Programming techniques

*Homework 2*

Moldovan Horia Andrei

Group 30424

Problem Specification :

Objective:

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

Description:

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.

Input data:

- Minimum and maximum interval of arriving time between customers;

- Minimum and maximum service time;

- Number of queues;

- Simulation interval;

- Other information you may consider necessary;

Minimal output:

- The average of waiting time, service time and empty queue time for 1, 2 and 3 queues for the simulation interval and for a specified interval (other useful information may be also considered);

- Log of events and main system data;

- Queue evolution;

- Peak hour for the simulation interval;

Problem Analysis:

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.

Problem Approach:

The problem asks from us to place clients in lines so in order to do that we will make use of the definition of the queue data structure. A queue is nothing more than a list of item which follow two simple rules:

1. Items can be added only at the end (tail) of the queue;
2. Items can be deleted only from the beginning (head) of the queue.

So a queue follows the FIFO(first in first out) concept.

If we were to follow the usual flow of the program then we will describe three main objects: the client, the queue and the store. The clients are organized in queues and the queues are organized in a store. The role of the queue is to manage the arrival and departure of clients and the store has the task to organize the queues. One of the tasks of the store would be for example to add the clients to queues with as few already waiting clients as possible, following one of the tasks of the queue which is to delete the clients from itself when its waiting time passes.

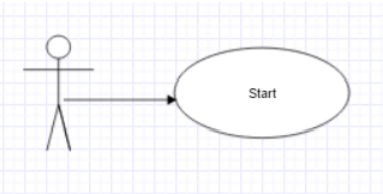
So the easiest implementation would be to store a list of clients in each queue and a list of queues in the store.

A client has three important attributes:

* It has an arrival time - at the queue;
* It has a waiting time - at the service;
* It has a waiting time - in the queue till it reaches its turn at the service

We will work with this attributes throughout the application in order to calculate the average waiting time for each working queue.

Use case

 The user has only one option when entering the application: starting simulation. From that point on, the application does its job regardless of the user interaction.

Use Case: Simulation Primary actor: User Main success Scenario:

• The user successfully launches the application.

• The user chooses the constraints for the simulation (minimum and maximum arrival times expressed in seconds, minimum and maximum service time expressed in second and the total interval of time for simulation).

• The user chooses the number of queues.

• The user hits the Start button.

• The simulation starts, user can track the animation and the logger in parallel to fully understand the events that happen.

Extensions: If the user provides some erroneous information such as minimums larger than maximums, negative inputs for any field or non-number information, a message of error is displayed and he has the option of restarting and providing some other input values.

Design

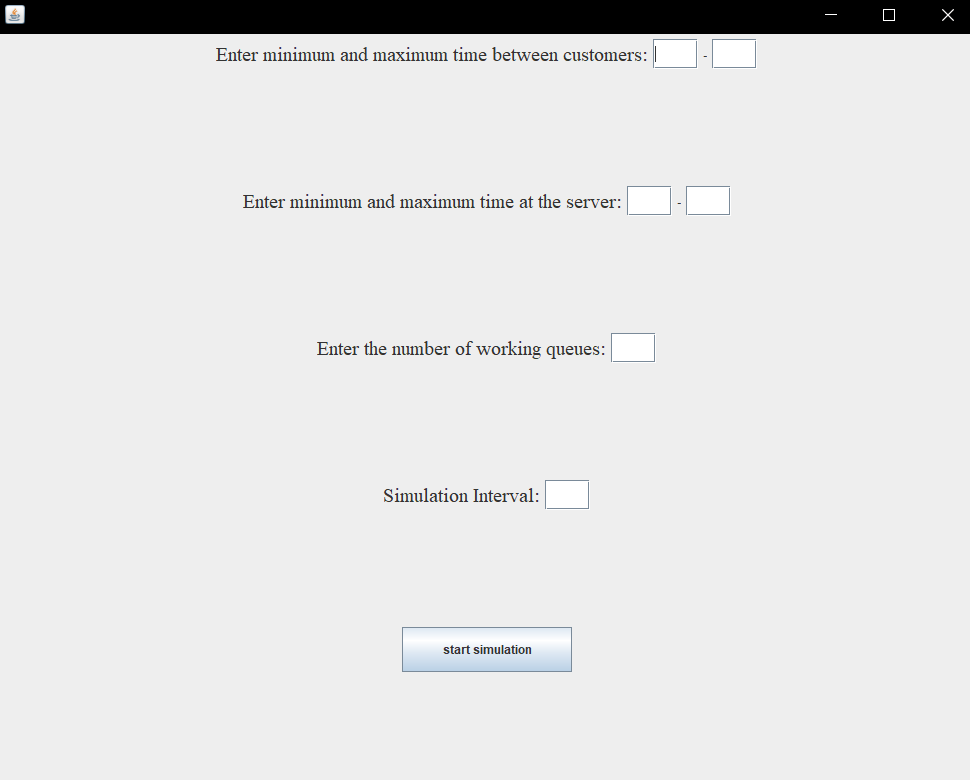
We can divide the project in two major parts: the logic of the application and the user interface. They both must run flawless in order for the application to work correctly. The design follows a Model-View-Controller pattern.

* The logic consists of the approach used(described above), and of the implementation(the classes with their attributes and methods), which glues all the parts of the MVC model together.
* -The GUI part was implemented according to the MVC pattern by splitting the written code in three big parts. -The Model contains the algorithms which run “in the back”, to which the user has no access.

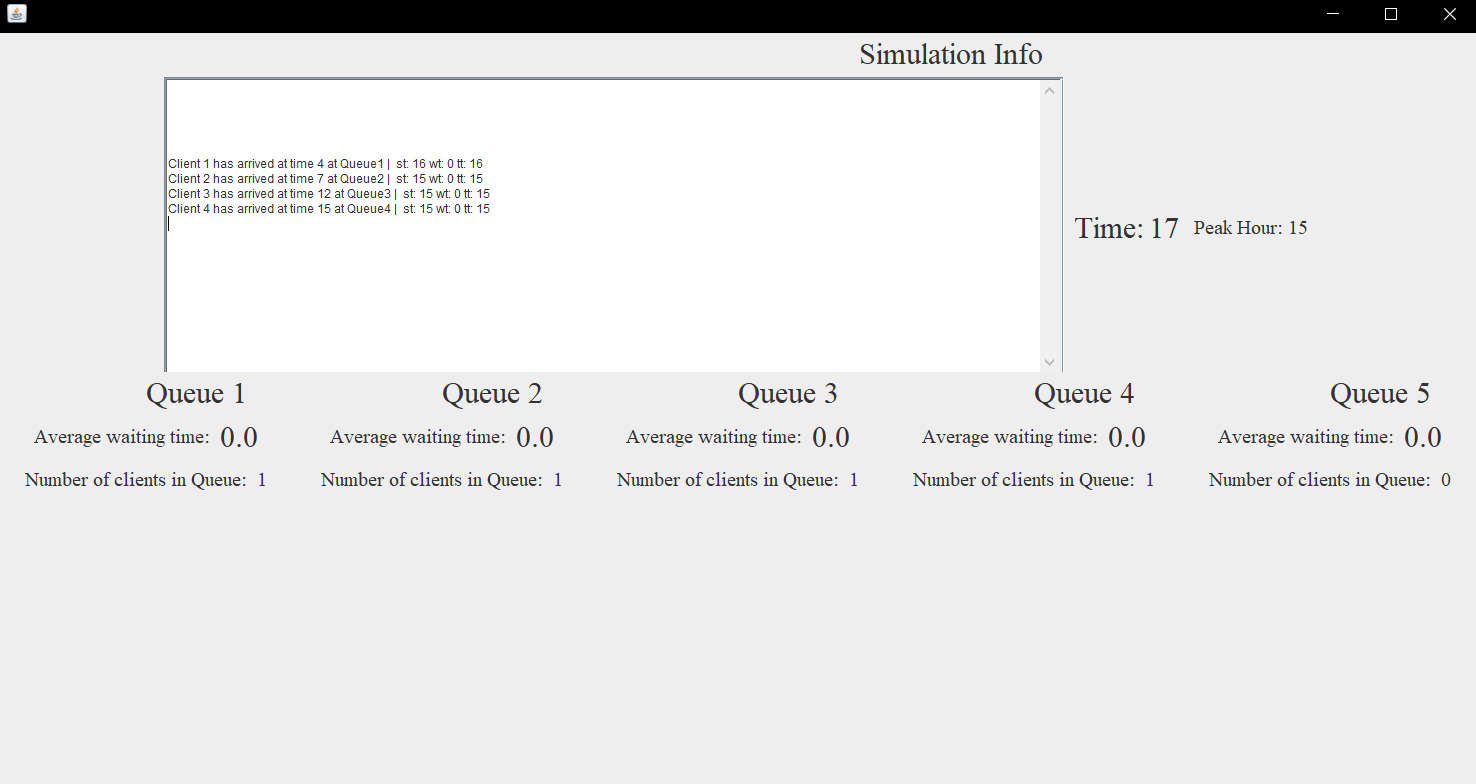
-The user has access only to the provided text fields which can be found in the View part.

The view is made of two frames:

-> The first frame which allows user communication

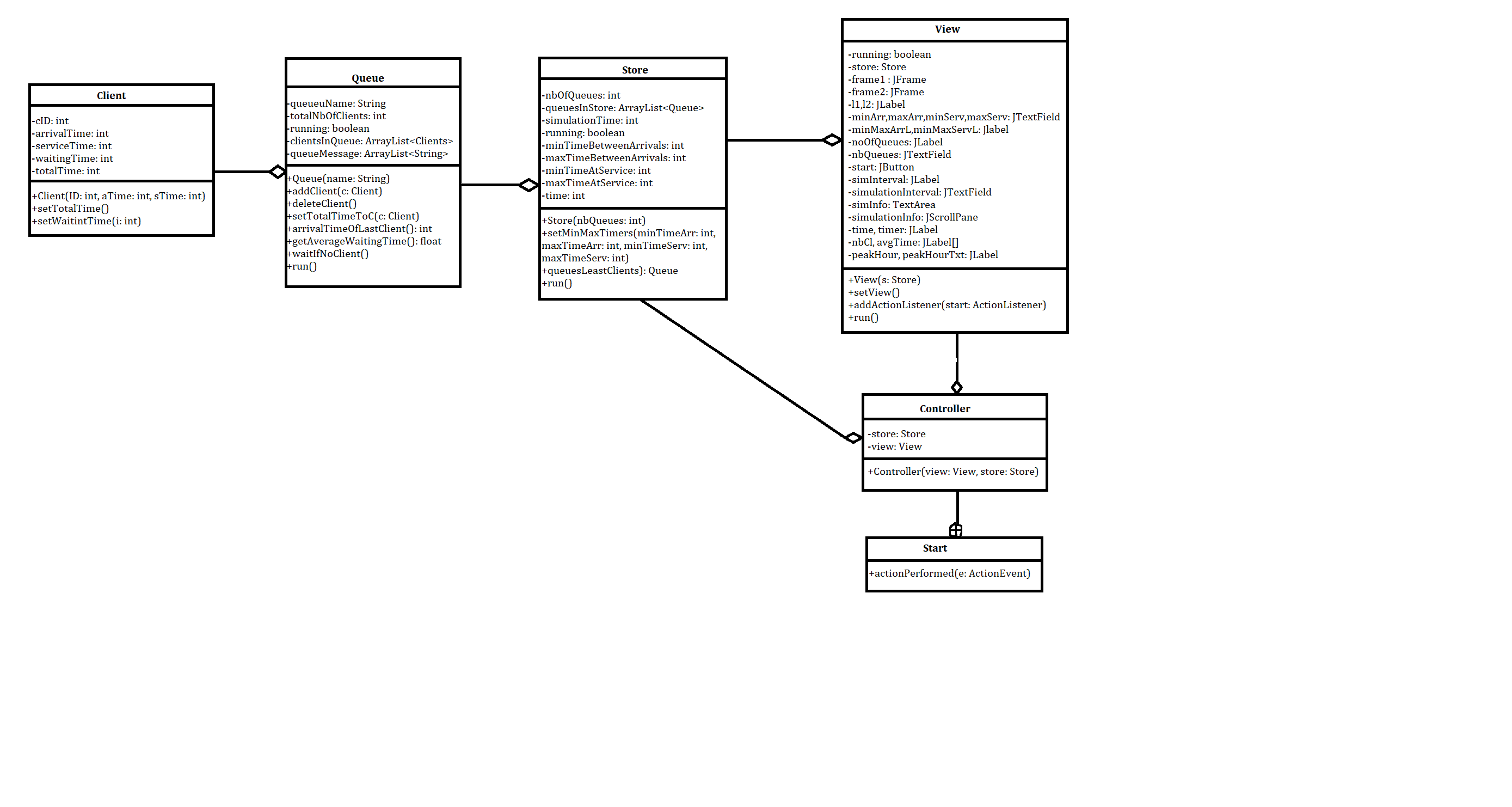


-> The second frame which is used only to display the simulation status of the program, no communication with the user.



The controller part of the Model-View-Controller implementation is used in order to manage the function of the Start button.

**UML Diagram**

****

**Implementation**

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

* ***Client.java***

The attributes of this class are cID, which represents the number of the client; arrivalTime, which represents the time at which the client enters the queue, serviceTime, representing the time the client has to wait at the service, waitingTime, 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.

* ***Queue.java***

The attributes of this class are queueName which hold the name of the Queue, totalNbOfClients, which tell how many clients are at the time in the queue, queueMessage, which captures all the information of the actions performed by the queue as a string format, and the most important of all, clientsInQueue, which is an arraylist of clients(clients with the attributes specified above) on which we perform the addition and the deletion operations. All the mentioned fields are declared as private, just as before.

The constructor of the class takes as parameter a string of characters and sets it as the name of the queue. Furthermore it initializes the list of clients.

This class extends the thread class which means that it is meant to run in parallel with the main program of the application.

The method of this class are:

-addClient: this method takes as parameter a client and it does not only add it to the list of clients of the queue but it also displays a message which says which client arrives at which queue, at which time, with which waiting-, serving-, and total time. The method uses another method setTotalTimeToC() which does nothing more than calculating the waiting time of the client. The thread method notifyAll() is used in order to notify the other methods of this class which use the wait() method of the action performed.

-deleteClient: this method waits if there are no clients in the queue till some other method uses the method notifyAll(). Then uses the method getAverageWaitingTime in order to find how much time is spend on average by the clients in the queue at waiting for their turn. This information plus the name of the client, the name of the queue and the time the client leaves the queue are displayed throughout this method.

Note: each time a client is added or deleted or added the number of clients in queue is decreased respectively decreased.

-run(): this method is the one we overrided from the thread class. The code found in the while loop of this method will be executed,as long as the boolean contidion is true, in parallel with other threads. The code does the following: it waits till some other method notifies it of some change, it paused the program afterwards for the amount of time the client has to wait in the queue and then it deletes the client by using the method described above.

* ***Store.java***

The attributes of this class are nbOfQueues, which holds the number of the working queues in the store, queuesInStore, an array of queues and the minimun and maximum integers which will be specified by the user in order to determine the spawning time and service waiting time of the clients. The time attribute is also present in order to increment it at each second. Once again, all attributes are declared as private.

This class extends Thread because it has to work concurrent with the other threads in order to be able to list each action performed at each one second.

The constructor of this class initializes the specified amount of queues with names in increasing order.

The methods of this class are:

- setMinMaxTimers: this method is used so the user can specify the desired intervals of time.

-queueLeastClients: this method returns the queue with the fewest number of clients in the store so it can lately decide where to add upcoming clients.

-run: this method is the one which was overriden from the thread class. In this method we constantly create new clients with the specified time intervals and we set the time the program has to wait between the arrival of two clients by using the sleep() function. In this method we also start all of our working queues, which means that each queue is ready to work in parallel and to execute its run() method.

* ***View.java***

The attributes of this class are all the frames, labels, buttons, and text fields used for the communication with the user. Each of them is resized and modified with the purpose of a more friendly appearance.The setView method does nothing more than taking the data introduced by the user in the first frame and creating a second frame which displays the desired simulation with the introduced parameters. All of the design of the first frame is done in the constructor method. This class implements the Runnable interface which allows us to use the run method once again. We need the presence of the threads so that we can increment the time at the specified amount and ‘refresh’ the data at the same time. The addActionListener method is used in order to add a functionality(which will be described int the controller class) to the start button.

* ***Controller.java***

This class is used for the transition between the first and the second frame described in the view class, done by pressing the start button.