# DOCUMENTATION

## ASSIGNMENT *ASSIGNMENT_2*

STUDENT NAME: MUREŞAN ANDREI
GROUP: 30425_2

## CONTENTS

# 1. Assignment Objective

## 1.1. Main Objective

Implement a queue system manager that assigns clients to proper queues, depending on one of the selected policy: shortest queue or shortest time in each queue. All queues must be a different thread.

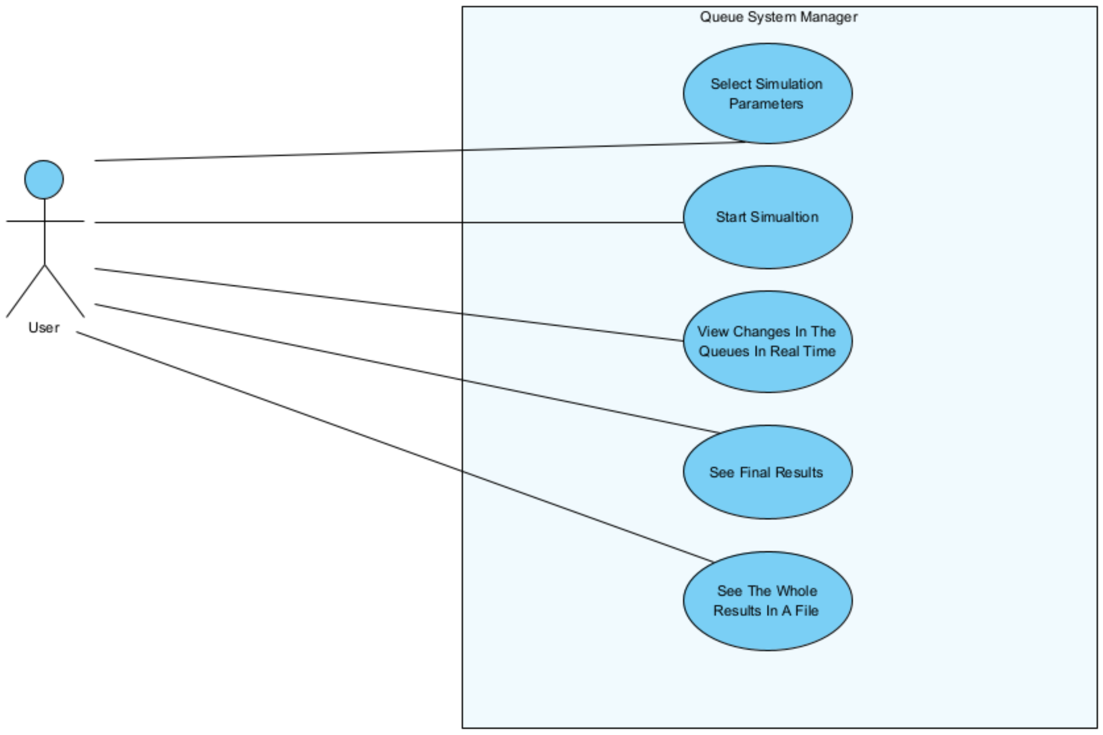
## 1.2. Sub-objectives

| Objective | Importance | Addressed | Details |
|---|---|---|---|
| **Use an object-oriented programming design** | Imperative | Design | Used encapsulation, abstractism, inheritance and composition |
| **Create a Random Client Generator** | Imperative | Implementation | Used Random and parameters given by the user to generate N clients |
| **Use Multithreading : one thread per queue** | Imperative | Implementation | Each Queue is on a separate thread, managed by a Scheduler class |
| **Appropriate synchronized data structures to assure thread safety** | Imperative | Design and Implementation | Used Java Swing |
| **Log of events displayed in a .txt file** | Imperative | Implementation | Create a txt file and display the result in it |
| **Max 300 lines in classes and 30 in methods. Except GUI** | Imperative | Design and Implementation | No details |
| **Use Java naming conventions** | Imperative | Implementation | No details |
| **Good documentation** | Imperative | | No details |
| **Strategy pattern and the two strategies (shortest time, shortest queue) for allocating clients to queues** | Needed for high grade | Design and Implementation | Created a Strategy Interface and two classes that implement it |
| **Implement a GUI** | Needed for high grade | Implementation | No details |
| **Display of simulation results (average waiting time, average service time, peak hour for the simulation interval) in the graphical user interface/.txt file corresponding to the log events** | Needed for high grade | Implementation | No details |
| **Run the application on the input data set and write them in a txt file** | Needed for high grade | Implementation | No details |

## 2. Problem Analysis, Modeling, Scenarios, Use Cases

The problem that this software aims to solve is minimizing the time spent by clients in a queue.

This software tries to mimic the underlying queue management system of some softwares that offers some services to a large number of clients. It creates two solutions: a shortest time solution and a shortest queue solution. In the next picture, the use case is presented. Using a GUI, a user can input the simulation parameters, start the simulation, view changes in the queue in real time, see final results and see the whole results in a txt file


Queue System Manager
Select Simulation Parameters
Start Simualtion
View Changes In The Queues In Real Time
See Final Results
See The Whole Results In A File
User

**Actors**:

- User: The entering the simulation parameters.
- System: The queue management system software.

**Preconditions**:

- The user has access to the simulation starter interface.
- The system is operational and ready to accept input.

**Postconditions**:

- The system displays the contents of each queue and the waiting clients
- The system displays the final results: average waiting time, peak hour and average service time.
- The system is ready for a new operation or to be closed.

**Main Flow**:

1. The user launches the queue management system simulation application.
2. The system presents an interface where the user can input the simulation parameters
3. The user enters the inputs in their designated input fields.
4. The user starts the simulation by pressing the "Start simulation" button.
5. The system processes the input and displays the queues contents in real life by using a text area.
6. The system displays the result of the operation to the user and saves them in a .txt file.

**Exception Flows**:

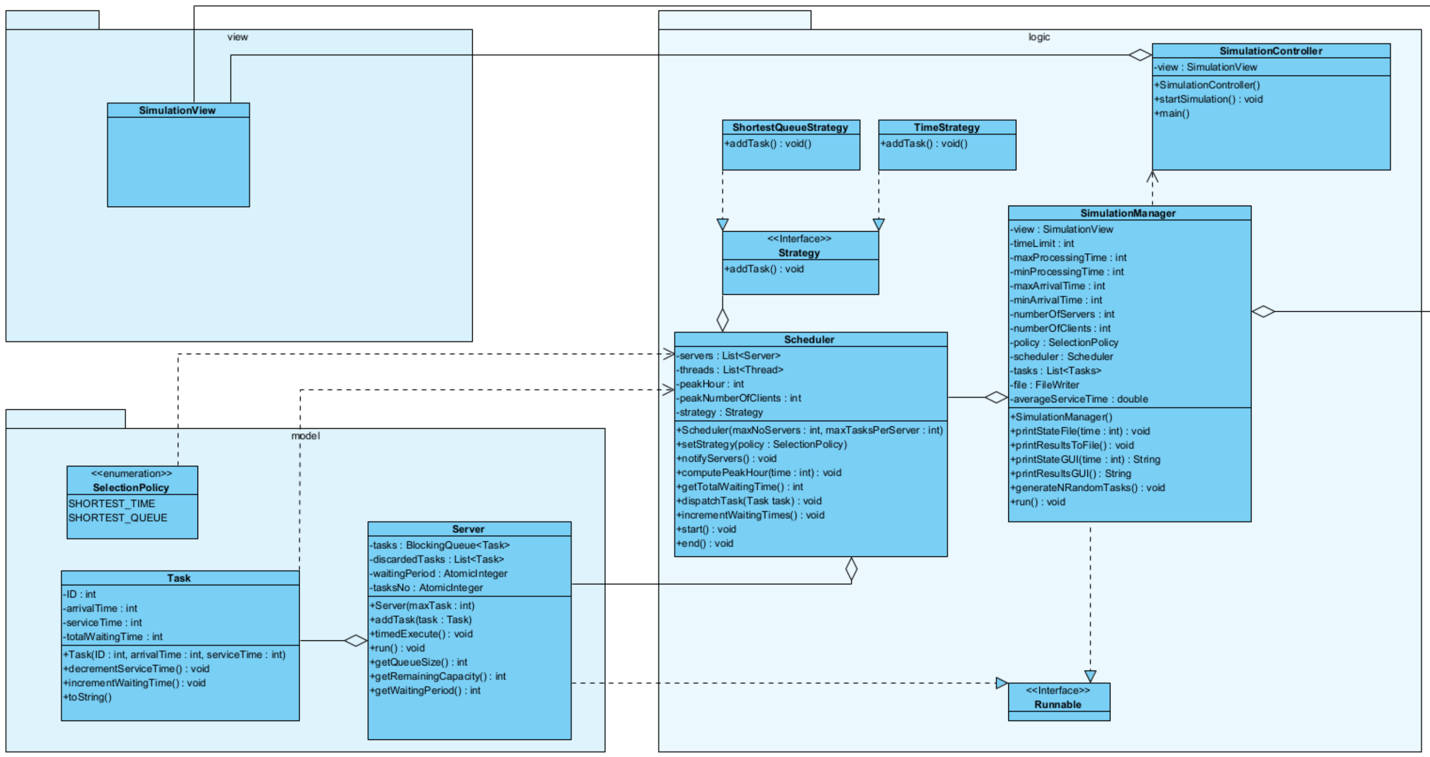
- **E1. Invalid Input**: If the user enters an invalid inputs , the system displays an error message and prompts the user to correct the input. The process then resumes from step 3.

E.G.

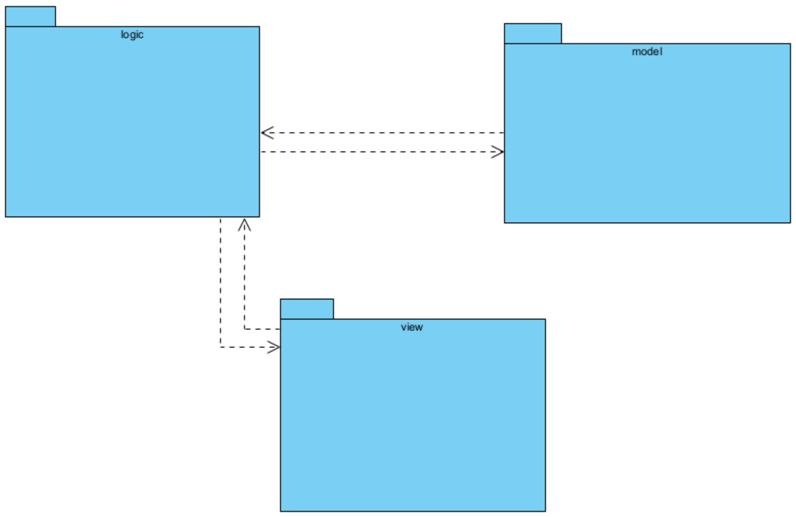
The user can enter the next inputs: maximum simulation time, maximum arrival time, minimum arrival time, maximum service time, minimum service time, number of queues, the strategy and number of clients. There exists some constrains on the inputs, like the minimum service time cannot be greater than the maximum service time, no time can be greater than the maximum service time and so on. If the user enters some false input, some error will display and the simulation won't start. Some inputs examples:

Max Simulation Time: 60
Max Arrival Time : 15
Min Arrival Time : 2
Max Service Time: 9
Min Service Time: 2
Number of queues: 2
Number of clients: 20

## 3. Design



The model, view, logic architecture was used in package organization. The Model consists of the Task class, Server class and SelectionPolicy enumeration. The Logic package consists of the classes Scheduler, SimulaitonManager, SimulationController and the Strategy interface with its two implementations: TimeStrategy and ShortestQueueStrategy. The View package consists of the only view present in this software, SimulationView. The package diagram is presented in the next diagram:



# 4. Implementation

## *1. MODEL*

### *1.1. Task*

This is the class that aims to model the client. It has three main fields and one additional field that helps in calculating the average waiting time. The three main fields are: ID, that is unique, arrival time that represents the simulation time at which the client is ready to be placed in one of the queues, and the service time, which represents the time which the client spends at the front of the queue, being served.

FIELDS:

- ID : int => represents the client's ID, which is unique
- arrivalTime : int => represents the simulation time at which the client can be placed in a queue
- serviceTime : int => the amount of simulation time the client spends at the front of the queue, being served
- totalWaitingTime : int => a field that aims to help in the calculation of the average waiting time

CONSTRUCTORS:

- ID : int, arrivalTime : int, serviceTime : int.

SOME IMPORTANT METHODS:

- decrementServiceTime(): void => decrements the service time so the server can knonw when to eliminate the task from the queue
- incrementWaitingTime() : void => increments the totalWaitingTime so we can calculate the averageWaitingTime
- toString() : String => returns the String representation of this task.

### *1.2. Server*

This is the class that aims to model the server. This class has a queue that keeps the tasks in it, and mimics the service on the client that is at the top of the queue. The server is controlled by the Scheduler, and lets it serve the client or/and add a client only at a time dictated by the Simulation Manager. Each server is represented by an afferent Thread. It implements the Runnable interface.

FIELDS:

- tasks : BlockingQueue<Task> => the queue in which the clients sit.
- discardedTasks : List<Task> => the list in which the served clients reside
- waitingPeriod : AtomicInteger => the sum of the service time. This field is used for the strategy of client placement
- tasksNo => AtomicInteger => keeps the size of the queue. Same as waiting period, is used for strategy of client placement.

CONSTRUCTORS:

- maxTask : int

SOME IMPORTANT METHODS:

- addTask(task: Task) : void => adds a task in the queue
- timedExecute() => method called by the scheduler to synchronize with the simulation time
- run() – the method overridden from the Runnable interface

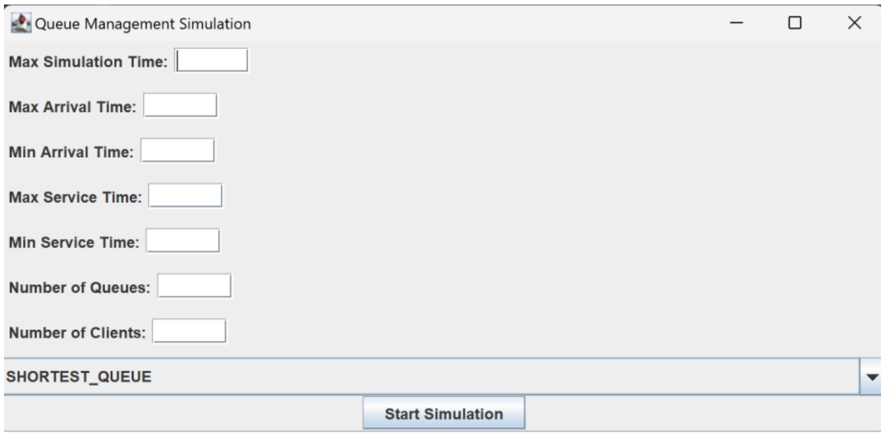
### *1.3. SelectionPolicy*

An enumeration with the possible strategies that the Scheduler uses to place the client in the proper queue(SHORTEST_QUEUE, SHORTEST_TIME).

## *2. VIEW*

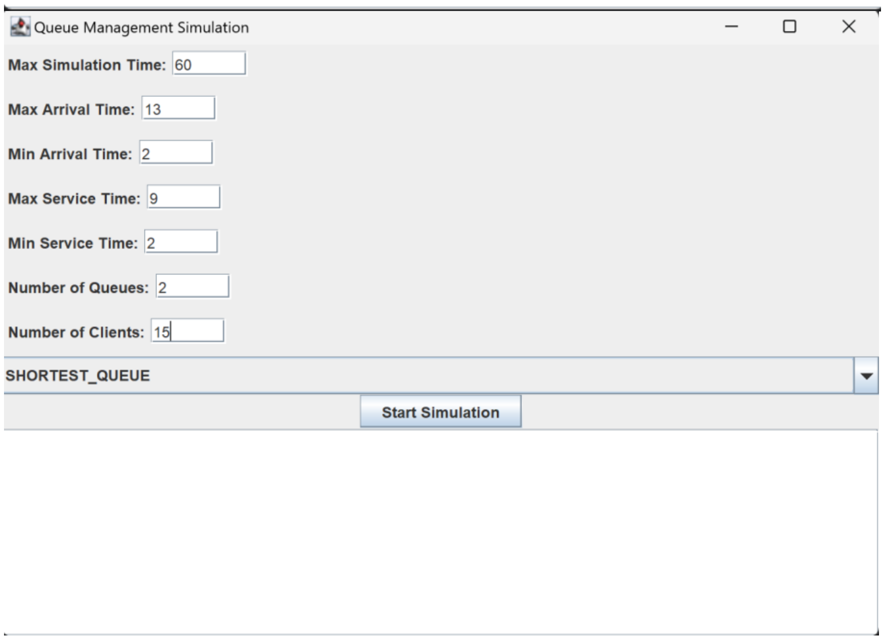
### *2.1. SimulationView*

This is the only view of the software. It is implemented using Java Swing. It consists in input fields, prompts for error, text area for displaying the results and button for starting the simulation, and a button “=” for executing the selected operation. The interface is controlled by the SimulationController class. Some examples of the graphical interface:


Queue Management Simulation

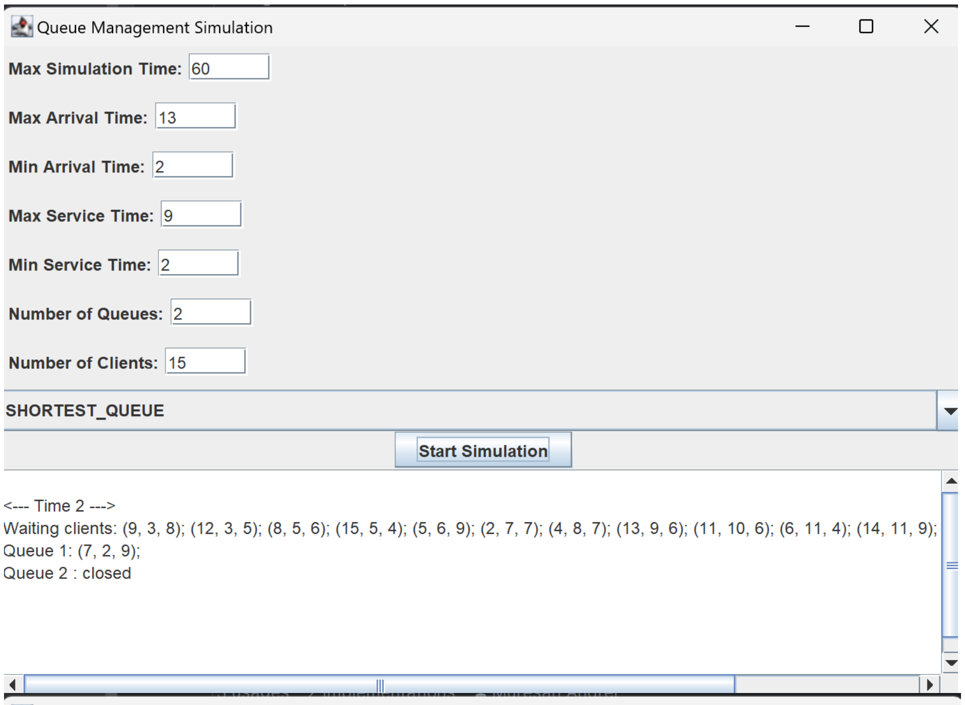
|  |  |
|---|---|
| **Max Simulation Time:** |  |
| **Max Arrival Time:** |  |
| **Min Arrival Time:** |  |
| **Max Service Time:** |  |
| **Min Service Time:** |  |
| **Number of Queues:** |  |
| **Number of Clients:** |  |

SHORTEST_QUEUE


Queue Management Simulation

|  |  |
|---|---|
| **Max Simulation Time:** | 60 |
| **Max Arrival Time:** | 13 |
| **Min Arrival Time:** | 2 |
| **Max Service Time:** | 9 |
| **Min Service Time:** | 2 |
| **Number of Queues:** | 2 |
| **Number of Clients:** | 15 |

SHORTEST_QUEUE



Max Simulation Time: 60

Max Arrival Time: 13

Min Arrival Time: 2

Max Service Time: 9

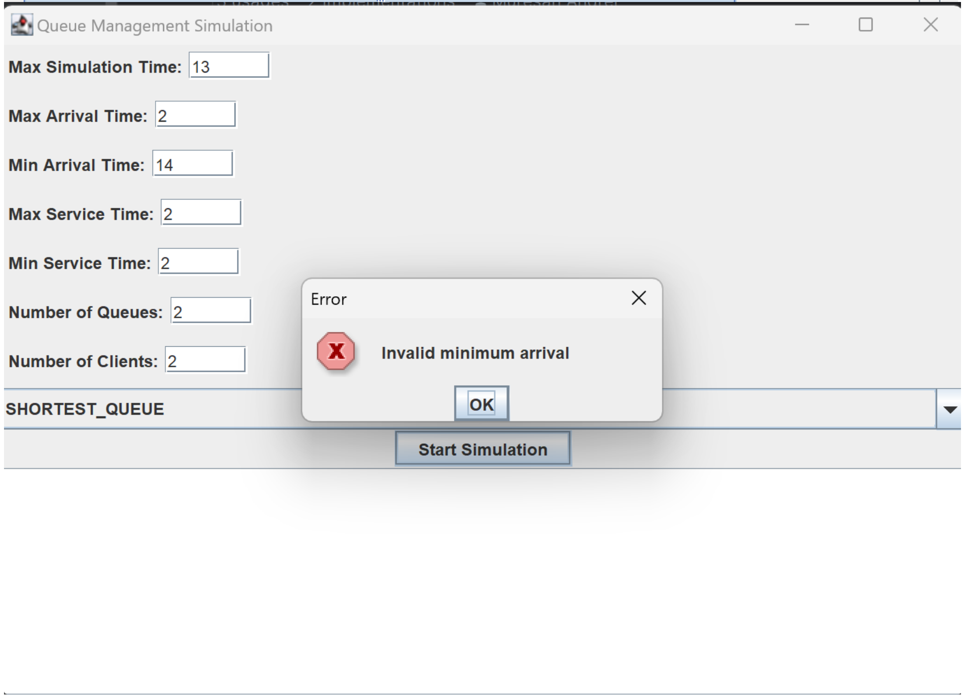
Min Service Time: 2

Number of Queues: 2

Number of Clients: 15

SHORTEST_QUEUE

```
<--- Time 2 --->
Waiting clients: (9, 3, 8); (12, 3, 5); (8, 5, 6); (15, 5, 4); (5, 6, 9); (2, 7, 7); (4, 8, 7); (13, 9, 6); (11, 10, 6); (6, 11, 4); (14, 11, 9);
Queue 1: (7, 2, 9);
Queue 2 : closed
```



Max Simulation Time: 13

Max Arrival Time: 2

Min Arrival Time: 14

Max Service Time: 2

Min Service Time: 2

Number of Queues: 2

Number of Clients: 2

SHORTEST_QUEUE

Invalid minimum arrival

## 3. LOGIC

### 3.4. Strategy

This is the interface used for strategy pattern. It has two implementations: TimedStrategy, ShortestQueueStrategy

SOME IMPORTANT ABSTRACT METHODS:

- addTask(servers : List<Servers>, task : Task) : void

*3.2. SimulationManager*

This is the class that holds the time, and by using the Scheduler class, it manages the simulation

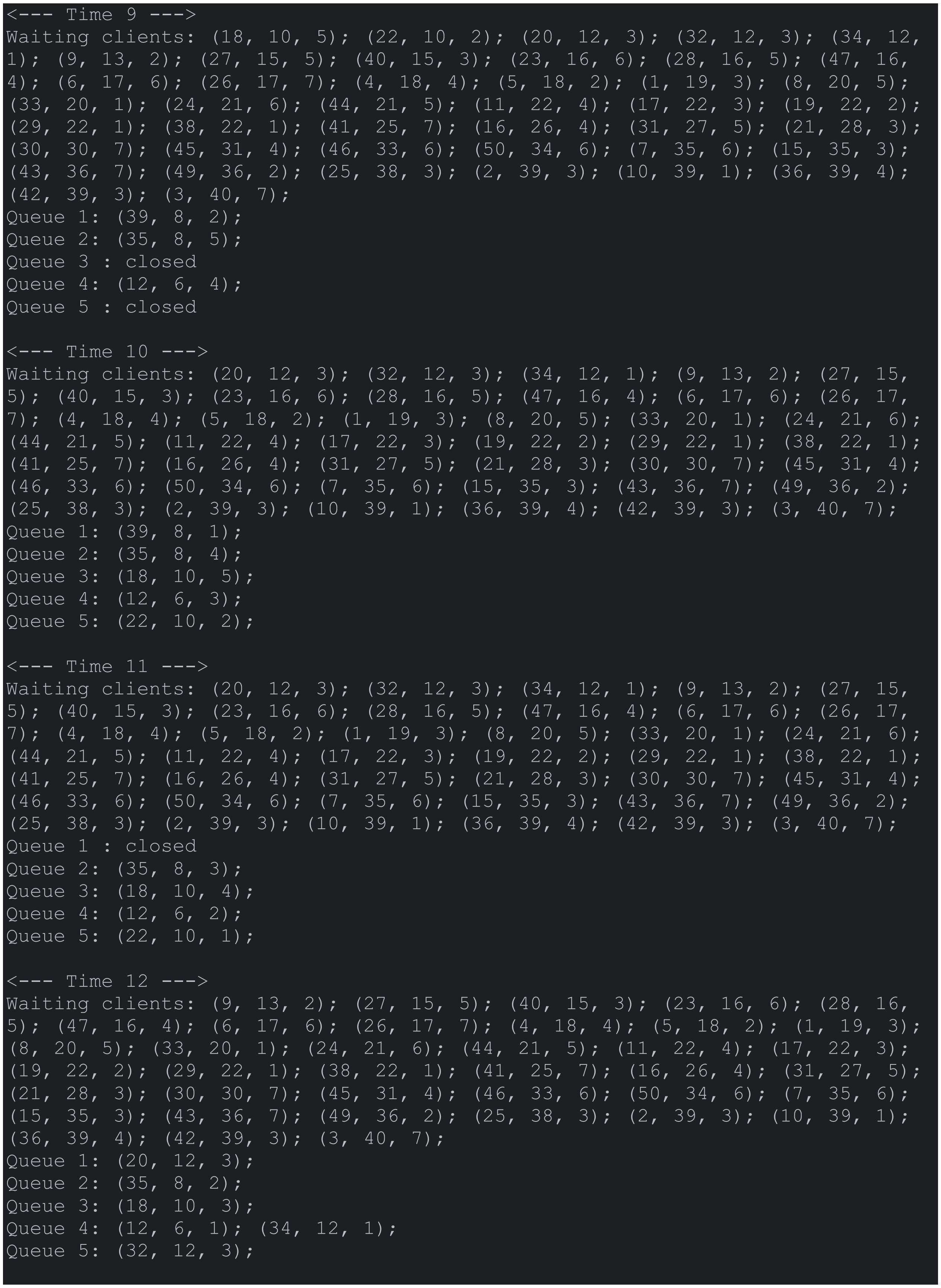
*3.3. SimulationController*

This is the class that controls the view, has the main function.

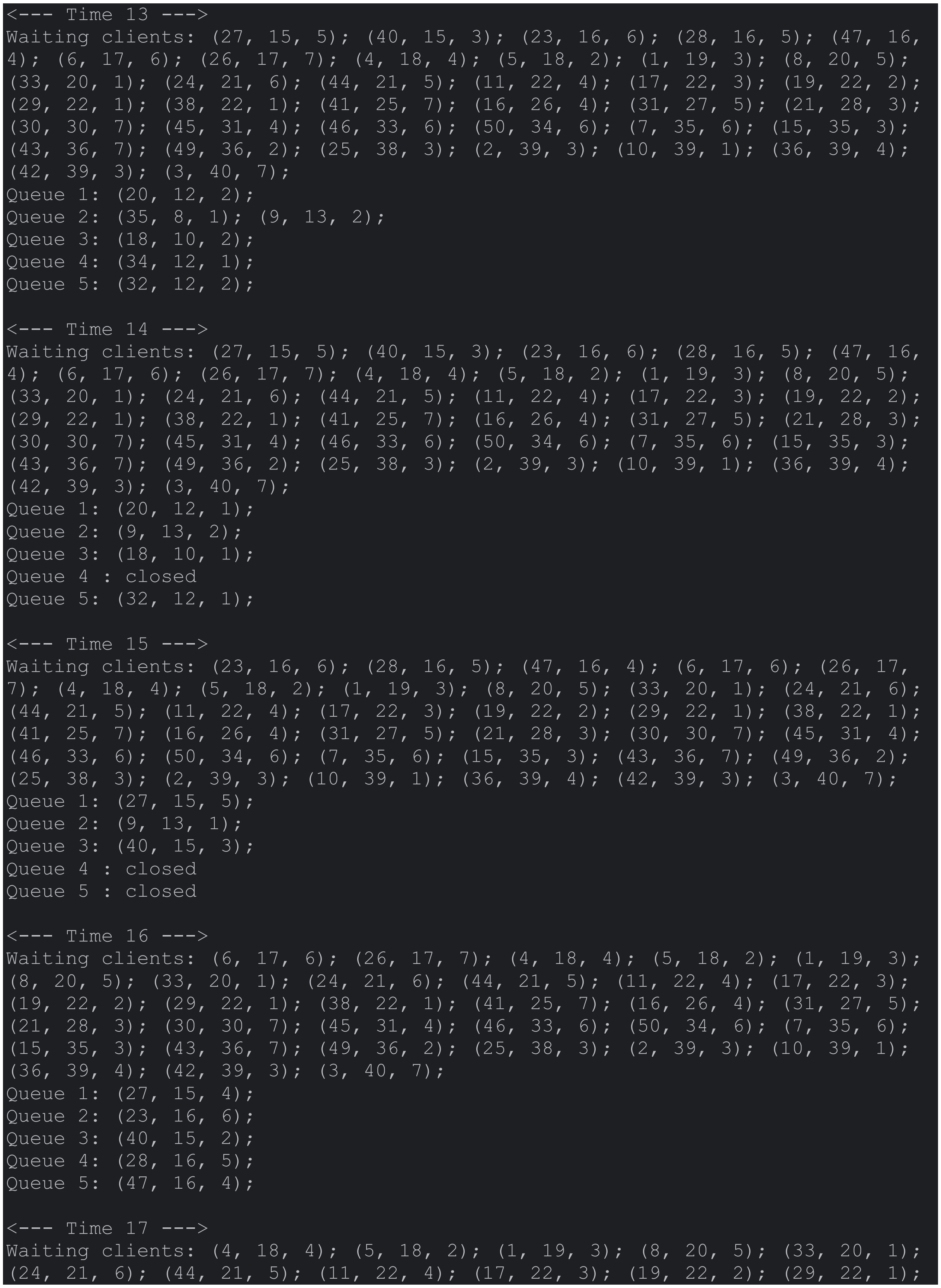
Results

```
<--- Time 0 --->
Waiting clients: (14, 2, 7); (37, 2, 6); (13, 4, 5); (12, 6, 7); (48, 6, 3);
(35, 8, 6); (39, 8, 2); (18, 10, 5); (22, 10, 2); (20, 12, 3); (32, 12, 3);
(34, 12, 1); (9, 13, 2); (27, 15, 5); (40, 15, 3); (23, 16, 6); (28, 16, 5);
(47, 16, 4); (6, 17, 6); (26, 17, 7); (4, 18, 4); (5, 18, 2); (1, 19, 3); (8,
20, 5); (33, 20, 1); (24, 21, 6); (44, 21, 5); (11, 22, 4); (17, 22, 3); (19,
22, 2); (29, 22, 1); (38, 22, 1); (41, 25, 7); (16, 26, 4); (31, 27, 5); (21,
28, 3); (30, 30, 7); (45, 31, 4); (46, 33, 6); (50, 34, 6); (7, 35, 6); (15,
35, 3); (43, 36, 7); (49, 36, 2); (25, 38, 3); (2, 39, 3); (10, 39, 1); (36,
39, 4); (42, 39, 3); (3, 40, 7);
Queue 1 : closed
Queue 2 : closed
Queue 3 : closed
Queue 4 : closed
Queue 5 : closed

<--- Time 1 --->
Waiting clients: (14, 2, 7); (37, 2, 6); (13, 4, 5); (12, 6, 7); (48, 6, 3);
(35, 8, 6); (39, 8, 2); (18, 10, 5); (22, 10, 2); (20, 12, 3); (32, 12, 3);
(34, 12, 1); (9, 13, 2); (27, 15, 5); (40, 15, 3); (23, 16, 6); (28, 16, 5);
(47, 16, 4); (6, 17, 6); (26, 17, 7); (4, 18, 4); (5, 18, 2); (1, 19, 3); (8,
20, 5); (33, 20, 1); (24, 21, 6); (44, 21, 5); (11, 22, 4); (17, 22, 3); (19,
22, 2); (29, 22, 1); (38, 22, 1); (41, 25, 7); (16, 26, 4); (31, 27, 5); (21,
28, 3); (30, 30, 7); (45, 31, 4); (46, 33, 6); (50, 34, 6); (7, 35, 6); (15,
35, 3); (43, 36, 7); (49, 36, 2); (25, 38, 3); (2, 39, 3); (10, 39, 1); (36,
39, 4); (42, 39, 3); (3, 40, 7);
Queue 1 : closed
Queue 2 : closed
Queue 3 : closed
Queue 4 : closed
Queue 5 : closed

<--- Time 2 --->
Waiting clients: (13, 4, 5); (12, 6, 7); (48, 6, 3); (35, 8, 6); (39, 8, 2);
(18, 10, 5); (22, 10, 2); (20, 12, 3); (32, 12, 3); (34, 12, 1); (9, 13, 2);
(27, 15, 5); (40, 15, 3); (23, 16, 6); (28, 16, 5); (47, 16, 4); (6, 17, 6);
```

```
(26, 17, 7); (4, 18, 4); (5, 18, 2); (1, 19, 3); (8, 20, 5); (33, 20, 1);
(24, 21, 6); (44, 21, 5); (11, 22, 4); (17, 22, 3); (19, 22, 2); (29, 22, 1);
(38, 22, 1); (41, 25, 7); (16, 26, 4); (31, 27, 5); (21, 28, 3); (30, 30, 7);
(45, 31, 4); (46, 33, 6); (50, 34, 6); (7, 35, 6); (15, 35, 3); (43, 36, 7);
(49, 36, 2); (25, 38, 3); (2, 39, 3); (10, 39, 1); (36, 39, 4); (42, 39, 3);
(3, 40, 7);
Queue 1: (14, 2, 7);
Queue 2: (37, 2, 6);
Queue 3 : closed
Queue 4 : closed
Queue 5 : closed

<--- Time 6 --->
Waiting clients: (35, 8, 6); (39, 8, 2); (18, 10, 5); (22, 10, 2); (20, 12,
3); (32, 12, 3); (34, 12, 1); (9, 13, 2); (27, 15, 5); (40, 15, 3); (23, 16,
6); (28, 16, 5); (47, 16, 4); (6, 17, 6); (26, 17, 7); (4, 18, 4); (5, 18,
2); (1, 19, 3); (8, 20, 5); (33, 20, 1); (24, 21, 6); (44, 21, 5); (11, 22,
4); (17, 22, 3); (19, 22, 2); (29, 22, 1); (38, 22, 1); (41, 25, 7); (16, 26,
4); (31, 27, 5); (21, 28, 3); (30, 30, 7); (45, 31, 4); (46, 33, 6); (50, 34,
6); (7, 35, 6); (15, 35, 3); (43, 36, 7); (49, 36, 2); (25, 38, 3); (2, 39,
3); (10, 39, 1); (36, 39, 4); (42, 39, 3); (3, 40, 7);
Queue 1: (14, 2, 3);
Queue 2: (37, 2, 2);
Queue 3: (13, 4, 3);
Queue 4: (12, 6, 7);
Queue 5: (48, 6, 3);

<--- Time 7 --->
Waiting clients: (35, 8, 6); (39, 8, 2); (18, 10, 5); (22, 10, 2); (20, 12,
3); (32, 12, 3); (34, 12, 1); (9, 13, 2); (27, 15, 5); (40, 15, 3); (23, 16,
6); (28, 16, 5); (47, 16, 4); (6, 17, 6); (26, 17, 7); (4, 18, 4); (5, 18,
2); (1, 19, 3); (8, 20, 5); (33, 20, 1); (24, 21, 6); (44, 21, 5); (11, 22,
4); (17, 22, 3); (19, 22, 2); (29, 22, 1); (38, 22, 1); (41, 25, 7); (16, 26,
4); (31, 27, 5); (21, 28, 3); (30, 30, 7); (45, 31, 4); (46, 33, 6); (50, 34,
6); (7, 35, 6); (15, 35, 3); (43, 36, 7); (49, 36, 2); (25, 38, 3); (2, 39,
3); (10, 39, 1); (36, 39, 4); (42, 39, 3); (3, 40, 7);
Queue 1: (14, 2, 2);
Queue 2: (37, 2, 1);
Queue 3: (13, 4, 2);
Queue 4: (12, 6, 6);
Queue 5: (48, 6, 2);

<--- Time 8 --->
Waiting clients: (18, 10, 5); (22, 10, 2); (20, 12, 3); (32, 12, 3); (34, 12,
1); (9, 13, 2); (27, 15, 5); (40, 15, 3); (23, 16, 6); (28, 16, 5); (47, 16,
4); (6, 17, 6); (26, 17, 7); (4, 18, 4); (5, 18, 2); (1, 19, 3); (8, 20, 5);
(33, 20, 1); (24, 21, 6); (44, 21, 5); (11, 22, 4); (17, 22, 3); (19, 22, 2);
(29, 22, 1); (38, 22, 1); (41, 25, 7); (16, 26, 4); (31, 27, 5); (21, 28, 3);
(30, 30, 7); (45, 31, 4); (46, 33, 6); (50, 34, 6); (7, 35, 6); (15, 35, 3);
(43, 36, 7); (49, 36, 2); (25, 38, 3); (2, 39, 3); (10, 39, 1); (36, 39, 4);
(42, 39, 3); (3, 40, 7);
Queue 1: (14, 2, 1); (39, 8, 2);
Queue 2: (35, 8, 6);
Queue 3: (13, 4, 1);
Queue 4: (12, 6, 5);
Queue 5: (48, 6, 1);
```



```
<--- Time 9 --->
Waiting clients: (18, 10, 5); (22, 10, 2); (20, 12, 3); (32, 12, 3); (34, 12,
1); (9, 13, 2); (27, 15, 5); (40, 15, 3); (23, 16, 6); (28, 16, 5); (47, 16,
4); (6, 17, 6); (26, 17, 7); (4, 18, 4); (5, 18, 2); (1, 19, 3); (8, 20, 5);
(33, 20, 1); (24, 21, 6); (44, 21, 5); (11, 22, 4); (17, 22, 3); (19, 22, 2);
(29, 22, 1); (38, 22, 1); (41, 25, 7); (16, 26, 4); (31, 27, 5); (21, 28, 3);
(30, 30, 7); (45, 31, 4); (46, 33, 6); (50, 34, 6); (7, 35, 6); (15, 35, 3);
(43, 36, 7); (49, 36, 2); (25, 38, 3); (2, 39, 3); (10, 39, 1); (36, 39, 4);
(42, 39, 3); (3, 40, 7);
Queue 1: (39, 8, 2);
Queue 2: (35, 8, 5);
Queue 3 : closed
Queue 4: (12, 6, 4);
Queue 5 : closed

<--- Time 10 --->
Waiting clients: (20, 12, 3); (32, 12, 3); (34, 12, 1); (9, 13, 2); (27, 15,
5); (40, 15, 3); (23, 16, 6); (28, 16, 5); (47, 16, 4); (6, 17, 6); (26, 17,
7); (4, 18, 4); (5, 18, 2); (1, 19, 3); (8, 20, 5); (33, 20, 1); (24, 21, 6);
(44, 21, 5); (11, 22, 4); (17, 22, 3); (19, 22, 2); (29, 22, 1); (38, 22, 1);
(41, 25, 7); (16, 26, 4); (31, 27, 5); (21, 28, 3); (30, 30, 7); (45, 31, 4);
(46, 33, 6); (50, 34, 6); (7, 35, 6); (15, 35, 3); (43, 36, 7); (49, 36, 2);
(25, 38, 3); (2, 39, 3); (10, 39, 1); (36, 39, 4); (42, 39, 3); (3, 40, 7);
Queue 1: (39, 8, 1);
Queue 2: (35, 8, 4);
Queue 3: (18, 10, 5);
Queue 4: (12, 6, 3);
Queue 5: (22, 10, 2);

<--- Time 11 --->
Waiting clients: (20, 12, 3); (32, 12, 3); (34, 12, 1); (9, 13, 2); (27, 15,
5); (40, 15, 3); (23, 16, 6); (28, 16, 5); (47, 16, 4); (6, 17, 6); (26, 17,
7); (4, 18, 4); (5, 18, 2); (1, 19, 3); (8, 20, 5); (33, 20, 1); (24, 21, 6);
(44, 21, 5); (11, 22, 4); (17, 22, 3); (19, 22, 2); (29, 22, 1); (38, 22, 1);
(41, 25, 7); (16, 26, 4); (31, 27, 5); (21, 28, 3); (30, 30, 7); (45, 31, 4);
(46, 33, 6); (50, 34, 6); (7, 35, 6); (15, 35, 3); (43, 36, 7); (49, 36, 2);
(25, 38, 3); (2, 39, 3); (10, 39, 1); (36, 39, 4); (42, 39, 3); (3, 40, 7);
Queue 1 : closed
Queue 2: (35, 8, 3);
Queue 3: (18, 10, 4);
Queue 4: (12, 6, 2);
Queue 5: (22, 10, 1);

<--- Time 12 --->
Waiting clients: (9, 13, 2); (27, 15, 5); (40, 15, 3); (23, 16, 6); (28, 16,
5); (47, 16, 4); (6, 17, 6); (26, 17, 7); (4, 18, 4); (5, 18, 2); (1, 19, 3);
(8, 20, 5); (33, 20, 1); (24, 21, 6); (44, 21, 5); (11, 22, 4); (17, 22, 3);
(19, 22, 2); (29, 22, 1); (38, 22, 1); (41, 25, 7); (16, 26, 4); (31, 27, 5);
(21, 28, 3); (30, 30, 7); (45, 31, 4); (46, 33, 6); (50, 34, 6); (7, 35, 6);
(15, 35, 3); (43, 36, 7); (49, 36, 2); (25, 38, 3); (2, 39, 3); (10, 39, 1);
(36, 39, 4); (42, 39, 3); (3, 40, 7);
Queue 1: (20, 12, 3);
Queue 2: (35, 8, 2);
Queue 3: (18, 10, 3);
Queue 4: (12, 6, 1); (34, 12, 1);
Queue 5: (32, 12, 3);
```



```
<--- Time 13 --->
Waiting clients: (27, 15, 5); (40, 15, 3); (23, 16, 6); (28, 16, 5); (47, 16,
4); (6, 17, 6); (26, 17, 7); (4, 18, 4); (5, 18, 2); (1, 19, 3); (8, 20, 5);
(33, 20, 1); (24, 21, 6); (44, 21, 5); (11, 22, 4); (17, 22, 3); (19, 22, 2);
(29, 22, 1); (38, 22, 1); (41, 25, 7); (16, 26, 4); (31, 27, 5); (21, 28, 3);
(30, 30, 7); (45, 31, 4); (46, 33, 6); (50, 34, 6); (7, 35, 6); (15, 35, 3);
(43, 36, 7); (49, 36, 2); (25, 38, 3); (2, 39, 3); (10, 39, 1); (36, 39, 4);
(42, 39, 3); (3, 40, 7);
Queue 1: (20, 12, 2);
Queue 2: (35, 8, 1); (9, 13, 2);
Queue 3: (18, 10, 2);
Queue 4: (34, 12, 1);
Queue 5: (32, 12, 2);

<--- Time 14 --->
Waiting clients: (27, 15, 5); (40, 15, 3); (23, 16, 6); (28, 16, 5); (47, 16,
4); (6, 17, 6); (26, 17, 7); (4, 18, 4); (5, 18, 2); (1, 19, 3); (8, 20, 5);
(33, 20, 1); (24, 21, 6); (44, 21, 5); (11, 22, 4); (17, 22, 3); (19, 22, 2);
(29, 22, 1); (38, 22, 1); (41, 25, 7); (16, 26, 4); (31, 27, 5); (21, 28, 3);
(30, 30, 7); (45, 31, 4); (46, 33, 6); (50, 34, 6); (7, 35, 6); (15, 35, 3);
(43, 36, 7); (49, 36, 2); (25, 38, 3); (2, 39, 3); (10, 39, 1); (36, 39, 4);
(42, 39, 3); (3, 40, 7);
Queue 1: (20, 12, 1);
Queue 2: (9, 13, 2);
Queue 3: (18, 10, 1);
Queue 4 : closed
Queue 5: (32, 12, 1);

<--- Time 15 --->
Waiting clients: (23, 16, 6); (28, 16, 5); (47, 16, 4); (6, 17, 6); (26, 17,
7); (4, 18, 4); (5, 18, 2); (1, 19, 3); (8, 20, 5); (33, 20, 1); (24, 21, 6);
(44, 21, 5); (11, 22, 4); (17, 22, 3); (19, 22, 2); (29, 22, 1); (38, 22, 1);
(41, 25, 7); (16, 26, 4); (31, 27, 5); (21, 28, 3); (30, 30, 7); (45, 31, 4);
(46, 33, 6); (50, 34, 6); (7, 35, 6); (15, 35, 3); (43, 36, 7); (49, 36, 2);
(25, 38, 3); (2, 39, 3); (10, 39, 1); (36, 39, 4); (42, 39, 3); (3, 40, 7);
Queue 1: (27, 15, 5);
Queue 2: (9, 13, 1);
Queue 3: (40, 15, 3);
Queue 4 : closed
Queue 5 : closed

<--- Time 16 --->
Waiting clients: (6, 17, 6); (26, 17, 7); (4, 18, 4); (5, 18, 2); (1, 19, 3);
(8, 20, 5); (33, 20, 1); (24, 21, 6); (44, 21, 5); (11, 22, 4); (17, 22, 3);
(19, 22, 2); (29, 22, 1); (38, 22, 1); (41, 25, 7); (16, 26, 4); (31, 27, 5);
(21, 28, 3); (30, 30, 7); (45, 31, 4); (46, 33, 6); (50, 34, 6); (7, 35, 6);
(15, 35, 3); (43, 36, 7); (49, 36, 2); (25, 38, 3); (2, 39, 3); (10, 39, 1);
(36, 39, 4); (42, 39, 3); (3, 40, 7);
Queue 1: (27, 15, 4);
Queue 2: (23, 16, 6);
Queue 3: (40, 15, 2);
Queue 4: (28, 16, 5);
Queue 5: (47, 16, 4);

<--- Time 17 --->
Waiting clients: (4, 18, 4); (5, 18, 2); (1, 19, 3); (8, 20, 5); (33, 20, 1);
(24, 21, 6); (44, 21, 5); (11, 22, 4); (17, 22, 3); (19, 22, 2); (29, 22, 1);
```

```
Waiting clients:
Queue 1 : closed
Queue 2 : closed
Queue 3 : closed
Queue 4 : closed
Queue 5 : closed

<--- Time 58 --->
Waiting clients:
Queue 1 : closed
Queue 2 : closed
Queue 3 : closed
Queue 4 : closed
Queue 5 : closed

<--- Time 59 --->
Waiting clients:
Queue 1 : closed
Queue 2 : closed
Queue 3 : closed
Queue 4 : closed
Queue 5 : closed

The average waiting time is: 6.24
The peak hour was at time: 22. The total number of clients at the peak hour
was: 13
The average service time is: 4.12
```

## 5. Conclusions

This project successfully demonstrates the development of an efficient queue management system using Java, highlighting the robust capabilities of multithreading to manage multiple service points concurrently. By assigning a distinct thread to each server, we were able to significantly enhance the system's responsiveness and throughput, ensuring that customer requests are handled swiftly and efficiently.

The implementation of threads allowed for the simulation of a real-world scenario where multiple servers operate simultaneously without interference, proving the system's scalability and reliability. Throughout the testing phase, the system maintained stable performance even under varying loads, which validates the effectiveness of our concurrency control mechanisms.

Moreover, the use of Java provided a platform-independent solution that can be seamlessly integrated into various operational environments with minimal adjustments. The experience gained from this project not only reaffirms Java's capability in handling complex multithreading tasks but also enriches our understanding of practical issues such as thread synchronization and resource sharing.

Future enhancements could include the incorporation of more sophisticated scheduling algorithms to further optimize resource allocation and improve customer wait times. Additionally, integrating a real-time monitoring dashboard would offer administrators better insights into queue dynamics and server efficiency.

In conclusion, this queue management system stands as a testament to the potential of Java and multithreading in creating powerful, scalable applications that can meet the dynamic needs of businesses requiring effective service delivery mechanisms.

## 6. Bibliography

1. *Udemy.com – a course that taught me threads.*