MINISTRY OF EDUCATION AND RESEARCH

TECHNICAL UNIVERSITY
OF CLUJ-NAPOCA, ROMANIA

FUNDAMENTAL PROGRAMMING TECHNIQUES

ASSIGNMENT 2

QUEUES SIMULATOR

1. Requirements

Design and implement a simulation application aiming to analyse queuing based systems for determining and minimizing clients' waiting time.

Queues are commonly used to model real world domains. The main objective of a queue is to provide a place for a "client" to wait before receiving a "service". The management of queue-based system is interested in minimizing the time amount their "clients" are waiting in queues before they are served. One way to minimize the waiting time is to add more servers, i.e. more queues in the system (each queue is considered as having an associated processor) but this approach increases the costs of the service supplier. When a new server is added the waiting customers will be evenly distributed to all current available queues.

The application should simulate (by defining a simulation time $t_{simulation}$) 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), $t_{arrival}$ (simulation time when they are ready to go to the queue; i.e. time when the client finished shopping) and $t_{service}$ (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 $t_{arrival}$ time is greater than or equal to the simulation time ($t_{arrival} \geq t_{simulation}$).

The following data should be considered as **input data read from a text file** for the application:

- Number of clients (N);
- Number of queues (Q);
- Simulation interval ($t_{simulation}^{MAX}$);
- Minimum and maximum arrival time ($t_{arrival}^{MIN} \leq t_{arrival} \leq t_{arrival}^{MAX}$);
- Minimum and maximum service time ($t_{service}^{MIN} \leq t_{service} \leq t_{service}^{MAX}$);

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, as shown in the following example.

In-Test.txt	Explanation
4 2 60 2,30 2,4	The application reads from In-Test.txt the simulation parameters: N=4 clients, Q = 2 queues, $t_{simulation}^{MAX} = 60$, a 60 second simulation interval. It also reads the bounds for the client parameters, respectively a minimum and maximum arrival time 2, 30, meaning that clients will go to the queues from second 2 up to second 30. Furthermore, the bounds for the service time are read from the final line, 2 and 4, meaning that a client has a minimum time to wait in front of the queue 2 seconds and a maximum time of 4 seconds. Using these parameters, a set of 4 clients are generated random, each client i being defined by the following tuple: (ID^i, $t_{arrival}^i$, $t_{service}^i$), with the following constraints:

	• $1 \leq ID^i \leq N$ • $t_{arrival}^{MIN} \leq t_{arrival}^{i} \leq t_{arrival}^{MAX}$ • $t_{service}^{MIN} \leq t_{service}^{i} \leq t_{service}^{MAX}$ A number of Q threads will be launched to process in parallel the clients. Another thread will be launched to hold the simulation time $t_{simulation}$ and distribute each client i to the queue with the smallest waiting time when $t_{arrival}^{i} \geq t_{simulation}$

The output of the simulation is a text file containing the status of the pool of waiting clients and the queues as the simulation time $t_{simulation}$ goes from 0 to $t_{simulation}^{MAX}$.

Out-Test.txt	Explanation
Time 0 Waiting clients: (1,2,2); (2,3,3); (3,4,3); (4,10,2) Queue 1: closed Queue 2: closed	At time $t_{simulation} = 0$, a number of 4 clients are generated. Client with ID = 1 has an arrival time equal to 2, meaning that it will be ready to go to a queue when $t_{simulation} \geq 2$. Furthermore, it has a service time equal to 2, meaning that is needs to stay 2 timesteps in the front of the queue. The same rules apply for the next 3 clients. The two queues are closed since there are not clients available.
Time 1 Waiting clients: (1,2,2); (2,3,3); (3,4,3); (4,10,2) Queue 1: closed Queue 2: closed	At time $t_{simulation} = 1$, none of the clients can be sent to the queues because none of them has the arrival time greater or equal to 2. The two queues are closed since there are not clients available.
Time 2 Waiting clients: (2,3,3); (3,4,3); (4,10,2) Queue 1: (1,2,2); Queue 2: closed	Queue 1 is opened and client with ID =1 is sent to the first queue since $t_{arrival}^{1} \geq t_{simulation} = 2$. Other clients are still waiting. Queue 2 is closed.
Time 3 Waiting clients: (3,4,3); (4,10,2) Queue 1: (1,2,1); Queue 2: (2,3,3);	Queue 2 is opened time $t_{simulation} = 3$, client with ID = 2 is sent to it since $t_{arrival}^{2} \geq t_{simulation} = 3$, and the waiting time at the second queue (0) is smaller than the waiting time at the first queue (1), where a client is still processed. The client from queue 1 has its service time decreased to 1 (coloured in yellow) because it is being processed. Other clients are still waiting.
Time 4 Waiting clients: (4,10,2) Queue 1: (3,4,3); Queue 2: (2,3,2);	At time $t_{simulation} = 4$, client with ID = 3 is sent to the first queue since $t_{arrival}^{3} \geq t_{simulation} = 4$. Furthermore, client with ID =1 was eliminated from the queue because its service time has dropped to 0 (it was 1 at the previous iteration and was decreased with one at the simulation step) The client from queue 2 has its service time decreased to 2 (coloured in yellow) because it is being processed.

	The final client is still waiting.
…	
Average waiting time: 2.5	The simulation is finished when there are no more clients in the waiting queue or at the service queues or $t_{simulation} > t_{simulation}^{MAX}$ The average waiting time is computed and appended to the file.

An example of input/output files is the following:

In-Test.txt

```
4
2
60
2,30
2,4
```

Out-Test.txt

```
Time 0
Waiting clients: (1,2,2); (2,3,3); (3,4,3); (4,10,2)
Queue 1: closed
Queue 2: closed

Time 1
Waiting clients: (1,2,2); (2,3,3); (3,4,3); (4,10,2)
Queue 1: closed
Queue 2: closed

Time 2
Waiting clients: (2,3,3); (3,4,3); (4,10,2)
Queue 1: (1,2,2);
Queue 2: closed

Time 3
Waiting clients: (3,4,3); (4,10,2)
Queue 1: (1,2,1);
Queue 2: (2,3,3);

Time 4
Waiting clients: (4,10,2)
Queue 1: (3,4,3);
Queue 2: (2,3,2);

…
Average waiting time: 2.5
```

1.2 Design considerations

- Use an object-oriented programming design

1.3 Implementation considerations

- Use the Java programming language
- Implement classes with maximum 300 lines
- Implement methods with maximum 30 lines
- Use the Java naming conventions (https://google.github.io/styleguide/javaguide.html)

2. Deliverables

- A **solution description document** (minimum 2000 words, Times New Roman, 10pt, Single Spacing) with the structure specified in the **Lab Description** document.
- **Source files** – will be uploaded on the personal ***gitlab*** account created at the **Lab Resources Gitlab** laboratory work, following the steps:
  - Create a repository on ***gitlab*** named according to the following template *PT2020_Group_FirstName_LastName_Assignment_2*
  - Push the source code and the documentation (**push the code not an archive with the code**)
  - A *PT2020_Group_FirstName_LastName_Assignment_2.jar* file containing the project configured to run from the terminal and read 2 parameters: the name of the input file and the name of the output file. The application should permit to be run with the following command:
    java -jar PT2020_Group_FirstName_LastName_Assignment_2.jar in.txt out.txt
- Three test files (in-test-1.txt, in-test-2.txt, in-test-3.txt) and their corresponding outputs (out-test-1.txt, out-test-2.txt, out-test-3.txt) for the following configurations:

in-test-1.txt	in-test-2.txt	in-test-3.txt
4 2 60 2,30 2,4	50 5 60 2,40 1,7	1000 20 200 10,100 3,9

- Share the repository with the user **utcn_dsrl.**

3. Evaluation

The assignment will be graded as follows:

Requirement	Grading
Minimum to pass • 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. • Documentation	5 points
Other two tests given in requirements (in-test-2.txt, in-test-3.txt)	1 point
Quality of documentation	2 points
Compute average waiting time	1 point
Other tests run when evaluating homework using the compiled code as .jar file uploaded and configured to run according to deliverable requirement	1 point

4. Bibliography

- http://docs.oracle.com/javase/tutorial/essential/concurrency/index.html
- http://www.tutorialspoint.com/java/util/timer_schedule_period.htm
- http://www.javacodegeeks.com/2013/01/java-thread-pool-example-using-executors-and-threadpoolexecutor.html