Fundamental Programming Techniques

Assignment 2:

Queues Simulator

Name: Filimon Adelin

Group: 30424

1.Objective

The objective of this assignment is to design and implement a simulation application aiming to analyze queuing-based systems for determining and minimizing clients' waiting time.

The secondary objectives are:

1) Design the client class and generator of clients class
2) Design the strategy classes using Strategy pattern
3) Design the server class, runnable component representing the queue
4) Design the scheduler class
5) Design the main component, simulation manager
6) Test the application
7) Generate the .jar file

Clients represent the model distributed in the queues. They are defined by ID, arrival time and service time.

Strategy classes represent the necessary models used for distribution of the clients in the queues. Using Strategy pattern, we can change the way they are distributed.

Server class have the role to process the clients. They should open/close dynamically.

Scheduler class make use of the strategy models and distribute each client in the proper queue.

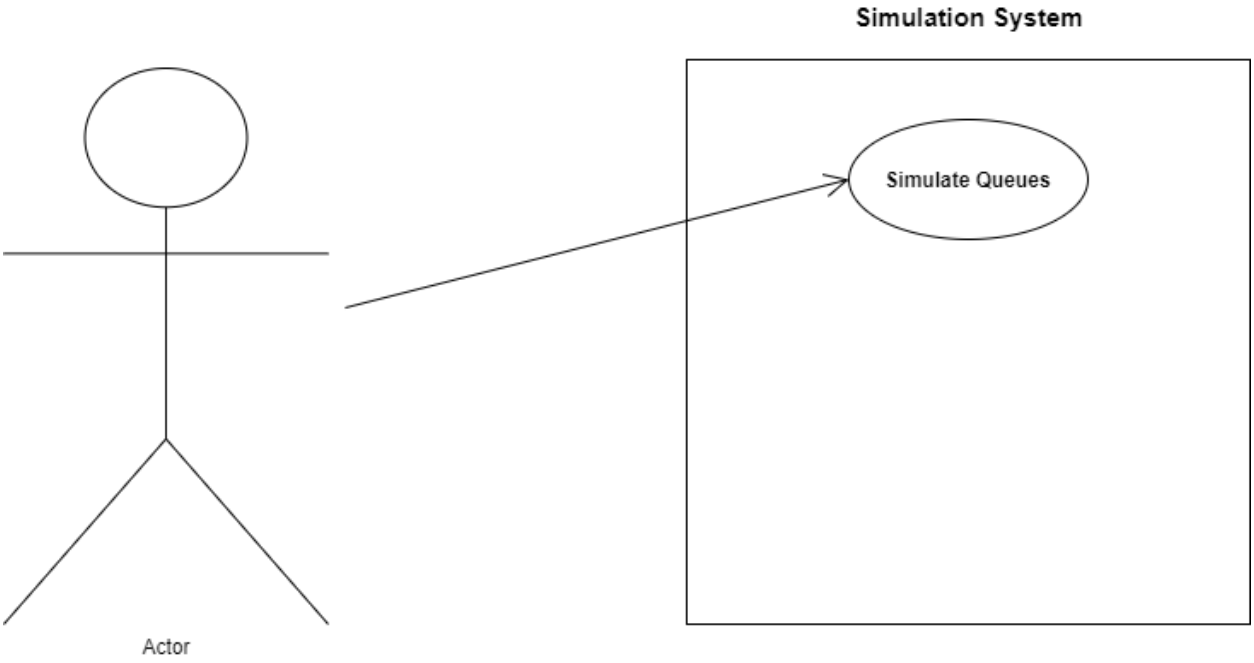
Simulation Manager maintain the simulation time and works in parallel with the servers to ensure the desired behavior.

The application is tested using some test cases, if they comply to the desired output, generate the jar file.

2.Problem analysis

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. 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 ($tarrival \geq tsimulation$). The output of the application is a text file containing a log of the execution of the application and the average waiting time of the clients.



Use case: Simulate Queues

Primary Actor: User

Preconditions: -

Main Success Scenario:

1) The user provides a valid input data as text;
2) The user provides an output file;
3) The application simulates queues with specified inputs as parameters and at every second should output information about waiting clients and queues;
4) At the end the application should output the average waiting time for clients.

Alternative sequences:

- Invalid input/output file: the application exits without doing anything.

3.Design of the application

The project is structured in 3 packages, client, simulation and strategy. The client package contains classes related to the elements of the queues, in this case, Client, and also provides a Client generator capable to generate multiple clients with random attributes by specifying the range of those attributes. Knowing that the application runs on multiple threads, the safety of the client attributes is ensured by using the atomic classes, especially AtomicInteger. The class also provides a waiting time attribute used in case the simulation ends without being

processed all the clients. The desired average waiting time should not include those clients. The Client Generator class offer methods for generating a single Client or multiple Clients as said before.

The strategy package implements the selection policy module of the application. Even if for this application the scheduler should choose the server with the smallest waiting time we could change that to choose the server with the smallest number of clients. We could add any mode of selection the server thanks to Strategy pattern. More specific, the package contains an interface, Strategy, with the definition of the method addClient. The classes ConcreteStrategyQueue and ConcreteStrategyTime implements this class and assign the desired implementation of the addClient method.

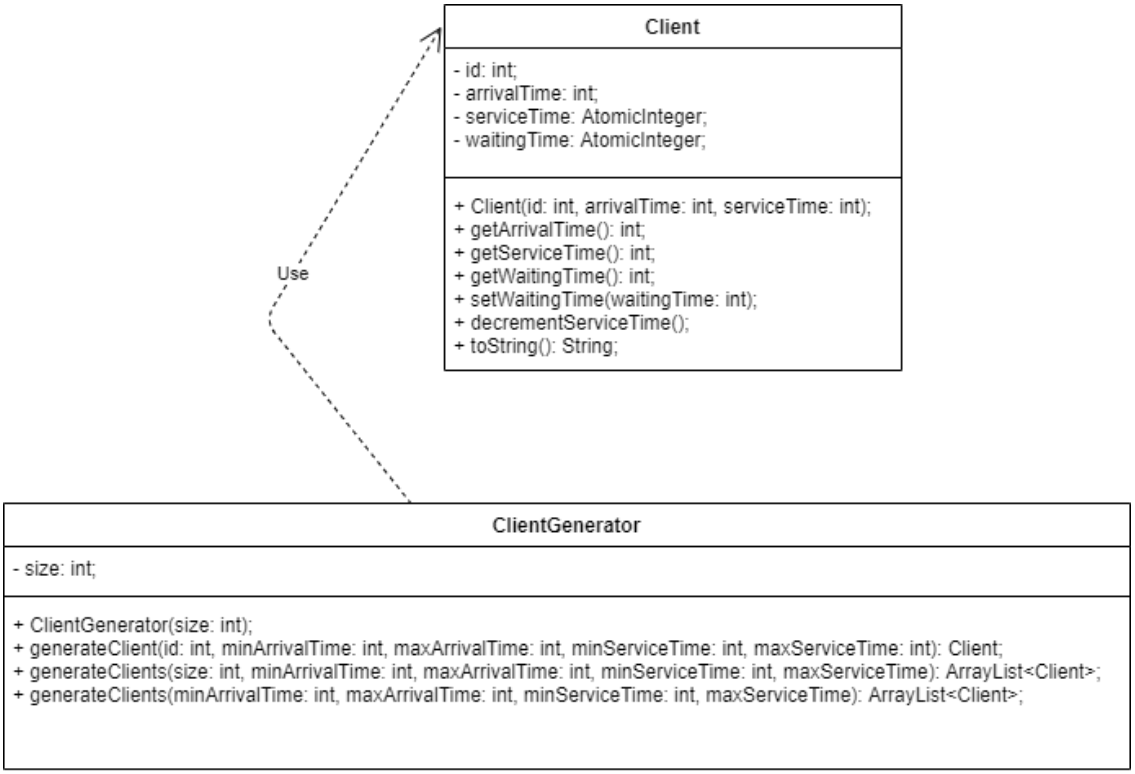
In the simulation package resides the Scheduler, Server and Simulation Manager, the classes which provides the dynamic of the application. They use the previous specified models for doing the actual work. Firstly, the Scheduler contains the list of the servers and strategy used. The methods which this class implements provides the communication with the servers. The Server class represents the Queue and runs in parallel with the Simulation Manager having the job to process the provided clients one at a time and also to open or close dynamically in function of the available clients. The Simulation Manager is a runnable too and have the job to generate the clients, maintain the time simulation and also to print to the output file.

The data structures used are the following:

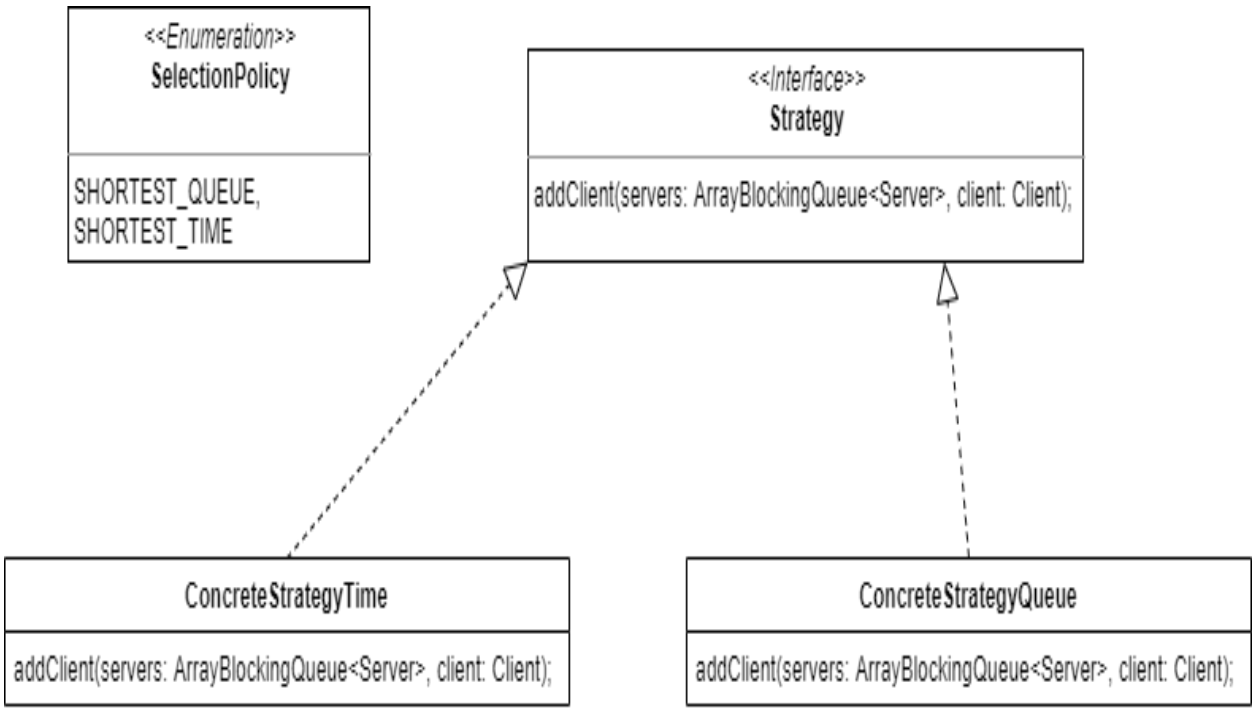
- ArrayList, used for generating the clients, this list doesn't need to be synchronized;
- ArrayBlockingQueue, used for servers, the list is synchronized since there exist a possibility to be accessed on different methods like printing or dispatching, etc. which could lead to unpredictable behavior;
- LinkedBlockingQueue, used in Server and Simulation Manager for maintaining the clients.

The UML diagram for every package is presented below:

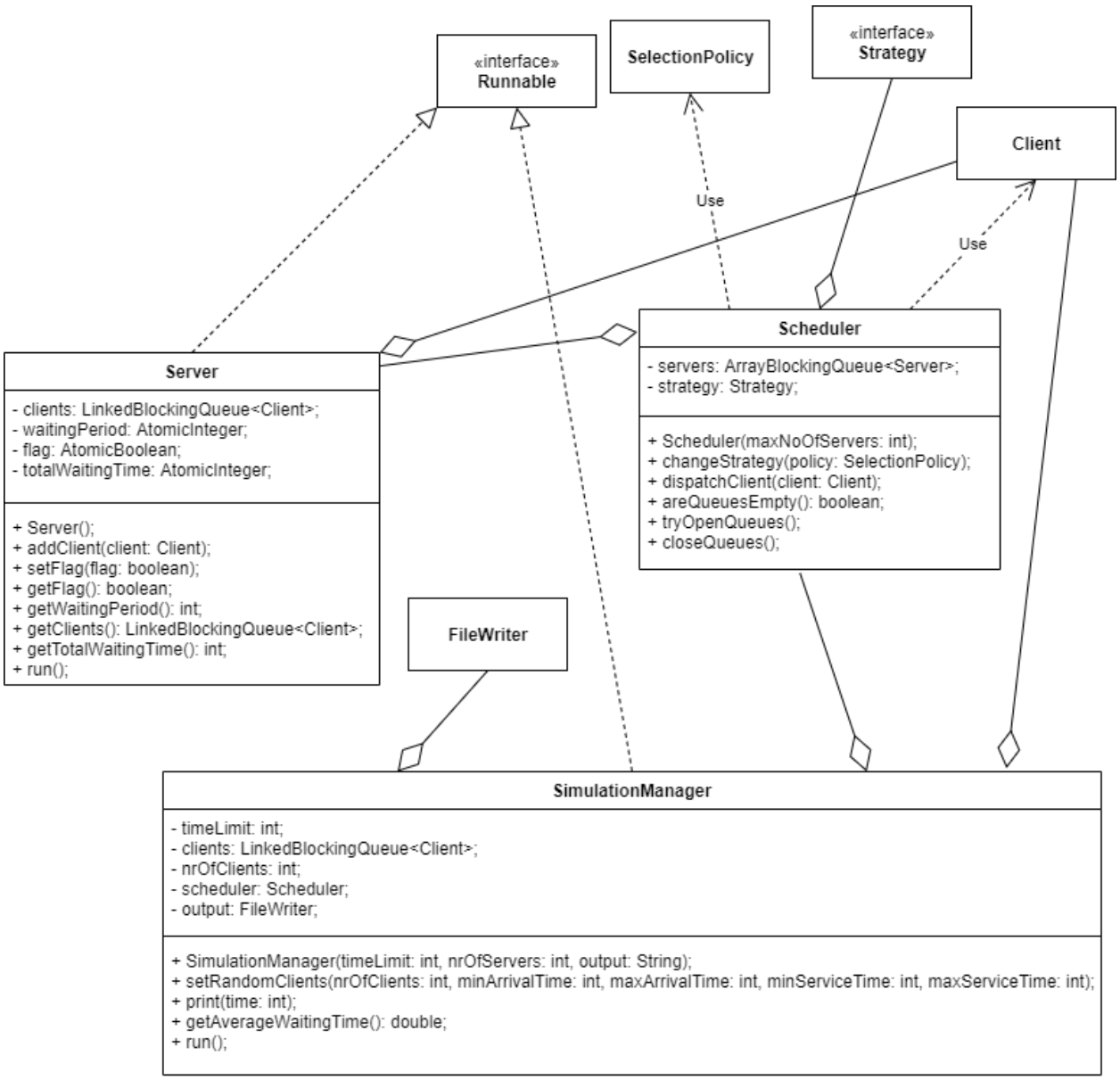
client package:



strategy package:



simulation package:



4.Implementation

Client:

Client: the client is described by the tuple (ID, arrival time, service time) also the client presents the field "waiting time" which represents the duration of time in which the client is in queue. This fields are private and presents thread safety (service time and waiting time) since they are used by another sources.

The constructor will check the provided parameters (id, arrival time, service time) and will throw IllegalArgumentException if something doesn't look good. If there are no problems, the constructor will initialize the fields.

The method decrementServiceTime have the job to process the client in the Server class, decrementing the service time.

The class overrides the toString method for printing purpose such that it will print the represented tuple (ID, arrival time, waiting time).

ClientGenerator:

This class have the purpose to generate random clients. The only field is a private integer, size. This have the role to reduce the number of parameters of method generateClients, in case it is provided at the creation of the object.

The method generateClient will generate a client with random fields (arrival time and service time) in function of the range provided in parameters but we should provide the id.

The method generateClients will generate an Array List of clients using the method generateClient. The user must provide a size.

Strategy

Is a interface with definition of the method addClient. The method takes as parameters the Array Blocking Queue which contains the servers and the Client to be inserted.

SelectionPolicy

Represents the enumeration which contains the ways in which the scheduler could place the clients in servers. At this moment it contains two constants: SHORTEST_QUEUE and SHORTEST_TIME. SHORTEST_QUEUE policy should consider the queue with the smallest number of clients and SHORTEST_TIME should consider the queue with the shortest waiting time. This assignment make use of the SHORTEST_TIME policy.

ConcreteStrategyQueue / ConcreteStrategyTime

Represents the implementations of the Strategy interface, they don't have fields or another method.

Server

The server class have the following fields:

- clients: the clients to be processed by this server, is thread safe and don't have a getter, only a method for adding to the list a client.
- waitingPeriod: this field is used for calculating the average waiting time and it is also used for SHORTEST_TIME policy. Basically, it represents the current waiting time, without its service time, a client could have if it is inserted in this moment to the queue.
- flag: represents the door for opening and closing the queue (kill the running thread). This field ensures the ability of the queue to open when there is a client to be processed and closing when there are no more clients to be processed. It is thread safe (AtomicBoolean).
- totalWaitingTime: the field returns the total waiting time of all clients processed by the queue, used in calculation of average waiting time.

The constructor initializes the fields, initially the flag is set to false and waiting period and total waiting time are set to 0.

The addClient method is used by the scheduler to add clients to the queue. Also the total waiting time and waiting period are updated.

The run method which is defined by Runnable interface have the job to process the clients. The sequence of instruction looks like this:

1. While the flag is set do:
    a. Check if the list of clients is empty, if it is break;
    b. Obtain the head of the client list;
    c. If service time of the client is 1 eliminate the client from the list
    d. Decrement the service time of the client
    e. Decrement the waiting time of the queue
    f. Sleep for 1 second
2. Set the flag to false.

The getTotalWaitingTime method traverse the entire list of clients, these clients are unprocessed and the total waiting time must not include them. This method returns the actual waiting time of this queue.

SimulationManager

This class presents the following fields:

- time limit: private final integer which represents the duration time of the simulation.
- clients: the list of clients which are generated by a client generator. It is thread safe.
- number of clients: the integer representing the number of clients, it is provided by the input file.
- scheduler: the scheduler of the simulation.
- output: a FileWriter used for printing the output of the simulation in a file whose path is provided by input.

The constructor initializes the time limit, number of clients, scheduler and output but not clients. The number of parameters for generating the clients is quite big so we do this step using another method.

The setRandomClients method generates the clients and assign them to clients list.

The print method has the job to print the current situation of the clients and queues in the output file.

The method getAverageWaitingTime sums up all the total waiting time from the servers and divide this result to the number of clients obtaining the desired average waiting time. This will be printed at the end of the simulation.

The run method is implemented such that this class implements the Runnable class. Next it is described the behavior of this method:

1. Initialize the simulation time to 0.
2. While simulation time is smaller or equal than time limit do:
    a. Check if the queues are empty.
    b. If waiting clients and queues are empty the simulation is over, break.
    c. Dispatch every client which have the arrival time equal with simulation time to a server.
    d. Remove those clients from waiting list.
    e. Print the current situation of the queues and waiting clients.
    f. Try to open the queues if there exist clients in server lists.
    g. Increment the simulation time.
    h. Sleep for 1 second.
3. Close the queues.
4. Write to the output file the average waiting time and close the file writer.

In the main method it is scanned the input file for obtaining the parameters of the simulation. The scanner of that file is closed and the obtained parameters are given to simulation manager and the thread is started.

5.Results

After the implementation of this application there were made 3 tests with following configurations:

Test1:

4 clients

2 queues

60 seconds (simulation limit)

2,30 (minimum arrival time, maximum arrival time)

2,4 (minimum service time, maximum service time)

In the output file we could observe the following details:

The clients generated in this test are: (1, 8, 3); (2, 22, 3); (3, 29, 3); (4, 16, 2);

The queues are closed until Time 8, when the first client enters the first queue.

All the clients are processed correctly and the simulation ends at Time 31.

Since in this simulation all clients are directly assigned to the queues as heads in server lists, they don't need to wait for another client and the average waiting time is determined only by their service time.

The average waiting time: 2.75 ((3 + 3 + 3 + 2) / 4)

Test2:

50 clients

5 queues

60 seconds

2,40 (minimum arrival time, maximum arrival time)

1,7 (minimum service time, maximum service time)

Details:

Generated clients: (1, 37, 3); (2, 11, 3); (3, 2, 6); (4, 34, 6); (5, 19, 3) + 45 clients.

The queues open at Time 2.

All clients are processed and the simulation is over at Time 51

The average waiting time is 5.78

Test3:

1000 clients

20 queues

200 seconds

10,100 (minimum arrival time, maximum arrival time)

3,9 (minimum service time, maximum service time)

Details:

Generated clients: (1, 63, 6); (2, 41, 5); (3, 56, 8); (4, 89, 9); (5, 58, 5) + 995 clients.

The queues open at Time 10.

Not all clients are being processed.

The average waiting time is 41.853

Since the tests seems legit, it is generated the jar file and another 2 tests are given. The jar file is tested from the console and it seems to work.

6.Conclusions

The assignment provides an introduction to the work with parallel programming. From this homework I learned to work with threads, to understand the concept of synchronization, to use synchronized data structures, to work with Strategy pattern which seems to be pretty useful in many applications.

The application could be improved by providing a graphical user interface, more strategies for dispatching the clients. The application could connect with a database with information about clients, in that case there are more use cases which the application could provide.

7.Bibliography

http://tutorials.jenkov.com/java-concurrency/creating-and-starting-threads.html