DOCUMENTATION

ASSIGNMENT *2*

STUDENT NAME: BOTA ALIN ALEXANDRU

GROUP: 30423

# CONTENTS

[1. Assignment Objective 2](#_Toc128043139)

[2. Problem Analysis, Modeling, Scenarios, Use Cases 2](#_Toc128043140)

[3. Design 4](#_Toc128043141)

[4. Implementation 6](#_Toc128043142)

[5. Results 12](#_Toc128043143)

[6. Conclusions 13](#_Toc128043144)

[7. Bibliography 13](#_Toc128043145)

# Assignment Objective

To create a simulation-based queues management application that assigns clients to queues in a way that minimizes the waiting time. The application should simulate the arrival of N clients, their entry into Q queues, waiting, being served, and leaving the queues. The waiting time for each client should be tracked, and the average waiting time should be computed. The user should be able to input the number of clients, the number of queues, simulation interval, and minimum/maximum arrival and service times. The application should assign each client to the queue with the minimum waiting time when the client's arrival time is greater than or equal to the simulation time.

For simplicity we have this example:

Graphical user interface, text, application

Description automatically generated

# Problem Analysis, Modeling, Scenarios, Use Cases

By problem analysis we refer to the process of identifying, defining, and understanding a problem in order to find a solution or a set of solutions. We clearly have an advantage because we’re using an object oriented language. It allows us to work on a superior, conceptual level, without being constrained by the technical work.

This strategy is also known by the bottom-up design. It has a big advantage because we can easily find structures that are connected in a way with the real world(objects). Clearly this concept doesn’t have only advantages, it also comes with a high level of complexity.

To solve this problem, the application needs to have a simulation time (timestep) and track the arrival time, service time, and waiting time for each client. The application should also keep track of the number of queues and the number of clients in each queue. The user should be able to input the number of clients, the number of queues, simulation interval, and minimum/maximum arrival and service times.

To assign clients to queues, the application should use a scheduling algorithm that takes into account the arrival time and service time of each client. One approach could be to assign each client to the queue with the shortest queue length, which can be computed by taking each available queue/server and get the number of clients that are served in the respectively moment.

The application should also compute the average waiting time for all clients and display it to the user at the end of the simulation. This can be calculated by summing up the waiting time for each client and dividing by the total number of clients.

**Use cases :**

-Service Provider: The application can be used by a service provider to manage customer queues and minimize waiting times.

-Event Management: The application can be used by event organizers to manage queues and optimize waiting times for attendees. Some examples : ticket lines, shop lines etc.

# Design

Dependencies, design, structure, variables and methods of classes :

Diagram

Description automatically generated with medium confidence

A picture containing graphical user interface

Description automatically generated

The ViewScreen Class -> Is responsible for the UI provided in the application. It has a listener to a button such that when the user presses the button, the simulation starts. Also provides seven textLabels that memorize the inputs that the application should work with.

The Controller Class -> Is in a strong relationship with all the other Classes provided in this program. This class is the “manager” of the application.

The Client Class -> Is simply a class that provides information about an individual client, such as : id, the time when arrives to the queue and the time needed to satisfy this client.

The ClientQueue Class -> Is a class which is responsible for multithreading the queues. Similar to the client class, provides information that the queue needs and also provides a method that allows the class to be accessed multiple times in parallel.  
 The Scheduler Class -> The scheduler class is in a strong relationship with the Client Queue class. It creates the number of servers/ queues that are provided by the input from the Controller Class. Also has a method that adds the client in one of the queues that has the minimum number of clients.

# Implementation

**View Screen**

****

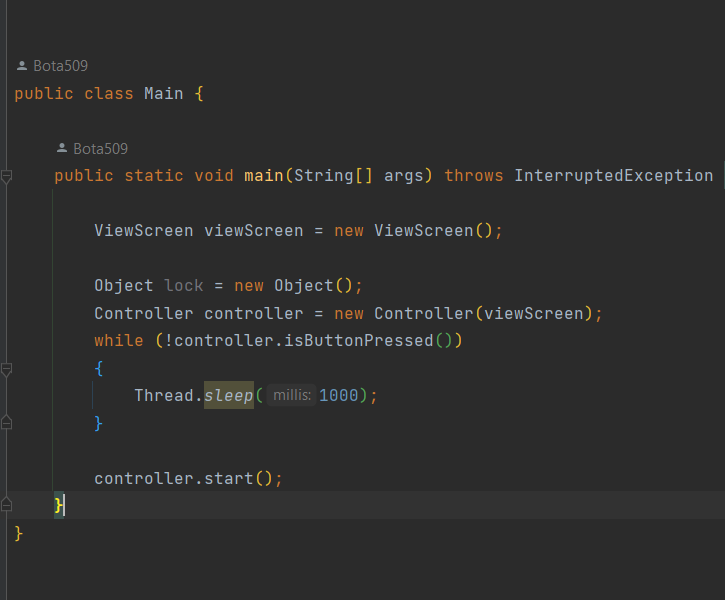
View Screen is simply a class that is responsible for the interface (UI) of the application. All the logic for the interface is put in the constructor of the class.

Text

Description automatically generated

It also provides a clear method, which is set when the button was pressed. It clears all the inputs provided on the UI. It also has a submit Listener method which will give a listener to the controller class, when the button was pressed.

**Main Class**

****

For the application to start it needs in the Main method a reference to the view Screen Class and the Controller Class(which is the manager of the application). A View Screen object is initialized which results with the interface shown on the display when the app is runed. An object of type controller is then initialized, and when the inputs are provided on the interface, by the user, and the button was pressed, a Boolean variable from the Controller Class is set to true -> the main thread off the simulation is activated.

**Client Class**

**Text

Description automatically generated**

The Client Class is responsible for the information of a specific client. It memorizes an id for each different client, a time Arrival variable which tells when the client should enter one of the available queues and a time Service variable which specifies how much time should be spend for a client to be served.

**ClientQueue Class**

This class has a similarly responsibility ass the Client Class. It provides the necessary information for each specific queue.

Text

Description automatically generated

It provides a time Arrival, a time Service, a total Time and an id. We use Atomic Integer instead of int for that multiple threads can safely access and modify the value of an integer variable without the risk of race conditions or data inconsistency. The id is used only for display purposes to specify which queue is displayed.

Text

Description automatically generated

The class is extended from a Thread Class which will allow us to do multithreading (meaning that each queue is a different thread). While this thread is alive and the queue has some clients , the thread will not be accessed until the current client was served and leaved. Then it eliminates the client from the queue.

**The Scheduler Class**

Text

Description automatically generated

The Scheduler Class provides a List of queues. It creates the number of queues/ threads that was specified when the input was provided on the interface and start all those threads.

Text

Description automatically generated

It provides an addInServiceQueue method which has a parameter of type Client. The method looks for the queue/thread that has the minimum number of clients and add in the specific queue the specific client. This is meant to minimize the process.

**Controller Class**

**Text

Description automatically generated**

When the constructor of the Class is called a writer variable of type File Writer is initialized, such that the output of the application will be displayed on a specific text file.

Text

Description automatically generated

The inner class SubmitListener is running when the button was pressed.

Text

Description automatically generated

When read method is called, each input that was specified on the interface is saved on a global variable to be easier to work with them.

Text

Description automatically generated

This method is used to generate a random list of clients with their arrival and service time. The method takes no input parameters and uses instance variables to set the minimum and maximum values for the arrival time and service time of the clients.

The method first creates a new Random object to generate random numbers. It then loops through numberOfClients times to create the specified number of clients.

Inside the loop, the method checks if the maximum arrival time is greater than the minimum arrival time and if the maximum service time is greater than the minimum service time. If this condition is true, the method generates random arrival and service times for the client using the nextInt() method of the Random class, which generates random integers within the specified range.

The method then creates a new Client object with the generated arrival and service times and adds it to the waitingClients list. Finally, the method adds the generated service time to the totalTimeService variable.

If the condition is false and the maximum and minimum values are not set correctly, the method prints an error message.

Text

Description automatically generated

This method is used to calculate the average waiting time for clients across all queues in the scheduler. It takes no input parameters and returns an integer value representing the average waiting time.

The method first initializes a variable waitTime to zero, which will be used to store the total waiting time across all queues. It then loops through each ClientQueue object in the scheduler's list of queues.

Inside the loop, the method adds the totalTime attribute of each queue (which represents the total waiting time for all clients in the queue) to the waitTime variable using the intValue() method to convert the totalTime attribute to an integer value.

After looping through all the queues, the method returns the average waiting time by dividing the waitTime variable by the numberOfQueues attribute of the scheduler, which represents the total number of queues.

Text

Description automatically generated

This method is used to write a string to a file and to the console. It takes a string as input parameter and does not return anything.

The method uses a writer object to write the input string to a file. The writer object is assumed to be initialized before this method is called, most likely by opening a file using a FileWriter or similar class.

Text

Description automatically generated

The simulation of the application ends when the real time of the simulation is smaller or equal than the maximum simulation time provided from the input and when there are no more clients to be served or waiting.

Firstly we iterate trough the list of clients that are waiting. When the time Arrival of a client is smaller or equal than the simulation time, the client is added in the queue to be served and removed from the waiting list.

Text

Description automatically generated

After the clients that needed to be added were added and then removed from the waiting list, we display the remaining waiting clients. We take a Boolean variable that is set to false when there are no more waiting clients. We calculate then the peak hour, by updating an variable with the maximum number of clients in a queue.

Text

Description automatically generated

We iterate trough the queues. If there are no more queues we set a Boolean variable to false. Also if there are no more clients waiting and all the queues are empty the simulation will stop.

Text

Description automatically generated

For the end of the simulation we display the average waiting of a client to enter a queue to be served, the peak hour and the average service time.

# Results

The results given for the provided input information :

Text, letter

Description automatically generated

The log below is shown :

Text

Description automatically generated

The rest of the results are put in the repository on the three text files: test1.txt, test2.txt, test3.txt;

# Conclusions

In conclusion the provided specification describes a queue management application that simulates a system where clients are assigned to queues in order to minimize the waiting time. The application should track the time spent by each client in the queues and compute the average waiting time. The user interface should allow the input of the number of clients, the number of queues, the simulation interval, the minimum and maximum arrival time, and the minimum and maximum service time.

A possible extension could be to allow for dynamic queue creation and deletion based on the current load of the system. Additionally, the application could be extended to include a visualization component to display the queues and clients as they move through the system.

# Bibliography

<https://www.youtube.com/watch?v=r_MbozD32eo>

<https://www.youtube.com/watch?v=jiWNk9S46j4>

https://stackoverflow.com/