

DOCUMENTATION

ASSIGNMENT 2

Queues Management System

STUDENT NAME: Stroie Anca-Gabriela
GROUP: 30422

CONTENTS

1.	Assignment Objective	3
2.	Problem Analysis, Modeling, Scenarios, Use Cases	3
3.	Design	4
4.	Implementation.....	5
5.	Results.....	8
6.	Conclusions	8
7.	Bibliography	8

1. Assignment Objective

The main objective of the assignment is to design and implement an application aiming to analyze queuing-based systems by simulating 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.

The sub-objectives are to:

- ❖ analyze the problem and identify requirements;
- ❖ design the simulation application;
- ❖ implement the simulation application;
- ❖ test the simulation application;

2. Problem Analysis, Modeling, Scenarios, Use Cases

2.1 Problem Analysis

Queues are commonly seen both in real world and in the models. The main objective of a queue is to provide a place for client to wait before receiving a service. The management of queue based systems is interested in minimizing the time amount its clients are waiting in queues.

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 supplier. When a new server is added the waiting clients will be evenly distributed to all current available queues.

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 real-time queues evolution
- ❖ The simulation application should display the peak hour, average waiting time

Non-functional requirements:

- ❖ Performance: the application should perform operations quickly and efficiently
- ❖ Reliability: the application should produce correct results for all valid simulation inputs
- ❖ Usability: the application should be user-friendly with a clear and intuitive interface

2.2 Modeling the problem

The user will be able to select the simulation data by introducing it in the graphical user interface.

In the model we use the following classes:

1. Clock class- this class implements the clock of the simulation using the Runnable interface

2. Server class- this class is responsible for managing and processing tasks in its queue

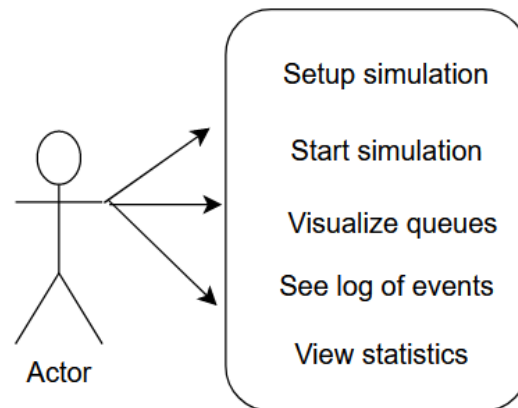
3. Task class- this class represents the client and define the id, arrival time and service time

2.3 Scenarios and use cases

The use case presents the user that interacts with the application and inserts the values for the number of queues, simulation interval, minimum and maximum arrival time , minimum and maximum service time.

The user press the button „Add Info” and starts the simulation

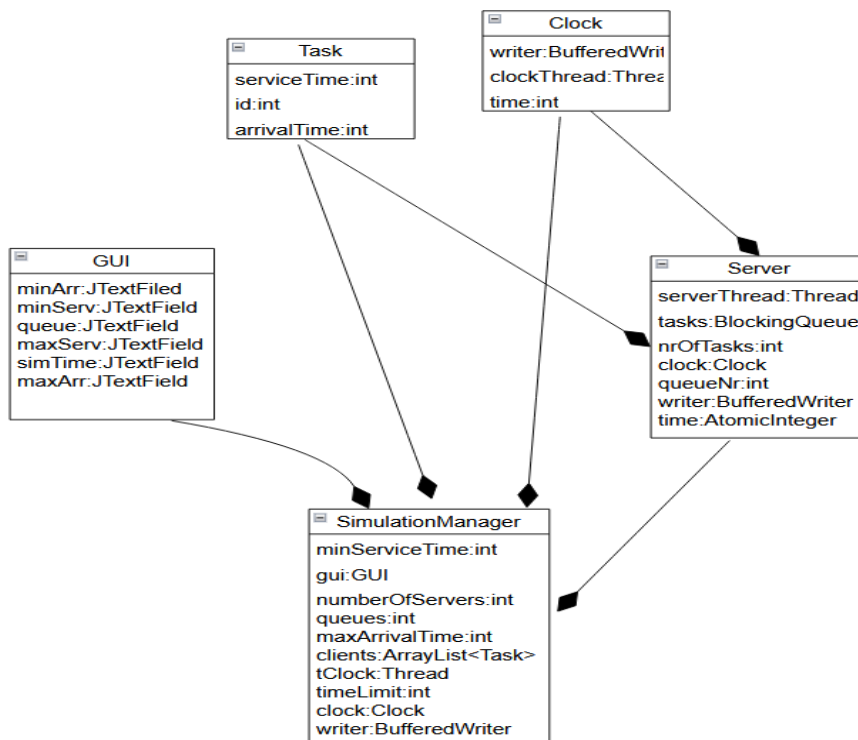
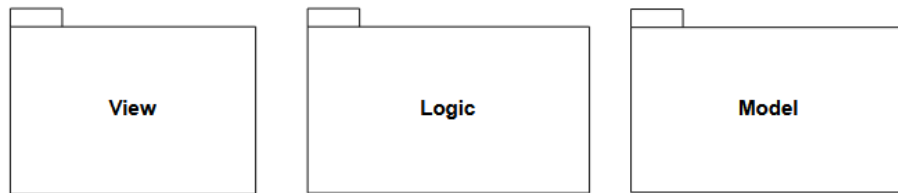
The real time display of the queue evolution will be shown in another page .



3. Design

The whole idea of splitting your program into classes is based on a general rule named divide and conquer. This paradigm can be used almost everywhere: you divide a problem into smaller problems and then you solve these little, simple and well-known problems .

Dividing your program into classes is one of the types of division which started to become common in last decade. In this programming paradigm we model our problem by some objects and try to solve the problem by sending messages between these objects.



4. Implementation

SimulationManager class

The class "SimulationManager" is responsible for managing the simulation of a queue management system. It handles the generation of random tasks, the initialization of servers, and the progression of time in the simulation.

- `setInfo()`: This method sets the simulation parameters based on the provided inputs.
- `generateNRandomTasks()`: This method generates random tasks based on the specified parameters and

adds them to the list of clients.

- `startSimulation()`: This method starts the simulation by creating the necessary threads, initializing the servers, and running the main simulation loop.
- `displayInfo()`: This method generates a string representation of the simulation parameters and the generated tasks.
- `displayWaitingClients()`: This method generates a string representation of the clients waiting to be serviced.
- `getQueueWithMinWaitingT()`: This method determines the server with the minimum waiting time and returns its index.
- `run()`: This method contains the main logic of the simulation. It checks for arrival times, assigns tasks to servers, updates the GUI, and progresses the simulation time.

Clock class

The class "Clock" represents a clock object that keeps track of the simulation time in the queue management system. It runs as a separate thread and increments the time by one second at regular intervals.

·`startClock()`: This method starts the clock by initiating the `clockThread`.

·`stopClock()`: This method stops the clock by setting the `isRunning` flag to false.

·`getTime()`: This method returns the current time value.

·`setTime(int time)`: This method sets the current time value.

·`run()`: This method is the main logic for the clock thread. It increments the time by one second at regular intervals and prints the updated time.

·`printClock()`: This method generates a string representation of the clock time.

Server class

The class "Server" represents a server in a queue management system. Each server is responsible for processing tasks from a blocking queue. It implements the `Runnable` interface, allowing it to run as a separate thread.

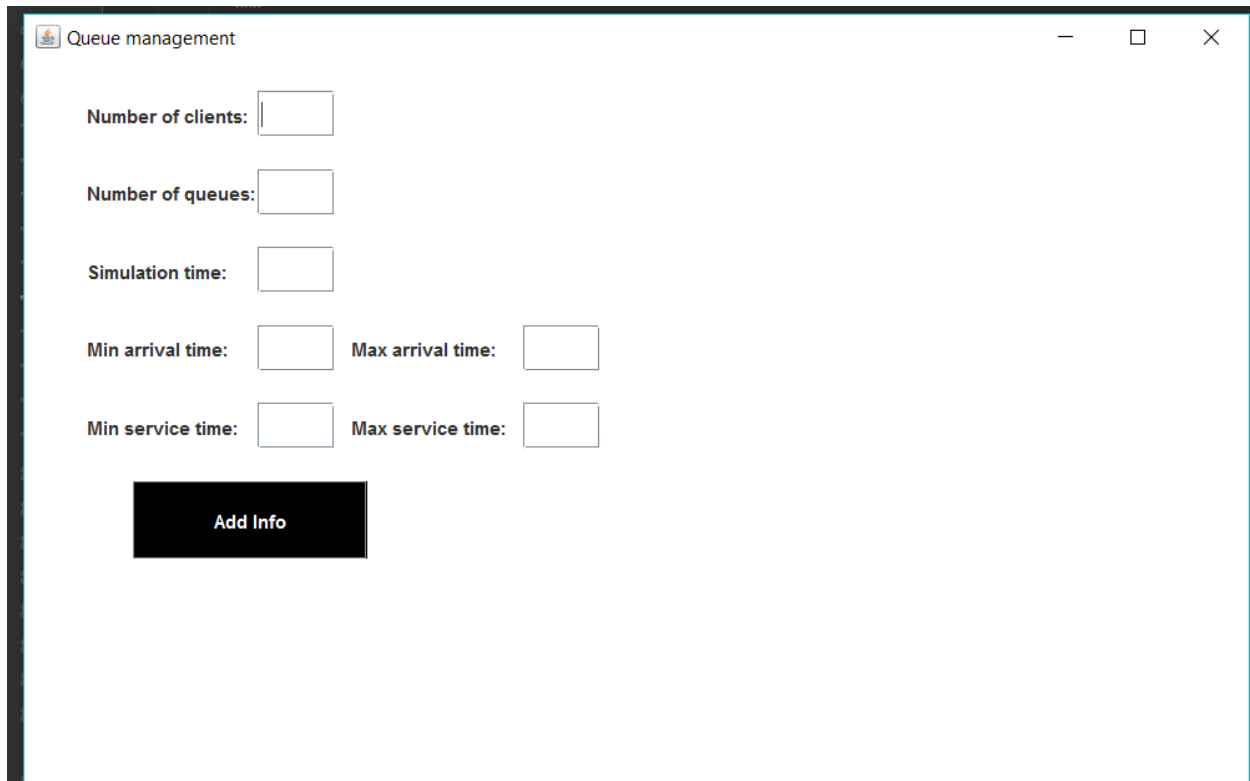
·`addTask(Task newTask)`: This method adds a new task to the server's blocking queue. It increments the `nrOfTasks` counter and notifies any waiting threads.

·`removeTask()`: This method removes a task from the server's blocking queue. It decrements the `nrOfTasks` counter and notifies any waiting threads.

·`getIndex()`: This method returns the server's queue number.

- `getWaitingTime()`: This method calculates and returns the total waiting time of all tasks in the server's queue.
- `run()`: This method contains the server's processing logic. It continuously checks the tasks in the queue, updates their service time, and removes completed tasks. It also prints the current state of the server's queue and waits for a specific duration.
- `displayQueue()`: This method generates a string representation of the server's queue, including the task information.
- `displayInTxt(String text)`: This method can be used to display the server's information in a text format.
- `getTasks()`: This method returns the blocking queue of tasks assigned to the server.

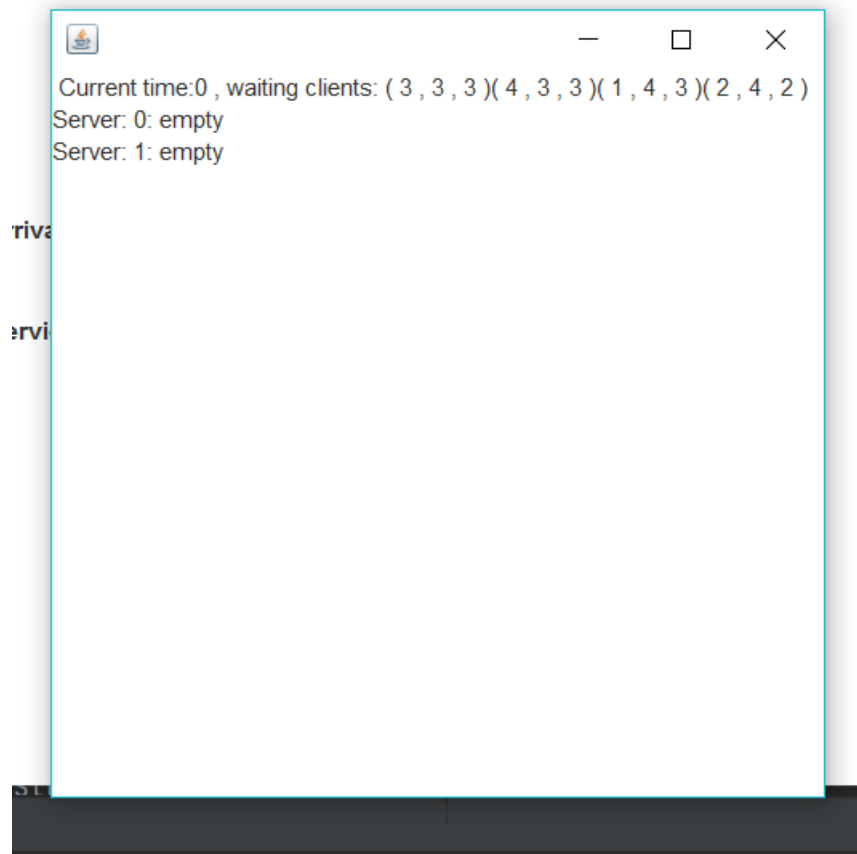
GUI class



The screenshot shows a Java Swing window titled "Queue management" with standard window controls (minimize, maximize, close) in the top right corner. The window contains several input fields for simulation parameters:

- Number of clients:** A single text input field.
- Number of queues:** A single text input field.
- Simulation time:** A single text input field.
- Min arrival time:** A text input field.
- Max arrival time:** A text input field.
- Min service time:** A text input field.
- Max service time:** A text input field.

Below these input fields is a black rectangular button with the text "Add Info" in white.



5. Results

In order to test the application, the user has to enter the data of the simulation and after the Add Info button is pressed, the log of event will appear.

6. Conclusions

This application is a user friendly queue management application. It was a helpful exercise to remember OOP concepts and GUI.

As future developments, it could be improved by allowing the user to introduce clients in real time during the simulation.

7. Bibliography

- ❖ <https://dsrl.eu/courses/pt/#>

❖ https://www.w3schools.com/java/java_threads.asp