

FACULTY OF AUTOMATION AND COMPUTER SCIENCE

FUNDAMENTAL PROGRAMMING TECHNIQUES

ASSIGNMENT 2: QUEUES SIMULATOR

Coordinator: Dr. Eng. Cristina Bianca Pop

Student: Bianca-Veronica Avram

2nd year, 2nd semester

2020

Contents

1. *Objectives*
2. *Problem analysis, modelling, scenarios, use case*
3. *Design*
4. *Implementation*
5. *Results*
6. *Conclusions*
7. *Bibliography*

*1.Objectives*

The requirement for this assignment resumes to implementing a simulation for a queue manager that deals with a set of clients that arrive at random moments of time and have different time periods for processing.

The main objective of this task is to approximate the time a client must wait in a queue, including the time spent in ordered to be served. This problem has its practical applications in what we informally call supermarkets and hypermarkets which are constantly trying to enhance their clients’ experience, so they came up with many options.

The simulation is provided a simulation time that represents the time in seconds during which the application should run.

The flow of the program is the following: first, a number of clients is generated, each having an identification number, arriving time and processing time. Also, a number of queues is generated and then the proper work begins.

The simulation is progressive and, at each step, clients that have the arriving time equal to the current time are sent to a queue that has the shortest waiting time. Then, each client is waiting in the queue until he is processed.

The main objective of the assignments is calculating the average waiting time for all the clients that have been processed in that simulation. This is done by adding all the waiting times from all the clients that have been successfully processed in the time provided and then dividing this number by the total number of clients.

As secondary objectives we have:

1. Identifying the classes that need to be implemented
   * This point is developed in the 3rd section
2. Creating the structure of the classes
   * Also in section 3
3. Dealing with the concurrent programming

* This is done by using java threads and special data structures that ensure thread safety
* More details will be presented in section 4

1. Merging the classes and providing functionality
   * This is done by using aggregation into the class that contains also the main method used to run the application (SimulationManager)
   * Also section 4
2. Managing the simulation as a whole and using command line parameters
   * Here the end of implementation is included
   * It can be found in section 4

*2.Problem analysis, modelling, scenarios, use case*

This problem does not provide such a rich utility for a regular user. It is to talk about usage, the application could be used by supermarkets supervisors or managers in order to track the flow of serving inside their stores and then coming up with efficient solutions in case some are needed. Here it is presented the use case diagram.



Remark: Developer is actually System

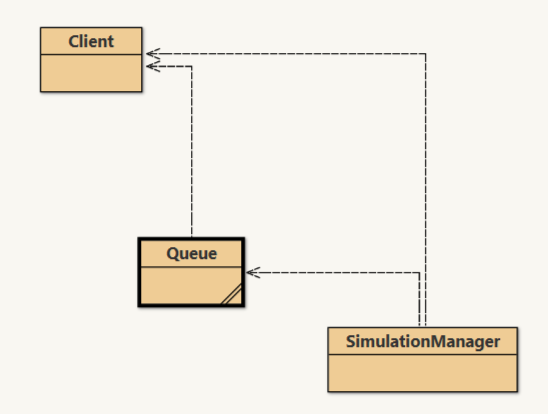
The supervisor is the person who wants to monitorize the clients flow in the store and provides some input data in order to simulate a possible scenario that can occur at the queues. So, a text file is written by the user and there he specifies the number of clients, the number of queues, the simulation time, which is explicitly the period of time in which the application will run, measured in seconds, an interval that contains the times when a client can come to the store and another interval that gives the possible values for the parameter that specifies how long it takes for a client to be processed, after he/she got the first in the queue.

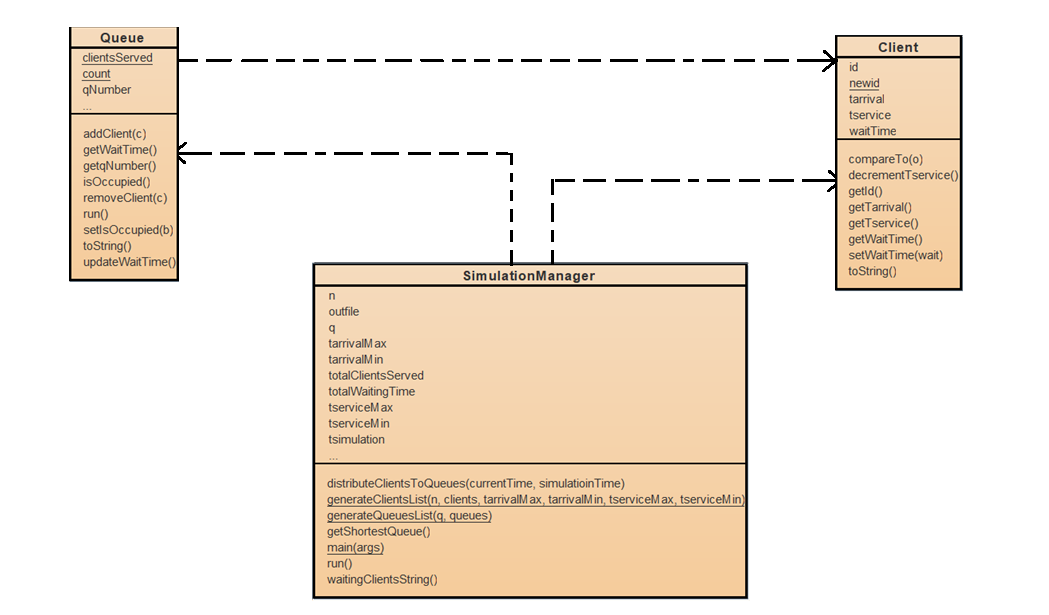
The application can be run from command line, using the .jar file and it needs two parameters. The first one is the input file and the second one is the output file, where the user can observe the results of the simulation. Once the application is run, the input file is processed and the data from there is transferred as fields to some objects of the created classes, so that the actual simulation can start.

The simulation goes on gradually, second by second, and each time there are some clients having the arrival time equal to the current simulation time, he is distributed to a queue. This queue is chosen such that the waiting time for this one client to be the shortest possible. Sometimes there is an empty queue where he can be placed, which is obviously the best case possible, but sometimes, there are other clients waiting in the queues, do the most convenient one must be chosen. The queues as thread objects process their clients, meaning that at each step of the simulation, the waiting time of all clients is decremented and the required service time for the first client in the queue is also decremented. The output will be a visual representation under string format of how the queues look in each moment of time, plus the full list of the generated clients for which the arrival time has not come yet.

For the termination of the simulation there are two possible scenarios. The first one is when all clients have been completely processed (that means that the last one has just been set service time to 0) and the simulation time is still on. Now the simulation has no point to continue, so it is stopped, as the result of the requirement can be generated with the data acquired so far. The second time is when the simulation ends naturally. That is when the simulation time expires. At this point, the clients that are still in queue are not taken into consideration for the result, as they did not completed their waiting time. The average waiting time is computed using only the clients who have been fully processed and eliminated from the queue by the time the simulation ended.

*3.Design*





This problem is not actually very complicated in terms of design. There are only three classes that can do the whole job. From the problem specification we can identify the client, who is the main actor of this problem, the queue, which is responsible with processing clients one by one and one more class that unifies these two and also fulfills the role of simulator for the given simulation time. Between these three there are only aggregation relationships.

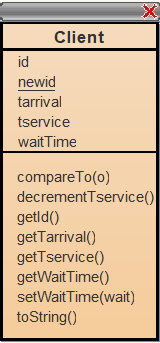
The SimulationManager class is the executor of this application. It processes the input data, transforms it into relevant structures for the application and runs a thread which constitutes the simulation time. As mentioned, threads are used. Both SimulationManager and Queue implement interface Runnable, which means that they have the run method implemented inside them.

Talking about threads, thread safety must be ensured, so some special data types are used. For the clients that wait in a queue a BlockingQueue is used. The wait time is an AtomicInteger and the flag that specifies if there are any clients at a certain queue is AtomicBoolean. This is a flag that starts and stops the thread run with a certain queue, so it is present in the while from queue’s run method.

The thread concept is very important in this problem, as it is actually required to simulate the real time. Threads are created each time an empty queue gets a client, the start method is called. As mentioned, this thread survives and does his job as long as the isOccupied flag is true, and that is as long as the waiting time for a new possible client at that queue is greater than 0. When this waiting time reaches 0, the flag is set on false and the thread practically dies (the run method reaches its end). It is not proper to state that the thread is the resurrected when a new client comes. A dead thread never comes back. What happens here is that, when a new client comes, a new thread is created having the queue as argument. So this new thread runs on the run method of the queue. There is a new thread, but on the same Runnable object.

*4.Implementation*

As shown in the previous section and mentioned in the objectives section, at this point the implementation details are to be discussed. A number of three classes was implemented and these are: Client, Queue and SimulationManager. Each of them will be taken separately and analysed from the point of view of the fields declared, of the methods and from the point of view of the functionality provided on its own.

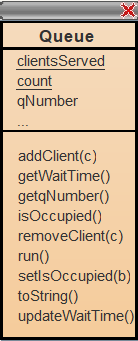


First of all, here is the class Client, the subject of this whole simulation. A client is the person who comes to the store at a certain time, has to wait in a queue until he can be processed and has a time allocated for being served. These being said, the fields of a Client object include a unique ID, which is based on a static field incremented in this class every time a Client object is created, tarrival, which is the moment of time when the client goes to a queue relative to simulation time, tservice, which is the time the client spends being the first one in the queue (in this time, tservice is decremented as time passes, until it gets to 0), and there is also the wait time, the total time the client spends in the store, from the moment when he arrives, until the moment he leaves the store (after being completely processes).

Initially, this class was designed to implement Comparable interface in order to be easier to sort the client list, but for some reason, it was not taken into consideration by Collection.sort().

This class has only one constructor with four parameters. As stated before, the ID is generated by incrementing newid and assigning it. The arrival time is a random number in the interval given by the first two parameters (the interval provided as data input). Tservice is obtained similarly, this time in the interval provided by the last two arguments. The wait time is initially 0.

The rest of the methods are mutators and accessors, as all the fields are private. And there is also an override of the toString method, which is used when printing a client in the output file.



The queue is the most controversial element of this whole application. Most of its fields are private, except one, clientsServed, which is public and used in order to keep track of the clients that have been completely processed. Count filed has the same role as newid from Client. Each queue is associated a number when created, so that there can be made a difference between them. Then there are the fields of types that ensure thread safety. They do not appear in the class card above, so they are listed below:

private AtomicInteger waitTime;  
private AtomicBoolean isOccupied;  
private BlockingQueue<Client> clients;

The wait time is the number that says how much time does a client need to wait in queue until he can be the first in line. This is the attribute that facilitates choosing the shortest queue. The next one, isOccupied, is the flag that was mentioned in the previous section, the flag that keeps the thread running on that queue active as long as there are still clients in the queue and stops the thread when the queue is empty. And, finally, there is the actual queue of clients which is implemented as a BlockingQueue and contains all the clients waiting in a row at that queue.

This class also has mutators and accessors which are not worth to be discussed. The constructor only initializes all the fields, setting waitTime on 0 and isOccupied on false. Also, there is the override of toString method which is used for listing into the output file. Now the most important methods will be presented:

1. public void addClient(Client c)

This method does one thing in 3 steps. First, it calls the add method of BlockingQueue for the client list of the queue and append the client given as parameter to the end of this list. Then the flag is set on true and the waitTime is incremented with the service time of the client.

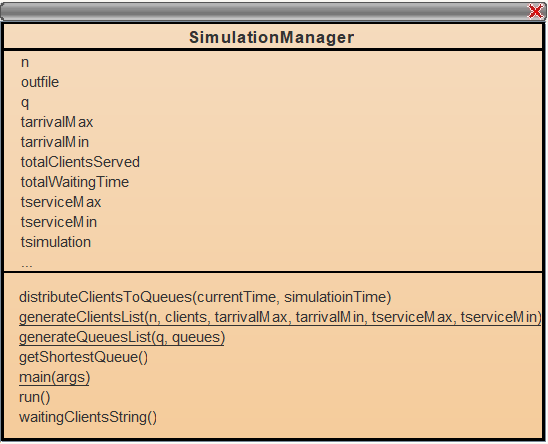
Exactly the same, there is removeClient() method that only calls the remove() method from BlockingQueue.

1. public void updateWaitTime()

This method is called inside the run method at every step of the simulation. The clients from the clients list have their wait time decremented and the first one has his service time decremented. Also, the wait time for the queue is decremented.

1. public void run()

This is the method that actually does the job. It is overriden from Runnable and it is a while conditioned by the isOccupied flag. The first client in the queue is taken and his tservice is decremented until 0. Also the rest of the clients in the queue and the queue itself is updated with the method from number 2.



SimulationManager is the brain of the application and it contains the main function. It has all the parameters given in the input file and it manages the correlation between clients and queues. More that that there are other fields of this class which are listed below:

public List<Client> clients = new ArrayList<Client>();  
public List<Queue> queues = new ArrayList<Queue>();  
public int totalWaitingTime;  
public int totalClientsServed;

The methods which were implemented will be presented below:

1. public static void generateClientsList(int n, List<Client> clients, int tarrivalMax, int tarrivalMin, int tserviceMax, int tserviceMin)

This method is used to create the initial clients list, based on the data provided by the user. They are generated randomly, with their constructor. Is just as simple as that.

1. public static void generateQueuesList(int q, List<Queue> queues)

Similarly whit the previous one, this method creates the list of queues.

1. public Queue getShortestQueue()

This method is used to get establish which is the queue that has the shortest waitTime to become the host of the client. It is based on an interation where the minimum is found.

1. public void distributeClientsToQueues(int currentTime, int simulatioinTime)

This method decides at each moment o time which client goes to which queue. It takes the clients that have the arriving time equal to the current time and adds them to the shortest queue. Also, here the queue thread are created on the mechanism that was presented in the previous section.

1. public void run()

This is the most important method of the application, as it organizes everything, starting from opening the output file, to going through simulation time with a while, to distributing the clients at the right moment and updating the queues, and finally to checking if the simulation was or was not completed. After the execution of this method, all the threads are stopped by setting their flags on false and the requested result is computed as a division between the total waiting time of all the processed clients and the number of these clients.

What is left is now the main method that gets as arguments the names of the input and output files, opens and gets the data form the input file, calls the methods enumerated before to generate the clients list and queues list and, finally, creates a thread in order to run the simulation.

*5.Results*

The results of this assignments can be seen in the provided text files. There is some input data, three cases, more exactly, so that the output of the simulation can be observed in a corresponding output file.

In what concerns the author observations, for the first test case, where there are only 4 clients, in most of the times the program was run, only one queue was used, as there was plenty of time between the arrival times of the clients such that there was no need for waiting for one another. Like this, at arrival time, the clients became directly the first in their queue.

The second test was also run multiple times and was useful in detecting some bugs in the code. The result was different as here the number of clients is close to the simulation time, which makes the 5 queues to be more busy.

And the third test was really complex, having 1000 clients waiting. This simulation ended without completely processing all the clients. Actually, plenty of them remained in the queues waiting, as there was not enough time for all of them to be processed. Even if there were plenty of clients arriving at the same moment of time, there might have happened that they were placed in the same queue. But, if thinking logically, there was still one who got there first and that makes the difference.

*6.Conclusions*

As a conclusion, the simulation parameters should be chosen very carefully, as they can radically influence the way the testing goes. There are plenty possible situations that can happen and all of them should be taken in account.

Also, regarding the assignment in general, it had a great role in initiating the very little piece of knowledge the author has about threads. Unknown solutions and data types were discovered and used in the most useful way possible. The concept of thread in java seems to be a very large domain that can perform a lot of tasks and is associated with special data types that ensure thread safety. Synchronization was used in order to maintain the flow of the instructions and to avoid the timing errors that can occur due to multiple threads being active at the same time.

Working with files also improved. The fact that the .jar file was required was a new challenge for the author, but it was completed successfully.

*7.Bibliography*

1. <http://coned.utcluj.ro/~salomie/PT_Lic/4_Lab/Assignment_2/Java_Concurrency.pdf>
2. <https://stackoverflow.com/questions/16137713/how-do-i-run-a-java-program-from-the-command-line-on-windows>
3. <http://inf.ucv.ro/documents/tudori/laborator8_53.pdf>
4. <https://www.geeksforgeeks.org/iterators-in-java/>
5. <https://www.geeksforgeeks.org/blockingqueue-interface-in-java/>
6. <https://www.geeksforgeeks.org/atomicboolean-set-method-in-java-with-examples/>
7. <https://docs.oracle.com/javase/7/docs/api/java/lang/Thread.html>
8. <https://www.geeksforgeeks.org/different-ways-reading-text-file-java/>
9. <https://app.creately.com/diagram/RByiN2QhN5T/edit>