Assignment 2

Student Name:Maios Patrick

Group:30424_1

1.Assignment Objective

The main objective of this assignment is to improve queue waiting times using multiple threads. A thread in Java is the direction or path that is taken while a program is being executed. Generally, all the programs have at least one thread, known as the main thread, that is provided by the JVM or Java Virtual Machine at the starting of the program's execution. We can we use this for is creating new threads that will allow us to send tasks to it, and the more we have, presumeably, the faster our program will work, and it would be easier to process more data. The Java framework allows us to instanciate those threads and run them at the same time. The main objective, as we talked about, is about making the threads work and finally giving the desired output when given a list of customers.

Some sub-objective we need to take into account:

1. Firstly, we will have a SimulationManager class that will be used in giving the problem's parameters. Some parameters we must take into account are time_limit (which gives out the maximum time the program can run), numberOfServices which gives out the number of Queue threads that we will be running and also the numberOfClients which gives out the number of clients that will be randomly generated using a generateRandomTasks method. Also, one more important detail is about how we want to store the answer, for that we will be using a file in which we give out the answer.
2. Secondly, we will use a Scheduler class that has a very important role in generating our required input data. The scheduler helps with the dispatching of the task and being able to change its selection policy that we'll talk about now.
3. The strategy interface implements 2 different methods of choosing how the tasks will be put into queues. I have decided to bring one method that choses the shortest list based on time needed to complete itself, and the second one being a method that sends the task through the shortest thread based on queue length.
4. Lastly, we will implement a class server that will asc as a Queue/Thread of sorts and will process tasks, add them to the list, remove them when needed, we have some constraints that we must keep in mind when working on the server thread, for example the maximum thread capacity.

2.Problem Analysis, Modeling, Use Cases

Use Case: setup simulation Primary Actor: user Main Success Scenario: 1. The user inserts the values for the: number of clients, number of queues, simulation interval, minimum and maximum arrival time, and minimum and maximum service time 2. The user clicks on the validate input data button 3. The application validates the data and displays a message informing the user to start the simulation Alternative Sequence: Invalid values for the setup parameters - The user inserts invalid values for the application's setup parameters - The application displays an error message and requests the user to insert valid values - The scenario returns to step 1

3.Design

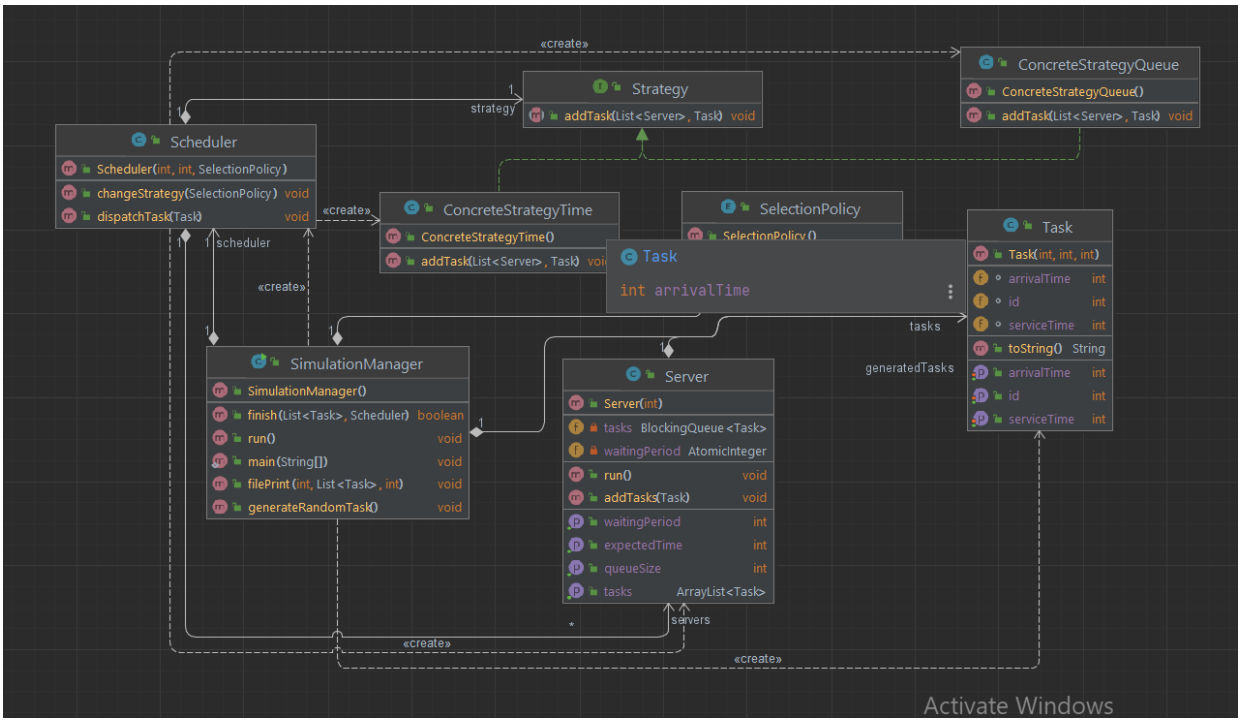
The aplication works best using OOP paradigms because of the real-life comparison to the real world, the problem was solved on the ground example of clients at a supermarket waiting in queues and our job being optimising those queue times in order to achive maximum speed and efficiency. On the main end, we would have the SimulationManager that would work as the ideea of people waiting in a line, with no open queues to be put into, other parameters declared here are just for the purpose of the simulation. The scheduler class is used as a helper for deciding where do people have to be put in the queue in order to achive maximum time efficiency and also the Server class that is more or less the queue, the clients being placed there, awaiting their turn and their completion.

The used data structures that are most proeminent in the whole program would be the AtomicInteger used in calculating the time the specific thread has to wait in order for it to be able to choose on which thread the client should go. It is most important to use AtimicInteger when working with threads in order to assure data consistency and not have jumbeled datas that will most likely overwrite the other ones, causing massive inconsistencies and even maybe program failture

Another data structure that I have used is the BlockingQueue which is just a queue that blocks itself if it's full. I have discovered some methods from the blocking queue that i coulde've used, like .take() which takes the first

element of the queue and then removes it, proved to be more useful the .peek() method that just shows the first element of the queue without removing it.

I have defined one interface called Strategy, the method addTask() and two implementations of it, Concrete Strategy Time and Concrete Strategy Queue. In the Concrete Strategy Time I compare the waiting times for all threads in order to find the smallest queue in regard to waiting time, additionally I used the waiting period method in the server class to take out the number of clock cycles it takes in order to start the new task. The method is optimal in my opinion because it choses the best time and reduces the wait time compared to the second one. In the Concrete Strategy Queue I compare the sizes of all queues and chose the shortest one to put my new client, this method is slower in my opinion, because you could end up with your program choosing (randomly) the queue with a very long waiting period, that will make the program slow and maybe even fail to complete it's computations before the time limit.We also used the Selection Policy enum in order to chose which one we would like to use



4 Implementation

TASK:

The task class is just a simple class that has data for the input, acting like clients. Tasks are identified by an unique ID, an arrival time which has the job of indicating when the client would join the queue, not sooner nor later, and service time which indicates the time they will spend in the queue when put at the head of the queue, for example, if one client has arrival time = 4 and service time =5 on the time 4 of the program, the user will enter the queue (presumeably at the head of the queue) and then wait 5 clock cycles, the be removed.

SERVER:
The server class takes a Blocking Queue of tasks and also uses the atomic integer fot it's calculations. The most important thing in the server class would be the run method, because Server is a thread, it is required to be ran, so in the run method, I chose to make some computations in order to check if the element shall be taken out and the next one checked for, or just let the program sleep for one more cycle. Some other methods that I used are the add task method which does just that, adds the task at the end of the queue

SCHEDULER:

The scheduler class has a very important role when it's constructed, because all of the threads are started from the scheduler class. The change strategy method changes which interface will be implemented when adding to the queues. The dispatch Task adds the task to it's corresponding server, waiting for it to be processed further.

SIMULATION MANAGER:

Simulation Manager class is used as the starter class for all, here wa also have the main method which calls out the Scheduler and starts the first thread, the main one. The most important method from this class would have to be run method again because the class implements runnable also and all computations are done there. But another very useful method would be generate random tasks that does just that, generates random tasks with random information, to assure the difference between clients and save us time with the input data. The file print method also is very important because it is required for us to send the simulation to a txt file for it to be valid.

5.Results

After a lot of time debugging and working out the nit-picks for the problem, the result given out by the program is the desired one, I can see the step-by-step traversal of the elements, them being added in the queue, their service time being decremented and them being taken out of queue, never to be seen again. The tasks generated randomly are also sorted by their arrival time to guarantee the time it takes for them to enter the queues would not be compromised by applying search algorithms in it, making it also faster.

6.Conclusion

This problem was harder than the previous one, but I managed to learn a lot when it comes to threads and how they work, I was really dumbfounded when I first started working with the threads, because I didn't really understand how the run function should work, and also had trouble understanding how the sleep method works for the processor because I thought that the thread would be stopped and not work according to the time inconsistencies. I've also learned how to write in a file in java, which was also nice, It really gave me something to work on and think about, and the information I gathered will stay with me for a very long time 😊!

7.Bibliography

https://dsrl.eu/courses/pt/

https://www.digitalocean.com/community/tutorials/atomicinteger-java

https://developer.ibm.com/tutorials/j-threads/

https://www.w3schools.com/java/default.asp

two words