DOCUMENTATION

ASSIGNMENT *2*

CONTENTS

1. Assignment Objective

(I) The main objective

The assignment's main objective is to create an application that implements queues procedures. The queues process the clients with an id, arrival time, and a processing time representing the time in which the client will be served from when he arrived at the queue.

(II) The sub-objectives

The sub-objectives are to see how the clients can be processed faster by using threads which will make the queues and the management of the clients in the waiting list work at the same. Because of the usage of threads, we will need to declare special data structures and variables.

Every queue will have a BlockingQueue data structure of clients which will provide the threads safety. The variables waitingPeriod and totalWaiting time variables will be used as AtomicInteger to avoid threads interference.

2.Problem Analysis, Modeling, Scenarios, Use Cases

(I) Problem Analysis

The application should perform the operation mentioned above using an interface that takes the information inserted by the user: the number of clients, number of queues, simulation time, and for each client that will be generated: minimum arrival time, maximum arrival time, minimum service time, maximum service time. The log of the events will be displayed live in the interface.

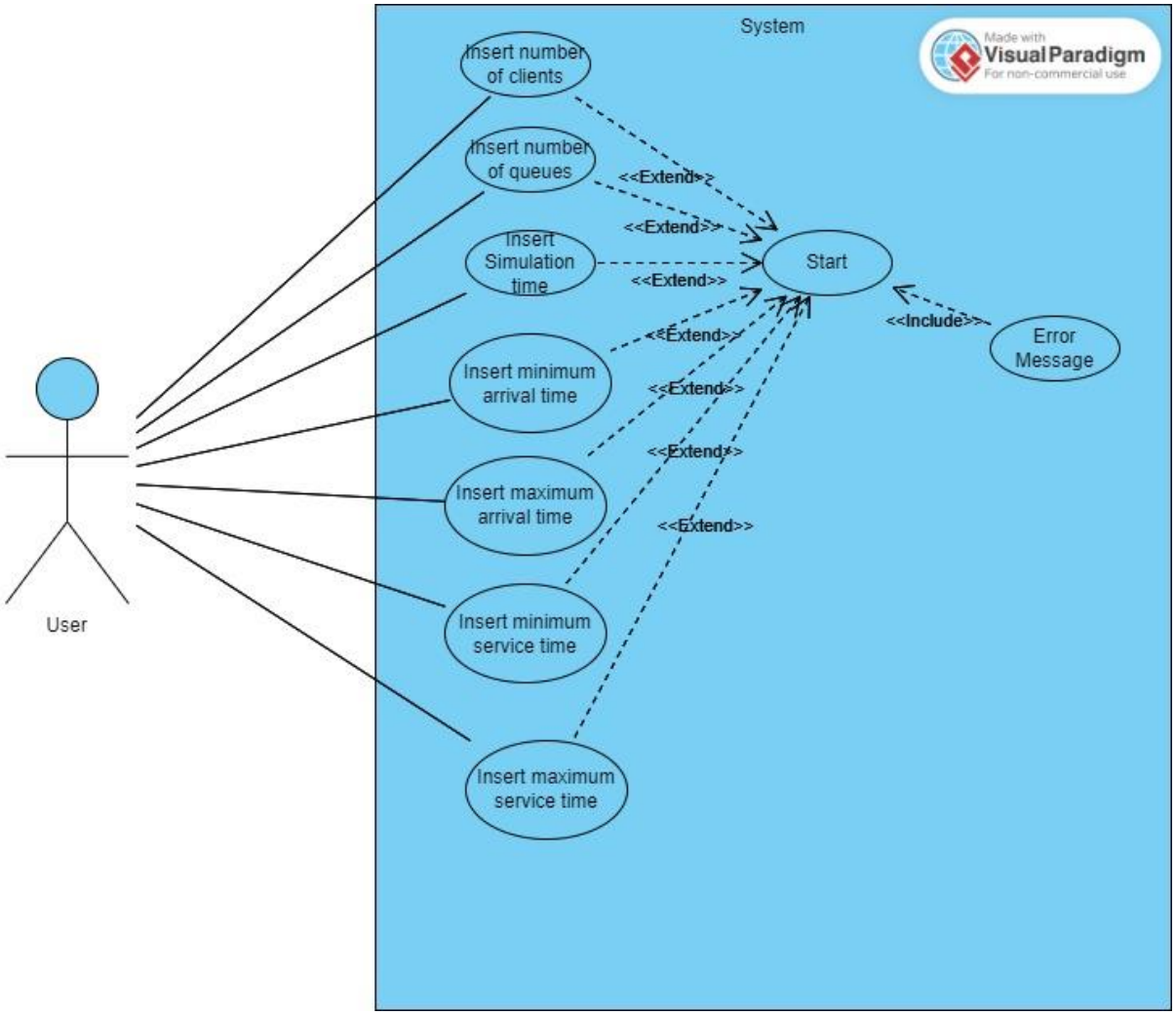
(II) Modeling

We will have 7 text fields for each data that needs to be inserted and one button which is „Start Simulation"

(III) Scenarios

We have only one scenario: the scenario where the user doesn't insert numbers on the text fields and inserts other special characteres.

(IV) Use cases



3. Design

The main object object in the application is the Server.
The Server class contains the following atrributes and methods:

-Atributes:

```
private int id;
private BlockingQueue<Client> clients;
private final AtomicInteger waitingPeriod;

private final AtomicInteger serviceTime;

private final AtomicInteger totalWaiting;
```

-Mehtods:

The main method here is the thread:

```
public void run() {
    while (true) {
        synchronized (this) {
            if (clients.size() > 0) {
                try {
                    Client client = clients.peek();
                    Thread.sleep(1000*client.getServiceTime());
```

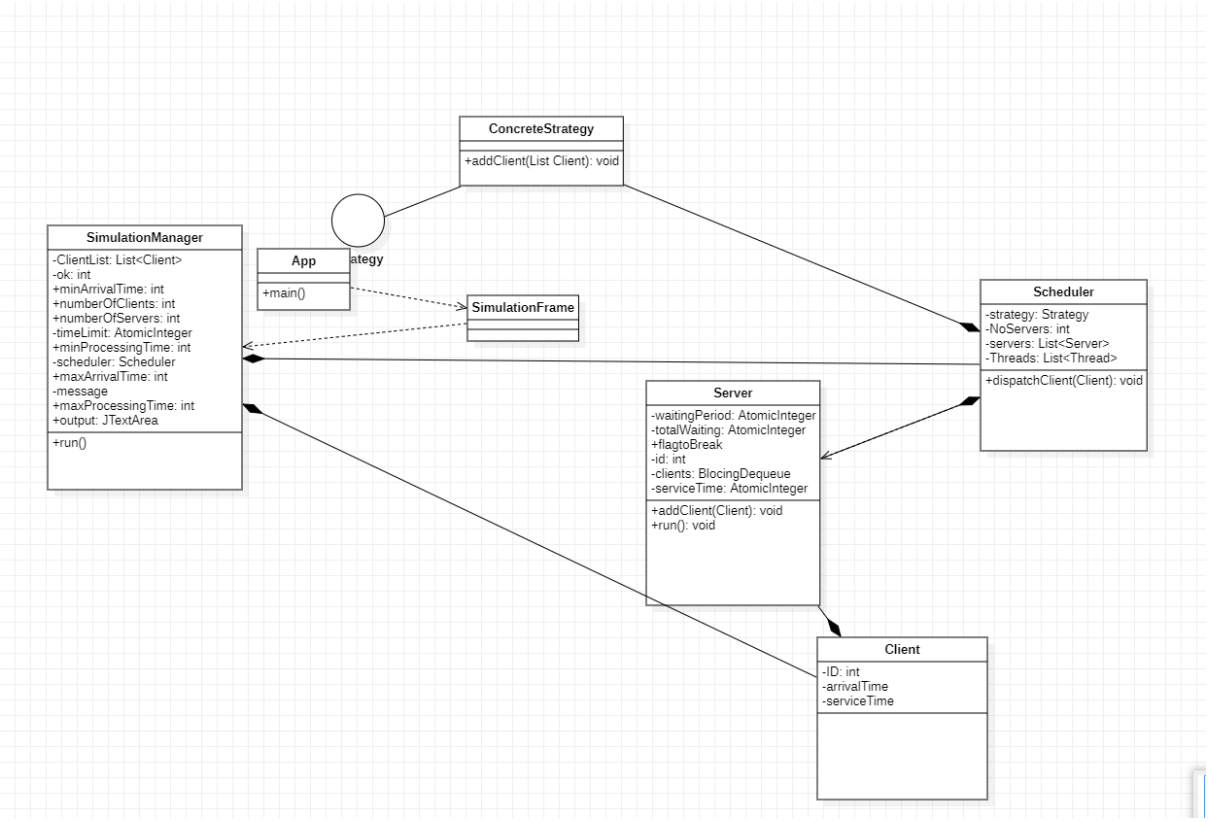
```
                    clients.poll();
                    if (waitingPeriod.intValue() > client.getServiceTime())
{
                        waitingPeriod.addAndGet(-client.getServiceTime());

                    } else {
                        waitingPeriod.set(0);
                    }

                } catch (InterruptedException e) {
                    throw new RuntimeException(e);
                }
            }
        }
        if(clients.isEmpty() && flagtoBreak==1)
            break;
    }
}
```

Another important class is the Scehduler class where the list of queus are stored.
The simulaitonManager is the class where the clients are proccessed in the queues and there is the other thread.

```
public void run() {
    try (PrintWriter writer = new PrintWriter("output.txt")) {
        AtomicInteger currentTime = new AtomicInteger(0);
        int peak = 0;
        int max = 0;
        int nrclients = 0;
        float avgWaitingTime = 0;
        float avgServiceTime = 0;
        float sum = 0;
        while (currentTime.intValue() < timelimit.intValue()) {
            if (ok == 0) {
                writer.println("Clients:");
                System.out.println("Clients:");
                message+="Clients:\n";
                for (Client client1 : clientList) {
                    writer.println(client1.toString() + ";");
                    System.out.println(client1.toString() + ";");
                    message+=client1.toString()+";"+"\n";
                    sum += client1.getServiceTime();
                }
                avgWaitingTime = sum / numberOfClients;
                writer.println();
                System.out.println();
                message+="\n";
                ok = 1;
            }

            List<Client> clientsToDispatch = new ArrayList<>();
            for (Client client : this.clientList) {
                if (client.getArrivalTime() == currentTime.intValue()) {
                    clientsToDispatch.add(client);
                }
            }

            for (Client client : clientsToDispatch) {
                scheduler.dispatchClient(client);
                clientList.remove(client);
            }
```

```
            if(this.clientList.isEmpty()){
                for (Server server : scheduler.getServers()) {
                    //writer.println(server.getWaitingPeriod().intValue());

                    server.flagtoBreak=1;
                }
            }

            for (Server server : scheduler.getServers()) {
                //writer.println(server.getWaitingPeriod().intValue());

                nrclients = nrclients + server.getQueueSize();
            }
            if (nrclients > max) {
                max = nrclients;
                peak = currentTime.intValue();
            }
            nrclients = 0;
            writer.println("Current time: " + currentTime + "\n");
            writer.print("Waiting clients:");

            System.out.println("Current time: " + currentTime + "\n");
            System.out.print("Waiting clients:");

            message+="Current time: " + currentTime + "\n\n";
            message+="Waiting clients:";

            for (Client client : clientList) {
                writer.print(client.toString() + "; ");
                System.out.print(client.toString() + "; ");
                message+=client.toString()+";";
            }
            writer.println();
            writer.println(scheduler.toString());

            System.out.println();
            System.out.println(scheduler.toString());


            message+="\n";
            message+=scheduler.toString()+"\n";
           output.setText(message);

            currentTime.getAndIncrement();
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }


        }
```

The UML diagram:



4. Implementation

(I) Clients Class

```
package model;

public class Client {
    private int ID;
    private int arrivalTime;
    private int serviceTime;

    public Client(int ID, int arrivalTime, int serviceTime) {
        this.ID = ID;
        this.arrivalTime = arrivalTime;
        this.serviceTime = serviceTime;
    }

    public void setID(int ID) {
        this.ID = ID;
    }

    public void setArrivalTime(int arrivalTime) {
        this.arrivalTime = arrivalTime;
    }

    public void setServiceTime(int serviceTime) {
        this.serviceTime = serviceTime;
    }

    public int getID() {
        return ID;
    }
```

```
    public int getArrivalTime() {
        return arrivalTime;
    }

    public int getServiceTime() {

        return serviceTime;
    }

    public String toString(){                                   //prints
the contents of the client object in the form
        return "("+getID()+", "+getArrivalTime()+",
"+getServiceTime()+")";   // (id, arrivalTime, serviceTime).
    }
}
```

(II) Server class

```
package model;

import java.util.ArrayList;
import java.util.List;
import java.util.concurrent.BlockingQueue;
import java.util.concurrent.LinkedBlockingDeque;
import java.util.concurrent.atomic.AtomicInteger;

import static java.lang.Thread.currentThread;

public class Server implements Runnable{
    private int id;
    private BlockingQueue<Client> clients;
    private final AtomicInteger waitingPeriod;

    private final AtomicInteger serviceTime;

    private final AtomicInteger totalWaiting;

    public int flagtoBreak=0;

    public Server(int id) {
        clients= new LinkedBlockingDeque<>();
        waitingPeriod=new AtomicInteger(0);
        serviceTime=new AtomicInteger(0);
        totalWaiting=new AtomicInteger(0);
        this.id=id;
    }
    public int getId() {
        return id;
    }

    public AtomicInteger getServiceTime() {
        return serviceTime;
    }

    public AtomicInteger getTotalWaiting() {
        return totalWaiting;
    }



    public void addClient(Client newClient){
        clients.add(newClient);
```

```
        waitingPeriod.addAndGet(newClient.getServiceTime());
        totalWaiting.addAndGet( newClient.getServiceTime());
    }


    @Override
    public void run() {
        while (true) {
            synchronized (this) {
                if (clients.size() > 0) {
                    try {
                        Client client = clients.peek();
                        Thread.sleep(1000*client.getServiceTime());
                        clients.poll();
                        if (waitingPeriod.intValue() >
client.getServiceTime()) {
                            waitingPeriod.addAndGet(-
client.getServiceTime());

                        } else {
                            waitingPeriod.set(0);
                        }

                    } catch (InterruptedException e) {
                        throw new RuntimeException(e);
                    }
                }
            }
            if(clients.isEmpty() && flagtoBreak==1)
                break;
        }
    }


    public String toString(){
        String print="";
        print +="Queue" + getId() +": ";
        if(clients.size()>0)
            for(Client client: clients){
                print+=client.toString()+"; ";
            }
        else{
            print+="closed";
        }
        return print;
    }
    public AtomicInteger getWaitingPeriod() {
        return waitingPeriod;
    }
    public int getQueueSize(){
        return clients.size();
    }


    public List<Client> getQueue() {
        return (List<Client>) clients;
    }
}
```

(III) Scheduler class

```
package bussiness.logic;
```

```java
import model.Client;
import model.Server;

import java.util.ArrayList;
import java.util.List;


public class Scheduler {
    private List<Server> servers;
    private int NoServers;
    private Strategy strategy;
    private final List<Thread> threads = new ArrayList<>();
    public Scheduler(int NoServers){
     this.NoServers=NoServers;
     servers=new ArrayList<>();
     for(int i=0; i < NoServers; i++){
         Server  server= new Server(i+1);

         Thread thread=new Thread(server);
         thread.start();
         threads.add(thread);
         servers.add(server);
     }
     strategy=new ConcreteStrategy();
    }
    public void dispatchClient(Client client){
        strategy.addClient(servers,client);
    }

    public String toString(){
        String printed = "";
        for(Server server : servers){
            printed += server.toString() + '\n';
        }
        printed += "\n";
        return printed;
    }

    public List<Server> getServers() {
        return servers;
    }
}
```

(IV) Strategy interface

```java
package bussiness.logic;

import model.Client;
import model.Server;

import java.util.List;

public interface Strategy {
    public void addClient(List<Server> servers, Client client);
}
```

(V) ConcreteStrategy

```java
package bussiness.logic;

import model.Client;
import model.Server;
```

```
import java.util.List;

public class ConcreteStrategy implements Strategy{

    @Override
    public void addClient(List<Server> servers, Client client) {
        //To do generate method stub
        Server toAdd= servers.get(0);
        for(Server server: servers){

if(server.getWaitingPeriod().intValue()<toAdd.getWaitingPeriod().intValue()
){
                toAdd=server;
            }
        }
        toAdd.addClient(client);
    }
}
```

(VI) Simulation Manager

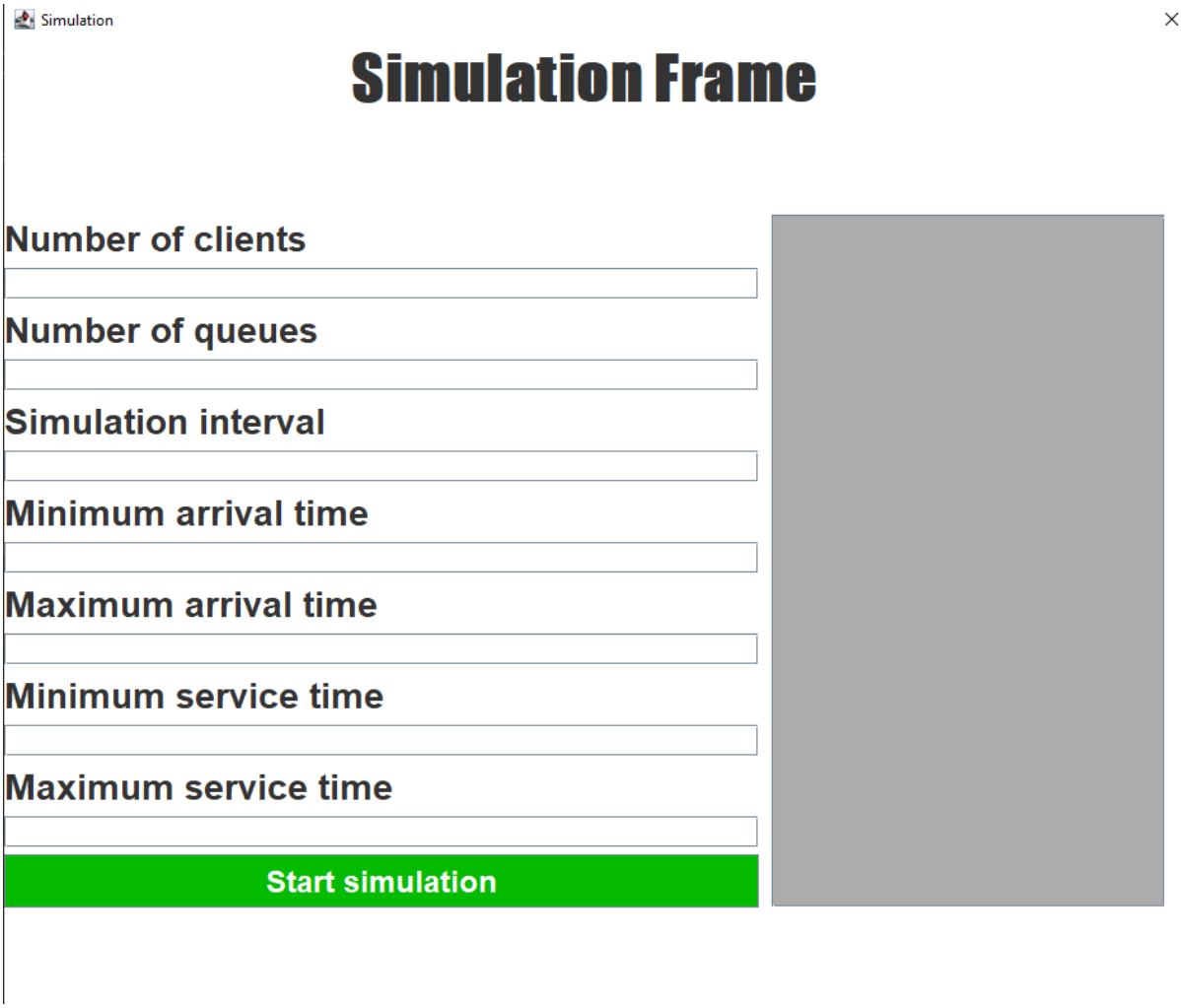
(VII)
```
package bussiness.logic;

      import gui.SimulationFrame;
      import model.Client;
      import model.Server;

      import javax.swing.*;
      import java.io.File;
      import java.io.FileNotFoundException;
      import java.io.PrintWriter;
      import java.util.ArrayList;
      import java.util.List;
      import java.util.Random;
      import java.util.concurrent.atomic.AtomicInteger;

      import static java.lang.Thread.currentThread;

      public class SimulationManager implements Runnable{
          private AtomicInteger timelimit=new AtomicInteger(0);
          public int minProcessingTime;
          public int maxProcessingTime;
          public int minArrivalTime;
          public int maxArrivalTime;
          public int numberOfClients;
          public int numberOfservers;

          private Scheduler scheduler;

          private String message="";
          private List<Client> clientList;

          private int ok=0;

          private JTextArea output;


          public SimulationManager(JTextArea jTextArea, int nrClients, int
      nrQ, Integer simTime, int minAtime, int maxAtime, int minPTime, int
      maxPTime){
              //initialize scheduler
              // => create and start numberOfServers threads
              // => initialize selection strategy=> createStrategy
```

```
            // genreate numberofClients using generateNRandomclients()
            // and store them to clientsList;


            this.output=jTextArea;
            this.numberOfClients=nrClients;

            this.timelimit.set(simTime);
            this.minArrivalTime=minAtime;
            this.maxArrivalTime=maxAtime;
            this.minProcessingTime=minPTime;
            this.maxProcessingTime=maxPTime;
            this.numberOfservers=nrQ;
            this.scheduler=new Scheduler(numberOfservers);
            for(int i=0; i< this.numberOfservers; i++){
                Thread serversTh= new
        Thread(this.scheduler.getServers().get(i));
                serversTh.start();
            }
            this.generrateNRandomClients(this.numberOfClients);




        }

        private void generrateNRandomClients(int numberOfClients){
            //generate N random tasks;
            // - random processing time
            // minProcessingTime< processing time < maxProcessingTime
            // -random arrivalTime
            // minArrivalTime< arrivalTime <  maxArrivalTime
            this.clientList=new ArrayList<>();
            Random rand= new Random();
            for(int i=1; i<=numberOfClients; i++){
                int pTime= rand.nextInt(maxProcessingTime-
        minProcessingTime+1) + minProcessingTime;
                int aTime= rand.nextInt(maxArrivalTime-minArrivalTime+1)
        + minArrivalTime;
                //int id=rand.nextInt(numberOfClients);
                Client client=new Client(i, aTime, pTime);
                clientList.add(client);

            }

        }

        @Override
        public void run() {
            try (PrintWriter writer = new PrintWriter("output.txt")) {
                AtomicInteger currentTime = new AtomicInteger(0);
                int peak = 0;
                int max = 0;
                int nrclients = 0;
                float avgWaitingTime = 0;
                float avgServiceTime = 0;
                float sum = 0;
                while (currentTime.intValue() < timelimit.intValue()) {
                    if (ok == 0) {
                        writer.println("Clients:");
                        System.out.println("Clients:");
```

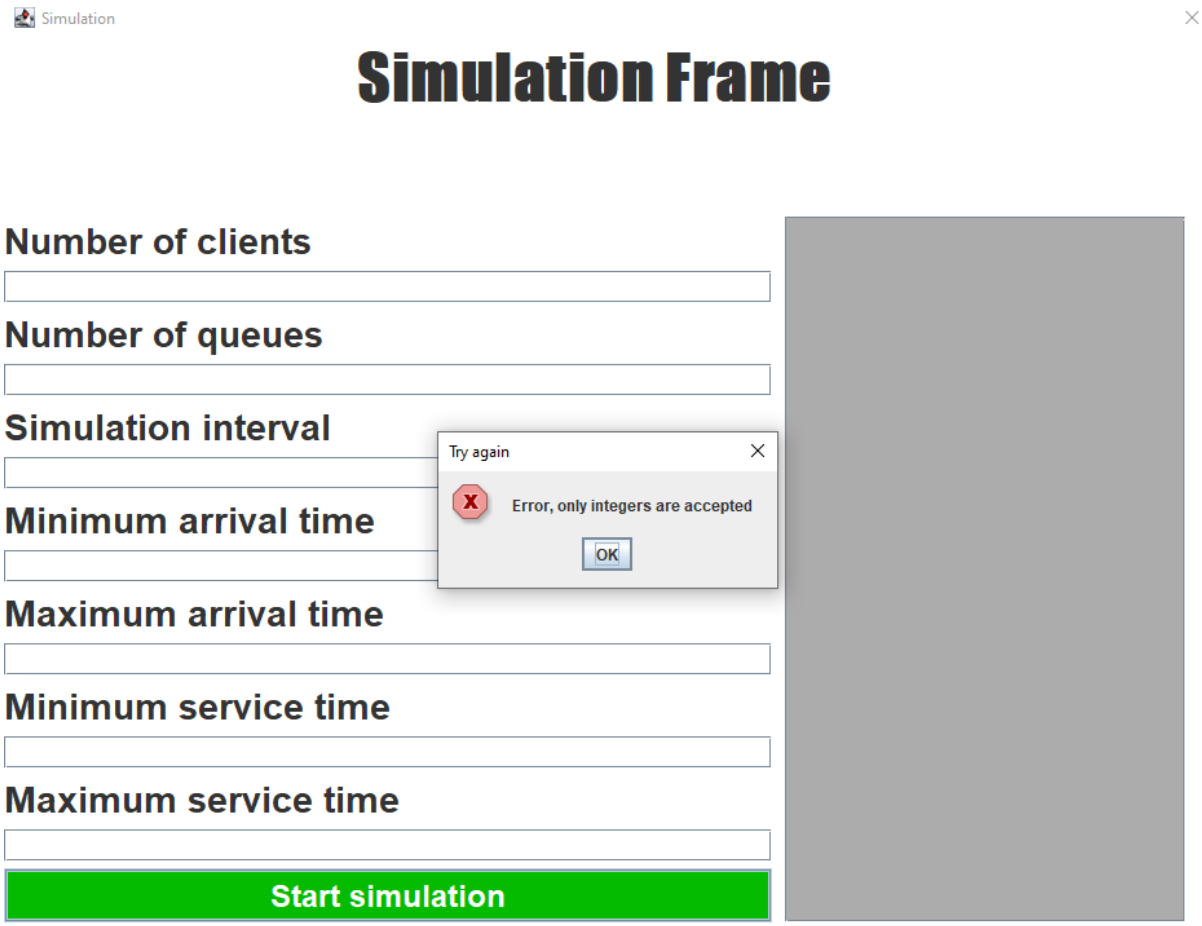
```
                    message+="Clients:\n";
                    for (Client client1 : clientList) {
                        writer.println(client1.toString() + ";");
                        System.out.println(client1.toString() + ";");
                        message+=client1.toString()+";"+"\n";
                        sum += client1.getServiceTime();
                    }
                    avgWaitingTime = sum / numberOfClients;
                    writer.println();
                    System.out.println();
                    message+="\n";
                    ok = 1;
                }

                List<Client> clientsToDispatch = new ArrayList<>();
                for (Client client : this.clientList) {
                    if (client.getArrivalTime() ==
    currentTime.intValue()) {
                        clientsToDispatch.add(client);
                    }
                }

                for (Client client : clientsToDispatch) {
                    scheduler.dispatchClient(client);
                    clientList.remove(client);
                }
                if(this.clientList.isEmpty()){
                    for (Server server : scheduler.getServers()) {

    //writer.println(server.getWaitingPeriod().intValue());

                        server.flagtoBreak=1;
                    }
                }

                for (Server server : scheduler.getServers()) {

    //writer.println(server.getWaitingPeriod().intValue());

                    nrclients = nrclients + server.getQueueSize();
                }
                if (nrclients > max) {
                    max = nrclients;
                    peak = currentTime.intValue();
                }
                nrclients = 0;
                writer.println("Current time: " + currentTime + "\n");
                writer.print("Waiting clients:");

                System.out.println("Current time: " + currentTime +
    "\n");
                System.out.print("Waiting clients:");

                message+="Current time: " + currentTime + "\n\n";
                message+="Waiting clients:";

                for (Client client : clientList) {
                    writer.print(client.toString() + "; ");
                    System.out.print(client.toString() + "; ");
                    message+=client.toString()+";";
                }
```

```
                writer.println();
                writer.println(scheduler.toString());

                System.out.println();
                System.out.println(scheduler.toString());


                message+="\n";
                message+=scheduler.toString()+"\n";
               output.setText(message);

                currentTime.getAndIncrement();
                try {
                    Thread.sleep(1000);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }


            }
            sum = 0;
            for (Server server : scheduler.getServers()) {
                sum += server.getTotalWaiting().intValue();
            }
            avgServiceTime = sum / numberOfClients;
            writer.println("Peak hour: " + peak);
            writer.println("Average service time: " + avgServiceTime);
            writer.println("Average waiting time: " + avgWaitingTime);
            System.out.println("Peak hour: " + peak);
            System.out.println("Average service time: " +
    avgServiceTime);
            System.out.println("Average waiting time: " +
    avgWaitingTime);
            message+="Peak hour: " + peak + "\n";
            message+="Average service time: " + avgServiceTime + "\n";
            message+="Average waiting time: " + avgWaitingTime + "\n";
            output.setText(message);
            writer.close();
            writer.flush();

        } catch (FileNotFoundException e) {
            e.printStackTrace();
        }
       //    System.exit(0);

    }

        public JTextArea getOutput() {
            return output;
        }
    }
```

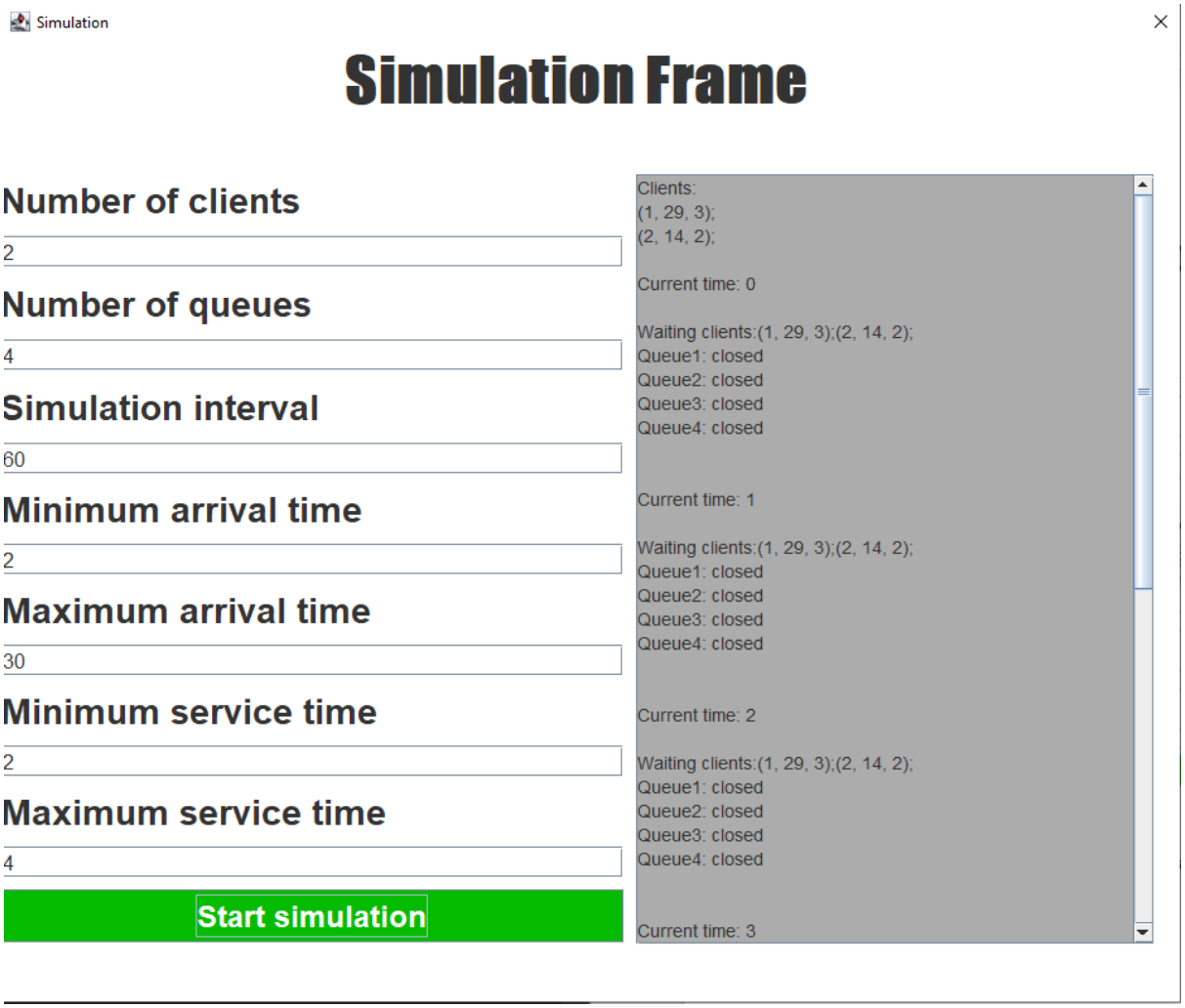
(VIII) The SimulationFrame



If the input is not correct the message will be shown



The text arrea will be filled with the log of events after the simulation starts



5. Results

The results can be seen on the log of events of this test: Clients:
The input is shown in the frame above:
(1, 10, 3);
(2, 7, 4);
(3, 21, 4);
(4, 22, 3);

Current time: 0

Waiting clients:(1, 10, 3); (2, 7, 4); (3, 21, 4); (4, 22, 3);
Queue1: closed
Queue2: closed

Current time: 1

Waiting clients:(1, 10, 3); (2, 7, 4); (3, 21, 4); (4, 22, 3);
Queue1: closed
Queue2: closed

Current time: 2

Waiting clients:(1, 10, 3); (2, 7, 4); (3, 21, 4); (4, 22, 3);
Queue1: closed
Queue2: closed

Current time: 3

Waiting clients:(1, 10, 3); (2, 7, 4); (3, 21, 4); (4, 22, 3);
Queue1: closed
Queue2: closed

Current time: 4

Waiting clients:(1, 10, 3); (2, 7, 4); (3, 21, 4); (4, 22, 3);
Queue1: closed
Queue2: closed

Current time: 5

Waiting clients:(1, 10, 3); (2, 7, 4); (3, 21, 4); (4, 22, 3);
Queue1: closed
Queue2: closed

Current time: 6

Waiting clients:(1, 10, 3); (2, 7, 4); (3, 21, 4); (4, 22, 3);
Queue1: closed
Queue2: closed

Current time: 7

Waiting clients:(1, 10, 3); (3, 21, 4); (4, 22, 3);
Queue1: (2, 7, 4);
Queue2: closed

Current time: 8

Waiting clients:(1, 10, 3); (3, 21, 4); (4, 22, 3);
Queue1: (2, 7, 4);
Queue2: closed

Current time: 9

Waiting clients:(1, 10, 3); (3, 21, 4); (4, 22, 3);
Queue1: (2, 7, 4);
Queue2: closed

Current time: 10

Waiting clients:(3, 21, 4); (4, 22, 3);
Queue1: (2, 7, 4);
Queue2: (1, 10, 3);

Current time: 11

Waiting clients:(3, 21, 4); (4, 22, 3);
Queue1: closed
Queue2: (1, 10, 3);

Current time: 12

Waiting clients:(3, 21, 4); (4, 22, 3);
Queue1: closed
Queue2: (1, 10, 3);

Current time: 13

Waiting clients:(3, 21, 4); (4, 22, 3);
Queue1: closed
Queue2: closed

Current time: 14

Waiting clients:(3, 21, 4); (4, 22, 3);
Queue1: closed
Queue2: closed

Current time: 15

Waiting clients:(3, 21, 4); (4, 22, 3);
Queue1: closed

Queue2: closed

Current time: 16

Waiting clients:(3, 21, 4); (4, 22, 3);
Queue1: closed
Queue2: closed

Current time: 17

Waiting clients:(3, 21, 4); (4, 22, 3);
Queue1: closed
Queue2: closed

Current time: 18

Waiting clients:(3, 21, 4); (4, 22, 3);
Queue1: closed
Queue2: closed

Current time: 19

Waiting clients:(3, 21, 4); (4, 22, 3);
Queue1: closed
Queue2: closed

Current time: 20

Waiting clients:(3, 21, 4); (4, 22, 3);
Queue1: closed
Queue2: closed

Current time: 21

Waiting clients:(4, 22, 3);
Queue1: (3, 21, 4);
Queue2: closed

Current time: 22

Waiting clients:

Queue1: (3, 21, 4);
Queue2: (4, 22, 3);

Current time: 23

Waiting clients:
Queue1: (3, 21, 4);
Queue2: (4, 22, 3);

Current time: 24

Waiting clients:
Queue1: (3, 21, 4);
Queue2: (4, 22, 3);

Current time: 25

Waiting clients:
Queue1: closed
Queue2: closed

Current time: 26

Waiting clients:
Queue1: closed
Queue2: closed

Current time: 27

Waiting clients:
Queue1: closed
Queue2: closed

Current time: 28

Waiting clients:
Queue1: closed
Queue2: closed

Current time: 29

Waiting clients:
Queue1: closed
Queue2: closed

Current time: 30

Waiting clients:
Queue1: closed
Queue2: closed

Current time: 31

Waiting clients:
Queue1: closed
Queue2: closed

Current time: 32

Waiting clients:
Queue1: closed
Queue2: closed

Current time: 33

Waiting clients:
Queue1: closed
Queue2: closed

Current time: 34

Waiting clients:
Queue1: closed
Queue2: closed

Current time: 35

Waiting clients:
Queue1: closed
Queue2: closed

Current time: 36

Waiting clients:
Queue1: closed
Queue2: closed

Current time: 37

Waiting clients:
Queue1: closed
Queue2: closed

Current time: 38

Waiting clients:
Queue1: closed
Queue2: closed

Current time: 39

Waiting clients:
Queue1: closed
Queue2: closed

Current time: 40

Waiting clients:
Queue1: closed
Queue2: closed

Current time: 41

Waiting clients:
Queue1: closed
Queue2: closed

Current time: 42

Waiting clients:
Queue1: closed
Queue2: closed

Current time: 43

Waiting clients:
Queue1: closed
Queue2: closed

Current time: 44

Waiting clients:
Queue1: closed
Queue2: closed

Current time: 45

Waiting clients:
Queue1: closed
Queue2: closed

Current time: 46

Waiting clients:
Queue1: closed
Queue2: closed

Current time: 47

Waiting clients:
Queue1: closed
Queue2: closed

Current time: 48

Waiting clients:
Queue1: closed
Queue2: closed

Current time: 49

Waiting clients:
Queue1: closed
Queue2: closed

Current time: 50

Waiting clients:
Queue1: closed
Queue2: closed

Current time: 51

Waiting clients:
Queue1: closed
Queue2: closed

Current time: 52

Waiting clients:
Queue1: closed
Queue2: closed

Current time: 53

Waiting clients:
Queue1: closed
Queue2: closed

Current time: 54

Waiting clients:
Queue1: closed
Queue2: closed

Current time: 55

Waiting clients:
Queue1: closed
Queue2: closed

Current time: 56

Waiting clients:
Queue1: closed
Queue2: closed

Current time: 57

Waiting clients:
Queue1: closed
Queue2: closed

Current time: 58

Waiting clients:
Queue1: closed
Queue2: closed

Current time: 59

Waiting clients:
Queue1: closed
Queue2: closed

Peak hour: 10
Average service time: 3.0
Average waiting time: 7.0

6. Conclusions

In conclusion, from this assignment, I learned how to use threads and how they work. How to procces the clients faster by using more queues. How threads affect the variables if they are not chose correctly.