Tehnical University of Cluj-Napoca



Facultatea de Automatică și Calculatoare

Assignment 2

Queues Simulator

Student: Loga Darius

Group: 30422

Section: CTI-E

TA: Diana Bâlc

**Table of Contents**

1. Aim of the assignment

2. Problem analysis

2.1 Use cases and scenarios

3. Design

4. Implementation

5. Results

6. Conclusions

7. Bibliography

**1. Aim of the assignment**

The main objective of this assignment is to design and implement an application that can simulate queues based system for determing and minimizing clients’ waiting time. The application should simulate, by defining a simulation time, a series of N amount of clients arriving for service, entering Q amount of 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 go to the queue) and time of service (time interval or duration needed to serve the client by the cashier). The application tracks the total time spend by every customer in the queues and computes the average waiting time.

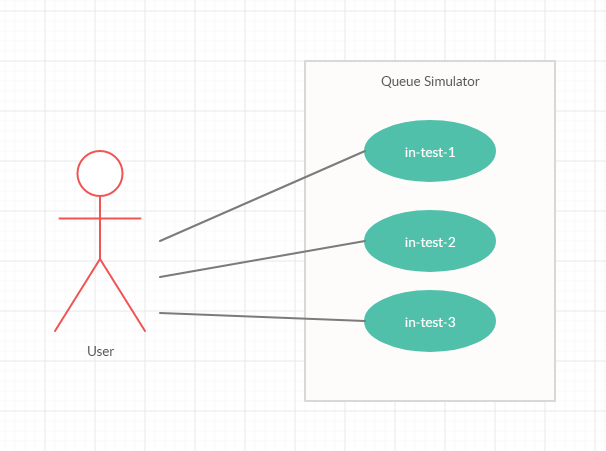
As for the secondary objectives that should be implemented to met the requirements for the main objective are the following: design and create the use case diagram with scenarios and also the UML class diagram, implement as many classes as you want to complete the project, but also include classes for reading the input and writing the output of the requirements met in the initial specifications, choose a strategy for inserting the clients in the desire queue and create a executable jar file to run the application using as parameters the input and output txt file.

**2. Problem analysis**

Just like in the real word, out there exists this kind of system in which one client arrives at a certain point and need to wait until in can be served at a desk or something like this. If more people or clients are coming to the waiting line, the waiting time increases if just one queue for that desk is open. For multiple desks, those queue number increases and the wainting time is reduced for the clients, but the service time remains the same for each people. In our requirements, the arrival time is given, just like you have a ticket and only when the time of the simulation appears, like displaying the number of the ticket on a display, you can go to a specific queue based on a strategy that we will have to implement. For implementing this application I used a series of specific concepts for object-oriented programming: abstraction, encapsulation, multiple threads, inheritance etc.

We can make the assumption that the given input for all three test, just the first one is compulsory, others are voluntary, are correct, no letters or other characters that do not belog in the parsing part, and well introduced in the form that on the first line is the number of clients, second one contains the number of queues, the third contains the maxium time allocated for running the simulation (in seconds) and last two lines are intervals, separating the minimum and maximum value using comma, one for the arrival time of each client and the last one for the service time of each of them. If the input is correct then the computating part and the output, written in the text files, are also going to be appropriate for the application and well disclosed.

**2.1 Use cases and scenarios**

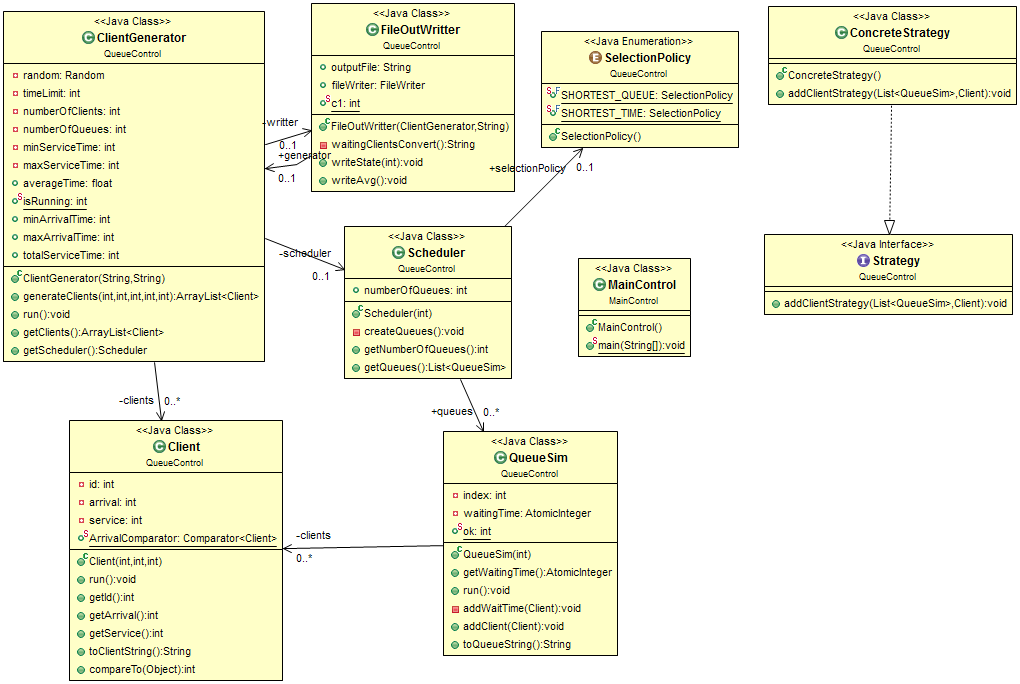
****

Upper we have the use case diagram for this application and the only actor that I have chosen is the user one and I will focus only on the first case, for the in-test-1, because the other ones are similar, but on a larger scale.

In the first case, the user can input the data into the in-test-1 text file, with the assumption that I have just made, introduce as parameters for the executable jar file, that I mentiond in one secondary objective, this input file and the seeked output file, out-test-1, let the application parse the input and do the implementation and in the output file, just writes the computed data in the formate specified in the assignment’s documentation. The alternatives cases, independent from the assumption, is that we neighter introduce the input or output text file, so the program will throw the IOException, terminate and we need to start again by introducing those files.

**3.** **Design**

Below is the UML class diagram from my application:



For this structure of implementing the classes, it was provided to us a guide pdf for better understading the implementation of the application. I divide my program into two packages: **QueueControl,** that includes classes similar with the one in that pdf file, that is listed in section 7 with the **Bibliography.** Ican mention these classes:Scheduler (**Scheduler**), Server (**QueueSim**) and Simulation Manager (**ClientGenerator**). I also wanted to include a strategy for the clients to be added in the queues, so I implemented the **ConcreteStrategy** class and one class for writing the output into a text file: **FileOutWritter.** For the **MainControl** package, I only put one class, **MainControl**, to start the application and to make use of the exec jar file.

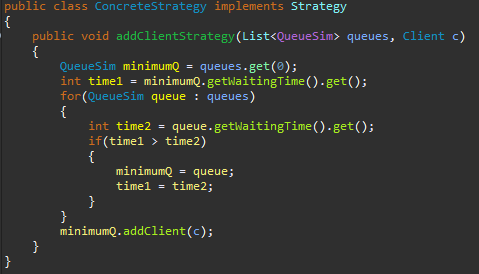
**4.** **Implementation**

I will start with the **QueueControl** package. The first class to be described is the **Client** class that has three parameters: ID, arrival and service that are tuples for defining one client and one static comparator used later for sorting the clients based on their arrival time in the waiting queue. The run() method, in which Client class implements the Runnable interface, is just made to sleep (wait) the client, for the entire duration of each simulation, with the value of their service time in the queue and the toClientString() method is just to return the client in a nicer format by a String type with the tuple **(id, arrival, service)**. The last method, the compareTo() is auto-generated and I will not use this one.

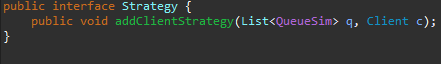
For the **QueueSim** class I implemented methods to get the wait time of each client that belongs to the **BlockingQueue** type and added into the **AtomicInteger** waitingTime and also to add clients to that queue with their waitingTime. The run() method, this class also implements the Runnable interface, identifies if the clients are added correctly to the BlockingQueue, and random, of course and if there are clients, the thread for the queue is sleeping (waiting) with the amount of time from the service time of each client after I applied the peek() method to get the head of the queue and after the sleep of the clients, the AtomicInteger decreases with the amount of each clients’ service time and remove the clients from the queue. If the AtomicInteger is 0, then the queue is empty, so therefore, no clients are there and the thread is paused. The last method is returning a string with the queues’ number and if they are closed, it display the messege “closed” and if not, the clients are shown.

The **SelectionPolicy** enum class that represents 2 **constants: SHORTEST\_QUEUE, SHORTEST\_TIME. I used the model from the provided pdf file to design this class. It only presents the strategies that will be implemented for the insertion of the clients in the queues. Below is a screenshot to represent this:**C:\Users\Darius\Desktop\selection.png

**The ConcreteStrategy class is just a class that has only one method to perfectly place the clients in the queue that has the smallest waiting time among all of them. This is the strategy that I have chosen to approach during my application. It also implements the custom interface that I will show below. Underneath, I will insert a screenshot with the method of this class:**



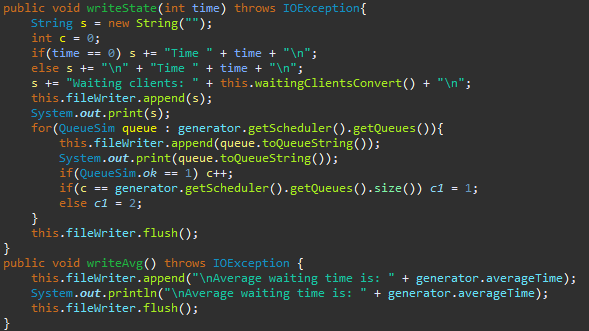
The **Strategy** interface could act as an abstract class, because it has only one method and the body is empty.



The **Scheduler** class is implemented to send tasks to the queues according to the desired strategy. It has an constructor that can create an ArrayList of **QueueSim**, it sets the selectionPolicy to the **SHORTEST\_TIME, it saves the integer values, from a variable into the instanciated variable of the classs, for the total number of the queues and calls the createQueues() method that basically creates a individual thread for each QueueSim from that ArrayList. One method was created to return the list of the QueueSim, but only to be read-only.**

One of the most important class is this package is the **ClientGenerator** one, because here is the actual generating of the simulation of this application. The getClients() and getScheduler() methods are done just as getters for those instance variables of the class. The constructor has two parameters used for the input and output file of this simulation. If one of these is missing, an exception is generated. Here the parsing of the input takes place and save the integers from that file into the variables created in the class and also save in other variables the Random(), Scheduler() and the FileOutWriter() constructors and the generateClients() method, that will explain it in a bit. The generateClients() method has parameter for the number of clients and minimum/maximum arrival and service time. In the body of this method we create an ArrayList to store the random generated clients. For the numberOfClients we compute, random, using the random class variable the arrival and service time and store each random service time into a class variable just to later compute the average waiting time that is: (the total service time of each client) / (number of clients) as a float and also, using the Collection class, to sort the clients by their arrival time. The ID of the client is not random generated. And the last method is the run() in which we apply the dispatch variable of type ConcreteStrategy() to use to insert clients in the queue with the smallest waiting time of all clients in that queue and for each time that is below the specified maximum time for the simulation, we create an ArrayList to store the clients that needs to be deleted from the waiting line and increase the time of the simulation. We can write, using the FileOutWriter variable only at the end of the iterations of the while loop and sleep the thread by 1000 miliseconds. If the queues are empty or not and in the waiting line are no more clients, the simulations should stop and write everything into the output file. The computed average waiting time would appear only if there are no clients in the waiting line and all the queues are closed.

The second most important class is the **FileOutWriter** class that has a constructor that requires to have the path to write the output to. If not, the IOException is thrown. It has also a method in which will return a string with all the clients generated using the toClientString() method to have a nicer output. The writeState() and writeAvg() methods are used to append the lines with the time, waiting clients and queues with clients or not to the output file to have a desired format, but I also made some printing to display in the console of the Eclipse and for the writeAvg(), only at the end of the simulation, to display the average waiting time for the whole simulation rather than for every time of the simulation. Below is a screenshot related to the format of the output:



**5. Results**

All of the results can be seen in the specific output files: out-test-1, out-test-2, out-test-3 and can be tested using the executable jar file with the command in any CMD:

java –jar (\*name of the jar\*) in-test-(\*1/2/3\*) out-test-(\*1/2/3\*)

If the input exists, the simulation has done its desired scope and no need for the IOExcepion to be thrown. What I would like to introduce to this application, as a further development is to add queues not by defining the number from the beginning, but the simulation to know if this amount of queues are enough for this number of clients, not to take to much time nor to consume a lot of resources and I would like to introduce a Graphical User Interface (GUI) for the input.

**6. Conclusions**

To sum everything up, the presented application is useful for simulating a real-life situation that can be found in any place that relates to a queue mechanism. My personal opinion is that this topic helped me to familiarize with a series of typical notions of object-oriented programming. I have acquired a lot of useful information, which will help me in solving the following problems / projects during this semester. At the same time, by applying the concepts of OOP, known so far at a theoretical level, I believe that I have improved my skills and now I can understand better and be more efficient when working with anything related to Java programming language.

**7. Bibliography**

**Personal resources**

<https://creately.com/diagram-type/use-case>

<https://javahungry.blogspot.com/2017/11/java-arraylist-of-object-sort-example-comparable-comparator.html>

<http://coned.utcluj.ro/~salomie/PT_Lic/4_Lab/Assignment_2/Assignment_2_rev.pdf>

<http://coned.utcluj.ro/~salomie/PT_Lic/4_Lab/Assignment_2/Java_Concurrency.pdf>

<https://docs.oracle.com/javase/tutorial/essential/concurrency/index.html>

<http://coned.utcluj.ro/~salomie/PT_Lic/5_Courses/>

<https://www.tutorialspoint.com/java/util/timer_schedule_period.htm>

<https://www.javatpoint.com/java-filewriter-class>