launched in the Main class, via the “launch” method.

```
while (currentTime < timeLimit && run.get())
```

```
int randArrivalTime = random.nextInt(maxArrivalTime - minArrivalTime) +
minArrivalTime;
```

Servers are created within the constructor of the scheduler, each one representing one thread. Since they represent threads the object must implement the “Runnable” interface, and they are started with the thread.start method. The data structure used in the server objects must be thread safe, so the following have been used: BlockingQueue, AtomicInteger and AtomicBoolean. In the overridden Run method the queue is doing the processing of the clients each second, that is the first client in the queue will stay until its service time ends and than it will be eliminated from the queue. The waiting period it is also update in the run method because it is of great importance in the allocation of the clients. Generally, the waiting period gets decremented each second but it is incremented when a new task is allocated to the queue. For waiting one second to process the clients, the Thread.sleep(1000) methos has been used, which imposes the thread to sleep for one second during the simulation. The Stop button is used for stopping the simulation and the threads. Each server object has a AtomicBoolean which is set to false if the user wants to terminate the simulation.

```
Thread thread=new Thread(servers.get(i));
thread.start();
```

Task object is basically the client. It contains basic information, such as the ID, the arrival time and the service time. A client with ID 1, arrival time 2 and service time 1 will be displayed by the simulation in the following format:(1,2,1). Before they are added to queues, the clients are sorted by the arrival time, as in real life clients will arrive at the shop for example. To sort them using Collections.sort(), I have overridden the CompareTo method, in order to automatically sort them by the arrival time. To be able to update the service time as each second passes, there is a decrementServiceTime method, which is called in the run method of the thread. For these objects that represent the data models, there are getters implemented to improve communication between classes.

Presenting the results in an output file is done using a FileWriter object, in the simulation manager thread. Each second of the simulation, the current time, the waiting clients list and the Q queues are displayed. Along with each queue, I have written also the corresponding waiting time at the moment and the size of the queue. For presenting the collections in a String format, I have overridden the toString method for the Task, Server, Scheduler classes. After the termination of the program the statistics are presented (peak hour, average service time, average waiting time). The peak hour is the moment in time when the most clients are enqueued, the average service time is the mean of service times for each client and the average waiting time is calculated using the strategy’s addTask method, accumulating in a variable the waiting time of the queue in which the respective client has been allocated, and then dividing by the number of clients. In the shortest queue strategy, the accumulator will contain the number of clients present in the queue before enqueuing. At the end of the simulation log and then the FileWriter object is closed.

5. Results

Testing the system has been done on multiple scenarios, including the ones presented. The verification can be done by examining the output files, with the queue and waiting list displayed. Down below are presented the three test cases. After the termination of the simulation, the peak hour which is calculated as the specific moment in time in which there were the most clients in all the queues, the average service time and the average waiting time are displayed at the end of the file.

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

SIMULATION LOG

Time: 0

Waiting clients: (1,3,2) (4,15,2) (2,23,3) (3,29,3)

Queue: waiting time->0 size->0

Queue: waiting time->0 size->0

Time: 1

Waiting clients: (1,3,2) (4,15,2) (2,23,3) (3,29,3)

Queue: waiting time->0 size->0

Queue: waiting time->0 size->0

Time: 2

Waiting clients: (1,3,2) (4,15,2) (2,23,3) (3,29,3)

Queue: waiting time->0 size->0

Queue: waiting time->0 size->0

Time: 3

Waiting clients: (4,15,2) (2,23,3) (3,29,3)

Queue: (1,3,2) waiting time->2 size->1

Queue: waiting time->0 size->0

Time: 4

Waiting clients: (4,15,2) (2,23,3) (3,29,3)

Queue: (1,3,1) waiting time->1 size->1

Queue: waiting time->0 size->0

Time: 5

Waiting clients: (4,15,2) (2,23,3) (3,29,3)

Queue: waiting time->0 size->0

Queue: waiting time->0 size->0

Time: 6

Waiting clients: (4,15,2) (2,23,3) (3,29,3)

Queue: waiting time->0 size->0

Queue: waiting time->0 size->0

Time: 7

Waiting clients: (4,15,2) (2,23,3) (3,29,3)

Queue: waiting time->0 size->0

Queue: waiting time->0 size->0

Time: 8

Waiting clients: (4,15,2) (2,23,3) (3,29,3)

Queue: waiting time->0 size->0

Queue: waiting time->0 size->0

Time: 9

Waiting clients: (4,15,2) (2,23,3) (3,29,3)

Queue: waiting time->0 size->0

Queue: waiting time->0 size->0

Time: 10

Waiting clients: (4,15,2) (2,23,3) (3,29,3)

Queue: waiting time->0 size->0

Queue: waiting time->0 size->0

Time: 59

Waiting clients:

Queue: waiting time->0 size->0

Queue: waiting time->0 size->0

Peak hour: 3

Avg service time: 2

Average waiting time: 0

6. Conclusions

The project was very useful for acquiring a more in depth knowledge of the different levels of system design and structure, identifying and classifying the requirements, developing useful UML diagrams to describe the models. I have gained some experience with working with threads, thread-safe data structures, synchronization and managing resources. Implementing the GUI was fun and I have also practiced my OOP programming skills. The project is not perfect but it encapsulates all operations and functionalities. Some further improvements can be made in memory management, efficiency and code reusability, providing the same results with less code.

7. Bibliography

https://www.youtube.com/watch?v=_FSXJmESFmQ

https://www.geeksforgeeks.org/

https://stackoverflow.com/