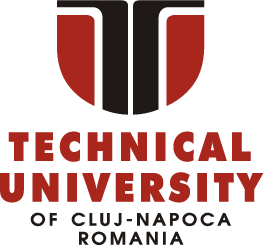
Technical University of Cluj-Napoca

Programming Techniques

Laboratory – Assignment 2

Queues Simulator

**

Teacher: prof. Ioan Salomie

Teacher Assistant: Cristina Bianca Pop

Student: Fleser Mihai

Group: E\_30424

1. Assignment Objective

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

* Analyze the problem and identify requirements …………………...………………. Chapter 2
* Design the Queues Simulator …..…………………………………………………. Chapter 3
* Implement the Queues Simulator ……………………………………….………….. Chapter 4
* Test the Queues Simulator ………………………………………………..……….. Chapter 5

1. Problem analysis, modeling, scenarios, use cases
   1. *Analyzing the problem*

PROBLEM: Simulate an application designed to analyze queueing based real-world systems

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.

The application should simulate (by defining a simulation time 𝑡𝑠𝑖𝑚𝑢𝑙𝑎𝑡𝑖𝑜𝑛) a series of N clients arriving for service, entering Q queues, waiting, being served and finally leaving the queues.

All clients are generated when the simulation is started, and are characterized by three parameters: ID (a number between 1 and N), 𝑡𝑎𝑟𝑟𝑖𝑣𝑎𝑙 (simulation time when they are ready to go to the queue; i.e. time when the client finished shopping) and 𝑡𝑠𝑒𝑟𝑣𝑖𝑐𝑒 (time interval or duration needed to serve the client; i.e. waiting time when the client is in front of the queue). The application tracks the total time spent by every client in the queues and computes the average waiting time. Each client is added to the queue with minimum waiting time when its 𝑡𝑎𝑟𝑟𝑖𝑣𝑎𝑙 time is greater than or equal to the simulation time (𝑡𝑎𝑟𝑟𝑖𝑣𝑎𝑙≥𝑡𝑠𝑖𝑚𝑢𝑙𝑎𝑡𝑖𝑜𝑛).

1. *Modelling the problem*

The user will be able to see how the simulation runs in real time on the interface. First, he will have to introduce the required data in the specific fields and then press the start button. After that, the clients will be randomly generated, depending on the input, and will be shown in the “console” area together with the given number of queues. As the time goes on, the console will show in each second how the clients are processed by the queues. The required input data is:

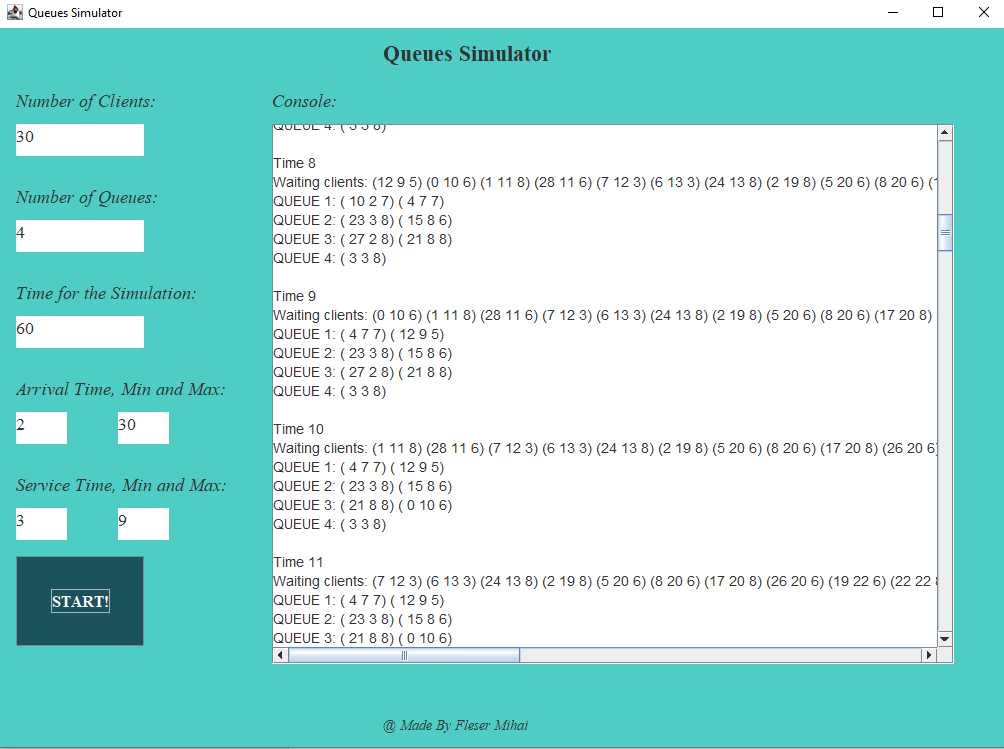
* Number of clients
* Number of queues
* Time for the simulation
* Minimum arrival time and Maximum arrival time
* Minimum service time and Maximum service time

Each client will have a random arrival and a random service time between the parameters specified above. The simulation will end when there are no more clients to be processed or when the time reaches the maximum time allowed from the user.

1. *Different scenarios and use cases*

Use cases are a technique for capturing, modelling and specifying the requirements of a system. A use case corresponds to a set of behaviors that the system may perform in interaction with its actors, and which produces an observable result that contribute to its goals. Actors represent the role that human users or other systems have in the interaction.

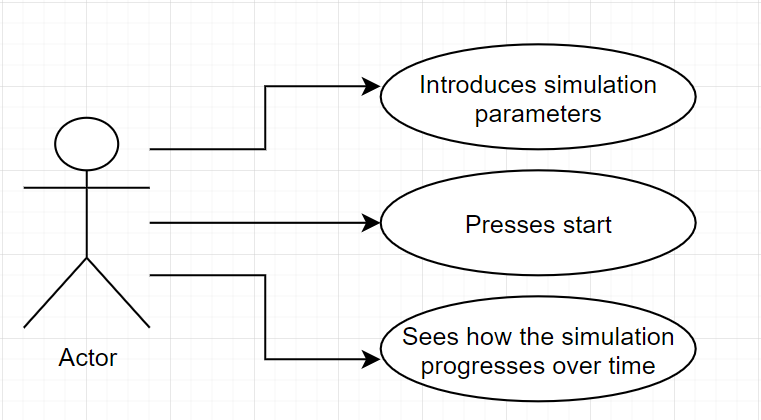
The use cases are strongly connected with the user steps. I tried to make the user interface as friendly as I could and the result looks like this:



The user will introduce all the data mentioned above in the corresponding text fields. After that, he will press the start button to begin the simulation.

The user must be careful to introduce the correct data in the GUI. He must only introduce numbers, because if he does not do that, an error will appear on the console saying that the input data is incorrect and the simulation will not start. After the simulation starts, if he presses the start button again, the simulation will restart itself with the new data introduced.

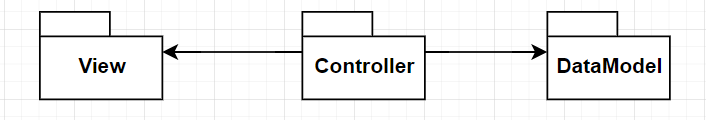
Use case Diagram:



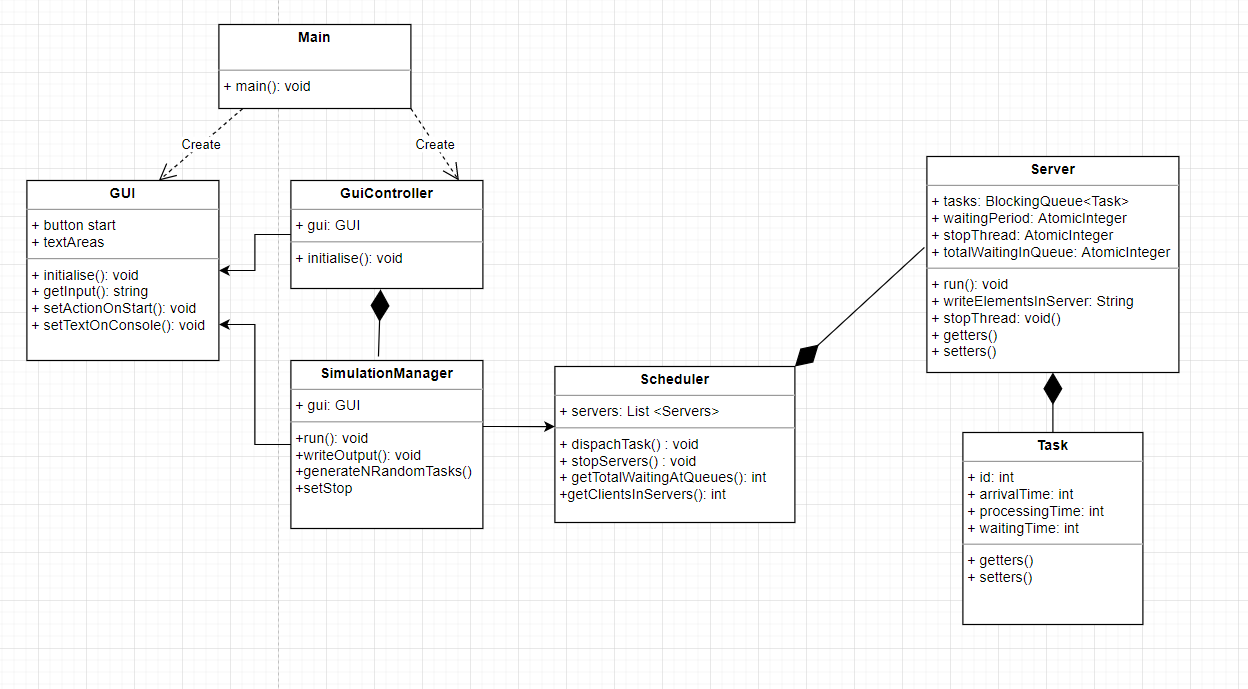
The use case presents the actor, which in our case is the user that interacts with the application. She/ He can start / restart the simulation and see in real time how it evolves.

* 1. Design

1. *UML package diagram*

**

* 1. *UML class Diagram*



* 1. *Data Structures*

For my project, I have used the following data Structures: AtomicInteger, BlockingQueue, List and String.

The AtomicInteger class protects an underlying int value by providing methods that perform atomicoperations on the value. I have used it as an atomic counter which is being used by multiple threads concurrently, meaning the waiting period that a client has to wait in order to be processed.

A blocking queue is a queue that blocks when you try to dequeue from it and the queue is empty, or if you try to enqueue items to it and the queue is already full. A thread trying to dequeue from an empty queue is blocked until some other thread inserts an item into the queue. A thread trying to enqueue an item in a full queue is blocked until some other thread makes space in the queue, either by dequeuing one or more items or clearing the queue completely. I have used it to store the tasks in the server.

The Scheduler uses a list of servers to simulate the application.

I used String for writing the output from the internal program to the user or for receiving the input from the user and then translating it into integers inside my program.

* 1. *Design Decisions and Packages*

I have done my project with the Design Pattern named model-View-Controller.

**Model–view–controller** (usually known as **MVC**) is a software design pattern commonly used for developing user interfaces that divides the related program logic into three interconnected elements. This is done to separate internal representations of information from the way information is presented to and accepted from the user.

**The Model** represents the central component of the pattern. It is the application's dynamic data structure, independent of the user interface. It directly manages the data, logic and rules of the application.

**The View** represents a collection of classes representing the elements in the user interface (all of the things the user can see and respond to on the screen, such as buttons, display boxes, etc.)

**The Controllers** receive input, usually as events that denote mouse movement, activation of mouse buttons or keyboard input. Events are translated to service requests, which are sent either to the model or to the view.

Therefore, I have created three packages containing one or more classes:

DataModel – the “brain” of the simulator, contains classes which store the data and perform different operations on it.

* Task class
* Server class

View – one class – The GUI class- interacts with the user.

Controller – 3 classes – the Scheduler, the SimulationManager and the GuiController

The entry point of the application in the Main class, which is separated from the others packages.

* 1. *Class Design*

As I said earlier, my Program consists of 3 parts, following the MCV architecture.

The Data Model:

* Task class – has 4 fields, meaning the id, arrivalTime, processingTime and the waitingTime
* Server class – has a queue of tasks which it has to process. It can add a new task in its queue. Each server run on a separate thread from another/

The View:

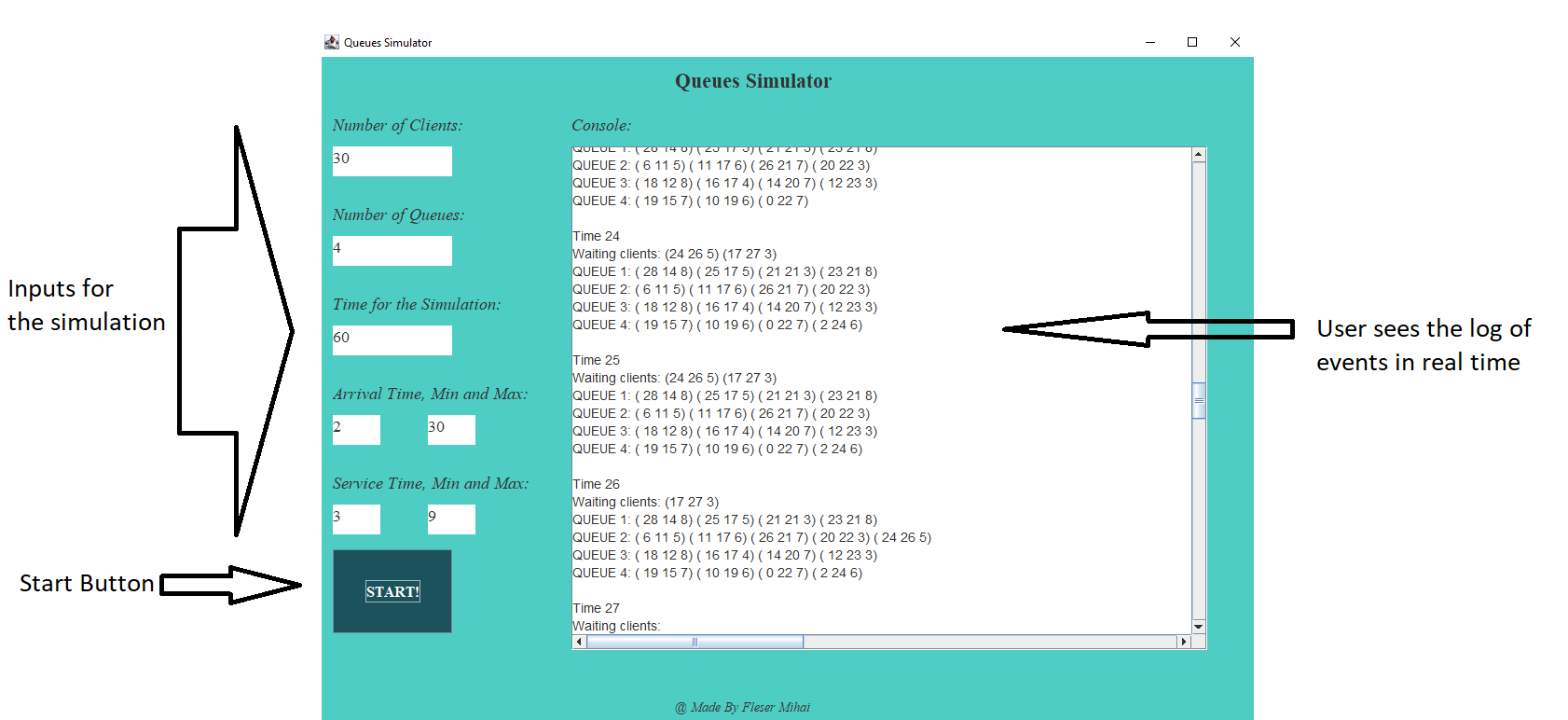
* Gui class – interacts with the user, takes the input and shows the output

The Controller:

* GuiController class – Handles the events that happens on the GUI, receives the input from the GUI and writes the output that the user wants to see. Is responsible for starting or stopping a simulation by commanding the SimulationManager.
* SimulationManager – generates the clients and tells the Scheduler when to add a new Task to the servers. Runs on a different Thread than the rest of the servers.
* Scheduler – Has a list of servers that he manages: starts or stops them and decide on which server to dispatch a new task.

The main class is the start point of the application, just creates a new GuiController which will do the rest.

* 1. *User Interface*



* 1. Implementation

As I said before, I used the Model-View-Controller.

**In the Model Package, I have implemented the following classes:**

public class Server implements Runnable{

private BlockingQueue<Task> tasks;

private AtomicInteger waitingPeriod;

private AtomicInteger stopThread;

private AtomicInteger totalWaitingInQueue;

public Server()

public int getWaitingPeriod()

public void stopThread()

public LinkedBlockingQueue getTasks()

public int getNumberOfTasks()

public void addTask(Task newTask)

@Override

public void run() { // This method is used for processing the tasks in the server.

while(stopThread.get() == 0)

{

try {

Task currentTask;

if(tasks.size() > 0)

{

Thread.sleep(tasks.peek().getProcessingTime() \* 1000);

currentTask = tasks.poll();

waitingPeriod.set(waitingPeriod.get() - currentTask.getProcessingTime());

for(Task t:tasks)

{

t.increaseWaitingTime(currentTask.getProcessingTime());

}

totalWaitingInQueue.set(totalWaitingInQueue.get() + currentTask.getWaitingTime());

}

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

public int getTotalWaitingInQueues()

public String writeElementsInServer()

}

public class Task{

private int id;

private int arrivalTime;

private int processingTime;

private int waitingTime;

public Task(int id, int arrivalTime, int processingTime)

public int getId()

public int getWaitingTime()

public void increaseWaitingTime(int amount)

public int getArrivalTime()

public int getProcessingTime()

public static Comparator<Task> monomialComparator = new Comparator<Task>() {

public int compare(Task t1, Task t2) {

//ascending order

if(t1.getArrivalTime() > t2.getArrivalTime())

return 1;

else if (t1.getArrivalTime() == t2.getArrivalTime())

return 0;

else

return -1;

}};

}

**In the Controller Package, I have:**

public class GuiController {

private GUI gui;

private SimulationManager simulationManager;

public GuiController(GUI gui)

public void initialise() // this method starts the simulation and sets an action on button

{

gui.initialise();

gui.setActionOnStartButton(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

try{

simulationManager.setStop(1);

gui.setTextOnConsole("");

int timeLimit = Integer.parseInt(gui.getInputTimeText());

int maxProcessingTime = Integer.parseInt(gui.getInputServiceMaxText());

int minProcessingTime = Integer.parseInt(gui.getInputServiceMinText());

int maxArrivalTime = Integer.parseInt(gui.getInputArrivalMaxText());

int minArrivalTime = Integer.parseInt(gui.getInputArrivalMinText());

int numberOfServers = Integer.parseInt(gui.getInputQueuesText());

int numberOfTasks = Integer.parseInt(gui.getInputClientsText());

simulationManager = new SimulationManager(gui,timeLimit,maxProcessingTime,minProcessingTime,numberOfServers,numberOfTasks,minArrivalTime,maxArrivalTime);

Thread thread = new Thread(simulationManager);

thread.start();

} catch (Exception exception)

{

simulationManager.setStop(1);

gui.addTextOnConsole("Error: can't parse one of the number\n");}

}

public class SimulationManager implements Runnable{

private GUI gui;

private String file;

private int peakHour;

private int maxClientsInQueues;

private float avarageWaitingTime;

private int timeLimit;

private int maxProcessingTime;

private int minProcessingTime;

private int maxArrivalTime;

private int minArrivalTime;

private int numberOfServers;

private int numberOfTasks;

private float avarageServiceTime;

private int stop;

private Scheduler scheduler;

private List<Task> generatedTasks;

public SimulationManager(GUI gui, int timeLimit, int maxProcessingTime, int minProcessingTime, int numberOfServers, int numberOfTasks, int minArrivalTime, int maxArrivalTime)

private void generateNRandomTasks()

{

for(int i = 0 ; i < numberOfTasks; i++)

{

int randomArrival;

int randomProcessing;

Random random = new Random();

randomArrival = random.nextInt(maxArrivalTime - minArrivalTime) + minArrivalTime;

randomProcessing = random.nextInt(maxProcessingTime - minProcessingTime) + minProcessingTime;

Task task = new Task(i,randomArrival,randomProcessing);

generatedTasks.add(task);

avarageServiceTime += randomProcessing;

}

avarageServiceTime = avarageServiceTime/ numberOfTasks;

}

public void setStop(int stop)

public void writeOutput(int currentTime, boolean lastTime) //writes output on console

@Override // this is the main function of the simulator.

public void run() { // dispatches the task, calculates the statistics and

PrintWriter writer = null; // writes on the GUI the log of events

try {

writer = new PrintWriter(file);

} catch (FileNotFoundException e) {

e.printStackTrace();

}

writer.print("");

writer.close();

int currentTime = 0;

int currentClientsInQueues;

while (currentTime < timeLimit && (generatedTasks.size() > 0 || scheduler.getClientsInServers() > 0) && stop == 0)

{

while(generatedTasks.size() > 0 && generatedTasks.get(0).getArrivalTime() == currentTime)

{

Task task = generatedTasks.remove(0);

scheduler.dispachTask(task);

}

currentClientsInQueues = scheduler.getClientsInServers();

if(currentClientsInQueues > maxClientsInQueues)

{

peakHour = currentTime;

maxClientsInQueues = currentClientsInQueues;

}

writeOutput(currentTime,false);

currentTime++;

try {

Thread.sleep(1000);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

if(stop == 0)

{

avarageWaitingTime = scheduler.getTotalWaitingAtQueues() / (float)(numberOfTasks - generatedTasks.size() - scheduler.getClientsInServers());

writeOutput(currentTime,true);

}

scheduler.stopServers();

}

}

public class Scheduler {  
 private List<Server> servers;  
 private int maxNoServers;  
 private int maxTasksPerServer;  
  
 public Scheduler(int maxNoServers, int maxTasksPerServer)  
  
 private Server getServerWithMinimumWaitingTime()  
  
 public int getClientsInServers()  
  
 public int getTotalWaitingAtQueues()  
  
 public void dispachTask(Task newTask)  
  
 public void stopServers()  
  
 public List<Server> getServers()  
  
}

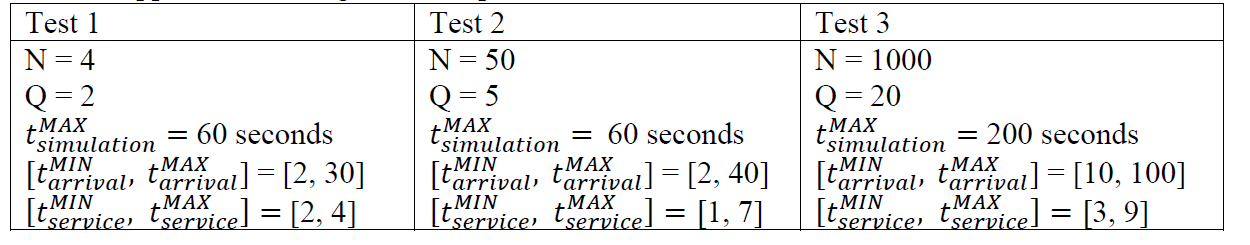
**In the View Package, I have the GUI class**  
  
public class GUI {  
  
 private JTextArea inputClientsText = new JTextArea();  
 private JTextArea inputQueuesText = new JTextArea();  
 private JTextArea inputTimeText = new JTextArea();  
 private JTextArea inputArrivalMin = new JTextArea();  
 private JTextArea inputArrivalMax = new JTextArea();  
 private JTextArea inputServiceMin = new JTextArea();  
 private JTextArea inputServiceMax = new JTextArea();  
 private JTextArea consoleText = new JTextArea();  
 private JButton start=new JButton("START!");  
  
 public GUI()  
  
 public String getInputClientsText()  
  
 public String getInputQueuesText()  
  
 public String getInputTimeText()  
  
 public String getInputArrivalMinText()  
  
 public String getInputArrivalMaxText()  
  
 public String getInputServiceMinText()  
  
 public String getInputServiceMaxText()  
  
 public void setActionOnStartButton(ActionListener actionListener)  
  
 public void addTextOnConsole(String s)  
  
 public void setTextOnConsole(String s)  
  
 public void initialise() //draws the GUI

And the Main class, the start point of the application:

public class Main {  
 public static void main(String[] args)  
 {  
 GUI gui = new GUI();  
 GuiController controller = new GuiController(gui);  
 controller.initialise();  
 }  
}

* 1. Results

I have run my application on the following input date sets from the table below:



The logs on events are generated in the text files included in the project, meaning “test1.txt”, “test2.txt”, “test3.txt”.

* 1. Conclusions

I liked this Project because it helped me to understand better how to work with threads and further develop my OOP knowledge. I have learned, again, that time management is quite important to not be stressed out and do the project in time. The previous assignment helped me a lot as I did not repeat some of my mistakes when making this one.

Future improvements:

* One idea will be to make a pause and a restart button for the simulation, enabling the user to have more control on the simulation.
* Another one will be to make it run as fast as possible without causing any trouble, or threads desynchronizing and so on.
  1. Bibliography
* Object-Oriented Programming - Lecture Slides of prof. Marius JOLDOS
* Programming Techniques – Lectures of prof. Ioan SALOMIE
* <https://docs.oracle.com/javase/tutorial/uiswing/>
* <https://docs.oracle.com/javafx/2/get_started/jfxpub-get_started.htm>
* <https://www.baeldung.com/javafx>
* <https://www.vogella.com/tutorials/JUnit/article.html>
* <https://stackoverflow.com/>
* <http://tutorials.jenkov.com/java-concurrency/>
* <https://howtodoinjava.com/java/multi-threading/>