**TECHNICAL UNIVERSITY OF CLUJ-NAPOCA**

**Laboratory Work:**

**Simulation**

**Name: Nistor Rares Constantin**

**Group: 30425**

**Content:**

1. **Introduction**

**1.1 Homework objective . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .page 3**

1. **Problem solving. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .page 3**
2. **Projection**

**3.1 UML diagram . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .page 4**

**3.2 Data structures . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . page 5**

**3.3 Class projection . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .page 6**

**3.4 User interface. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .page 7**

**3.5 Packages . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . page 8**

**3.6 Algorithms . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . page 8**

1. **Implementation and testing. . . . . . . . . . . . . . . . . . . . . . . . .page 9**
2. **Results . . . . . . .. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .page 10**
3. **Conclusions. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . page 10**
4. **Bibliography. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . page 11**
5. **Introduction**
   1. **Homework objective**

The objective of this homework is the following:”Design and implement a simulation application aiming to analyze queuing based systems for determining and minimizing clients waiting time”. In other words we must simulate how a queue, at a store for example, works and to extract useful information from the specific simulation.

1. **Problem solving**

This homework was a pretty difficult one. To successfully solve the problem, or part of the problem, first I started to gather information about threads, what are threads, what are they useful for, disadvantages, advantages and how to use them in java. This was a little harder than I expected. After learning a few things about threads I started to search how to correctly implement them in java, how to synchronize them with what kind of data structures I can use them. In concurrent programming, there are two basic units of execution: *processes* and *threads*. In the Java programming language, concurrent programming is mostly concerned with threads. However, processes are also important.

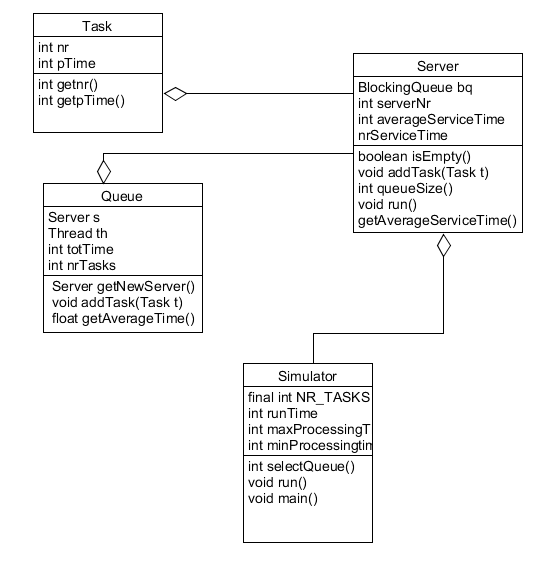
Multithreaded execution is an essential feature of the Java platform. Every application has at least one thread — or several, if you count "system" threads that do things like memory management and signal handling. But from the application programmer's point of view, you start with just one thread, called the *main thread*. This thread has the ability to create additional threads, as we'll demonstrate in the next section.

After sketching a class diagram I started to create classes, implement methods and slowly writing the code.

1. **Projection**

**3.1 UML diagram**

Class Diagram :



Q q q q q q q q q q q q q q q q q q q q q q q q q q q q q q q q

Q q q q q q q q q q q q q q q q q q q q q q

Q q q q q q q q q q q q q q q q q q q q q q q q q q q

Q q q q q q q q q q q q q q q q q q q q q q q q q q q q q q q

Q q q q q q q q q q q q q q q q q q q q q q q q q q q q q q q q

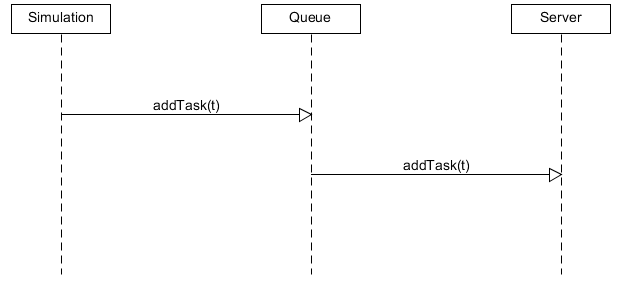
Q q q q q q q q q q q q q q q q q q q q q q q q q q q q q q q

Q q q q q q q q q q q q q q q q q q q q q q q q q q q q q q q

Q q q q q q q q q q q q q q q q q q q q q q q q q q q q q q

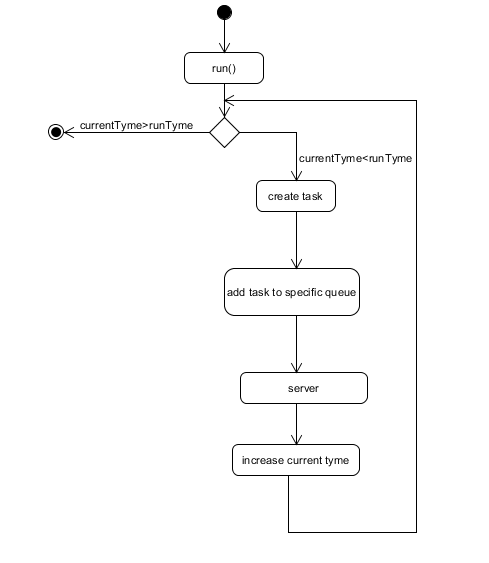
The class diagram is essential for every java project. This diagram is usually done at the beginning of the project so that you can know how to structure your code. This diagram is important because it shows you how many and what classes you have what type of variable you should use how your methods should look like and what king of parameters they should have. It also shows you what king of connection is between the classes.

Sequence diagram



This sequence diagram show how a new task is added to the server, first the simulation creates the task then it adds it to the Queue and when the Server is empty the queue adds it to the server.

Activity diagram



The activity diagram shows how the program works and what steps it take until it finishes.

**3.2 Data structures**

In this homework I use variables of type Integer, Float and structures of type BlockingQueue and vectors. The most important structure here is the BlockingQueue. It was an essential structure in this project because it is strongly related to the Threads.

A BlockingQueue is typically used to have on thread produce objects, which another thread consumes. Here is a diagram that illustrates this principle:

|  |
| --- |
| A BlockingQueue with one thread putting into it, and another thread taking from it. |
| **A BlockingQueue with one thread putting into it, and another thread taking from it.** |

The producing thread will keep producing new objects and insert them into the queue, until the queue reaches some upper bound on what it can contain. It's limit, in other words. If the blocking queue reaches its upper limit, the producing thread is blocked while trying to insert the new object. It remains blocked until a consuming thread takes an object out of the queue.

The consuming thread keeps taking objects out of the blocking queue, and processes them. If the consuming thread tries to take an object out of an empty queue, the consuming thread is blocked until a producing thread puts an object into the queue.

**3.3 Class projections**

Now we will talk about how the program is structured. There are 4 classes. Tasks, Server, Queues and Simulator.

Tasks :

The Task class represents, for example a customer at a shop. It has 2 variables nr which is the task number and is unique for each task and the pTime which is the time that it takes for the Task to be processed.

As methods the class has a getter for nr, getnr(), and a getter for pTime, pTime().

Server:

The server is like the cash desk of a shop. It has 4 variables serverNr, averageServiceTime, nrServiceTime and bq which is a BlockingQueue. This class implements the Runnable interface which means that is must implement the method run. In the method run is the body of the thread in this thread we try to simulate what happens when the customer reaches the cash desk and display the information on the screen. Except this method it also has the methods: addTask(), queueSize() and getAverageServiceTime(). They are pretty simple fuction.

Queue

This class represents the queue at a shop in this class new tasks, customers, are added in the BlockingQueue, queue. As variables it hats totTime and nrTasks alos a variable s of type server and a variable th of type thread. The getServer method returns the server to which it leads addTask and a task to the BlockingQueue, and getAverageTime returns the average time spent at the queue.

Simulator:

This class is responsible for generating the Tasks and adding them to the right queue. This class has the following variables: an integer NR\_SERVERS which represents the number of servers that are created, runtime which represents the amount of thyme that the application will simulate the shop, maxProcessingTime represents the maximum amount of time a task can stay in the Server before being eliminated and minProcesstime represents the minimum amount of time a task can stay in the Server before being eliminated. It also has the following methods selectQueue which selects the queue in which the next task should go the main method used to test the homework. This class also extends Runnable so it implements the run method. In these method elements tasks are given to the Queue class to put them in right BlockingQueue.

**3.4 User interface**

Unfortunately because this homework was pretty challenging , or at least it was for me, I was unable to generate a graphic user interface, the user doesn’t need to introduce any values, they are hardcoded, he just needs to watch the simulation.

**3.5 Packages**

This project has only one package the default one, since the project has only 4 classes I didn’t consider it necessary or useful to divide the project in more packages*.*

**3.6 Algorithms**

We will now talk about some of the algorithms that were used in the methods. The large majorities of the algorithms are pretty easy or just return a value. The main algorithms that we will talk about are the ones in the run methods. But first let’s talk about the run method itself and the Runnable Interface.

The Runnable interface should be implemented by any class whose instances are intended to be executed by a thread. The class must define a method of no arguments called run.

This interface is designed to provide a common protocol for objects that wish to execute code while they are active. For example, Runnable is implemented by class Thread. Being active simply means that a thread has been started and has not yet been stopped.

In addition, Runnable provides the means for a class to be active while not subclassing Thread. A class that implements Runnable can run without subclassing Thread by instantiating a Thread instance and passing itself in as the target. In most cases, the Runnable interface should be used if you are only planning to override the run() method and no other Thread methods. This is important because classes should not be subclassed unless the programmer intends on modifying or enhancing the fundamental behavior of the class.

When an object implementing interface Runnable is used to create a thread, starting the thread causes the object's run method to be called in that separately executing thread. The general contract of the method run is that it may take any action whatsoever.

And now let’s talk about the algorithm that I use in the run method. In the class Server I save in the task t the first element put in the BlockingQueue, then I print that the task t and the server that is processing it. After I print the output I use sleep to make the thread sleep 100 times the processing time of t to simulate that the server is busy that time. After the sleep is over I increase thenr of tasks that were processed and I print that the task was processed.

In the simulator class, in the rum method I initialize 2 variables currentTime which is at the beginning 0 and sel. I use the Math.random() method to generate a random number between maxProcessingTime and minProcessingTime, after I create the task t which has as number the currentTime and processtime the nr early generated. Using selectQue I select the queue in which to put the task, the task will be put in the first empty queue or, if there is no empty queue in the queue with the fewer tasks, and after is use addTask to introduce the task in the queue. Next I use Thread.sleep to simulate that a second has passed and the next client is entering and I increase the currentTime. At the end I print the average waiting time for each queue and average service time for each server and I use System.exit(0). I use this method to terminate the program. It is necessary because it is almost certain that after runtime has passed there are still threads in the queue and I don’t want them to continue to enter the servers.

Also the algorithm Read(), allows the user to introduce from the keyboard the minimum processing time for a task, the maximum processing time and the amount of time that the program will simulate a shop.

1. **Implementation and Testing**

Although the implementation is not without its flows I tried to implement the algorithms as simple as possible and to respect as much as possible the OOP principles. The most interesting and challenging part in this project was to use the Runnable interface and the run function, and getting to know a little more about threads and synchronization. The testing part consisted in running the algorithm a few time and observing if there are duplicates values or if the average value are well calculated. Although I know that testing shows the presence, not the absence of bugs ( Dijkstra ), and the value are randomly generate there were no bugs so far and I hope there won’t be in the future either

1. **Results**

The result of this homework is a user friendly application that simulates a Shop with customers and cash desks that is aiming to analyze queuing based systems for determining and minimizing clients waiting time. I don’t think there will be any problems understanding the output of this application.

1. **Conclusions**

In conclusion the homework helped me to better understand the notions of threads, synchronization and allowed me to work with interesting interfaces like Runnable. I hope that this homework is a decent one because I know it’s far from perfect, or even good.

For example here are some change and improvements that I would make:

* Add a graphic user interface because it is much easier to understand what is happening in a user interface when you can model the appearance of the output as you like than to the way that it is printed in the console.
* The next thing that I would modify Is to add arriving times to the task, so that the tasks arrive at a certain or random tine not in every second
* And the last think that I would modify is to let the user introduce from the keyboard the number of task and server that he wants to simulate.

1. **Bibliography**

For this homework I used different sites to help me understand concepts that I did not yet understood or to help me create different diagrams.

* <http://stackoverflow.com/>
* <https://www.youtube.com/>
* <http://docs.oracle.com/javase/tutorial/essential/concurrency/index.html>
* <http://www.javacodegeeks.com/2013/01/java-thread-pool-example-using-executors-and-threadpoolexecutor.html>