Cristian Flaviu - Remus

Group 30423

Homework2

1) Objectives :

The main purpose of this program is to implement a simulation application aiming to analyze queuing based systems for determining and minimizing clients' waiting time.

In order to do that we had find a way to store information about the clients (their arrival time, service time etc.) and also information about every queue from the existing "shop".

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 (I choose the number of clients);

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;

The secondary objectives are:

a) **Designing a Client.** This will be our "consumer" that is generated randomly and is queued at one of the servers at its arrival time and is taken out of the queue when its at the top and its processing time passes. This is detailed later on;
b) **Designing a Server.** The server is used to queue and offer service to the Clients. A server uses its own Thread in synchronization with the other servers and the main Controller. This is detailed later on .
c) **Designing a Scheduler.** This is used to "deploy" the Clients to one of the existing servers. This is detailed later on;
d) **Designing a strategy.** Some strategies are required in order to schedule the Clients in the best possible manner. For the purpose of simplicity, here I used only one: by time (shortest waiting time in queue) . This is detailed later on.

e) Implementing the required output. We are required to find different parameters like the average waiting time, the peak time, the empty queue time etc.

f) Implementing the Graphic User Interface (GUI). Final step where we make a simple GUI for the user to interact with.

2) Problem analysis, scenarios and use cases :

The program expects interaction with a user in order to introduce date and specify the flow of instruction. The user have to introduce in a specified text field the value of the minimum arrival time, maximum arrival time, minimum service time, maximum service time and also the simulation time. All the date introduce should be integer numbers.

The minimum arrival time should have a smaller value than the maximum arrival time and also the minimum service time should have a smaller value than the maximum service time and also a big value for the simulation time in order for the program to work properly .

a) Use case "Run the simulation"

The user have to open the program and introduce properly as mentioned above the required data. After that he/she has to press the Save Button(from the right bottom of the page) in order to save the date and start the simulation. In the left side of the screen the evolution of the servers(queues) will be presented. When a client will be introduce in a queue his/her id will be displayed next to the server it belongs for a certain period of time (until that client is served).

Alternative scenarios :

The user does not respect the specified format (he has introduced other characters than numbers , a minimum time greater that the maximum time or a really small value for the simulation time(eg:0)). In this case the system won't produce any output and an error message will be displayed.

3) Design (design decisions, UML diagrams, data structures, class design, interfaces, relationships, packages, algorithms, user interface)

Firstly, we must design a Client entity that can hold information like the arrival time of the Client and the service time, both of which are given as input. Other information like a waiting time (to know how much it must wait until given service), finishing time (the time when the client live the queue) the and a name (easily identify the Client) I considered to be necessary.

Another step is the creation of a server which “consumes” and queues Clients. To queue the clients, I used an ArrayBlockingQueue. This structure permits the safe dequeuing and queuing of elements by threads. An Atomic Integer totalServiceTime is used to continuously store info about the total service time of all added to the queue. The server has a name stored in a String serverName in order to identify each server more easily. It also has other attributes used for calculating the desired outputs.

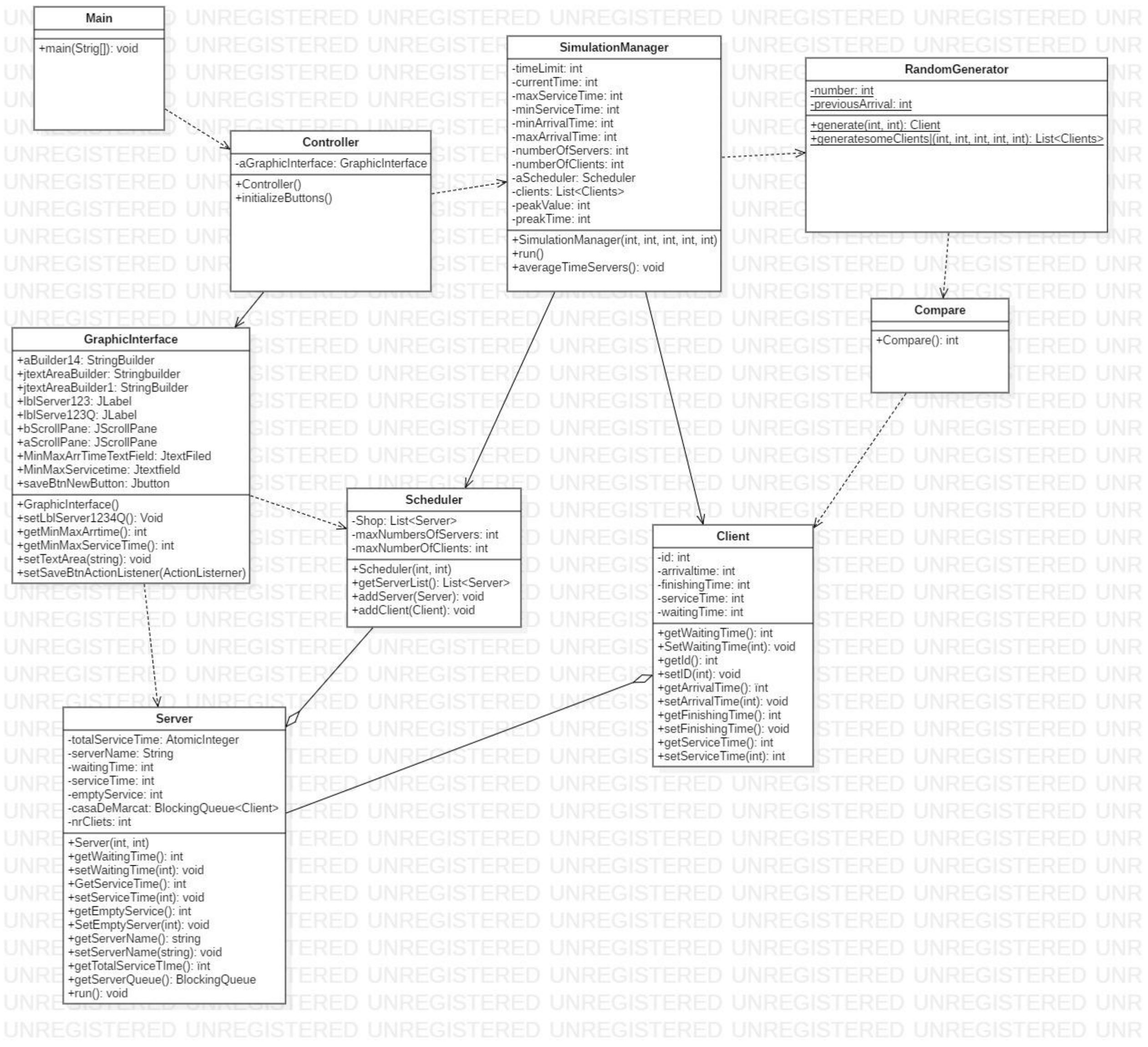
Next, we have a scheduler used for storing a list of all available servers(shop).It has 2 attributes, maxNumberOfClients which represent the maximum amount of clients which can be placed in a queue at a time and maxNumberOfServer which represent the maximum amount of server available.

In order to generate a random list of clients we the Class RandomGenerator which generates the clients using two static method and attributes. One for generating a single Client and one for generating the required number of clients.

The GUI simply displays the required information in different JScrollPanes and JLabels for the servers and one for the generated Clients and one for the Log.

There are 4 packages :

- structures which holds the following classes : Client ,Scheduler and Server
- randomgenerator which holds the following classes : Compare, RandomGenerator
- main which holds the following classes : Main, SimulationManager
- gui which hold the following classes: Controller, GraphicInterface



4) Implementation

a) Client

```
private int id;
private int arrivaltime;
private int finishingTime;
private int serviceTime;
private int watingTime;
```

The id represent the name of the Client each Client will have a different id. The arrivalTime is the time when a Client get to queue .ServiceTime represent the time needed by the client in order to be served. WaitingTime represent the time a client needs to waits until he/she will be served. The finishingTime is the time when the client leaves the queue.
Each attribute has a getter and a setter .

b) Server

```
 private AtomicInteger totalServiceTime;
        private String serverName;
        private BlockingQueue<Client> casaDeMarcat;
        ….
```

CasaDeMarcat represent a list(queue) where all the Clients will be stored. The serverName will be name of the server ,usually the will be “Server ” .

```
`       public void addClient(Client aClient) {
        casaDeMarcat.add(aClient);
        totalServiceTime.set(totalServiceTime.get() + aClient.getServiceTime());

        aClient.setFinishingTime(totalServiceTime.get()+SimulationManager.getCurrentTime());
        aClient.setWatingTime(totalServiceTime.get());
        serviceTime=serviceTime+aClient.getServiceTime();
        watingTime=watingTime+aClient.getWatingTime();

        }
```

This method will add a Client to the CasaDeMarcat and will also calculate and set the finishing and the waiting time for the client which has just been added.

```
public void run() {

        while (SimulationManager.getCurrentTime() < SimulationManager.getTimeLimit()) {
                try {

                        if(casaDeMarcat.size()>0)
                                {totalServiceTime.set(totalServiceTime.get() - 1);

                                                                                                }
                        elseDeMarcat.peek() != null)
                        {

                                if(casaDeMarcat.peek().getFinishingTime()==SimulationManager.getCurrentTime())
                                {

                                        if(serverName.contains("1")) {

                                        System.out.println("Client " + casaDeMarcat.peek().getId() + " finish at
server: " +serverName+"\n");
                                        GraphicInterface.setTextArea("\nClient " + casaDeMarcat.peek().getId() + "
finished at server: " +serverName+"\n");
                                        GraphicInterface.aBuilder.delete(0,
(""+casaDeMarcat.peek().getId()).length()+1);
                                        GraphicInterface.setLblServer1Q();

                                        }
                                        else if(serverName.contains("2"))
                                        {System.out.println("\n Client " + casaDeMarcat.peek().getId() + " finish at
server: " +serverName+"\n");
                                        GraphicInterface.setTextArea("\nClient " + casaDeMarcat.peek().getId() + "
finished at server: " +serverName+"\n");
                                        GraphicInterface.aBuilder2.delete(0,
(""+casaDeMarcat.peek().getId()).length()+1);
                                        GraphicInterface.setLblServer2Q();

                                        }
                                        else if(serverName.contains("3"))
                                        {System.out.println("\nClient " + casaDeMarcat.peek().getId() + " finish at
server: " +serverName+"\n");
                                        GraphicInterface.setTextArea("\nClient " + casaDeMarcat.peek().getId() + "
finished at server: " +serverName+"\n");
                                                GraphicInterface.aBuilder3.delete(0,
(""+casaDeMarcat.peek().getId()).length()+1);
```

```
                                                                          GraphicInterface.setLblServer3Q();
                                                                  }
                                                                  else if(serverName.contains("4"))
                                                                  {System.out.println("\nClient " + casaDeMarcat.peek().getId() + " finish at
server: " +serverName+"\n");
                                                                  GraphicInterface.setTextArea("\nClient " + casaDeMarcat.peek().getId() + "
finished at server: " +serverName+"\n");
                                                                          GraphicInterface.aBuilder4.delete(0,
(""+casaDeMarcat.peek().getId()).length()+1);
                                                                          GraphicInterface.setLblServer4Q();
                                                                  }
                                                                  casaDeMarcat.take();

                                                          }
                                                  }
                                                  Thread.sleep(1000);
                                          }

                                          catch (InterruptedException e) {

                                                  e.printStackTrace();
                                          }

                                  }

                          }
```

Because this Class implements the interface Runnable it has to implement the method run.
This method will run until the simulation time will be smaller the current time and will check if the client from top the top of the server has the finishing time equal to the current time, if yes then the client will be deleted from the queue. The totalServiceTime will be decremented at each second with one unit if in the Server exist at least one client. It also contains some suggestive message for the console in order for the user to understand what is happening at each step.

c) Scheduler

```
public void  addServer(Server server)
        {
                if(shop.size()<maxNumberOfServers)
                        {shop.add(server);
                        Thread aThread=new Thread(server);
                        aThread.start();
                        }

                else
                        System.out.println("to many servers");
        }
```

This method will add a Server to the shop if the number of server is smaller than the maxNumberOFServers.

```
public void addClient(Client aClient )
        {
        Server auxServer=new Server(100,"a");
        int minimalTime=10000;

        for(Server aServer:shop)
        {
        if(aServer.getTotalServiceTime()<minimalTime && aServer.getServerQueue().size() <maxNumberOfClients)
                        {
                                auxServer=aServer;
                                minimalTime=aServer.getTotalServiceTime();



                        }
```

```
                                }
auxServer.addClient(aClient);
```

This method will add a client in the most convenient server for the client(where the totalTimeService is the smaller and also where the server has not the maximum amount of clients)

d) GenerateSomeClients

```
        public  static  Client generate(int minInterval,int maxInterval,int minServiceTime,int maxserviceTime)
        {
                Client aClient =new Client();
                Random rand=new Random();

                int arrivalTime=previousArrival+minInterval+rand.nextInt(maxInterval-minInterval);
                int serviceTime=minServiceTime+rand.nextInt(maxserviceTime-minServiceTime);

                previousArrival=arrivalTime;

                aClient.setId(number++);
                aClient.setArrivaltime(arrivalTime);
                aClient.setServiceTime(serviceTime);
                aClient.setFinishingTime(0);

                return aClient;
                }
```

This method will generate a Client which will have an arrivalTime bigger than the arrival time of the previous client with the value of the minInterval and also smaller than previousArrivalTime +maxInterval.The service time will be also between minServiceTime and MaxServiceTime.

```
public static List<Client> generateSomeClients(int numberOfclients,int minInterval,int maxInterval,int minServiceTime,int maxserviceTime)
        {

                List<Client> aList=new ArrayList<Client>();

                for(int i=0;i<numberOfclients;i++)
                {
                        aList.add(generate( minInterval,maxInterval,minServiceTime,maxserviceTime));

                }
                aList.sort( new Compare());
                return aList;
        }
```

This method will generate a list of Clients of size NumberOfClients, with some characteristic (minServiceTime, MaxServiceTim , etc).

e) Simulation Manager

```
         public void run() {

                        while (currentTime < timeLimit) {

                                System.out.println("\n-----------time " + currentTime + "----------\n");
                                GraphicInterface.setTextArea("\n-----------time " + currentTime + "----------\n\n");
```

```
                                            for (Client aClient : clients) {
                                                    if (aClient.getArrivaltime() == currentTime) {
                                                            aScheduler.addClient(aClient);
                                                    }

                                                    if (aClient.getFinishingTime() != 0 && aClient.getFinishingTime() == currentTime) {

                                                    }
                                            }

                                            try {

                                                    Thread.sleep(1000);
                                                    currentTime++;
                                            } catch (InterruptedException e) {

                                                    e.printStackTrace();
                                            }
                                            for (Server aServer : aScheduler.getServerList()) {
                                                    if (aServer.getServerQueue().peek() != null)

                                                    {

                                                    System.out.println("Id:" + aServer.getServerQueue().peek().getId() +  " finishTime  "+
                        aServer.getServerQueue().peek().getFinishingTime() + "\n");
                                                    }


                                            }

                                    }

                            }
```

This class implements the interface Runnable so it has to implements the method run. This method when appealed will run as long as the current time is smaller than the simulation time. It will iterate through all the clients and check if their arrival time is equal to the current time, if so the Client will be added to a server.

f) GraphicalInterface

This class contains the following attributes :

```
public static StringBuilder aBuilder=new StringBuilder();
public static StringBuilder aBuilder2=new StringBuilder();
public static StringBuilder aBuilder3=new StringBuilder();
public static StringBuilder aBuilder4=new StringBuilder();
public static StringBuilder jtextAreaBuilder=new StringBuilder();
public static StringBuilder jtextAreaBuilder1=new StringBuilder();

private JLabel lblServer1;
public static JLabel LblServer1Q;
private JLabel lbServer2;
public static JLabel LbServer2Q;
JLabel lblServer3;
private static JLabel LblServer3Q ;

JLabel lblServer4;
private static JLabel LblServer4Q ;
private static JTextArea textArea;
private static JTextArea textArea_1;
private JScrollPane bScrollPane;
private JScrollPane aScrollPane;
```

```
        private JTextField MinArrTimeTextField;
        private JTextField MaxArrTimetextField;
        private JTextField MinServiceTimeTextField;
        private JTextField MaxServiceTimetextField;
        private JTextField SimulationIntervaltextField;
        private JLabel lblNewLabel;
        private JLabel lblNewLabel_1;
        private JLabel lblNewLabel_2;
        private JLabel lblNewLabel_3;
        private JLabel lblNewLabel_4;

        private static JLabel lblServer1Time;
        private static JLabel lblServer2Time;
        private static JLabel lblServer3Time;
        private static JLabel lblServer4time;
        private JButton saveBtnNewButton;
        private JLabel lblTextpeakTime;
        private JLabel lblTexTPeakValue;
        private static JLabel lblpeakTime;
        private static JLabel lblPeakValue;
        private JLabel lblWatingtime;
        private JLabel lblEmptyserver;
        private static JLabel lblEmptyServer1;
        private static JLabel lblEmptyServer2;
        private static JLabel lblEmptyServer3;
        private static JLabel lblEmptyServer4;
        private static JLabel lblAvgservivetime;
        private static JLabel lblServiceTime1;
        private static JLabel lblServiceTime2;
        private static JLabel lblServiceTime3;
        private static JLabel lblServiceTime4;
        private JLabel lblNewLabel_5;
        private static JLabel lblWaitingTime1;
        private static JLabel lblWaitingTime2;
        private static JLabel lblWaitingTime3;
        private static JLabel lblWaitingTime4;
```

Each of this attribute has a getter and a setter. Each StringBuilder is used for storing information which will be displayed on the interface.

g) Controller

```
        GraphicInterface aGraphicInterface = new GraphicInterface();
public void initializeButtons() {
                aGraphicInterface.setSaveBtnActionListener(e -> {
                        int maxServicetime = 0;
                        int minServiceTime = 0;
                        int minArrtime = 0;
                        int maxArrtime = 0;
                        int timeLimit = 0;

                        maxServicetime = 
Integer.parseInt(aGraphicInterface.getMaxServiceTimetextField().getText());
                        minServiceTime = 
Integer.parseInt(aGraphicInterface.getMinServiceTimeTextField().getText());

                        minArrtime = 
Integer.parseInt(aGraphicInterface.getMinArrTimeTextField().getText());
                        maxArrtime = 
Integer.parseInt(aGraphicInterface.getMaxArrTimetextField().getText());

                        timeLimit = 
Integer.parseInt(aGraphicInterface.getSimulationIntervaltextField().getText());

                        SimulationManager aManager = new SimulationManager(minArrtime, maxArrtime, 
minServiceTime, maxServicetime,
                                        timeLimit);
```

```
				Thread aThread = new Thread(aManager);
				aThread.start();
		});
		}
```

The class Cotroller is used for interpreting the data introduced by the user and for creating a new instance of type SimulationManager.

1. Results

There were no tests done on this Project as it is impossible to predict the generated Clients parameters and to check it against some values.

2. Conclusion

This homework brought a nice view of the Multithreading “domain” of java. I believe I learned more about this by doing this practical homework than only by reading on the subjects. Further improvements of the code can be done making a cleaner GUI and by “cleaning” the code a little bit. Also, more strategies can be added, and more info can be extracted from the simulation.

3. Biography

Mostly what was presented in PT2019_tema2_Vio at the lab .

https://www.youtube.com/watch?v=F-CkaU8aQZI