**DOCUMENTATION**

ASSIGNMENT NUMBER 2

STUDENT NAME: VOIVOD OVIDIU – FLORIN

GROUP: 30423

**CONTENT**

1. Assignment’s objective ……………………………………………………………………………………... 3
2. Problem analysis, modeling, scenarios, use cases ………………………………………………………….. 3
3. Design ………………………………………………………………………………………………………. 5
4. Implementation ……………………………………………………………………………………………... 7
5. Results ……………………………………………………………………………………………………..... 9
6. Conclusion …………………………………………………………………………………………………. 10
7. Bibliography ……………………………………………………………………………………………….. 10

1. **Main purpose of the project**

**1.1 Main objective**

The main purpose of the project is to design and implement an application aiming to analyze queueing-based systems by:

* Simulating a series of N clients arriving for service, entering Q queues, waiting, entering in queue, waiting in queue to be served, being served and finally leaving the queue.
* Computing the average waiting time, average service time, and peak hour.

**1.2 Sub – objectives**

* Analyze the problem and identify the requirements and implementation
* Design the simulation application for the queues management system
* Implement the simulation application
* Test the functionalities and assure the correctness and good functioning of the application

1. **Problem analysis, modeling, scenarios, use cases**

**2.1 Functional requirements**

* The queues management application should allow users to setup the simulation data (no. of clients, no. of queues, simulation time, min and max arrival times, min and max service times)
* The queues management application should allow user to start the simulation
* The queues management application should display the real time data, clients waiting, clients waiting in every queue, clients being processed at the current moment.
* The queues management application should inform user if the input data is wrong.
* The queues management application should allow user to refresh the input data fields.

Diagram

Description automatically generated

**2.2 Use cases**

**Use Case:** simulation setup

**Primary Actor:** user

**Main Success Scenario:**

1. The user inserts the values for the number of clients, number of queues, minimum arrival time, maximum arrival time, minimum service time and maximum service time
2. The user clicks on “Start” button
3. The application validates the input set by the user
4. If everything is alright the application starts the simulation of the queues management app

**Alternative Sequence:** Invalid values for the setup parameters (input data for simulation)

* The user inserts invalid values for the application’ setup parameters
* The application shows a pop-up window which displays a message to announce the user that the input data is not correct, and he’ll have to insert valid values.
* The scenario returns to step 1

**Use Case:** reset

**Primary Actor:** user

**Main Success Scenario:**

1. The user inserts the values for the number of clients, number of queues, minimum arrival time, maximum arrival time, minimum service time and maximum service time
2. The user clicks on “Reset” button
3. The application resets the input fields of the GUI, deleting the input inserted by the user.
4. The scenario returns to step 5

**Alternative Sequence:** User presses “Reset” while simulation is running

* User presses reset button while simulation is already running.
* Simulation won’t be affected but the input data fields will be reset and will not have any data in them.
* Simulation finishes and the scenario returns to step 1 or 5

**2.3 Non – Functional Requirements**

**-** The simulation application should be intuitive and easy to use by the user.

- The application should clearly display the simulation such that any user would understand what it is shown in the simulation.

**3.Design**

**3.1 OOP Design**

The application is designed based on a MVC (model view controller) design pattern. In the view part is the implementation of the GUI, the visual part, only what the user can see. In model we have the classes which represent the “models” for the application, the real-world objects. In the Controller we have the “business logic”, the connections between model and view. The controller contains the most important parts for the simulation application. Here we take the input parameters for the simulation and use them for creating the clients, the queues, the threads and also here we assign every client to the most suitable queue in order to be processed.

**3.2 Package Diagram**

**A picture containing logo

Description automatically generated**

**3.3 Class Diagram**

Diagram

Description automatically generated

**3.4 Data Structures**

ArrayLists are the main data structures used in the Queues Simulation Application’s implementation. Array List in Java is used to store dynamically sized collection of objects. Compared to Arrays which have fixed sizes, array lists size grows automatically when new elements are stored in it.

Array Lists is a part of Java’ s collection framework which implements List interface. They have almost the same characteristics as the traditional array, with some improvements and features which facilitate a correct and easy implementation of the applications. Array Lists are dynamic arrays which grow or shrink their sizes according to the number of elements stored in the collection. They maintain the insertion order of the elements and allow us to access the stored elements by their index, like in the traditional array.

BlockingQueue is another data structure used in the application’s implementation. This data structure represents a traditional queue which is mainly used when we work with threads. It supports operations that wait for the queue to become non – empty when retrieving an element and also wait for space to become available when storing an element.

**3.5 Defined Interfaces**

In the implementation of the simulation application we have defined an interface called “Strategy”. It defines a “addClient” method which we use in 2 classes which implement this interface. Both of those classes have a method which deals with the strategy in which we add the clients in queues, first according to the length of the queues, and second according to the waiting time for the queues.

We also use the Runnable interface which is an interface used when we want to work with threads. This interface contains the run method. This method contains the code which will be executed by the thread we start for the class which implements this method.

The comparable interface is used to override the compareTo method, for facilitating the easy sorting of the arraylist, which stores the waiting clients, by their arrival times.

1. **Implementation**

This application is implemented using the MVC (model view controller) design pattern, which helps us make a separation between the classes which have different roles in the implementation of the full application. The whole application is divided in 3 main packages. Every package corresponds to one of the MVC’s components and contains important code which contribute to the good and complete functioning of the application.

* 1. **Model**

Encapsulates the core date and functionality of the application. It contains all the classes and methods which contribute to the functioning of the application.

* + 1. **Client**

It’s the class which defines the clients, as the name says. A client object contains and is defined by 3 Integer attributes: id, arrival time and service time. This class implements the “Comparable” interface because we want to override the “compareTo” method here. We have to override this method to be able to sort the arraylist of clients waiting to be sent to a queue, by their arrival time. If we wouldn’t done this, the compiler wouldn’t have known what sorting criteria it should use for sorting the clients from the arraylist. This class contains also the “toString” method in which we represent the attributes of the client as s string. This is very useful when we make the in real time simulation because we can display on the GUI every client along with its details.

* + 1. **InputValidator**

We use this class in the app implementation to determine whether the input given by the user through the graphical user interface is correct. This class has a method called validator which takes as input a string. This method checks if the input given by the user contains only numbers or it contains other characters. This is done with the help of Pattern and Matcher classes. If the input string contains only numbers, the method returns “true”, otherwise it will return “false”.

* + 1. **Queue**

This is a very important class in the implementation of the simulation app. Queue class implements two interfaces: Runaable and Comparable. This class implements the runnable interface because we want to create a thread for every Queue object created, to handle with the clients from the respective queue, which will be done in parallel with the other queues and also with the thread from Simulation Manager. The Comparable interface is used to override the compareTo method which compares the waiting periods of the queues. Every Queue has, as attributes a blocking queue, in which we store the clients which are waiting in that specific queue, and the waiting period, which is an atomic integer. We use blocking queues and atomic integers because they support working with threads. They help us avoid desynchronization and wrong operations on data due to working with more threads.

An important method implemented in this class is the “run” method. This method is executed by another thread for every new object of this class. In this method we access the client has to be processed next in that queue. We decrement it’s service time with 1 every second until his processing service time has passed, and then we remove that client from the queue and update the waiting time for the specific queue. The toString method iterates through the blocking queue and calls the to string method for every client from the queue. This method helps us with the in real time display of the queues in the simulation.

* 1. **Controller**
     1. **ConcreteStrategyQueue**

This class implements the Strategy interface. This class contains only a addClient method. This method decides in which queue should a client be the most suitable to be placed. “AddClient” method iterates through an arraylist of queues and searches for the queue with the smallest size. When it finds it, it adds the clients that needs to be added to that queue which is the moist suitable to receive a new client.

* + 1. **ConcreteStrategyTime**

This class also implements the Strategy interface. It only contains a addClient method, like the previous presented class. Here this method does the same thing as the method from the previous class, but it uses a different strategy. In this class, the addClient method iterates through the arraylist of clients and searches for the queues with the smallest waiting time. This strategy is better than the one implemented in the previous class because we can have a queue with 3 clients and the sum of service times of those clients to be smaller than of just a client from another queue. That means that a newly added client in the queue which has already 3 clients, will have to wait less to be processed than a newly added client in the queue with only one client but a greater service time.

* + 1. **Scheduler**

This class’s constructor receives the number of queues which will be in the simulation as parameter. When scheduler’s constructor is called with that parameter, it automatically creates a “n” number of queues for the simulation, along with n threads, one for every queue.

This class contains also a changeTask method, which let the developer choose which method from the 2 presented above should be used for adding the clients in queues.

The “dispatchTask” method calls the method for adding clients to the queues, according to the chosen strategy for adding clients in queues.

* + 1. **SimulationManager**

The SimulationManager class takes the input data parameters from the GUI and uses them to set up the simulation of the queues management. It takes the input as String parameters in the constructor. Also this class has a method which generates N random clients along with their random attributes. This class implements the Runnable interface, which means it will have a “run” overridden method. This method will be run by a new thread. Here we take the clients from the waiting list at their specific arrival times and add them in the most suitable queue according to the chosen strategy. In this method, we also update the simulation output GUI at every second. In this method, we also compute the values necessary for the average waiting time, peak hour and average service time. This is the method which controls everything that happens in the simulation on the thread which controls the time passing.

* + 1. **Main Controller**

This is the class which handles the buttons pressed from the GUI. Here everything that will happen when a button is pressed is implemented. We have 2 action listeners, one for the Start button and one for the Reset button.

* 1. **View**
     1. **MainView**

It’s the class where the GUI is implemented. This GUI is used by the user to give the input data and by the application to show the in real time simulation of the queue management. For the user input the GUI contains some text fields. The output of the simulation is displayed in 2 text areas, one for the queues and one for the waiting clients.

1. **Results**

Text

Description automatically generatedText

Description automatically generatedThe results are displayed in 3 different .txt files, called log1.txt, log2.txt and log3.txt. Every log contains the simulation steps for a different set of inputs. The logs contain the status of every queue along with the waiting clients and those that are being processed in that moment, at every second of the simulation. At the end of the simulation every log contains the average waiting time, average service time and peak hour.

1. **Conclusions**

Working for this application, I’ve learned how to implement programs which use parallel programming, by working with more threads. The first step was to learn how to work with threads, because I have never worked with them before. This was a good project to learn how threads work in Java, how to synchronize them and how to avoid errors which might occur by thread desynchronization.

**Future Improvements:**

As a future improvement I would add a pause button which would pause the simulation. I would also probably try some more efficient ways to synchronize the threads. A good improvement would be to make different content panes for the input data and simulation steps. Also, I would like to make a prettier design for the GUI.

1. **Bibliography**

<https://dsrl.eu/courses/pt/>

<https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/BlockingQueue.html>

<https://stackoverflow.com>

<https://www.w3schools.com/java/java_files_create.asp>