Programming Techniques

Homework 2: Queues Simulator

Student: Soós Roland

Group: 30421

**1.Objectives**

Design and implement a simulation application aiming to analyze queuing-based systems for

determining and minimizing clients’ waiting time.

The application should simulate (by defining a simulation time 𝑡𝑠𝑖𝑚𝑢𝑙𝑎𝑡𝑖𝑜𝑛) 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), 𝑡𝑎𝑟𝑟𝑖𝑣𝑎𝑙 (simulation time when they are ready to go to the queue; i.e.

time when the client finished shopping) and 𝑡𝑠𝑒𝑟𝑣𝑖𝑐𝑒 (time interval or duration needed to serve the

client by the cashier; i.e. waiting time when the client is in front of the queue). The application

tracks the total time spend by every customer in the queues and computes the average waiting

time. Each client is added to the queue with minimum waiting time when its 𝑡𝑎𝑟𝑟𝑖𝑣𝑎𝑙 time is greater

than or equal to the simulation time (𝑡𝑎𝑟𝑟𝑖𝑣𝑎𝑙 ≥ 𝑡𝑠𝑖𝑚𝑢𝑙𝑎𝑡𝑖𝑜𝑛).

Secondary objectives:

-Designing the code in the Object Oriented Programming style;

-Structuring the ideas for the design by creating an UML diagram and representing the efficient organization of the classes;

-Creating a multi-thread implementation of a number of queues (ex: simulate the multiple checkouts from a store);

-Opening and closing the queues in a dynamic way such that when a certain queue has no customer, the queue should be closed, and open otherwise;

-Using appropriate synchronized data structures to assure thread safety;

-Getting used to the creation and configuration of the .jar files

**2. Analyzing the problem, modelling, scenarios**

It is necessary to get a detailed implementation for the simulation of the queues in order to get as close as possible to a real situation, thus the simulation results and statistics to have value and importance. First we need to generate N number of customers, each one of them with random data of arrival time( the moment the person enters the queue) and service time( the amount of time spent at the peak of the queue and being given their required services). These customers will be dispatched to queues and our program will have the task to do it as efficient as possible, having a limited number of available queues.

This homework does not require a graphical user interface/representation, and because of this, it is not designed on the model-controller-view manner, it is simply a one package project. All the classes (total of 5) are put in the same package called Simulation. Each class has a very specific functionality, which I will describe in more details in the other chapters of this documentation. A short walkthrough of the design would be:

* “Client” class, represents a customer object and it has data such as ID, arrival time, service time and waiting time
* “Server” class, represents a queue
* “Manager” class, it does the management of the servers (queues)
* “Data” class, reads the data input from a text file and processes it into segments (number of clients, number of queues, …). It also generates random customers
* “Main” class, this class has only a main method, that runs the application accordingly to the required specifications

For processing all the queues in the same time we will use threads. A thread of execution is the smallest sequence of programmed instructions that can be managed independently by a scheduler, which is typically a part of the operating system. Multiple threads can exist within one process, executing concurrently and sharing resources such as memory, while different processes do not share these resources. In particular, the threads of a process share its executable code and the values of its dynamically allocated variables and non-thread-local global variables at any given time. In this implementation I used the run() method and override it.

A scenario represents a sequence of steps taken by the user to correctly interact with the application. The possible scenarios are described in the following section:

* The user gets the desired results. To get the these results the user must rebuild the .jar file and then introduce correctly in the terminal the command that should run the jar file (ex: java -jar PT2020\_Soos\_Roland\_Assignment\_2.jar <input file> <output file>). If the input file is not corrupted, then the application should create a new file for the output (or update an existing one) and write the results in in.
* The user does not get the desired results:
  + The command is not typed correctly. In this case in the terminal will appear an error status and the output file will not be written
  + There is a problem with the input file. Either it does not exist, in which case there will not be any output file, or the input file is corrupted, if the data written inside does not respect the correct format

Correct format:

<number of clients>

<number of queues>

<simulation time>

<minimum arrival time>,<maximum arrival time>

<minimum processing time>,<maximum processing time>

**3. Design and UML Diagrams**

For designing the project, I used two types of data structure in order to maximize the efficiency.

The first data structure used in this assignment is the queue. A queue is an abstract data type representing a collection of items, which are stored in a fixed way, to which it can be realized adding operations named “enqueue” and extracting or removing operations from the head of the queue named “dequeue”.

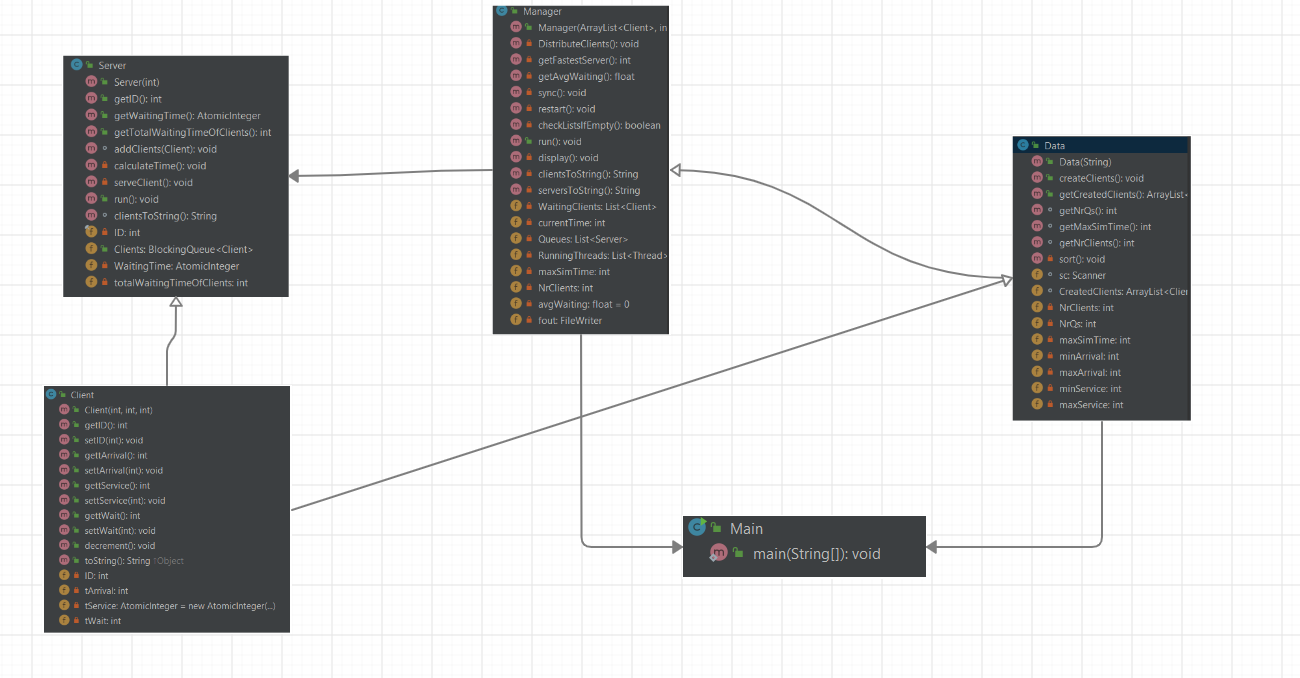
The second data structure is a linked list. The list is an abstract data type that represents a countable number of ordered values, where the same value may occur more than once. If the same value occurs multiple times, each occurrence is considered a distinct item. The difference between the list and the queue is that the list can be traversed, while in the queue we have access only to the first element of that data structure. This property of the queue is beneficial for our implementation since in our queues we work only with the first customer.

The simulation of a queue is done using the classes Server and Client. The Client class will represent customers as objects that will fill up a Server type object, which will represent our waiting line, more exactly, our Queue. The simulation of multiple queues needs the help of the Manager class, which manages a list of servers and a list of customers that are waiting to be put in one of the queues.

The customers are generated using the Random() class. The Data class reads the bounds of the arrival time and service time, for ex (4,22 \n 2,6), it means that the customers are expected to arrive between 4 and 22 and the time needed to serve them is expected to be between 2 and 6. The data generated for these customers have values from given intervals.

Regarding the threads, the Main class creates the main thread that will create the simulation, while the threads responsible for the queues are created in the Manager class, such that they are connected to their respective Server object.

UML Diagram:



**4. Implementation**

**4.1 Client Class**

This class has 4 parameters:

* Private int ID – it holds the id of a customer, so that it will b easier to track down its movement and the way the simulator puts him in the queues
* Private int tArrival – it holds the time the customer is expected to arrive at the queue
* Private int tWait – represents the total time spent waiting in the queue
* Private Atomic Integer tService – this variable will pe decremented each step in the queue when being at the head of the queue. It represents the time spent on being served as a client

Some of the implemented methods of this class are the getters and setters for all the data mentioned above and obviously there is a constructor that will take 3 parameters Client(int ID, int tArrival, int tService). The class is very straightforward and very simple, so I needed only to add two more things to it: a method that will decrement the service time, I simply called it decrement() and it does a simple command tService.decrementAndGet(), and an override of the toString() method that I will use when writing into the output file.

**4.2 Server Class**

The Server Class will play the role of a waiting line, or as a self implemented queue. The class stores integers ID and totalWaitingTimeOfClients, an AtomicInteger WaitingTime and a BlockingQueue Clients. The main functionality of the class comes from implementing Runnable.

The algorithm of the class is very simple. It takes a client and puts it in the queue, if empty, in the head, if not, in the first empty position counting from the head, then a method will calculate the WaitingTime ( the amount of time needed to get to serve all the costumers in front of this customer) and at each step, update the service time of the customer that is at the head of the queue.

The methods implemented of this class are:

* AddClient(client) – adds a Client type object to the queue Clients, calculates the waiting time of the queue with the calculateTime() method, then sets the waiting time of the client and adds to the total amount of waited time( this is necessary in order to able to calculate the average time spent waiting)
* claculateTime() adds up the service time of all the clients that are currently in the respective queue
* serveClient() decrements the service time of the customer that is at the head of the queue and decrements the waiting time
* run() method is an override method and will be called by the server’s thread. This method verifies if the queue is empty, if it’s not, then calls the serveClient() method, then takes out the client from the head of the queue when it has service time equal to zero.
* toString() method is just to help us print easily the output into the file

**4.3 Manager Class**

This class implements Runnable as well, and it will have also a run() method overridden. The class will act as a scheduler of queues, and it has the role to search for the fastest queue for each client that is waiting to be put into the queue. Finding the fastest server is done by checking the waiting time and choosing the server that has the smallest waiting time. For this step I have implemented 2 methods: getFastestServer() and DistributeClients(); getFastestServer() method will return an int with the serverID of the server in order to know exactly which server should we look at. DistributeClients() method will use the getFastestServer() for every customer that has arrival time equal to the current time. When a server is found, the client is inserted there, and finally removed from the waiting list.

The Manager class has:

* List<Client> WaitingClients – clients that haven’t arrived yet (tArrival > currentTime)
* Int currentTime – the time of the simulation
* List<Server> Queues – a list of servers that will serve clients
* List<Thread> RunningThreads – a list of threads that will run the servers
* Int maxSimTime – a value that represents the maximum simulation time (current will never be greater than maxSimTime)
* Int NrClients – the number of customers in total that took part of the simulation
* Float avgWaiting – the average waited time by a client
* FileWriter out – a writer object used to write the output in to the file

Two very important methods that glue together the functioning of this class are: sync() and restart(). These two methods are responsible with syncing and restarting the threads in order to complete the simulation properly. The sync() method will join all the threads from the RunningThread list using .join(millis: 10), while the restart() will check if there are terminated threads and changing them with new threads with the respective server, thus keeping the thread list running and working without interrupting the overall flow.

The display() will simply write the updated time, waiting list and the queues and its objects stored, into the output file.

**4.4 Data Class**

This class is a very simple class, all it does is to get the input file, read its content, transform the lines into actual data of integers (number of queues, number of clients, maximum simulation time, arrival time interval, service time interval) and in addition to this, to generate clients with data corresponding to the values read. It also has a sort() method, that will sort all the clients from the list of clients created in increasing order by arrival time.

**4.5 Main Class**

This class contains only a main method that will create the main thread, a data object and a manager object and then start the simulation: (Manager) M.start();

**5. Results**

The results of this application can be found as output txt files written during the simulations. I was able to run all 3 test files and get corresponding results.

**6. Conclusion**

This assignment was an opportunity to understand better the concept of threads in java and to design a project that will be based on this concept. I learned some new things and new ways to approach different types of problems and I think it was quite a lot of fun.

**7. Bibliography**

Java\_Concurrency.pdf

<https://en.wikipedia.org/wiki/Queue_(abstract_data_type)>

<https://blog.jetbrains.com/idea/2010/08/quickly-create-jar-artifact/>

<https://www.javacodegeeks.com/2013/01/java-thread-pool-example-using-executors-and-threadpoolexecutor.html>

<https://www.baeldung.com/java-thread-stop>