*Homework 2:*

*Queues Simulator*

***Programing Techniques***

* *Rus Rares Tudor*
* *Group 30422*

*Contents:*

1. *Requirements . . . . . . . . . . .. . . . . . .. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . ..3*
2. *Problem Analysis. . . . . . . . . . .. . . . . . .. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .3*
3. *Use Cases . . . . . . . . . . .. . . . . . .. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4*
4. *Projection . . . . . . . . . . .. . . . . . .. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .5*
5. *Implementation . . . . . . . . . . .. . . . . . .. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6*
6. *Results . . . . . . . . . . .. . . . . . .. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8*
7. *Conclusions . . . . . . . . . . .. . . . . . .. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9*
8. *Bibliography . . . . . . . . . . .. . . . . . .. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .. . 9*

*--Requirements :*

The main objective of this assignment is to design and implement a system for simulate a queuing based system for determining a task’s waiting time and also to minimalize it, this application should also calculate the average time for waiting time spent at the queues. To achieve such a goal, we have to implement a thread based algorithm and a handler to start each of the task’s thread and for the servers known as queues also. In order to successfully fulfil this task we have to understand what a queue is, so here is the definition:

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 systems 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 a series of clients arriving for service, entering queues, waiting, being served and finally leaving the queue. It tracks the time the customers spend waiting in queues and outputs the average waiting time. To calculate waiting time we need to know the arrival time, finish time and service time. The arrival time and the service time depend on the individual clients – when they show up and how much service they need. The finish time depends on the number of queues, the number of clients in the queue and their service needs.

*--Problem Analysis:*

Abstraction in its main sense is a conceptual process where general rules and concepts are derived from the usage and classification of specific examples, literal signifiers, first principles, or other methods.

Conceptual abstractions may be formed by filtering the information content of a concept or an observable phenomenon, selecting only the aspects which are relevant for a particular subjectively valued purpose. For example, abstracting a leather soccer ball to the more general idea of a ball selects only the information on general ball attributes and behavior, excluding, but not eliminating, the other phenomenal and cognitive characteristics of that particular ball. In a type–token distinction, a type (e.g., a 'ball') is more abstract than its tokens.

In order to design an application which fulfills the above mentioned description we have to understand the fact that we can not just work with the usual synchronous flow of the program we are used to. We have to make use of the well known thread concept because we have to do more things at the same time, in parallel. For example we can not just trace the working of only one queue and then move one to the next queue and so on. This approach would be fatal for our program. The correct approach would be to trace the execution of all the working queues simultaneously and in order to do this we make use of the threads.

So what are threads?

Def: thread = the smallest sequence of programmed instructions that can be managed independently by a scheduler.

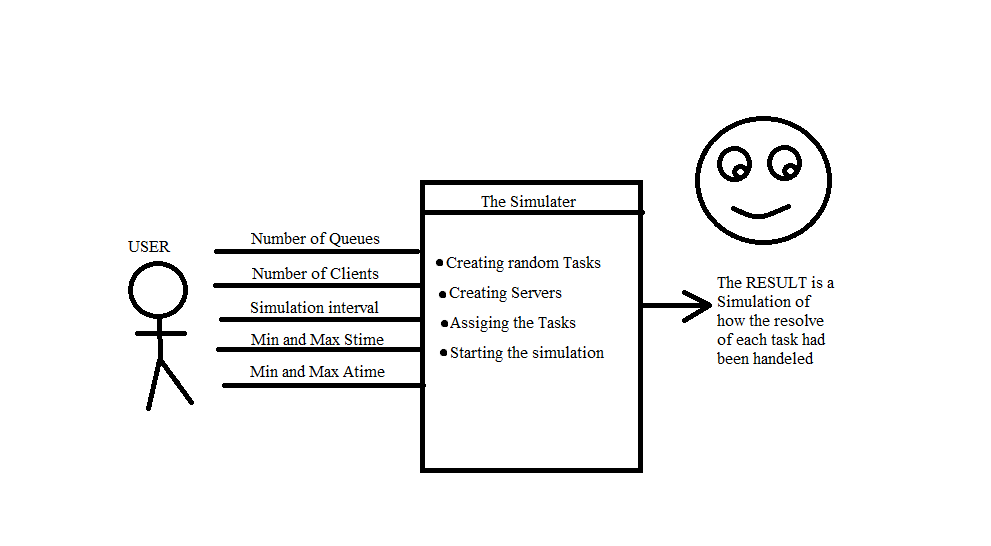
So in other words a thread is a set of one after the other written instructions which execution time runs in parallel with the execution time of the main program (because even if we don’t use threads in a specific program there will always exist the main thread known as the line of code executed in the main method of the program) or with the execution time of other threads.

If we understand this simple concept then the more complex thread operations, which will occur in the design of the program, will be easily understand.

As presented above, a queue is nothing more than an array of specified objects, in our case this objects are tasks and each task has three parameters and those are: an ID which is an integer an arrival time that is also an integer, this parameter determines how much time an task is waiting until entering a queue, and the last parameter is service time which tell to the handler how long the task is remaining in the queue.

.

*--Use Cases:*



A user is required to start the application by entering five inputs and those are: Number of queues, Number of clients, the simulation time, and two intervals from where the application will generates random integer numbers for the parameters of the Task. When the user runs the application the output should appear.

Perform Addition/ Subtraction/ Multiplication/ Division of two polynomials:

- *User Main SUCCES scenario*:

-User launches application successfully.

-User provides relative inputs based on the task.

-User presses the running button so the application starts.

-The date introduced by the user are correctly processed.

-The running is performed successfully with no exceptions or errors.

-The result is displayed in a label provided by the application.

*- User Main FAILED scenario*:

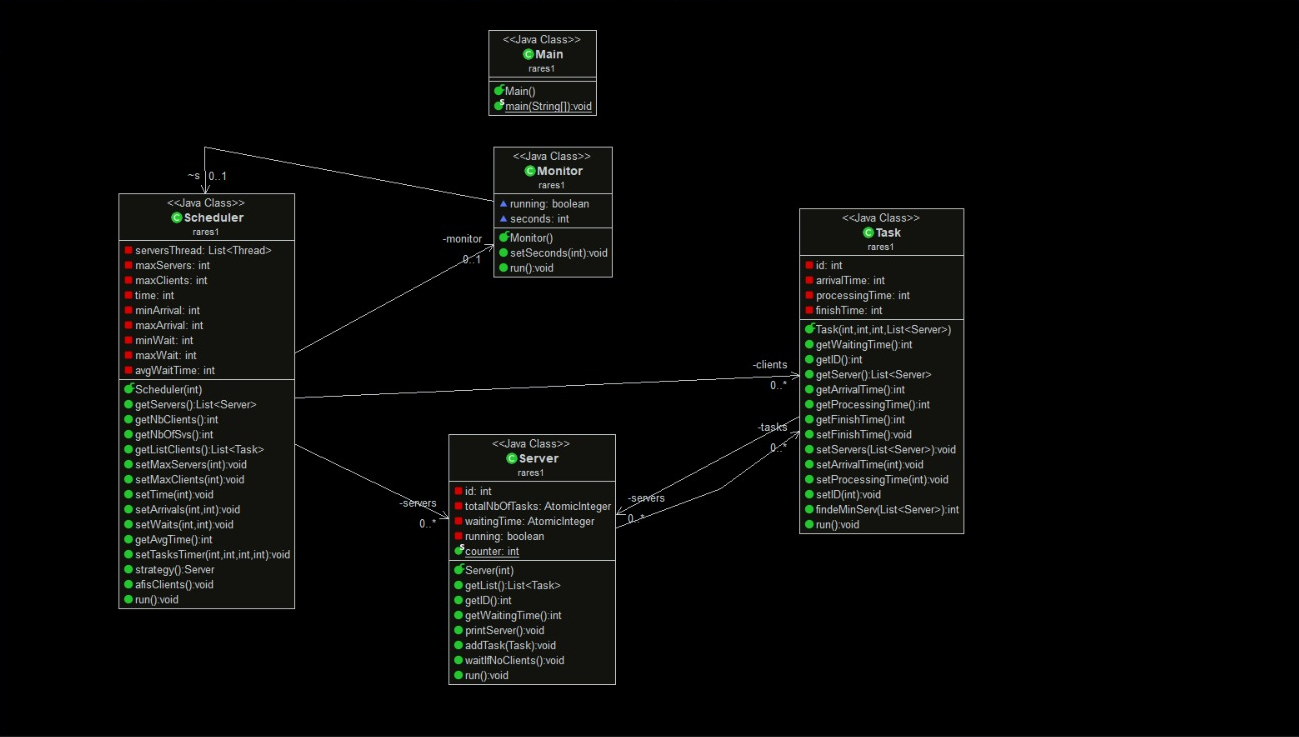
-User can’t launches application successfully.

-User provides wrong inputs for the requirements.

-The inputs are unsuccessfully transformed into the Task parameters.

-The result is not displayed in a label provided by the application.

*--Projection:*



*--Implementation:*

In the following, we will take a closer look at the role and the implementation of each class with its attributes, constructors and methods.

*- Task.java*

The attributes of this class are ID, which represents the number of the client; arrival Time, which represents the time at which the client enters the queue, service Time, representing the time the client has to wait at the service, waiting Time, representing the time the customer has to wait till he reaches hes turn at the service and the total time which is a sum of the last two. All of the attributes are declared as private for security purposes. They can be accessed from outside this class only by using special methods(getters and setters).

The constructor of this class takes as parameter two integers and sets them as the arrival time and as the service time of the client. So other said, when created, the client already knows its arrival time and the time he has to spend at the service.

*- Server.java*

This class has as parameters the following: id as integer, tasks as a list of tasks which represent the task in the queue, waitingTime as an atomic integer which represents the total time of waiting at this queue, running which keeps the tread running. This class also has a constructor, and settees and gutters for each parameter, also all the variables are declared privet in security motives. Here are also some methods that help us handle the requirement, and those are: getWaitingTime which returns the total time of the queue which calls the function, that means each task in the queue is traversed and it’s processing time is added to the returned variable of the method. Another method is printServer which travers all the queue and prints the elements of each task present in the queue, add Task is here for adding a task in the queue and also printing a message when this is happening, another method is waitIfNoClients whichi stops the thread until another task is added to the queue. The most important method here is the run one, because this method runs the three of the server, it is also synchronized with all the threes in the application and override it when is the case.

-*Scheduler.java*

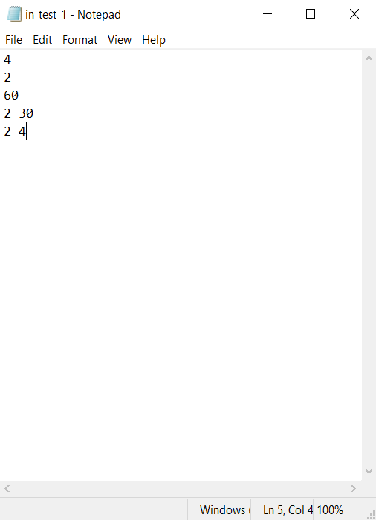
This class is the core of the application because its handling booth tasks and servers list and starts the threads for each of them. It has the following parameters the following: a list of servers and another of tasks and all the parameter given by the user as input: maxServers, maxClients, stime as the simulation time, and also 2 set of variables for the arrival time and waiting time tasks as a list of tasks which represent the task in the queue, waitingTime as an atomic integer which represents the total time of waiting at this queue, running which keeps the tread running. This class also has a constructor, and settees and gutters for each parameter, also all the variables are declared privet in security motives. In concert of methods are as following, one of the most important is strategy which returns the suitable server where the current task should be added. Another one is afisClients which prints the clients in the queue. The most important method here is the run one, because this method runs the three of the server, it is also synchronized with all the threes in the application and override it when is the case.

-*Monitor.java*

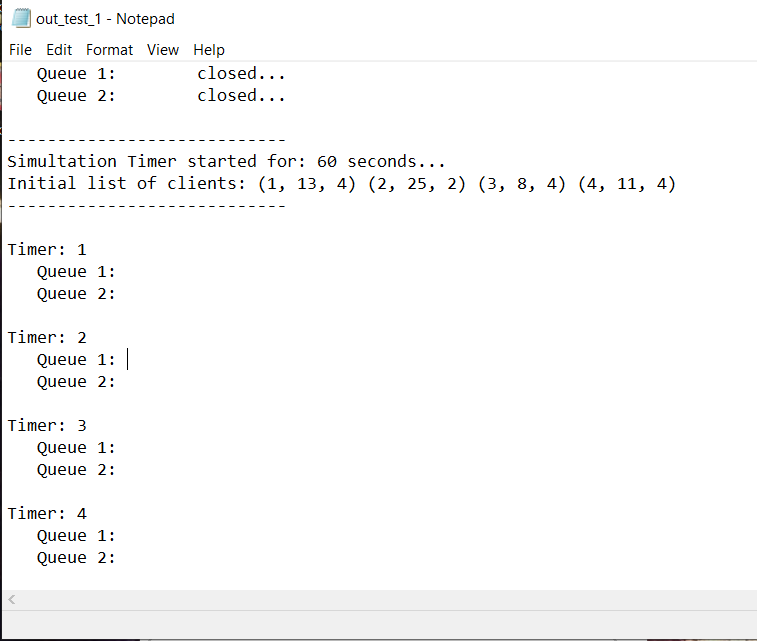
This is the head of the application and it starts the scheduler so all the threads start it also has some cool prints so the interface of the user may look a bit nicer. Here we have no parameters and just one constructor. All that’s happening in this class is in the run method.

*--Results:*

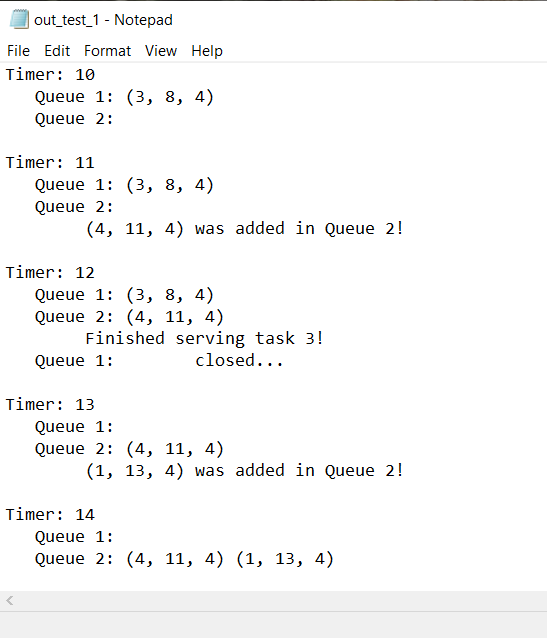
Here is a sneak peak of how the input and the output files should look, but of cores it may change from a user to another because the task are generate random and each of the user may give as input any values.



The input file:



The output file:

**

**

*--Conclusion:*

Java offers the real possibility that most programs can be written in a type-safe language. ... It extends Java with a mechanism for parametric polymorphism, which allows the definition and implementation of generic abstractions. The paper gives a complete design for the extended language

I personally, learned and better understood how to use classes and modifiers (which sets the class as public or private), modifiers are also used for variables and methods and also I have my first interaction with the threads and that’s awesome, but in my opinion are kind hard to understand and to work with them, but I had a lot of fun and I like it, because who doesn’t like an tuff chalange.

What’s concerning the further developments, I have some ideas to adjust the interface to make it more attractive because in our days this is what an user is looking for, he or she has no concerns about how the application works or why it takes so much memory. Regarding to possible updates of the application, I have in mind to implement an algorithm which finds the roots of the polynomial and returns a graphical schema and also to increase performance so the application can work faster.

*--Bibliography:*

- <https://www.geeksforgeeks.org/>

- <https://www.wikipedia.org/>

- <https://stackoverflow.com/>