DOCUMENTATION

ASSIGNMENT *2*

STUDENT NAME: Soponar Darius Cristian

GROUP: 30424

# CONTENTS

[1. Assignment Objective 3](file:///C:\Users\Darius\Downloads\QueueSimulator-Documentation.docx#_Toc128043139)

[2. Problem Analysis, Modeling, Use Cases 3](file:///C:\Users\Darius\Downloads\QueueSimulator-Documentation.docx#_Toc128043140)

[3. Design 3](file:///C:\Users\Darius\Downloads\QueueSimulator-Documentation.docx#_Toc128043141)

[4. Implementation 3](file:///C:\Users\Darius\Downloads\QueueSimulator-Documentation.docx#_Toc128043142)

[5. Conclusions 3](file:///C:\Users\Darius\Downloads\QueueSimulator-Documentation.docx#_Toc128043144)

[6. Bibliography 3](file:///C:\Users\Darius\Downloads\QueueSimulator-Documentation.docx#_Toc128043145)

1. Assignment Objective

Main objective: The main purpose of this application is to simulate a number of tasks with randomly generated properties within a number of selected queues.

Subobjectives:

* Generating the random id, service time and arrival time.
* Using for every queue and for the waiting tasks separate threads with common resources.
* Implementing the 2 types of Strategies: Shortest Queue Strategy and Shortest Time Strategy.
* Creating a graphical interface for our user

1. Analysis, Modelling, Use Cases

**2.1 Problem analysis**

A thread in Java is the direction or path that is taken while a program is being executed. Generally, all the programs have at least one thread, known as the main thread, that is provided by the JVM or Java Virtual Machine at the starting of the program's execution. Basically a thread is a sub-process, and in Java we implement this using either the interface Runnable, or the extension of the class Thread. We override the run() method of these classes and that is where we code what commands our thread must execute.

**2.2 Modeling**

First we create 2 classes: Task and Server. In the Task class we store data for our tasks: the id, arrival time and service time. The word task in this project can be changed with the word client. The server class is where we store tasks that must be processed.

**2.3 Use Case**

In order for our user to interact with the application, he has the graphical interface of the application(GUI).

When the user runs the application he will see 7 text boxes with the following labels: Number of Clients, where he has to write the number of tasks he wants to process; Number of queues, where he has to write the number of queues he wants to use for the processing of the clients; Simulation interval, where he has to put the maximum number of seconds for his simulation; Minimum arrival time, where he introduces the lower bound for the arrival time of any client; Maximum arrival time, where he introduces the maximum bound for the arrival time of any client; Minimum service time, where he can write the lower bound for the service time of any client; Maximum service time, where he can write the maximum bound for the service time of any client.

The first page of the graphical interface has 1 button: “Start simulation”. When the user presses it, the input data is validated and then the user is shown the next page of the GUI and that is the simulation page. Here he can see the real-time evolution of the desired simulation in the following manner: On the left he has a box with a label at the top, “Waiting clients”, where he can see the clients that are still waiting to be put in a queue. On the right he can see the number of queues that he wanted to be generated, and in each queue either the word “empty”, if the queue is empty, or the clients that are processed. On the bottom left, he can see the timer, with the label “Timer”, that counts from 0 up to the maximum time limit that he has introduced before. When the simulation is done, our user gets to see the third page of the GUI and that is the Results page: The log of events is displayed along with the average waiting time, average service time and the peak hour of his simulation. On the top of the screen he can see a button labeled “Go home”, and if pressed he can introduce new data for a new simulation and run the simulation again.

1. Design

***UML Diagram***

A picture containing table

Description automatically generated

A picture containing chart

Description automatically generatedTable

Description automatically generated

1. Implementation
2. ***Model***:

Class Task:

In this class we have the following fields: private int id, where we store the id of our task; private int arrivalTime, where we store the arrival time of our task; private int serviceTime, where we store the time it takes for our task to be finished when it gets to the front of the queue.

It also has the basic methods: setters, getters and constructors.

* *String toString()*: method used to parse the Task from its class form to a String.

Class Server:

In this class we have the following fields: private BlockingQueue<Task> tasks, where we store the tasks that are going to be processed on that Server; private AtomicInteger waitingPeriod, where we store the total waitingPeriod for a queue; private Boolean finishedSimulation, which is a Boolean that helps us to stop the threads that are running after the simulation is finished.

It also has the basic methods: setters,getters and constructors.

Methods:

* addTask(Task) – this method is used to add a task to the tasks variable and add to the waitingPeriod variable the Service Time of that task
* run() – it is an overridden method, since this class implements the Runnable interface. In this method we check whether a Task is done being processed and if so , we delete that task from the blocking queue.

Interface Strategy:

In this interface we declare the methode addTask(task List,task) which then will be implemented by the other two classes.

Class ConcreteStrategyTime:

In this class when we want to add a Task to our Server we check in which server the waiting time for our task is the shortest, then we add the task to the selected server.

Class ConcreteStrategyQueue:

In this class when we want to add a new task to the server we check in which server the number of tasks to be processed is the smallest and after that we add the task.

Class Scheduler:

This class has the following fields: private List<Server> serverList, where we store the list of Servers we want to use for our Task processing; private Strategy strategy, where we store the strategy we want to use for our simulation; private List<Thread> threads, where we have the threads we need for the Servers to work.

Methods:

* public void dispatchTask(Task) – method used to send a task to a certain queue, based on the strategy the user has chosen.
* public void startThreads() – method used to start all the threads needed for our simulation.
* public void stopThreads() – method used to stop all the threads we used for our simulation.
* public void stopServers() – method used to stop all the servers we used for our simulation, by setting their finishedSimulation Boolean to True.

Class SimulationManager:

In this class we have the following fields: timeLimit where we store the nr of seconds,maxService where we store the upper bound ,a minService for the opposite reason, nrServers for the number of servers, nrOfTasks where we store the number of tasks, minArrival and maxArrival for the upper and lower bounds of the arrival time of a task, The scheduler for running the simulation, etc.

Methods:

* public void run() – It is an overridden method, since this class implements the Runnable interface, and represents the main thread of our application. In this thread we implement the simulation: We go through the task list, while having a variable(currTime), for storing the time we are at for each step. Each time we find a task, which has an Arrival Time equal with the Current Time, we dispatch it to a Server, based on the strategy we have chosen. When the Current Time is equal to the Time Limit, or the queues are all empty, including the queue with the waiting clients, the thread stops, and we write the event log and final results in the GUI.
* public void generateNTasks() – This method generates N tasks with random Arrival Times and random Service Times, based on the bounds introduced by the client.
* public boolean emptyServers() – This method checks whether the servers are empty or not, including the server with the waiting clients.

**b)View:**

Class HomePage:

In this class we created the necessary text fields and buttons for the GUI and implemented the methods needed to run the Interface.

Class SimulationGUI:

In this class we dictate the details of the way the interface looks, the font size, color and the overall aspect of the GUI.

1. Conclusions:

From this assignment I learned how to work with threads, since I haven’t worked with threads in Java before. I also learned the difference between the interface Runnable and the class Thread, and how even if they look the same when used, they are slightly different. I discovered the synchronization techniques that are used in Java, along with the most used Thread-safe types of data. Now I know how to use threads efficiently and how to optimize them.

I also believe that I improved my front-end development skills, because I tried to make the application as user friendly and easy to use as possible.

Another important thing I learned was the Strategy design pattern. I have heard before of it, but never actually used it in practice, but I’m glad I did for this assignment, since I believe it is a very useful approach to different types of applications, especially video games.

For future development, I believe I can make the GUI a bit more visually pleasing, since this is a very primitive form just for visualizing the evolution in real-time of the simulation. I could add a Stop button, which would stop the simulation for as long as the user wants, and then he can continue it, and so on. Also I could make the formatting of the text a little bigger.

1. Bibliography:

* Website for learning Swing: <https://www.javatpoint.com/java-swing>.
* Website for learning Java: https://www.w3schools.com/java/java\_threads.asp
* Book about threads and thread-safety in Java: https://leon-wtf.github.io/doc/java-concurrency-in-practice.pdf