**DOCUMENTATION**

**HOMEWORK 2**

STUDENT NAME: Sandu Bogdan-Emilian

GROUP: 30423

**TABLE OF CONTENTS**

1. Homework objective……………………………………………..
2. Problem analysis, modeling, scenarios, utilization cases........…..
3. Design……………………………………………………………
4. Implementation…………………………………………………..
5. Results……………………………………………………………
6. Conclusions………………………………………………………
7. Bibliography……………………………………………………..
8. **Homework objective**

Design and implement a simulation application aiming to analyse queuing based systems for determining and minimizing clients’ waiting time.

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

One way to minimize the waiting time is to add more servers, i.e. more queues in the system (each queue is considered as having an associated processor) but this approach increases the costs of the supplier. When a new server is added the waiting clients will be evenly distributed to all current 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 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;

- Strategy chosen for clients distribution;

Output data:

 - Average of waiting time, service time;

- Log of events;

 - Queue evolution;

 - Peak hour for the simulation interval;

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

The application simulates clients waiting to receive a service just like in the real world; they must wait in queues, each queue processing clients simultaneously. The idea is to analyse how many clients can be served in a certain simulation interval, by entering parameters in an intuitive, user-friendly, application graphical interface.

Input: the customers are generated randomly, each having its own service time and arrival time. Each client’s data is generated taking into account the minimum and maximum parameters introduced by the user. Also the number of clients to be generated is given as input by the user.

Input data given by the user:

* The maximum simulation interval;
* The number of queues to be generated;
* The number of clients to be randomly generated;
* The minimum and maximum arrival time of each client;
* The minimum and maximum service time of each client;
* Diagram

  Description automatically generatedThe selection policy for the way the clients are distributed to the queues;

**Use Case: START**

**Primary Actor: User**

**Success Scenario Steps:**

1. The user introduces the data required for the simulation and select the clients distribution policy
2. The user presses the “Start” button
3. The app checks if the input is correct
4. The application starts simulating the queues and displays live the results at each step on the screen
5. After the simulation finishes some information about it is displayed on the screen

**Alternative Sequences:**

If the input introduced by the user is not valid, an adequate message is displayed on the screen.

**Use Case: Reset Input**

**Primary Actor: User**

**Success Scenario Steps:**

1. The user presses the “Reset Input” button
2. All the input fields are reseted

**Alternative Sequences:**

If the simulation is running the “Reset Input” doesn’t stop the simulation

1. **Design**

The design was realized with OOP concepts in mind. For the simulation to work, correct input must be provided by the user.

The project contains the following packages:

* GUI: contains classes related to the graphical user interface and the controller
* Controller
* SimulationFrame
* BusinessLogic: contains classes, interfaces and enumerations related to the logic used behind the app
* ConcreteStrategyQueue
* ConcreteStrategyTime
* Scheduler
* SelectionPolicy
* SimulationManager
* Strategy
* Model: contains classes related to the data used to model the problem
* Server
* Task
* Utils: contains a class used for writing the output generated by the simulation to a file and a class used to help generate random Clients
* TaskUtil
* OutputWriter

Data Structures used:

* Lists
* BlockingQueues
* Atomic variables

In object-oriented programming development, model-view-controller (MVC) is the name of a methodology or design pattern for successfully and efficiently relating the user interface to underlying data models. The MVC pattern is widely used in program development with programming languages such as Java, C and C++.

The MVC pattern has been heralded by many developers as a useful pattern for the reuse of object code and a pattern that allows them to significantly reduce the time it takes to develop applications with user interfaces.

The model-view-controller pattern proposes three main components or objects to be used in software development:

- *Model*, 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.

- *View*, which is a collection of classes representing the elements in the user interface (all of the things the user can see and respond to on the screen, such as buttons, display boxes, and so forth)

- *Controller,* which represents the classes connecting the model and the view, and is used to communicate between classes in the model and view.

1. **Implementation**

All the classes were created with the concept of data encapsulation in mind, therefore all the classes attributes are private and getters and setters are used.

* Task: is a class used for modeling the data used in the application. It implements the Comparable Interface. It has the following attributes: ID (type int), arrivalTime(type Int), serviceType(type int) and a static field used for generating Clients IDs
* Task(int arrivalTime, int serviceTime)

Is the only constructor of this class. It generates Task with the given parameters which are randomly generated with the use of the TaskUtil class.

* decrementServiceTime()

It simply decrements the service time of each task by 1.

* Getters and Setters
* Server: is a class used for modeling the data used in the application. It represents a queue and it implements the Runnable interface. It has the following attributes: task(type BlockingQueue), waitingPerdiod(type AtomicInteger), exit(type Boolean), totalWaitingTime(type AtomicInteger), totalClients(type AtomicInteger), totalServiceTime(type AtomicIteger)
* addTask(Task newTask)

Adds a task in the tasks field, updates totalServiceTime, updates totalWaitingTime, updates totalClients and updates waitingPeriod.

* Server(int maxTasksPerServer).

The only constructor for this class, it initializes all the attributes of the class.

* displayTasks()

It displays the current state of the queue in a user friendly way. Also build a string which will be outputted in a file. It decrements by 1 the serviceTime of the Client at the front of the queue.

* run()

It sleeps the current thread for the serviceTime of the task situated at the front of the queue if it is not empty, after that it removes it. If the queue is empty is sleeps the thread for 1 sec.

* Getters and Setters
* Scheduler: is a class which contains all the Queues of the simulation and the strategy used to distribute the Clients.
* Scheduler(int maxTasksPerServer, int nrOfServers)

The only constructor of the class, it creates the servers and adds them to the servers attribute.

* changeStrategy(SelectionPolicy policy)

It changes the policy by which the Clients are distributed in the Queues

* dispatchTask(Task task)

It dispatches the Client to the right Queue based by the strategy used by the server.

* Getters and Setters
* SimulationManager: is a class used to simulate the Queue Management System. It has the following attributes given by the UI: timeLimit, maxProcessingTime, minProcessingTime, maxArrivalTime, minArrivalTime, numberOfServers, numberOfClients (all of type int), selectionPolicy. It contains the scheduler, a list of generatedTasks, output String for displaying info and the View.
* SimulationManager(int maxProcessingTime, int minProcessingTime, int maxArrivalTime, int minArrivalTime, int numberOfClients, int numberOfServers, int timeLimit, SelectionPolicy selectionPolicy, SimulationFrame view)

The only constructor of this class, it initializes its attributes and starts the Queues in separated Threads.

* generateNRandomTasks()

Generated the Clients with the help of TaskUtil class, after the Clients are generated, they are sorted in the ascending order of their arrivalTime.

* run()

It runs the SimulationManager, it dispatches the Clients to the Queues when it can, it generates the Output String, updated the GUI and sleeps the Thread it is running it for 1 second while the currentTime of the SimulationManager is less than the timeLimit and there are still Clients in the simulation.

* ConcreteStrategyQueue and ConcreteStrategyTime:

Are 2 classes which contain only 1 method addTask. They implement the Strategy Interface.

* addTask(List<Server> servers, Task t)

It adds the Client to the corresponding Queue based on the implementation.

* SimulationFrame: it is a class used for implementing the GUI with Java Swing. It contains different graphic elements. Attributes: contentPanel, inputPanel, resultPanel, nrQueues, nrClients, minArrivalTime, maxArrivalTime, minServiceTime, maxServiceTime, resultLiveLabel, resultLive, maxSimulationLabel, maxSimulation, queuesNr, clientsNr, minAT, maxAT, misST, start, resetInput, controller, strategy.
* prepareInputPanel()

This method is used to place the general bulk of components. It contains buttons for each action, each button has a representative action command and it has an action listener. Each button also has a representative name. It contains a JComboBox used to select the distribution strategy. Seven text fields are used to receive the user input, each text field also has a representative label. The components are placed in a grid layout (9 by 2) in a pleasing visual order which makes the user experience intuitive. The panel in which these components are placed is being called inputPanel.

* prepareResultPanel

This method is used to place result related components. The panel in which these components are placed is called resultPanel and it has a grid layout (2 by 1). It contains two labels, the first one indicating the panel role and the second the current step of the simulation.

* TaskUtil

Is an utility class used to help generate random Clients for the queues.

* getRandomNumber(int min, int max)

Is a static method used to generate a random number in the given interval.

* OutputWriter

Is an utility class used to write a List of String to the output file output.txt.

* writeOutput

Is a static method used to write the output to the file.

Diagram

Description automatically generatedGraphical user interface

Description automatically generated with medium confidence

1. **Results**

The results of the simulation are found in multiple places, one place is the terminal which displays each stept after the “Start” button is pressed. Also the visual interface displays ONLY the current step of the simulation in order not to overcrowd it. The last place where the results could be seen is in the output.txt file which contains formatted nicely all the steps that the user can see during the simulation.

1. **Conclusions**

Making the threads to be synchronized correctly was the most difficult part of the project. I had no prior experience working with threads and debugging such an application proved to be a difficult task at first but I got used to it. Other problems that I encountered was while I was working on the GUI. In order to have a responsive GUI I had to make use of the SwingWorker class which I had some problems with it but I solved them in the end.

The GUI could be improved a lot and be made more appealing to the eye. Also, another button “Stop Simulation” could be implemented in order to stop the simulation while running and resume it while needed. Also the way the output is displayed could be improved such that each step could be seen clearly and in the same time to not overcrowd the GUI.

Main aspects learned during the making of this project: how to work with multi-threaded applications and how to debug them, a better understanding of the MVC model, gained experience working with Java Swing and Maven.

1. **Bibliography**

* <https://docs.oracle.com/javase/tutorial/uiswing/>
* <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/uml-class-diagram-tutorial/>
* https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/BlockingQueue.html#take()
* <https://docs.oracle.com/javase/9/docs/api/javafx/concurrent/Worker.html>
* https://www.tutorialspoint.com/java/java\_multithreading.htm
* Lots of stackoverflow threads