Technical University of Cluj-Napoca

Programming Techniques

Laboratory – Assignment 2

Queue Management System

*Logo, company name

Description automatically generated*

Teacher: prof. Cristina Bianca Pop

Teacher Assistant: Cristina Bianca Pop

Student: Siladi Ciprian

Group: 30424\_2

# Objective

The main objective is to design and implement a simulation application aiming to analyze queue-based systems for determining and minimizing their client’s waiting time. The application must use multi-threading and synchronization mechanisms. Also, the application should have an easy to understand and intuitive graphical interface for a good interaction with the user. Through this graphical interface, the user should be able to enter the specific details about the simulation and see relevant information during the execution, as well as at the end of it.

In order to achieve this, we have to take into consideration the following objectives:

* Design a client
* Design a queue
* Create a thread for each queue
* Design a system that manages the clients and the queues
* Design strategies for dispatching the clients
* Design a graphical user interface

# Analysis, modelling, scenarios, use cases of the problem

## Problem Analysis

A simulation application can be useful in various situations when a process needs to be modeled and tested. By using a simulation application, a process can be represented and its outcome can be visualized in order to guarantee its proper working.

The system I implemented can be helpful for analyzing real life scenarios, where we have clients or tasks that need to be distributed to different queues/servers by helping the administrators to decide how many queues should be open at a time of the day based on the average waiting time and peak hours, but also helping the users to choose the queue with the shortest waiting time.

## Modelling

First of all, it is necessary to have the basic objects in order to implement the requirements: a

Client/Task and a Queue/Server. In order to simulate the process, we need to have the time of arrive and also how long it takes to finish the task. We also need to have an entity which handles the simulation of the program, generates clients, decides how and when to dispatch clients, calculates statistics, prints the output to the corresponding file and so on.

Therefore we have the following classes:

* Client: the object of the simulation which has as attributes the arrival time and the service time, as well as a unique id in order to identify each client.
* Queue: the entity that contains the clients and serves them accordingly.
* Strategy: a functional interface with the purpose of deciding at which queue to add a new Client. This can have multiple implementations, but for this project I chose to implement an approach similar to a real one: adding the client to the shortest queue in size, but also an ideal approach which adds the client to the queue with the least waiting time.
* Scheduler: the entity which dispatches the clients in the queues according to the chosen strategy
* Simulation Manager (the “brain” of the operation): the object which controls the entire simulation. It also handles the transmission of data to the graphical user interface and the file writing of the outputs.

## Scenarios, Use Cases

Scenarios include the simulation of this real life situation. Data is provided to define the restrictions and

the system starts the simulation at the press of a button with the information provided by the user.

The input data is the following:

* Number of clients
* Number of queues
* Duration of the simulation
* Minimum and maximum interval of arriving time between two clients
* Minimum and maximum interval for service time
* Strategy selector between shortest time and shortest queue
* Output file

All these data, except the strategy selector which is a choice box and the output file which should be a

string are assumed to be integers that are correctly introduced in the text fields.

As an output, the user can see at each second of the simulation time, the waiting clients, the queues and

the clients who are waiting to be served along with their id, arrival time and service time left and the waiting period left in the respective queue.

After the time limit is reached the average waiting time and the average service time are displayed on the screen through an alert as well as the peak hour (second).

To identify clearly they way the objects interact with each other I will describe the main scenario for using the application:

|  |  |
| --- | --- |
| Main scenario | |
| Actors | User |
| Pre-conditions | The system is in waiting for the client to introduce the correct data and to press the start simulation button. |
| Post-conditions | The system displays at each second how the simulation is done and also displays at the end the final results. It also writes in a .txt file the step by step descripiton of the simulation. |
| Main succes scenario | 1. The user introduces correctly the required information (time limit, number of queues, number of clients, minimum and maximum values, etc). 2. Client presses the start simulation button. 3. System starts the simulation, showing at each second the waiting clients and the clients in the queues. 4. Simulation ends and the statistics are displayed on screen as well as written in the output file. |
| Main failure scenario: | 1. Client introduce the wrong data in the text fields 2. Start simulation button doesn’t work |
| Alternative scenarios: | 1. Client introduces the values. 2. Presses start simulation. 3. Presses stop simulation before the timi limit is reached and the statistics up to that point are displayed on the screen. |

Although the scenarios are fairly simple, I have designed a flux diagram for a better understanding of the flow of the program:

Diagram

Description automatically generated

# Design

## Class design and UML Diagram

The design of this assignment is based on using and handling threads. The designed classes are not so complicated, their functioning and utilities are easy to understand since the complexity of the project is not so big.

A picture containing text, map, sky

Description automatically generated

The first steps were to design the entities like Client or Queue, which was not difficult since their responsibilities are straightforward. After these classes, designing the structure of threads and the system where they are working was a key step. For this classes named Scheduler and SimulationManager were used.

In the class Queue a thread was implemented, which is running if there is a client to serve. If not, the thread stops and when a new client is added, a new instance of the thread is created in the scheduler and started.

The Scheduler class is responsible for creating the queues and starting their threads. It has another important role, which is working as a dispatcher of the clients that arrive. It has a method to change the strategy based on which one the user chooses.

The SimulationManager class is responsible for creating a thread where a simulation can take place. The simulation’s step is one second, so each second clients arrive and are dispatched in a certain queue. In order to achieve these, the class has another tasks, like generating the clients, take information from the queues, and finally send the filtered information to the output file.

The FileService class is a simple class, used for reading information from an input file and displaying information into an output file.

The Strategy interface is responsible for the strategy used to dispatch the clients. It is implemented by the ConcreteStrategyTime class which chooses to place the client in the queue with the smallest waiting time (this information is calculated by each queue) and also by the ConcreteStrategyQueue which chooses to place the client in the queue with the smallest size.

## Packages and Relationships

This assignment is structured similarly to a project that uses Model View Controller pattern.

In the package **model** we have the Client and the Queue classes.

In the package **controller** we have all the components which are part of the simulation or prepare the simulation. Therefore, here we find the classes FileService, Scheduler, SimulationManager, Controller ( for the GUI) ,ConcreteStrategyTime, ConcreteQueueTime and the Strategy interface.

The package view consists of the FXMLoader from the Application and the Controller class which controls the GUI interface.

## Algorithms and Data Structures

The algorithms used by a Queue is the following: take a client and serve him (put the thread to sleep

until the client is served) and decrement the waiting time of the queue.

The algorithm used by Scheduler is finding the index of the queue which has the smallest waiting time.

The algorithm iterates through the list of queues, finds the minimum waiting time and saves the index of that queue. After the desired queue is found, the new client is inserted there.

The SimulationManager class uses algorithms for generating clients, running the thread and calculating statistics. When a client is generated, it is provided with a ClientID, randomly generated arriving time and service time, between the given boundaries.

# Implementation

## Class Client

Table

Description automatically generated

The class Client represents a client (in other words a task which needs to be completed). As attributes it has: arrivalTime, serviceTime, id all of them being integers.

In the constructor of the class, values for arrival time, service time and client ID are assigned.

The class has getters for every attribute, functions which help setting different values to these attributes, according to the simulation’s state, and has an overridden toString method which converts the class into an easy to read string.

## Class Queue

Text

Description automatically generated

The class Queue represents a queue, implementing its corresponding functionalities. In order to be able to run threads with this type of object, the class implements the runnable interface. The attributes of this class are the following:

* private AtomicInteger waitingPeriod – the waiting time of the queue, updated when clients arrive or have been served.
* private BlockingQueue <Client> clients – represents the clients that are waiting in the queue

The addClient() method adds a client to the queue.

The run() method is overridden from the Runnable interface, where a client is served (the algorithm was discussed before).

The getQueueSize() method returns the size of the BlockingQueue, similarly the getWaitingPeriod() method gets the current waiting period.

The setStop() method can be used to stop the running of the queue.

## Class Scheduler

Text, table

Description automatically generated

Scheduler class is responsible for dispatching clients and creating the queues.

As attributes it has an ArrayList of queues (private ArrayList<Queue> queues), an integer representing the maximum number of queues generated (private int maxNoQueues), and a Strategy variable representing the current used strategy.

In the constructor the queues are initialized and their threads are started.

The changeStrategy() is a method for changing the current strategy, between the shortest time strategy or the shortest queue strategy.

The dispatchClient() method takes a client and places him in the corresponding queue.

The notifyThreads() method notifies the threads that run the queue and therefore starts the running of that specific thread, similarly stopThreads() method uses the setStop() function from the Queue class and stops the running of the threads.

## Class SimulationManager

Table

Description automatically generated

Class SimulationManager is a class that controls the threads and generates the random clients, it implements the Runnable interface.

The Constructor initializez the values required for the simulation by the given parameters (I did not write all of them because there are a lot).

The setController() is a method is a method required for the proper functioning of the GUI implemented in JavaFX, because JavaFX runs on a single thread, therefore we need to specify which thread controls the JavaFX thread.

The setOutputFile() is a method which gets a String and creates the output file with the information about the simulation.

The run() method is responsible for the simulation. It basically consists of a while loop, which loops until the given time limit is reached. At every step, the current time is listed, then the clients with an arriving time equal to current time are added to the corresponding queues. The waiting clients and queues are listed, and the thread sleeps for one second in order to simulated the 1 second time period. After the loop is finished, the average waiting time, average service time and peak hour are calculated and sent to the output.

## Class FileService

Text, table

Description automatically generated

The FileService class is a simple class responsible for writing in the output file the step by step

Simulation.

It has 3 static methods since we don’t need to instantiate the class in order to be able to call those methods:

* setFile() sets the name of the file in which the program should insert the information.
* write() is the method that inserts the given string in the file.
* closeFile() is the method that closes the file.

## Interface Strategy

Text, letter

Description automatically generated

The Strategy interface is responsible for selecting the working strategy. It has an addClient() method which adds a client to a queue

## Class ConcreteStrategyTime and ConcreteStrategyQueue

Table

Description automatically generated with low confidenceA picture containing table

Description automatically generated

The classes implement the Strategy interface.

They have an addClient(queues, client) method, which receives a client and a list of queues, and places the client in the queue based on which strategy it is chosen. ConcreteStrategyTime inserts the client into the queue with the smallest waiting time, while ConcreteStrategyQueue inserts the client into the queue with the smalles size.

## Class Controller

Text

Description automatically generated

The Controller Class is responsible for controlling the GUI which was loaded by the FXMLoader from the Main class.

The startSimulation() method is a method which is started at the press of the Start Simulation button and retrieves the inserted data from the interface and starts a thread with Simulation Manager.

The stopSimulation() method is a method which is started at the press of the Stop Simulation button and it is used for the sudden stop of the simulation at the users decision and displays the results so far.

The following 4 setter methods are used at the end of the execution for displaying information such as:

Average waiting time, average service time, peak hour.

## Class Main

Text

Description automatically generated

This class simply loads the .fxml file required for the GUI and launches the controller.

The GUI looks has an easy to use and intuitive design which makes the simulation very easy for the user.

Graphical user interface, application

Description automatically generated

# 5.Usage and testing

The simulation of the application can be done by providing the data in the user interface or by changing it directly in the SimulationManager class. We will discuss the first one.

The user has to provide correct inputs for the simulation and press the start simulation button. After the execution, the log of the program will be printed(Which can also be seen directly in the user interface). The user can see at each time moment the clients who are waiting to be added in a queue sorted by their arrival time and the clients who are currently in queues and being processed. After the simulation time limit is reached, the average waiting and service time is printed as well as the peak hour.

The output file will look like the example beneath:

Text

Description automatically generated

Text

Description automatically generated

# 6.Conclusions

During the development of this assignment, some research was needed for understanding the concepts of concurrent programming. After this assignment I can say that I am more familiar with creating and using threads.

A newly learned concept was the Strategy Design Pattern, which allows the user to choose between multiple algorithms given to do some task.

Future developments:

* Other statistics (can be easily calculated)
* History of the statistics, methods of using these data to actually help businesses

# 7.Bibliography

* <https://dsrl.eu/courses/pt/materials/A2_Support_Presentation.pdf>
* <https://stackabuse.com/reading-and-writing-files-in-java/>
* <https://www.geeksforgeeks.org/killing-threads-in-java/>
* <https://www.tutorialspoint.com/java/java_multithreading.htm>
* <https://docs.oracle.com/javase/tutorial/essential/concurrency/runthread.html>
* <https://www.tutorialspoint.com/design_pattern/strategy_pattern.htm>