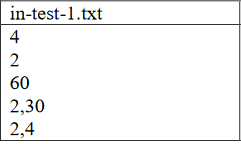
**Queues Simulator**

**Avram Andrei-Alexandru**

**30424**

1. **Assignment’s objective**

The main objective of this assignment is to design and implement a simulation application that is capable of analyzing queuing-based systems for determining and minimizing client’s waiting time. A secondary objective is the design and implementation of an efficient strategy to place the clients in a queue. For this strategy two approaches could have been chosen: Suppose that a client (or any number of them) arrives, we can direct this client to a queue that has the least amount of clients in it or we can direct him/them to the queue that has the least waiting time (by this waiting time I refer to the total service time of the clients that are already in that queue). In this application I have chosen the second approach, the implementation is explained [here](#optimalQueue).

The input and the output of the simulation are stored in two separate text files that are supplied via the command line arguments in this order: inputFile.txt outputFile.txt. (Note that file names are given here as an example). In order for the application to work properly the input file must respect **exactly** the format that was posted online, the format is also presented below:

The content of this file is: 4 – number of clients, 2 – number of queues, 60 – simulation time, 2 respectively 30 – the minimum and maximum arrival time,

1. **Problem analysis, modelling, scenarios, use-cases**

This application can have many real-world use cases. For example, it can be used to roughly model and simulate an administrative building that has many clients and persons that serve them daily. It can be used to see if your setup can be improved any further or in fact if it actually makes sense to open more queues if that action does not translate in a notable increase of your processing performance. Another use case is in the virtual world. If you have a number of servers and a number of requests to process that arrive to you randomly (but still follow some guidelines) you may need an application to see how fast you can process these requests and if adding more servers with an increase in cost actually makes sense.

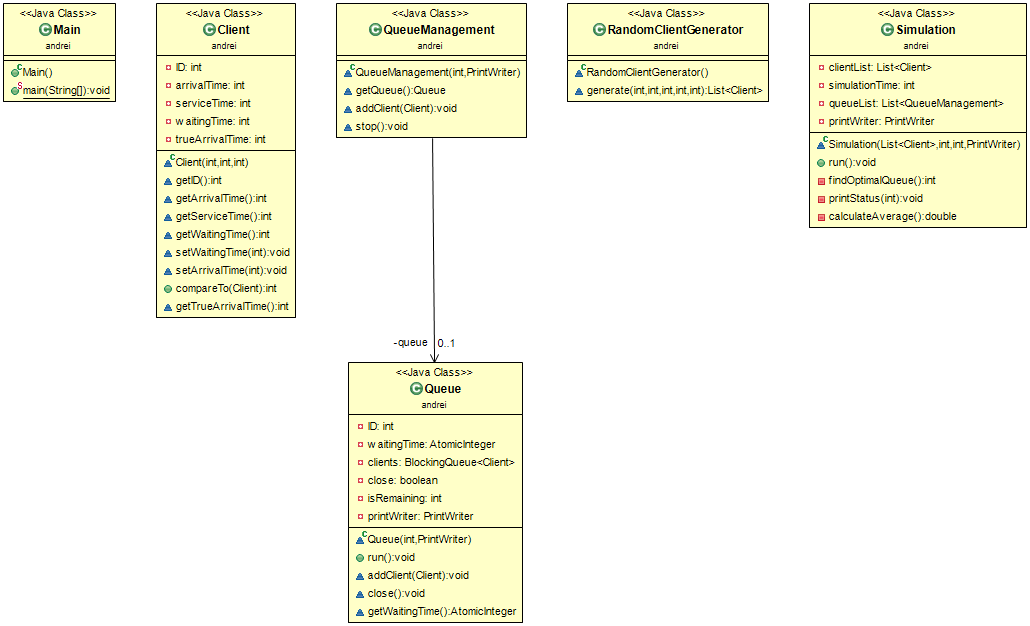
Functional requirements:

|  |  |
| --- | --- |
| ID Requirement | Description |
| 1 | The input data will be provided in the form of a text file given as a parameter |
| 2 | The user will wait a number of seconds for the simulation to be done |
| 3 | The output is provided in the form of a text file also given as a parameter |

To use this application, one must follow these steps:

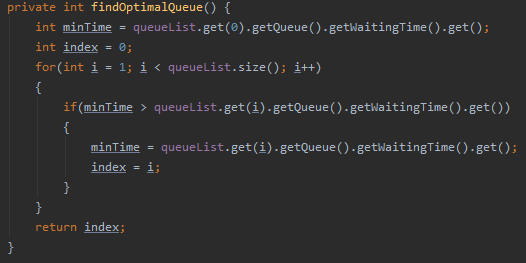
1. Make an input file that follows the format.
2. Run the application providing as command line parameters the input and output file.
3. Wait for the simulation to be done.
4. Check the results that are stored in the output file.
5. **Design (design decisions, UML diagrams, data structures, class design, interface, relations, packages, algorithms)**

For the design of this application I have chosen to use one package. The UML diagram for the whole

application is presented below:

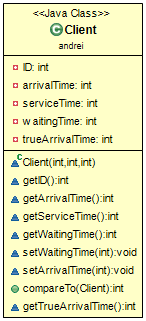
The actual client processing is done in the Queue class. But each Queue has its own manager to say, a QueueManagement object. The number of Queues and QueueManagement objects are the same and is equal to the number of queues given in the text file. At the start of the simulation, which is done in the Simulation class which is also a thread, an ArrayList of QueueManagement objects is created and using these objects the number of required queues are created and started. The Queue and Simulation classes both extend the Thread class in order to be able to create threads and run them independent of each other.

Data structures used in this project are: Lists to hold the QueueManagement objects and BlockingQueues to hold the clients in a queue.

 Next, I will present the strategy for distributing clients in queues. Each Queue has a field named waitingTime which is an AtomicInteger. I made it AtomicInteger to ensure that the add and subtract operation is done in one operation. First when a client is added to the queue the waitingTime increases with the client’s service time. When a client is done the waitingTime decreases with that client’s service time. In this way we can make an efficient distribution of clients among queues. The algorithm to find a queue to place a client is very simple:

First, we set the minimum waiting time to be equal to the first queue waiting time. After that we iterate from the second queue up to the last queue and we see if we can find a queue that has a lower waiting time than the minimum. If a lower time is found the minimum value takes that queue waiting time. This method returns the index of the QueueManagement object in the ArrayList that holds the queue with the minimum waiting time.

1. **Implementation**
2. The Client class



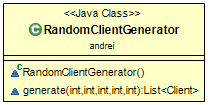
Each client is uniquely identified using an ID. The client class has only integer fields

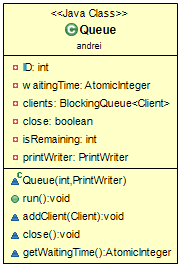
which store besides the ID, the arrival time, service time, the waiting time (how much this particular client had to wait to reach the front of the queue and to be processed) and the true arrival time. I have made a second variable to hold the arrival time because the first one gets modified during the simulation. It is modified in order for my internal algorithm to work.

This class implements the Comparable interface because I needed to sort the list of clients, that we generated randomly, based on their arrival time. If client A has a lower arrival time than client B then it will be in front of B in the list.

As for methods that are in this class, nothing is special, only getters and setters and a compareTo method that takes a parameter a Client object and returns 1, 0, -1 if thisclient has a bigger, equal or lower arrival time compared to the parameter client.

1. RandomClientGenerator class

This class is used to generate a given number of clients with their arrival time and service time generated randomly. The arrival and service time are generated within a range. This class has only one method that returns a List of clients. The number of clients needed to be generated and their time bounds must be supplied to the method.

1. Queue class

This class is implementing the queue where the clients are being served. The fields and their meaning are explained below:

**ID** – used to uniquely identify each queue

**waitingTime** – is the number of seconds a client that arrives at this queue needs to wait before is being processed.

**clients** – is the blocking queue that holds the clients that have arrived at this queue and are waiting to be processed

**close** – is used to shut down the thread

**isRemaining** – used to see if the client in the front of the queue is leaving right now. So that if a client is done in the same time another client comes to be processed, no waiting time is required between these two operations.

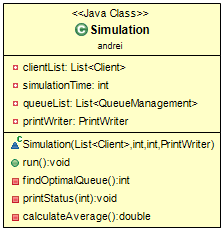
**printWriter** – used to write to the output file.

Furthermore, I will explain the two most important methods found in this class.

The addClient() method adds the client given as an argument to the queue of clients that are waiting to be processed. Also, here the waiting time for each client is set to the value equal to the waitingTime field found in this class.

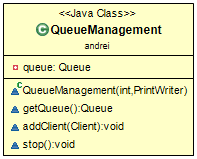
The run method is the method that is driving the thread. Inside this method is while loop that is executing until the thread has been given the signal to shut down (close field originally is set to false but after the close() method is called the close field is set to true). In this loop only two things can happen. If the queue that holds the clients that are waiting to be processed is empty the Queue sleeps for one second. Otherwise, we take the client that is in front of the queue and the thread sleeps for a number of seconds that is equal to the client service time. After that number of seconds have passed the client is done and the waiting time for the queue is decremented with the service time of the client that was previously serviced.

1. Simulation class



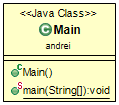
This class, which itself is a thread, is used to run the simulation. First when this class is instantiated, in the constructor every thread gets created and started (using appropriate methods found in QueueManagement class). The findOptimalQueue() method is used to find a queue that has the smallest waiting time. How this is done is explained [here](#optimalQueue). The printStatus() method is used to write to the output file the status of the clients every second. In the run method there is a for loop that goes from 0 until the simulation time. In this for loop first, for every client that has arrived (arrival time <= current simulation time) we find an optimal queue to direct to place the client. After that the thread sleeps for one second. If all the clients have arrived and all of them have been serviced the simulation stops. After the simulation had stopped the average waiting time is computed and printed in the output file and all the queues are shut down.

1. QueueManagement class



This class is used to create and start a queue. The constructor has two parameters: an integer that represents the queue ID and a PrintWriter that is passed to the queue and is used inside the run method of the queue to print to the output file the status of the queue. The get queue method returns the queue, the add client method sends to the queue a client, and the top method signals the queue to stop.

1. Main class

 This method only contains the main method used to start the program. In the main method all the needed information is read from the input file and stored in corresponding variables.

1. **Executable jar**

For this application I made an executable jar that can be run from terminal in the following way:

java -jar nameOfExecutable.jar inputFile.txt outputFile.txt

**Important** in order to run this jar file you need to have installed on your system java 8.

1. **Conclusions**

After completing this assignment, I have learned how to create and work with multiple threads in java. In a future development of this application one can extend the functionality of clients to have more complex tasks, not just waiting for a period of time and one can add different types of clients and multiple simulations that run at the same time.

1. **Bibliography**

<https://www.tutorialspoint.com/java/java_multithreading.htm>

<https://www.youtube.com/watch?v=YdlnEWC-7Wo&list=PLBB24CFB073F1048E>

<https://www.tutorialspoint.com/java/java_thread_synchronization.htm>