DOCUMENTATION

ASSIGNMENT *2*

STUDENT NAME: TUDOR ALEXIA-CRISTINA
GROUP: 30424

CONTENTS

1. Assignment Objective 3
2. Problem Analysis, Modeling, Scenarios, Use Cases 3
3. Design 4
4. Implementation 5
5. Results 8
6. Conclusions 10
7. Bibliography 11

1. Assignment Objective

Design and implement an application aiming to analyze queuing-based by simulatimg a 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.

2. Problem Analysis, Modeling, Scenarios, Use Cases

- Problem Analysis

A queue is a useful data structure in programming. It is similar to the ticket queue outside a cinema hall, where the first person entering the queue is the first person who gets the ticket.

Queue follows the First In First Out (FIFO) rule - the item that goes in first is the item that comes out first. In programming terms, putting items in the queue is called enqueue, and removing items from the queue is called dequeue.

In computer science, a thread of execution is the smallest sequence of programmed instructions that can be managed independently by a scheduler, which is typically a part of the operating system. The implementation of threads and processes differs between operating systems. Threads can help improve the efficiency and performance of a computer program by allowing multiple tasks to be executed simultaneously. Threads make debugging and maintenance easier because each thread can be managed independently.

This queue-based management, in which clients are assigned based on the number of clients already in a queue, is used to provide a minimum time of service. However, in order to provide a better solution, increasing the number of queues in the system may be beneficial; however, this approach will raise the cost of the service.

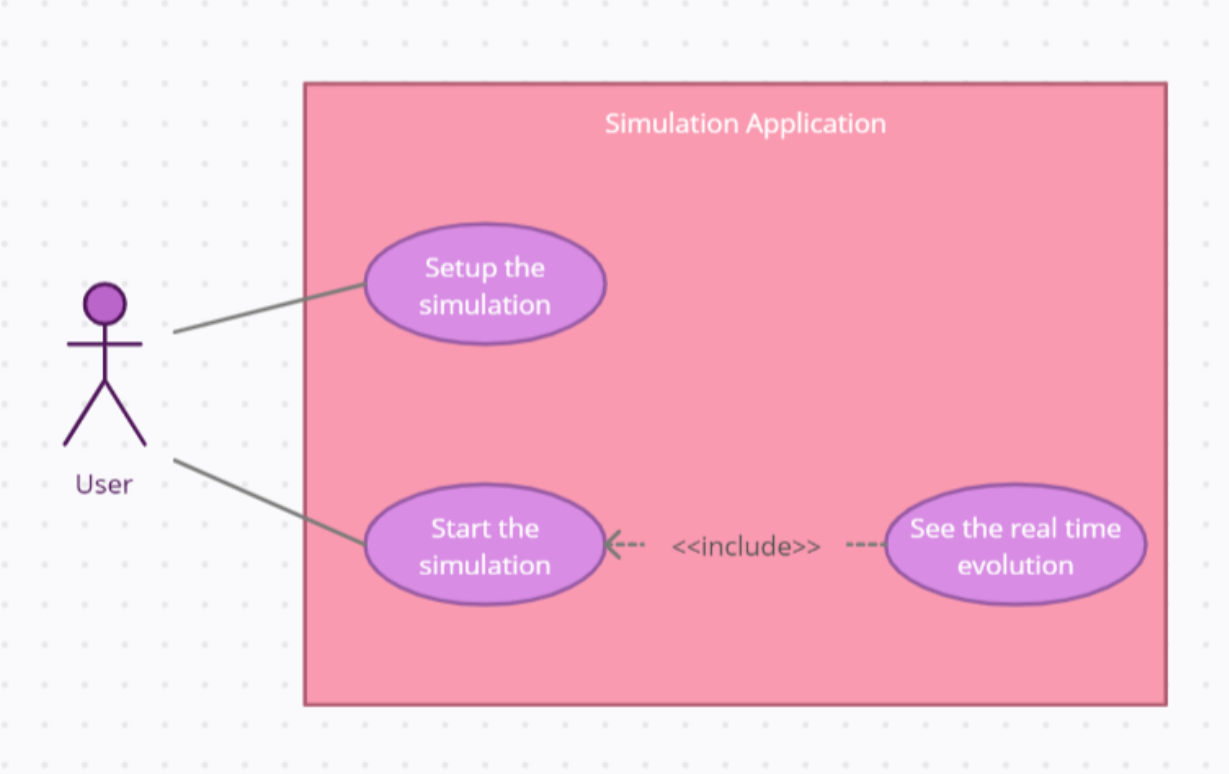
- Modeling

The user can simulate the application by entering the following inputs into the graphical user interface: time limit, number of clients, number of servers, minimum and maximum arrival time and minimum and maximum service time.

By pressing the “generate” button, the simulation starts. The given inputs would be stored, a list of random generated tasks (clients) would be created, and the threads would be running and displaying the real-time evolution of each server (queue) and the average waiting time, average service time and the peak hour in a text field.

- Scenarios and Use Cases

The user should insert valid information in the Text Field and select the checkbox of one of the strategies: Shortest Time or Shortest Queue, then click the “generate” button to display the results in the Log of events TextArea in real-time.



3. Design

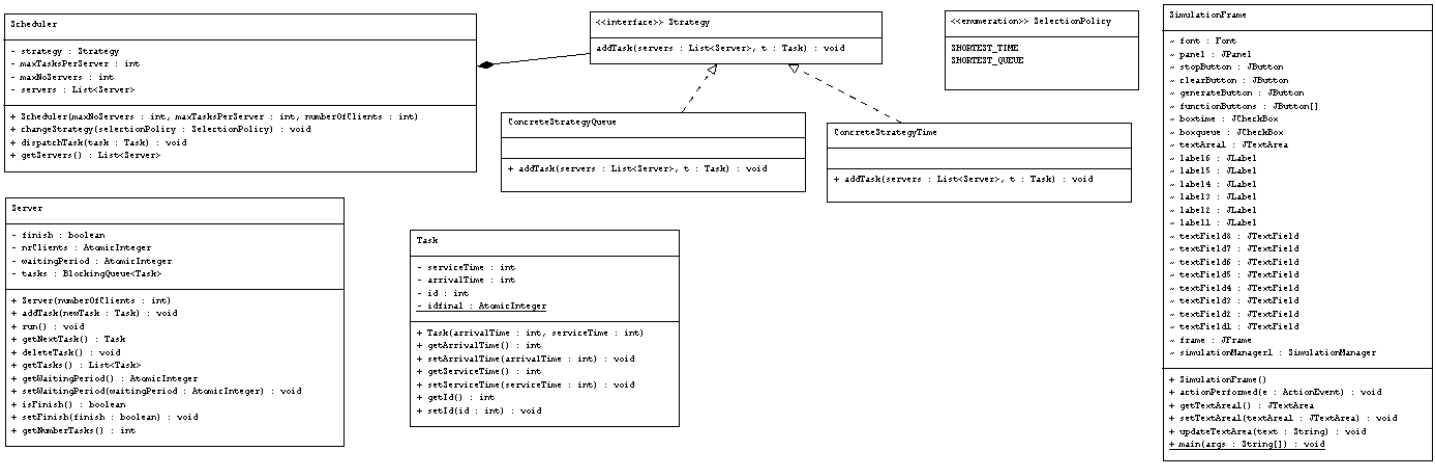
- Packages

Java packages are used to group together multiple modules and to connect related classes and interfaces.

The packages and classes are designed based on the Model-View-Controller(MCV) design pattern for a better implementation.

-Business Logic: contains the classes implementing the application data and the threads
-Model: contains the classes modeling the application data
-GUI: contains the classes implementing the graphical user interface

- Class Diagram



- Data Structures

The data structures that I've used are primitive data types: integers and even AtomicInteger; List of objects such as: ArrayList and queues like: BlockingQueue. I've created new objects such as: Task, Server, Scheduler. I've used BlockingQueues instead of Queues due to providing blocking put and take methods. AtomicInteger helps because if two threads are both reading and writing to a shared variable, then using the volatile keyword does not work.

- Algorithms

I have not used specific algorithms, but I've developed approaches to assist me in improving code efficiency. As a result, I chose appropriate algorithms for the problems, used efficient data structures, and avoided redundancy and unused code. In addition, I used object-oriented programming principles to make my code easier to read and maintain.

4. Implementation

- Class Design
- Model

A. Task

Constructor:

public Task(int arrivalTime,int serviceTime){}: sets the given arrivalTime and serviceTime.

Methods:
Getters and Setters.

B. Server

Constructor:

public Server(int numberOfClients) {}: based on the number of clients given, we create a ArrayList with the capacity of that same number and set the waitingPeriod and number of clients.

Methods:

public void addTask(Task newTask) {}: increments the waiting period and the counter of clients in the queue and adds the given task in the queue (at the end).

public void run() {}: runs the threads, checks if the first element of the queue has the service time equivalent to 0, if so we remove it and decrement the waiting period.

public Task getNextTask()}: gets the first task from the queue.

public List<Task> getTasks(){}: gets the tasks and stores them into a List of Tasks

public void deleteTask(){}: removes the first task.

public int getNumberTasks() {}: calculates the number of tasks in the queue.

- BusinessLogic

A. Strategy

Methods:

public void addTask(List<Server> servers, Task t){}: calls the addTask method.

B.Scheduler

Constructor:

public Scheduler(int maxNoServers, int maxTasksPerServer, int numberOfClients) {}: creates servers and threads based on the maxNoServers given and initializes the other two.

Methods:

public void changeStrategy(SelectionPolicy selectionPolicy) {}: changes the strategy of how to insert a task.

public void dispatchTask(Task task){}: sends the task to be added in the queue

public List<Server> getServers() {}: returns the list of servers.

C. SelectionPolicy

Is an enum with the given strings: SHORTEST_QUEUE, SHORTEST_TIME. One of them is selected for the respective strategy to be implemented.

D.ConcreteStrategyQueue

Methods:

public void addTask(List<Server> servers, Task t){}: receives the call when the right SelectionPolicy is selected. Adds the taks in the server's queue where the number of clients is minimum.

E.ConcreteStrategyTime

Methods:

public void addTask(List<Server> servers, Task t){}: receives the call when the right SelectionPolicy is selected. Adds the taks in the server's queue where the time of waiting is minimum.

F.SimulationManager

Constructor:

public SimulationManager(SimulationFrame frame, int timeLimit, int minArrivalTime, int maxArrivalTime, int maxProcessingTime, int minProcessingTime, int numberOfServers, int numberOfClients, SelectionPolicy selectionPolicy) {}: initializes the values, creates the strategy and the generated tasks.

Methods:

private void createStrategy(SelectionPolicy selectionPolicy){}: changes the current SelectionPolicy with the given one.

private List<Task> generateNRandomTasks(int numberOfClients, int minProcessingTime, int maxProcessingTime, int timeLimit){}: generates random clients with random arriving and service time.

public void run() {}:runs the threads, calculates the average service time, peak hour and the log of events and displays it in a text file.

public void startSimulation(SimulationFrame frame, int timeLimit, int minArrivalTime, int maxArrivalTime, int maxProcessingTime, int minProcessingTime, int numberOfServers, int numberOfClients, SelectionPolicy selectionPolicy) {}: creates the threads and starts the simulation based on the given parameters.

- GUI

A.SimulationFrame

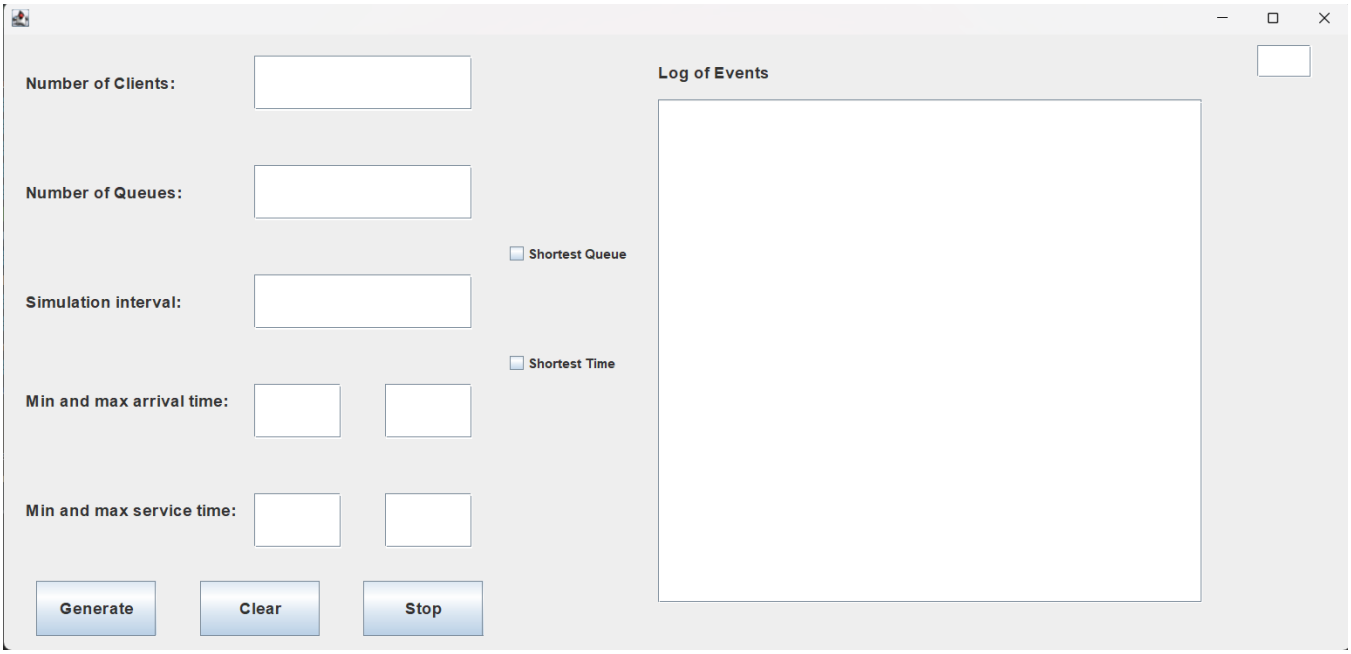
Constructor:

public SimulationFrame() {}: default constructor, generates the JButtons, JTextFields, JTextArea, JLabels, JPanel and JFrame.

Methods:

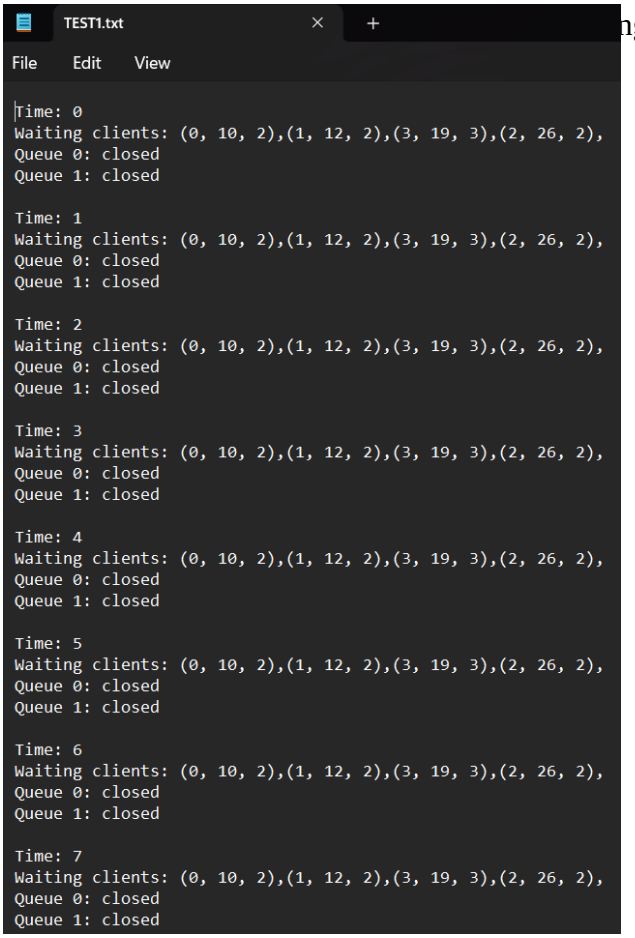
public void actionPerformed(ActionEvent e) {}: interprets the JButton pressed by the user and starts the simulation.

public void updateTextArea(String text){}: modifies the text in the TextArea.



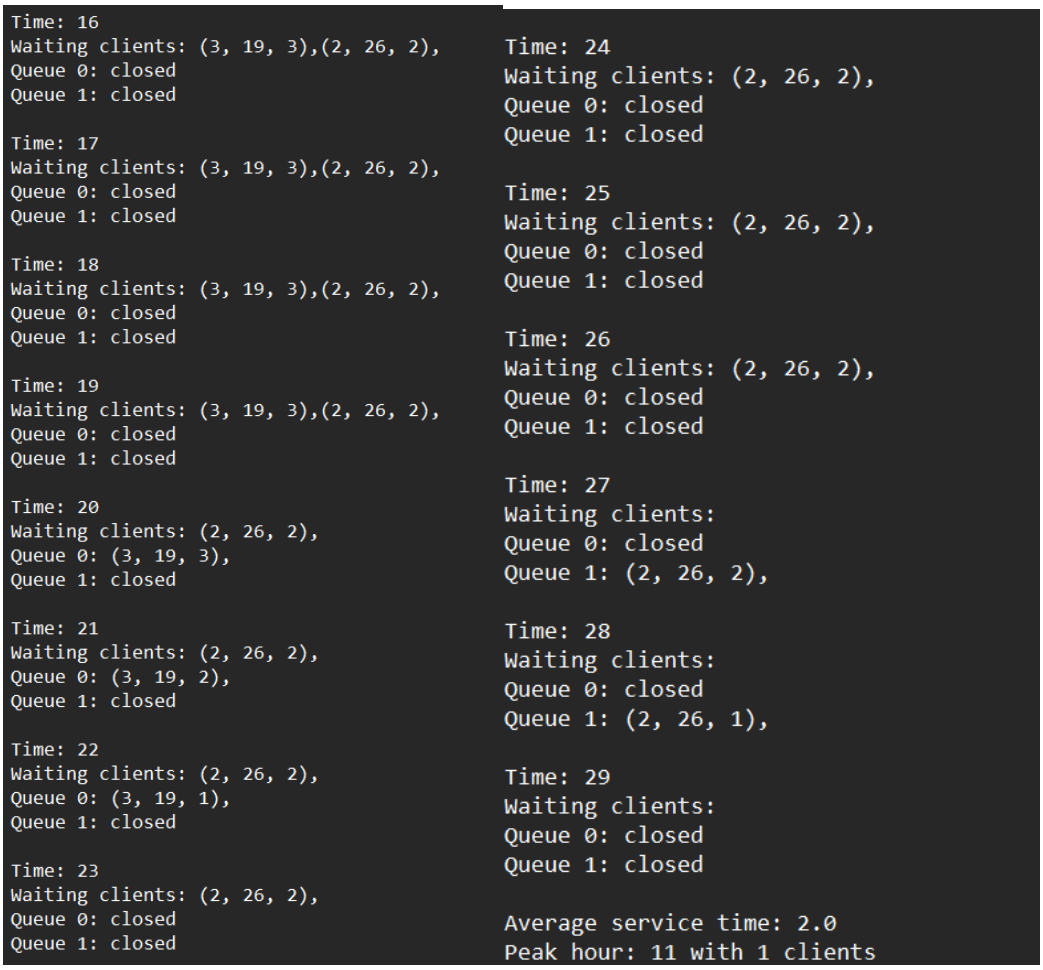
The interface is user friendly, with big and clear defined JLabels for the user to input the data in the correct TextField.

5. Results



ng the first set of test inputs:

```
Time: 8
Waiting clients: (0, 10, 2),(1, 12, 2),(3, 19, 3),(2, 26, 2),
Queue 0: closed
Queue 1: closed

Time: 9
Waiting clients: (0, 10, 2),(1, 12, 2),(3, 19, 3),(2, 26, 2),
Queue 0: closed
Queue 1: closed

Time: 10
Waiting clients: (0, 10, 2),(1, 12, 2),(3, 19, 3),(2, 26, 2),
Queue 0: closed
Queue 1: closed

Time: 11
Waiting clients: (1, 12, 2),(3, 19, 3),(2, 26, 2),
Queue 0: (0, 10, 2),
Queue 1: closed

Time: 12
Waiting clients: (1, 12, 2),(3, 19, 3),(2, 26, 2),
Queue 0: (0, 10, 1),
Queue 1: closed

Time: 13
Waiting clients: (3, 19, 3),(2, 26, 2),
Queue 0: closed
Queue 1: (1, 12, 2),

Time: 14
Waiting clients: (3, 19, 3),(2, 26, 2),
Queue 0: closed
Queue 1: (1, 12, 1),

Time: 15
Waiting clients: (3, 19, 3),(2, 26, 2),
Queue 0: closed
Queue 1: closed
```



6. Conclusions

This assignment provided a good challenge for me, helping me understand and improve my OOP skills. Along with fundamental OOP knowledge and programming skills, these kinds of projects also necessitate excellent time management skills and a good understanding of the problem's requirements and constraints. It was challenging, but it was impressive how I learned about project management - designing, implementing, and showing real time evolution - in a short time.

Working with threads proved to be one of the most challenging parts of the assignment. Nevertheless, I was able to gain a lot of knowledge and insight into their use and implementation.

A future development could be a better graphical user interface, with suggestive queue displayment.

7. Bibliography

The references that were consulted by the student during the implementation of the homework will be added.

Example:

1. *Java Checkbox - https://www.javatpoint.com/java-jcheckbox*
2. *Java Array - https://docs.oracle.com/javase/tutorial/uiswing/components/textarea.html*
3. *Java Threads - https://www.w3schools.com/java/java_threads.asp*
4. *Introduction to Threads - https://www.simplilearn.com/tutorials/java-tutorial/thread-in-java*
5. *Queues - https://www.programiz.com/dsa/queue*
6. *StringBuilder Java Class- https://www.javatpoint.com/StringBuilder-class*
7. *Iterators in Java - https://www.geeksforgeeks.org/iterators-in-java/*