

**Fundamental Programming Techniques**

**Assignment 2: Queues Simulator**

**Student:** Diana-Maria Jurcan

**Group:** 30422

**Laboratory professor:** Marcel Antal

**Course professor:** Cristina Bianca Pop

# CUPRINS

1. Assignment objective……………………………………….3
2. Problem analysis, modelling, scenarios, usage cases……….3
3. Design……………………………………………………….4
4. Implementation………………………………………………7
5. Results……………………………………………....……...10
6. Conclusions………………………………………………...13
7. Bibliography……………………………………………......13
8. **Assignment objective**

The objective of the current laboratory topic was to propose, design and implement a queues management application which assigns clients to queues such that the waiting time is minimized. Queues are commonly used to model real world domains. 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 their "clients" are waiting in queues before they are served. 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 service supplier.

In other words, it is a simulator that generates, depending on the input data, a simulation in which the states of the available queues are presented, in every second of the simulation.

1. **Analysis**

The queues management application should simulate (by defining a simulation time) a series of N clients arriving for service, entering Q queues, waiting, being served and finally leaving the queues. All clients are generated when the simulation is started, and are characterized by three parameters:

* ID (a number between 1 and N),
* time of arrival (simulation time when they are ready to enter the queue)
* time of service (time interval or duration needed to serve the client, i.e., waiting time when the client is in front of the queue).

The application tracks the total time spent by every client in the queues and computes the average waiting time. Each client is added to the queue with the minimum waiting time when its time of arrival is greater than or equal to the simulation time.

The following data should be considered as input data for the application that should be inserted by the user in the application’s user interface:

- Number of clients (N);

- Number of queues (Q);

- Simulation interval (𝑡𝑠𝑖𝑚𝑢𝑙𝑎𝑡𝑖𝑜𝑛 𝑀𝐴𝑋);

- Minimum and maximum arrival time (𝑡𝑎𝑟𝑟𝑖𝑣𝑎𝑙 𝑀𝐼𝑁 ≤ 𝑡𝑎𝑟𝑟𝑖𝑣𝑎𝑙 ≤ 𝑡𝑎𝑟𝑟𝑖𝑣𝑎𝑙 𝑀𝐴𝑋);

- Minimum and maximum service time (𝑡𝑠𝑒𝑟𝑣𝑖𝑐𝑒 𝑀𝐼𝑁 ≤ 𝑡𝑠𝑒𝑟𝑣𝑖𝑐𝑒 ≤ 𝑡𝑠𝑒𝑟𝑣𝑖𝑐𝑒 𝑀𝐴𝑋).

**Use cases**

The queue simulator, which consists of simulating a series of N customers who arrive at Q queues for a certain "service", are served and finally leave the queues. All clients are randomly generated when the simulation has started. Clients are characterized by 3 parameters: id (a number from 1 to N), arrival time (simulation time when customers are ready to sit in front of queues) and service time (time the customer has to wait to be served by the cashier).

The purpose of this application is to distribute customers optimally, depending on the waiting time of each queue.

1. **Design**

* **Use case diagram**

The requirements are descriptions of the services provided by the system and the constraints that are generated during the requirements engineering process. The specific functional requirements for this issue are as follows: Processing tasks received from clients, arranging clients in queues as efficiently as possible, calculating the average queue time, using threads as server simulators, Closing and dynamically opening servers.

Diagram

Description automatically generated

* **UML Diagram**

Unified Modeling Language or UML for short is a standard language for describing models and software specifications. UML was originally developed to represent the complexity of object-oriented programs, the foundation of which is the structuring of programs into classes, and their instances (also called objects). However, due to its efficiency and clarity in the representation of abstract elements, UML is used beyond the IT domain. On the next page you can see the UML diagrams generated with IntelliJ.

Graphical user interface

Description automatically generated

Graphical user interface

Description automatically generated

In the UML diagram above are presented all the classes, with the attributes and the relations between them, the main classes are:

* Client
* Queue
* Scheduler
* RandomClientGenerator
* FileActions
* Main

1. **Implementation**

**Data structures**

The most important data structures that have been used are the following:

- Array: Client[] or Queue[].

- ArrayList <Client>.

**Classes and methods:**

In the following, all the important classes and methods will be briefly presented:

* **Class Client:**

- Abstracts the customer model, encapsulating information such as: clientID, arrival time and service time;

- The only more important method would be the overwritten method toString () which returns a form string (ID, arrivalTime, serviceTime)

- Here, other methods that help within the Scheduler class, will be:

- decrement service time, which is self explanatory

- increment waiting time

- get queue status which we will use for determining if a client belongs to a queue. If it is set to false, the client does not belong to any queue.

* **Class Random Client Generator:**

- It comes with the help of the simulation, generating the specified number of clients, with random values, bordered by some limits required in the input file. The algorithm is made using the generateClients() method. In this method, an object of type Random will be instantiated, and the random Arrival time and random Service time will be assigned a random value between the ranges of [minimum arrival time, maximum arrival time] and [minimum service time, maximum service time].

- In this class, it will also be computed the total Service time and total Waiting time, which we will need in class Scheduler for computing the average waiting time and average service time. They will be set to static variables.

* **Class Queue:**

- Abstracts the model of a queue.

- Two important methods are: run() and serveClient() and addClient(). The main execution is run ():

The steps followed in the run method:

* If the current thread is still running, wait a second and call serveClient method, else print that the thread has been interrupted.
* serveClient method will call wait() if there are no clients in the queue, which will make the thread “go to sleep”. If service time of the first client is different from 0, the service time will be decremented, as well as the waiting time. Otherwise, we remove the client from the queue and set its status to false and decrement one last time the waiting time. If it was the last thread, end thread and notify all threads that are waiting.
* The method addClient will be synchronized, just like serve Client, and it will set the status of a client to True, meaning the client belongs to the queue, and the waiting time will get the service time. Finally, the notifyAll() method will be called again, to notify all threads that are waiting.
* Status() method returns “closed” when the queue is empty and it returns the list of clients in a string otherwise.
* **Class Scheduler:**
* Contains method run(), which is the most important for the whole functioning of our program and the methods emptyQueues() which verifies if a queue is empty and method indexMinimumQueue() which will return the index of the queue with the minimum waiting time.
* Method run(): will start the thread and while the variable time, which will increase by each passing through, is smaller than the maxSimulation will do the following:
  + - The boolean variable areThereAnyClientsLeft will be set to false. And wewill write the current time and the current list of waiting clients.
    - For the current list of waiting clients, we go through every client and check if the arrival time is bigger than the time the boolean variable are There Any Clients Left will be set to true.
    - Otherwise, if the arrival time is equal to the current time, we get the index of the queue with the minimum waiting time and check if a thread didn’t already start, and start a thread for a queue only if there are clients in it and we add the current client in the queue.
    - If the arrival time is smaller than the current time and the queue status if true(the clients already belongs to a queue), we increment the waiting time of the client.
    - We move on to the list of queues. For every queue, we print in the file the number of the queue and the content of the queue. Also, we compute the peak time. For this we have to compute the number of clients per queue. We will also get a variable maxClients which we will verify if it is smaller than the number of clients per queue, and put the biggest number of clients per queue in that variable. The peak hour will be the current time for that queue with the maximum number of clients.
    - Wait a second.
    - Print the current time.
    - Increase the current time.
    - If the are no more clients, and all the queues are empty or the current time is equal to the simulation time, then we print the average waiting time, which we compute by getting the total waiting time which was calculated in the randomClientGenerator class and dividing it by the total number of clients. Moreover, the average service time will be computed the same and the peak time will be the peak Hour static variable calculated earlier.
    - After printing everything, we interrupt the thread and we print that the thread has been interrupted and we close the file and exit the application.
* **Class FileActions:**

Contains methods that help us in reading from file and writing in it.

* readFromFile() method will read every line from the file and split the line in strings if they have a comma between them. Then we assign to every variable that we need for our given task, the values that we read from the file. We instantiate a new queue for all queues and generate random clients for the list that we have.
* printInFile() method will create a new PrintWriter, without automatic line flushing, with the specified file (the output file). The file writer will be transmitted to scheduler class which will write the specific outputs in the outputs file every step of the way.

**Graphical User Interface**

Unfortunately, the current project does not have a graphical user interface, the execution being done using user input and files that are located in the folders of the IntelliJ project.

1. **Results**

The program was tested using the input files specified in the topic content: in-test-1 .txt, in-test-2 .txt, in-test-3 .txt, and the result will be saved in the files: out-test-1.txt, out-test-2.txt, out-test-3 .txt.

The files in question are available in the directory resources in the hierarchy of project directors. In order to verify the obtained results, both the output files and the input files can be verified, these having the data specified in the topic description.

For the first case: we will have the following data saved in file in-test-1.txt: **A picture containing text

Description automatically generated**

The output will be present in out-test-1.txt:

Text

Description automatically generated

With the computation of the average waiting time, average service time and peak hour.

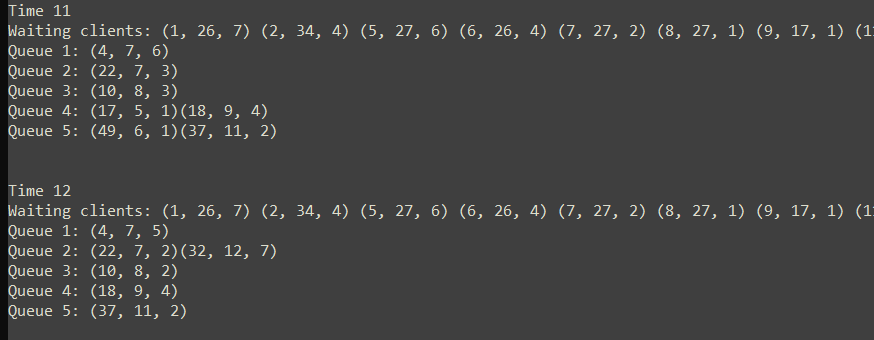
Text

Description automatically generated

For the next cases, we can see the results below, accompanied by the final computations as well.

Text

Description automatically generated with medium confidenceFor case II:



Graphical user interface, application

Description automatically generated with medium confidenceText

Description automatically generated

For case II:

Text

Description automatically generated

1. **Conclusions**

The following skills were learned in this topic: working with Threads, using data structures in order to execute the program under the aspect of "safe thread". All project objectives have been achieved. This topic strengthened my knowledge of object-oriented programming, as the implementation of the topic required structuring classes and creating objects: Queue, Scheduler.

I learned to use threads and simulate a real-time application with them. As a possibility for further development, an interface that resembles the tail of a supermarket could be implemented. Buttons, text fields and other interface related features could also be implemented easily and this would also make this project respect the model controller view architecture more clearly.

1. **Bibliography**

* <https://stackoverflow.com/>
* <https://docs.oracle.com/javase/tutorial/uiswing/>
* <https://www.vogella.com/tutorials/JUnit/article.html>
* <https://www.baeldung.com/junit-5>
* <https://www.geeksforgeeks.org/>
* <https://en.wikipedia.org/wiki/Use_case>
* <https://google.github.io/styleguide/javaguide.html>
* <https://www.lucidchart.com/>
* <https://docs.oracle.com/javase/tutorial/essential/concurrency/index.html>
* <https://www.tutorialspoint.com/java/util/timer_schedule_period.htm>
* <https://www.javacodegeeks.com/2013/01/java-thread-pool-example-using-executors-and-threadpoolexecutor.html>
* Courses, laboratories, homework, and OOP project from year 2, semester 1.