***QUEUES MANAGEMENT APPLICATION USING THREADS AND SYNCHRONIZATION MECHANISM***

***Fundamental Programming Techniques***

***Student: Fuia Daria***

***Group: 30424***

***Specialization: CTI Eng, 2nd year***

***CONTENTS***

1. **Assignment Objective ………………………………………….**
2. **Problem Analysis, Modeling, Scenarios, Use Cases…………..**
3. **Design……………………………………………………………**
4. **Implementation…………………………………………………**
5. **Results…………………………………………………………..**
6. **Conclusions……………………………………………………...**
7. **Bibliography…………………………………………………….**

***1. Assignment Objective***

The main goal of this project is to design and implement a customer management system for those waiting in queues, based on the number of open queues, with a strategy aimed at ensuring that each customer spends as little time as possible in the queue..

In order to fulfill this, I needed to part it in more secondary objectives:

* Determining the attributes and behavior of the objects in the created classes (instance variables and methods)
* Implementing the methods of each class
* Reading data from the graphical interface, where the following will be specified: the number of clients, the number of queues, how long the queues will be open, the time interval in which a client can arrive at the queue through two minimum and maximum values, the time interval required to process a client's requests, also through minimum and maximum values, a random number will be chosen from these intervals
* Creating an output file that displays the evolution of the queues and clients throughout the program execution in order to test the 3 provided tests
* Creating the graphical interface, so the user would be able to use the application

***2. Problem Analysis, Modeling, Scenarios, Use Cases***

* ***Problem Analysis***

The queue data structure is a particular case of a list that operates on the principle of FIFO (first in - first out, the first item that is added is the first to be served).

The main operations that can be performed on such a data structure are:

* Traversal
* Adding to the end (add)
* Removing the first element (take)
* Processing the first element (element)
* ***Modeling***
* The concept of a thread defines the smallest processing unit that can be programmed for execution by the operating system. It is used in programming to optimize program execution by executing distinct pieces of code in parallel within the same process.
* ***Use Cases***

In order for the user to be able to use the calculator, a graphical interface was implemented to help him.

He could use it following the next steps:

The user has entered the necessary data for running the application in the following order:

Number of clients

Number of queues

Time for which the queues will be open

Minimum arrival time, maximum arrival time

Minimum processing time, maximum processing time

After entering the data, the user started the application.

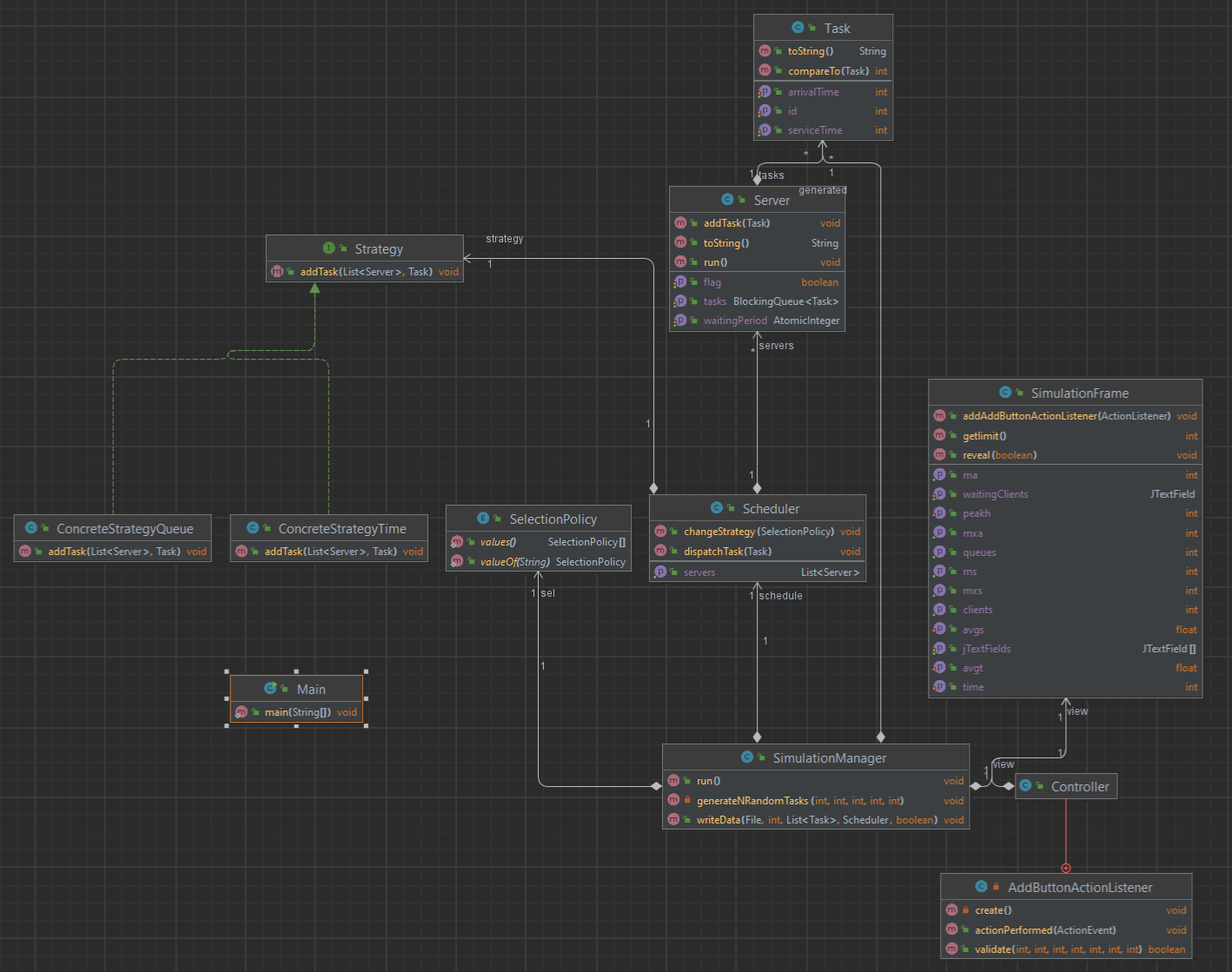
The program will generate the output, which will contain the time interval, randomly generated clients, and their distribution to queues. It will be possible to track exactly which queue the client X was assigned to, how long he waited in the queue, and how many clients he had in front of him.

***3. Design***

For implementing this project I chose to use the MVC architecture, where it is separated in 3 large packages:

* Models: here are the Task, Server and the Scheduler classes where the objects are contoured
* Views: here is implemented the graphical interface, where each JComponent is positioned somewhere in the frame
* Controllers: here is implemented the backend of the application with all the methods and also here are handled the actions of the user when he interacts with the GUI.
* If the problem statement does not mention the creation of Junit tests, then they can be skipped for this project.

*UML Diagram*

**

***4. Implementation***

I will describe each class and its methods separately:

* The Task class is the code representation of a real-world client and serves as the basis for the Server class, as a server will contain multiple tasks.

This class has 3 attributes: ID (int), arrivalTime (int), serviceTime (int).

The class implements the Comparable interface because tasks need to be sorted according to arrivalTime, and to use the sort method from Collections, I have overridden the compareTo<Task> method.

To be able to display each task according to the problem specifications, I have also overridden the toString method.

* The Server class has 2 attributes: waitingPeriod (the waiting period at a queue), which is an AtomicInteger, and a list of Tasks represented by the BlockingQueue type.

In addition to getter and setter methods, the class also contains the addTask(Task) method, which adds the task to the queue and increments the waiting time at the queue by the processing time of the task passed as a parameter. It also implements the Runnable and overrides the run() method.

* The Scheduler class can be considered as a representation of a manager from the real world.

The class has 3 attributes: servers (List<Server>) which is a list of servers, maxNoServers (int) which is the maximum number of servers, maxTasksPerServer (int) which represents the maximum number of tasks and a strategy.

In addition to the getter methods, the class also contains the dispatchTask(Task) method which is responsible for choosing in which queue the task received as a parameter should be assigned based on the waiting time at each of the n servers.

* The SimulationManager contains the overridden run() method and also some others.

The method generateNRandomTasks, as its name suggests, will generate n random tasks that will respect the conditions from the input given by the user, respectively to belong to the interval [minArrivalTime, maxArrivalTime] and the interval [minProcessingTime, maxProcessingTime], and will add them to the task list of the class.

The method writeData receives as parameters the output file, the task list, the current time, the application manager, and a Boolean variable that decides whether to display the average wait time, average service time and the peak hour in the queue.

The purpose of this method is to write the output data to the file received as a parameter so that it meets the expectations of the user of the application.

The data in the file will be displayed as follows:

• Time: current time

• Waiting clients: clients waiting to be processed

• Queue 0: ....

• Queue 1: ...

• .....

• Queue n: ...

• Average waiting time

• Average service time

• Peak hour

***5. Results***

I attached in the repo the results obtained by running those 3 tests provided in the assignment.

***6. Conclusions***

In this project, I learned how to work with threads. I found out what threads are, how they can be useful, and in what context they should be used. I also tried to have well-organized code.

Even if I encountered ups and downs during this project, I think it was really useful to spend this time coding as I was able to improve the way I work on a project.

***7. Bibliography***

* <https://www.youtube.com> -> for small problems that I encountered
* <http://docs.oracle.com/javase/tutorial/essential/concurrency/index.html>
* <http://www.tutorialspoint.com/java/util/timer_schedule_period.htm>