DOCUMENTATION

ASSIGNMENT *ASSIGNMENT_2*

STUDENT NAME: Sîrca Florentina Raluca
GROUP: 30423

CONTENTS

1. Assignment Objective

Design and implement a simulation application that analyzes queuing-based systems to determine and minimize clients' waiting time.

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

One way to minimize the waiting time is to add more servers, i.e., more queues in the system, but this approach increases the costs of the supplier. When a new server is added, the waiting clients will be evenly distributed among all currently available queues.

The application should simulate a series of clients arriving for service, entering queues, waiting, being served, and finally leaving the queue. It tracks the time the clients spend waiting in queues and outputs the average waiting time. To calculate the waiting time, we need to know the arrival time, finish time, and service time. The arrival time and the service time depend on the individual clients – when they show up and how much service they need. The finish time depends on the number of queues, the number of other clients in the queue, and their service needs.

Input data:

- Minimum and maximum interval of arriving time between clients;

- Minimum and maximum service time;

- Number of queues;

- Simulation interval;

Minimal output:

- Average of waiting time and service time for (nrQueques) queues for the simulation interval, for a specified interval;

- Log of events and main system data;

- Queue evolution;

- Peak hour for the simulation interval;

2. Problem Analysis, Modeling, Scenarios, Use Cases

- General overview

The objective of this application is to replicate the experience of customers waiting to receive a service, such as in a supermarket or bank, mirroring real-world scenarios. In this simulation, customers are required to wait in queues, with each queue processing clients concurrently. The primary goal is to analyze the number of clients that can be served within a specific simulation interval. To facilitate this analysis, the application provides an intuitive and user-friendly graphical interface where users can input relevant parameters.

- Input and Output

The customers are generated randomly, each having its own service time and arrival time, the number of clients depends on the input values.

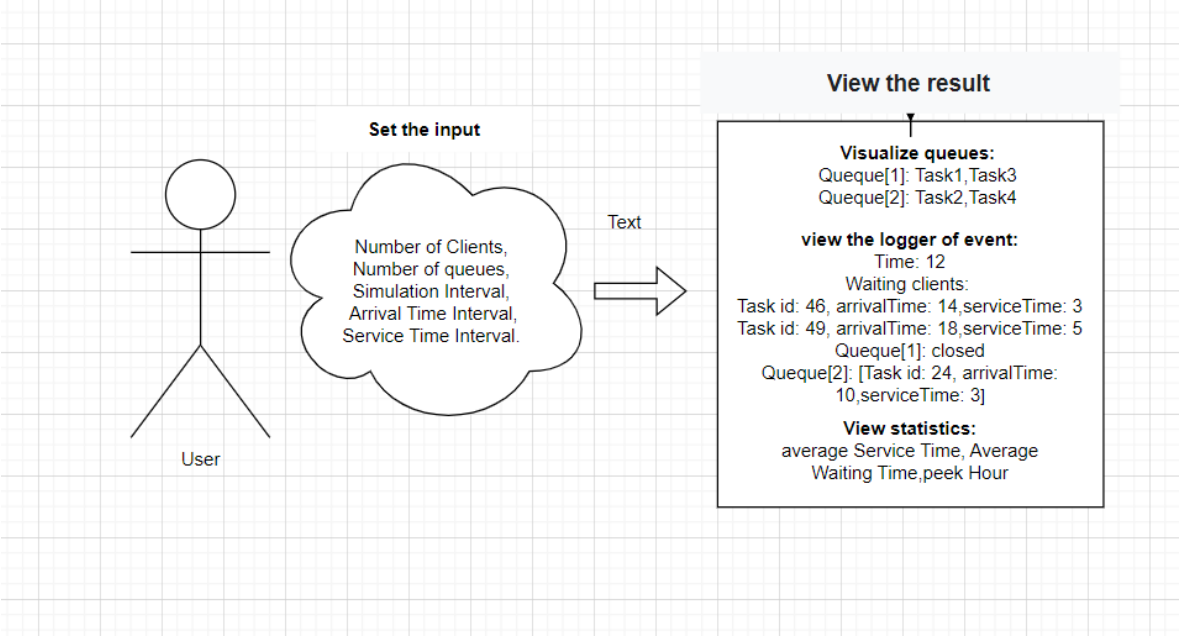
<u>The user can set:</u>

- The maximum number of queues available to process customers;
- The maximum number of clients which will arrive;
- Minimum and maximum arrival interval: the delay between customers arriving to receive a service (in seconds), a value is chosen randomly;
- Minimum and maximum service time: the number of minutes needed for a client to be processed, a value is chosen randomly;
- Simulation interval: the starting and finishing time of the simulation.

<u>The user can read:</u>

- How many customers were served (not generated), during the simulation interval;
- The average service time of the served customers;
- The average waiting time of the served customers;
- The total waiting time of all served customers;
- The “peak hour”, when the most clients were served;

- Use Cases



- Scenario

Preconditions: The user has introduced all the data; I assume that all the introduced data is correct, and it contains only digits.

Normal Scenario: The user has perfectly introduced all the required data inputs and it presses the “Validate Input Data” button, after this the application displays the log and the real time evolution of the queues. After this can see the “Statistics” and see what data has been calculated during the serving process.

Alternative Scenario: If the user doesn’t fill in all the input boxes, an error message will be displayed to remind him to do so, but again, the introduced input must be correct because it is not verified.

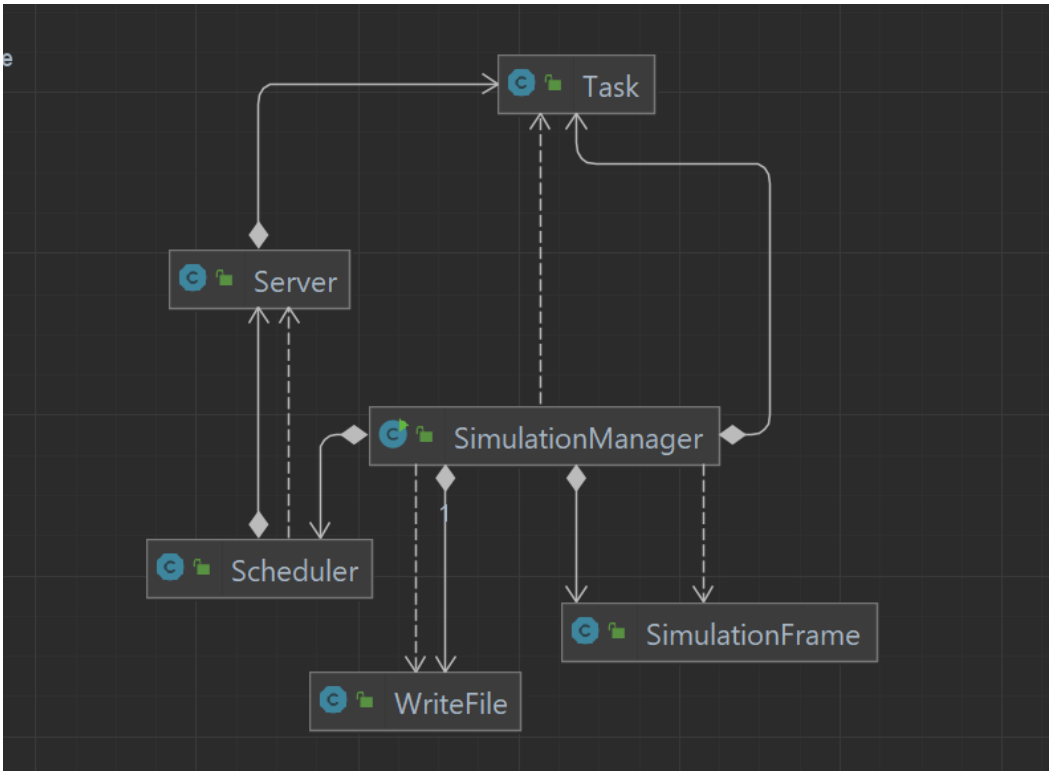
3. Design

- Design

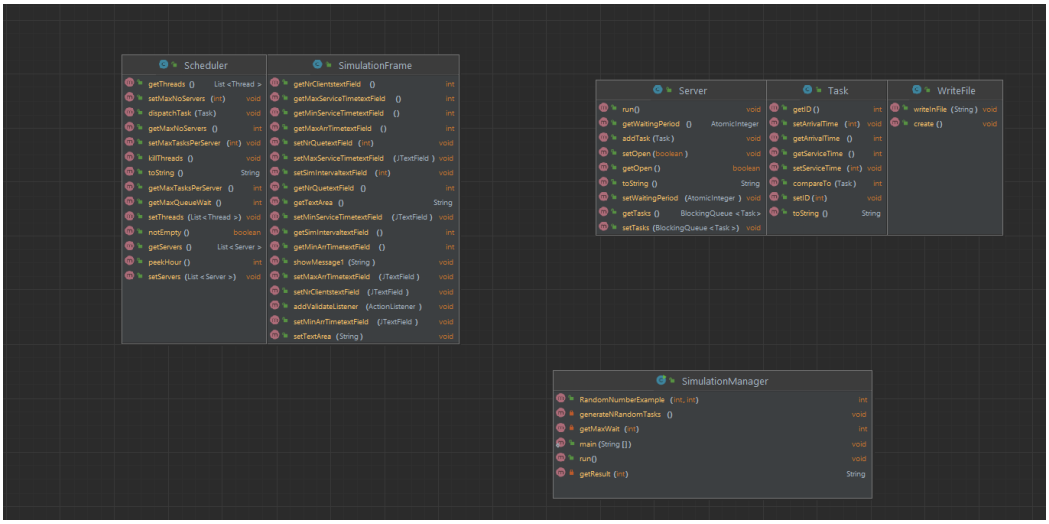
The system is designed using multiple classes, which each hold data to be used for the application, along with runnable classes to implement threads. All input metadata is inserted into a class Simulation Manager, which processes it and creates the system accordingly to the requirements. This class communicates directly with the Server, which actually starts and stops all threads, which is a Runnable class that keeps track of the time. The Server, after it generates the data and starts the clock, it starts all the threads. Once the client enters the Queue thread, it waits until the service time passes, only then the Queue is ready to accept another client. When all the clients are processed, the time thread is stopped, and the queue threads are all finished.

Every time a second passes, the log is updated, and in the end, the output file is updated, adding details about what is happening within the system; all of this is happening using a Write File class.

- UML Diagram



- Package Diagram



4. Implementation

Each class will be described (fields, important methods). Also, the implementation of the graphical user interface will be described.

Packages:

- BusinessLogic:

- *public Scheduler(int maxNoServers, int maxTasksPerServer);*
- *public SimulationManager(SimulationFrame view);*

- GUI

- *public SimulationFrame();* this is a user interface consisting of various elements that are visually presented to the user and can be interacted with, such as buttons, display boxes, and other components.

- Models: which represents the underlying, logical structure of data in a software application and the high-level class associated with it. This object model does not contain any information about the user interface.

- *public Server();*
- *public Task(int ID, int arrivalTime, int serviceTime);*
- *public void writeInFile(String name);*

Scheduler is a class that creates lists based on the number given as input. It also generates a list of threads that start when the lists are created.

- In this class there is also the *public void dispatchTask(Task t)* method that enters the task in the queue with the minimum number;
- the *public boolean notEmpty()* method checks if the queue is empty, information that is used in the Simulation Manager.
- The *public int peakHour()* method checks every second what is the maximum number of people placed in queues.
- The *public void killThreads()* method sets each list so that the thread in it can be stopped.
- The *public int getmaxWaitingTime()* will calculate the maximum waiting time in the queue from the entrance to serving.
- The *public String toString()* will display each queue with its tasks.

SimulationManager: It is a controller, which plays a crucial role in connecting the model and the view in an application. It serves as a mediator between the classes in the model and the view, facilitating communication and coordination between them.

- The *public int RandomNumberExample(int maximum, int minimum)* method generates a random number in the given interval.
- The *private void generateNRandomTasks()* method generates a task based on the random numbers method for id, arrival time and service time.
- The *public synchronized void run()* method that starts the main thread created by hand and that will make the whole project run. The method has a while that works for simulationTime seconds, this method, calculates for each client its perfect queue , calling the above mentioned method and then inserts the customer there, also a String is being

constructed which will appear on the GUI and in the file. Also here, peak time, average service time and average waiting time are calculated.

- The *private String getResult(int currentTime)* method displays the waiting list at the given time.
- The *public static void main(String[] args)* method is the main method.

Server: this is the class that implements the queue we need. The queue contains tasks sorted by arrival time. It has the characteristic of being able to be turned on and off, depending on how it is located. It is started at the beginning, and when the task ends its activity it will be stopped.

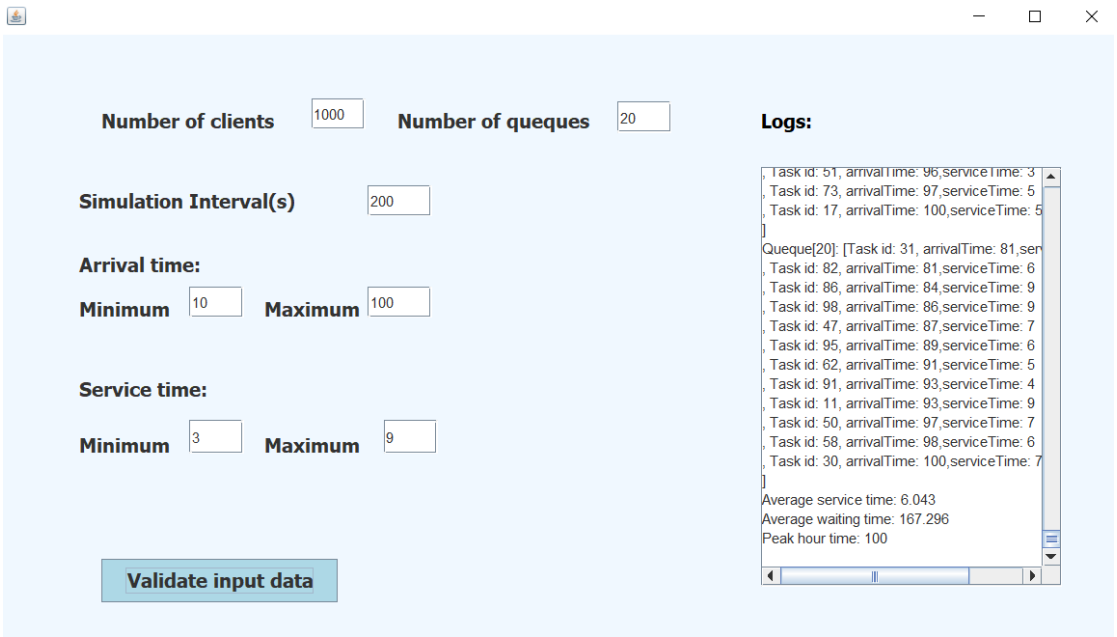
- The *public void addTask(Task newTask)* method that adds tasks to the list of tasks and increments the waitingPeriod for each list.
- The *public String toString()* method displays each task from the created queues.
- The *public synchronized void run()* method will get the client from the queue and will reduce his service time, until the moment he finishes and the task will leave the queue.

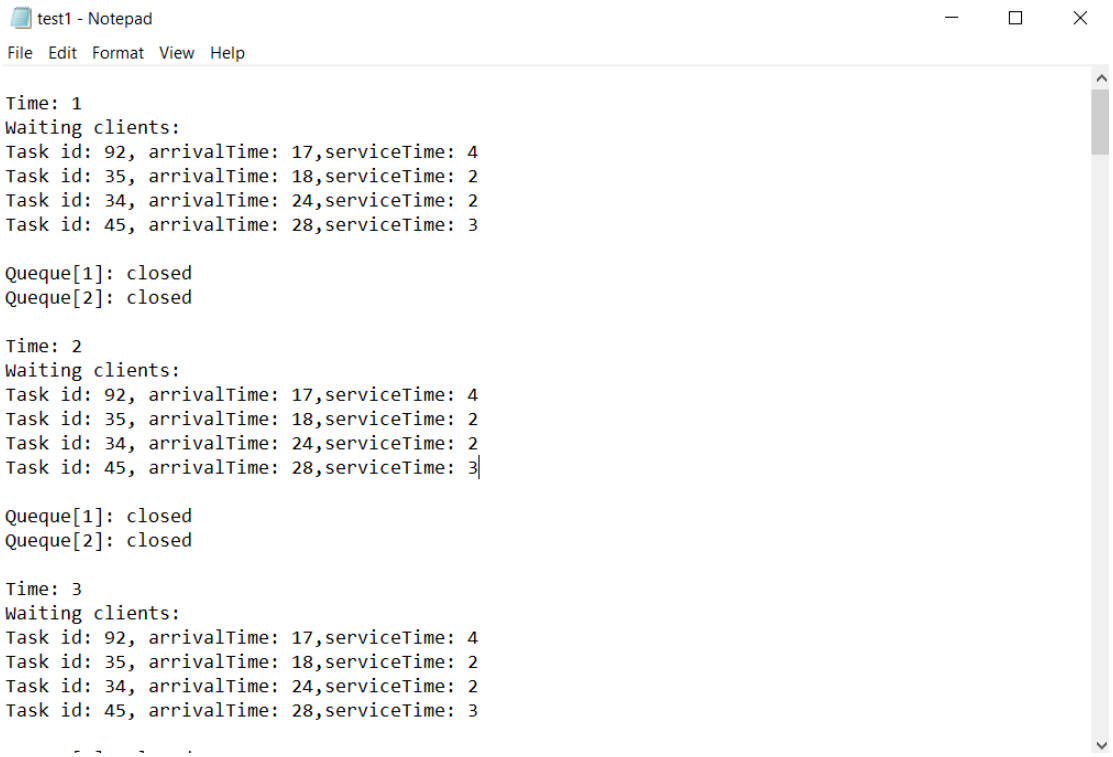
Task: it is a class that contains data about the task details, such as id, arrival Time and service Time.

- The *public Task (int ID, int arrivalTime, int serviceTime)* method sets the details about each task.
- The *public String toString()* method displays each characteristic of the task.

WriteInFile: it is a class that will open(or create if does not exist) a file where will store all the information about the functionality

5. Results





Based on the results, we can make the following observations:

- The Difference between Number of Clients and Number of Queues: The greater the difference between the number of clients and the number of queues, the larger the average service time and the average waiting time. This suggests that having a significant difference between the number of clients in the system and the available queues can lead to congestion and increased waiting times.

- Average Service Time: The average service time seems to be independent of when the clients entered the queue. This implies that the order in which clients join the queue doesn't have a significant impact on the average time it takes to serve them. The service time may be determined by other factors such as the efficiency of the service or the complexity of the task being performed.

- These observations highlight the importance of managing the number of clients and queues effectively to reduce waiting times in the system. It may involve strategies such as optimizing resource allocation, improving service efficiency, or implementing queue management techniques to ensure a smoother and more efficient client experience.

6. Conclusions

Undertaking this project provided a valuable opportunity to revisit the fundamental concepts of object-oriented programming (OOP) that were initially learned in the first semester. However, it also presented an avenue for acquiring new knowledge in the field. Initially, I found the project

to be both useful and challenging. As a result, there are several key takeaways that I would like to share.

First and foremost, I realized the immense significance of time management. Maintaining a strong sense of organization allows for a gradual progression and facilitates productivity. Effectively allocating time proved to be an invaluable asset throughout the project.

Furthermore, I discovered that appropriately modeling the problem from the project's inception expedites the implementation process. By establishing a solid foundation, the subsequent development stages were smoother and more efficient. Consequently, I arrived at the realization that confronting coding issues independently and engaging in research to rectify them not only enhances one's understanding of new concepts but also improves the utilization of existing knowledge.

Lastly, a crucial aspect I learned was the ability to create interfaces using code. Prior to this project, I had relied on tools like WindowBuilder during the previous semester. However, about a month ago, I recognized the necessity of being able to construct interfaces through code.

Throughout this project, I also had the opportunity to enhance my understanding of queue processing and its implementation within a system. Additionally, I delved into the realm of threads and synchronization, broadening my knowledge in these domains. Furthermore, I gained valuable insight into the significance of utilizing a logger and had the chance to explore various libraries dedicated to date and time functionalities.

Overall, this project provided an enriching experience that allowed me to consolidate my grasp of OOP concepts while simultaneously acquiring new knowledge. It served as a platform for personal growth and development in various areas, ultimately equipping me with a broader skill set and a deeper understanding of the subject matter.

7. Bibliography

1. *Bruce Eckel, Thinking in Java (4th Edition), Publisher: Prentice Hall PTRUpper Saddle River, NJUnited States, ISBN:978-0-13-187248-6 Published:01 December 2005.*
2. *What are Java classes? - www.tutorialspoint.com*
3. *http://docs.oracle.com/javase/tutorial/essential/concurrency/index.html*
4. *http://www.tutorialspoint.com/java/util/timer_schedule_period.htm*
5. *http://www.javacodegeeks.com/2013/01/java-thread-pool-example-using-executors-andthreadpoolexecutor.html*