|  |  |
| --- | --- |
| **Project title** | **QUEUES SIMULATOR** |
| **Author(s)** | **Pop Vlad-Iulian-Ioan** |
| **Group** | **30421** |

1. **Task Description**

Design and implement a simulation application aiming to analyse 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 requirements**

* Object-oriented programming design
* Classes with maximum 300 lines
* Methods with maximum 30 lines
* Java naming conventions
* Random Client Generator
* Multithreading: one thread per queue
* One test run and saved: in-test-1.txt
* .jar file uploaded and configured to run according to deliverable requirement
* Appropriate synchronized data structures to assure thread safety (avoid synchronized keyword as much as possible)
* Queues should open/close dynamically. Initially all queues are closed. When clients are distributed to the queues, they become open as needed. When a queue becomes empty, it is closed, and the corresponding thread is paused.
* Other two tests given in requirements (in-test-2.txt, in-test-3.txt)
* Compute average waiting time

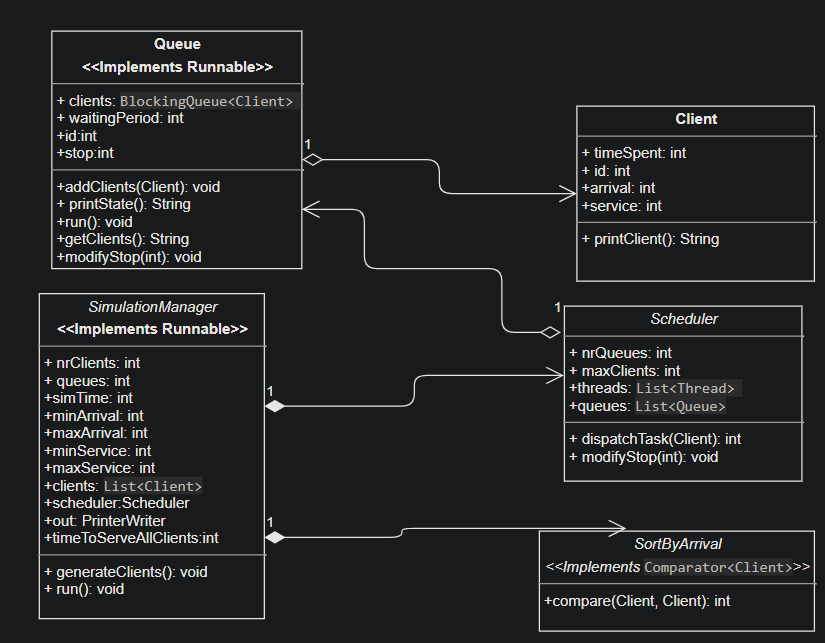
1. **Problem Analysis**

The user must provide for the arguments of the program the name of the input file as the first argument and the name of the output file as the second argument. The input file must obey a certain pattern. The number of clients on the first row, the number of queues on the second row, the simulation time in seconds on the third row, on the fourth row the range for the arrival time of each client e.i. the moment he is done shopping, the lower bound and the upper bound must be separated be a comma and on the fifth row the range for the time it take to for the client’s produces to be scanned when he is in front of the queue also the lower bound and the upper bound must be separated be a comma.

The time of the simulation is given in seconds. The implementation will consider one second equal to 10 miliseconds from practical and saving time reasons.

* Use case: the input and output files can exist
* Primary actor: User
* Main Success Scenario: The output is show in the output file
* Use case: the data in the input file do not respect the pattern
* Primary actor: User
* Main Success Scenario: The output is show in the output file will show garbage values or the program will stop running
* Use case: the number of arguments is wrong
* Primary actor: User
* Main Success Scenario: The output to the console will be: " The number of arguments is wrong"
* Use case: the input file cannot be found
* Primary actor: User
* Main Success Scenario: The output to the console will be: "No such input file found"
* Use case: the output file cannot be found
* Primary actor: User
* Main Success Scenario: The output to the console will be: "No such output file found"

1. **Designing**

****

1. **Implementation**

**Client:**

This class is supposed to represent the client. It has 4 fields. The timeSpent field represents the amount of time the client will spend in the queue. The id is the identification. The arrival time is the time when the client will enter the queue. The service field is the amount of time it takes for the client to be served as the head of the queue.

This class has 2 constructors. One constructor gets as input the id, arrival and service.

The other constructor receives as input the id and the range in which the arrival and the service fields will be generated. Those 2 fields will be random generated with the help of the Random class.

The printClient method is used for nice printing the id, arrival and service fields in a nice way.

**Queue:**

This class queue in which the clients will wait to be served. The field clients is a BlockingQueue of Clients implemented with ArrayBlockingQueue of Clients. The field waintingPeriod is represents the amount of time a new client will have to wait to reach the head of the queue. The id is the identification. The stop field is used in the run method to kill the thread, if the field is 0 the run method will run normally and if it will be 1 the main while from the run method will be stoped and so the run method will reach its end terminating the thread.

The constructor receives as parameters the id of the queue and the capacity of the queue e.i. how many clients it can hold at one time.

The addClient method is used for introducing clients in the queue, computing the waitingTime corresponding to the new client and notifying the thread so it can be unblocked and resume its execution. The method is synchronized with the class.

The printState method will return a string which will help us in pretty printing the state of the queue. The state will be closed if there is no client in the queue. If there are any clients in the queue they will be appended to the returned string with the help of the printClient method defined in the Client class. The clients field is traversed with a foreach.

The run method processes the state of the queue at each step from the run time. It is synchronized with the class. While the queue is considered as not closed e.i. it has clients the service time of the head of the queue will decrease at each step and the time the client spends it the queue will increase. After this operations the thread will sleep for 10 miliseconds and will take if from the beginning till the queue is closed. If the queue becomes closed the thread will be blocked. It can be unblocked only by the addClient method or the modifyStop method. In the later case it will be unblocked only so it could end its execution. The execution will end by breaking the big while that encapsulates the method, which runs to infinity, so the flow of the thread will reach the end of the run method.

The modifyStop method will help the thread to terminate by notifying it. It is synchronized with class. It will only be called from a method from the Scheduler class.

The getClients method is used for listing the clients of the queue and was mainly implanted for testing and debugging purposes.

**SimulationManger:**

This class keeps the track of the simulation time. The simulation time is assumed in seconds but I implemented it as 1 second to be equal to 10 milisecond for the sake of testing. It would be bothersome for a test case with a very large running time, it is more practical this way.

The field nrQueues represents the number of queues. The queue field represents the number of queues that the program will use. The simTime field is the time or steps that the program will execute. The minArrival and maxArrival fields represent the range of time in which the a client can arrive in a queue. The minService and maxService represent the range of time in which a client can spend as the head of the queue. The clients field represents the list of clients that the store will have. It is a List of clients implemented with the ArrayList class. The scheduler field is an instance of the Scheduler class which will manage the queues. The field out is an instance if the PrinterWriter class and will be used to print the output into the desired output file.

The field timeToServeAllClients represents an alternative time stamp for the the program to end. If there are no more clients to waiting and the queues are empty the simulation will stop.

The constructor of this class receives as arguments the Scanner object used to read the input file, and the PrinterWriter object used to print the output in the desired file.

The input will be extracted and stored in the different fields of the class in the constructor. The constructor also calles the method generateClients for randomly generating clients based on the input.

The generateClients method randomly generates the number of clients that the input specified. The created clients are added to the clients field and then sorted with the help of the sort method of the Collections class. The arguments of the sorting method are the list of clients and an object of type SortByArrival. Also in this method is the field timeToServeAllClients initialized with the sum of the arrival and service time of the last member of the sorted clients list.

The run method does the simulation of the required task and prints the result with the help of the out field. The simulation will end if the current time becomes equal to the simulation time or if the current time becomes equal to the timeToServeAllClients field. We always remember the position till where every client has been put in a queue. If the arrival time of client is equal to the current time of the simulation it will be added to the queue and will update the timeToServeAllClients field if needed, if time that the client will spent in the queue is larger than it. We count the clients served for the average waiting time. The number of clients served will not necessarily be the number of client specified in the input file because some of them may not be served in the available simulation time. The thread will be put to sleep for 10 miliseconds and then the rest of the output will be printed. At the end of the method the scheduler modifyStop is called so that the queues threads can be terminated and the output file will be closed. The method is synchronized to the queues field of the scheduler object.

**Scheduler:**

This class will start the queue threads and will help at adding clients to the right queues. The field nrQueues represents the number of queues that will be created.

The field maxClients represents the number of clients a queue can hold at one time. I used the total number of clients for this. The field threads is used to store all the threads that will be create. The List structure is used for it and it is implemented with the ArrayList class. The field queues is used to store the queues that will be created and its type is similar to the threads field.

The constructor of this class receives as its parameters the number of queues to be created and their capacity. The constructer stores all the threads and the queues in the class fields and calls the start method for each one of them.

The dispatchTask method is to identify the queue with the smallest waiting time and add the Client object given as its argument to the right queue. It is also used for calculating the amount of time the client will stay in the queue and will return this number as the potenttialy new timeToServeAllClients from the SimulationManager class.

The modifyStop method is used for ending the queue threads and it is called from the run method of the SimulationManager. It calls the method modifyStop from the Queue class for each of the queue threads waiting.

**SortByArrival**

The class’s sole reason is to help at sorting the clients by arrival time. It overrides the compare class of the Comparator class.

1. **Testing**

The testing of the program is done through the 3 input files given as example in the task description.

1. **Conclusinons**

This program helped me get hands on experience with the thread concepts and threads in java.

1. **Bibliography**

The pdf. Support that the laboratory teacher provided to us.

Multiple sites from the World Wide Web.