Technical University of Cluj-Napoca

Faculty of Automation and Computer Science

2nd Semester 2016-2017

Programming Techniques

Homework 2

Student: Andrei Branga

Group: 30422

Table of contents

1. Assignment objectives

(EN) Lab – Homework 2

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;

2. Problem analysis, modelling, scenarios, use cases

2.1. Problem analysis

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.

2.2. Modelling

A)Client

```
public class Client
{
    private UUID clientID;
    private int serviceAmount; //in service units


    public Client(){
        this.clientID=IDgenerator.getUniqueID();
        this.serviceAmount=0;
    }

    public Client(int serviceAmount)
    {
       this.clientID=IDgenerator.getUniqueID();
       this.serviceAmount=serviceAmount;
    }

    public int getServiceAmount() {
```

```
        return serviceAmount;
    }

    public void setServiceAmount(int serviceAmount) {
        this.serviceAmount = serviceAmount;
    }

    public UUID getClientID() {
        return clientID;
    }

    @Override
    public String toString() {
        return "Client{" +
                "clientID=" + clientID +
                ", serviceAmount=" + serviceAmount +
                " (serviceunits) }";
    }
}
```

This class holds basic data about the client (the id and the service amount needed).

B)Client Profile

```
public class Client_Profile {
    private Client client;
    private volatile LocalTime arrivalTime;
    private volatile LocalTime takenInServiceTime;
    private volatile LocalTime leavingTime;
    private volatile int
waitingInQueue,waitingToBeServiced,totalWaitingTime;//in seconds
    private boolean serviced=false;
    private boolean takenInService=false;

    public Client_Profile(){};

    public Client_Profile(Client client)
    {
        this.client=client;
        arrivalTime=LocalTime.now();
    }

    public void takeInService()
    {
        takenInService=true;
        takenInServiceTime=LocalTime.now();
        waitingInQueue=takenInServiceTime.getSecond()-
arrivalTime.getSecond();
    }

    public void finnishService()
    {
        serviced=true;
        leavingTime=LocalTime.now();
        waitingToBeServiced=leavingTime.getSecond()-
takenInServiceTime.getSecond();
```

```
        totalWaitingTime=waitingInQueue+waitingToBeServiced;
    }

    public Client getClient() {
        return client;
    }

    public LocalTime getArrivalTime() {
        return arrivalTime;
    }

    public LocalTime getTakenInServiceTime() {
        return takenInServiceTime;
    }

    public LocalTime getLeavingTime() {
        return leavingTime;
    }

    public int getWaitingInQueue() {

        if (takenInService)
        {
            return waitingInQueue;
        }
        return 0;
    }

    public int getWaitingToBeServiced() {
        return waitingToBeServiced;
    }

    public int getTotalWaitingTime() {
        return totalWaitingTime;
    }

    public boolean isServiced() {
        return serviced;
    }

    public boolean isTakenInService() {
        return takenInService;
    }

    @Override
    public String toString() {
        return "Client_Profile{" +
                "client=" + client.toString() +
                ", arrivalTime=" + arrivalTime.toString() +
                ", takenInServiceTime=" + takenInServiceTime.toString() +
                ", leavingTime=" + leavingTime.toString() +
                ", waitingInQueue=" + waitingInQueue/60 +" minutes " +
                ", waitingToBeServiced=" + waitingToBeServiced/60 +"
minutes " +
                ", totalWaitingTime=" + totalWaitingTime/60 +" minutes "+
                ", serviced=" + serviced +
                ", takenInService=" + takenInService +
                '}';
    }

    public int getServiceAmount()
```

```
    {
        return client.getServiceAmount();
    }

}
```

Now, the basic client is taken and for him is created a client profile with all the data related to servicing him.

This data is manipulated by the servicers.

C)Client Queue

```
public class ClientQueue {
    Queue<Client_Profile> clientQueue=new LinkedList<Client_Profile>();
    private int queueSize=-1;
    LocalTime peakHour;
    public ClientQueue(){};

    public synchronized  void enqueue(Client_Profile client_profile)
    {
        clientQueue.add(client_profile);
        if(clientQueue.size()>queueSize)
        {
            peakHour=LocalTime.now();
            queueSize=clientQueue.size();
        }
    }

    public synchronized Client_Profile dequeue()
    {
        Client_Profile client_profile=clientQueue.remove();
        return client_profile;
    }

    public int getSize()
    {
        return clientQueue.size();
    }

    public int getWaitingTime()
    {
        LocalTime time=LocalTime.now();
        int total=0;
        for (Client_Profile cp:clientQueue
             ) {
            total+=(time.getSecond()-cp.getArrivalTime().getSecond());

        }
        return total;
    }

    public int getServiceAmount()
    {
        int amount=0;
        for (Client_Profile cp:clientQueue
```

```
             ) {amount+=cp.getServiceAmount();

        }
        return  amount;
    }

    public LocalTime getPeakHour()
    {
        return peakHour;
    }

    public boolean isEmpty()
    {
        return clientQueue.isEmpty();
    }

    @Override
    public String toString() {
        String ret="ClientQueue: \n";
        if(clientQueue.isEmpty())
        {
            ret+="<<EMPTY QUEUE>>\n";
            return ret;
        }
        for (Client_Profile c:clientQueue
             )
{ret+="..................................................\nclient id:
"+c.getClient().getClientID()+"\n";
             ret+="arrival time: "+c.getArrivalTime().toString()+"\n";
             ret+="current wating time: "+(LocalTime.now().getSecond()-
c.getArrivalTime().getSecond())/60;
             ret+="\n..................................................";

        }
        return ret;
    }


}
```

For every servicer, a client queue is created to manage.

Clients are taken from the client pool and assigned to the available queues.

D)Servicer

```
public class Servicer {
    ArrayList<Client_Profile> client=new ArrayList<Client_Profile>();
    private int serviceUnitsPerSec;
    private boolean isBusy=false;
    public volatile int noOfClients=0;
    public Servicer(){}
```

```
    public Servicer(int serviceUnitsPerSec)
    {
        this.serviceUnitsPerSec=serviceUnitsPerSec;
    }

    public void takeInService(Client_Profile client_profile)
    {
        if(client.isEmpty())
        {
            this.client.add(client_profile);
            isBusy=true;
            client.get(0).takeInService();
            noOfClients++;
        }
    }
    public boolean isDone()
    {
        if(client.isEmpty())
        {
            return true;
        }
        return client.get(0).isServiced();
    }

    public void service()
    {
        if (!isDone() && !client.isEmpty())
        {

client.get(0).getClient().setServiceAmount(client.get(0).getClient().getSer
viceAmount()-serviceUnitsPerSec);
        }
    }

    public Client_Profile freeClient()
    {
        client.get(0).finnishService();
        isBusy=false;
        return client.remove(0);
    }

    public synchronized int getNoOfClients() {
        return noOfClients;
    }

    public Client_Profile getClient() {
        if(client.isEmpty())
        {
            return null;
        }
        else
        {
            return client.get(0);
        }
    }

    public synchronized boolean isBusy() {
        return isBusy;
    }
}
```

Every queue has its own servicer which manages it.

E)Logger Manager

```
public class LoggerManager {

    public static String servicesOpen()
    {
        return LocalTime.now().toString()+" "+"Queues and services are now
open... \n";
    }

    public static String servicesClosed()
    {
        return LocalTime.now().toString()+" "+"All services are now
closed... \n";
    }

    public static String newClientInQueue(Client_Profile client_profile,int
i)
    {
        return LocalTime.now().toString()+" "+"New client in queue
"+i+":"+client_profile.getClient().getClientID()+" ...\n";
    }

    public static String newClientInService(Client Profile client profile)
    {
        return LocalTime.now().toString()+" "+"New client in service
:"+client_profile.getClient().getClientID()+" ...\n";
    }

    public static String clientLeft(Client_Profile client_profile)
    {
        return LocalTime.now().toString()+" "+"Client left
:"+client_profile.getClient().toString()+" ...\n";
    }

    public static String servicerNoOfClients(int i,Servicer s)
    {
        return LocalTime.now().toString()+" "+"Servicer "+i+" number of
clients serviced: "+s.getNoOfClients()+"...\n";
    }

    public static String peakHour(ClientQueue q)
    {
        return "Peak hour: "+q.getPeakHour().toString()+"\n";
    }
}
```

Logger manager was created for easying the logging processes.

F)QueueServicerPair

```
public class QueueServicerPair implements Runnable
{
    public LocalTime startSimulation=LocalTime.now();
    public LocalTime endSimulation;
    public volatile ClientQueue queue1=new ClientQueue();
    public volatile Servicer servicer1=new Servicer(5);
    public volatile double avgWaitingTime=0;
    public volatile double waitingtime=0;

    public QueueServicerPair(int SecOfSimulation)
    {
        this.endSimulation=LocalTime.now().plusSeconds(SecOfSimulation);
    }


    @Override
    public void run() {
       while (LocalTime.now().isBefore(endSimulation))
       {
           if(!queue1.isEmpty() && servicer1.isDone())
           {
               servicer1.takeInService(queue1.dequeue());

System.out.println(LoggerManager.newClientInService(servicer1.getClient()))
;
               try
               {
                   String filename= "log.txt";
                   FileWriter fw = new FileWriter(filename,true); //the
true will append the new data

fw.write(LoggerManager.newClientInService(servicer1.getClient()));//appends
the string to the file
                   fw.close();
               }
               catch(IOException ioe)
               {
                   System.err.println("IOException: " + ioe.getMessage());
               }
           }
           try {
               Thread.sleep(1000);
           } catch (InterruptedException e) {
               e.printStackTrace();
           }
           if(!servicer1.isDone())
           {

               servicer1.service();
           }
           if(servicer1.getClient()!=null
&&servicer1.getClient().getServiceAmount()<=0)
           {
               Client_Profile cp=servicer1.freeClient();
               int aux=cp.getWaitingInQueue();
               if(aux>=0)
               {
```

```
                    waitingtime+=aux;

                    if(servicer1.getNoOfClients()!=0)
                    {

avgWaitingTime=waitingtime/servicer1.getNoOfClients();
                    }
                    else
                    {
                        avgWaitingTime=0;
                    }
                }
                System.out.println(LoggerManager.clientLeft(cp));
                try
                {
                    String filename= "log.txt";
                    FileWriter fw = new FileWriter(filename,true); //the
true will append the new data
                    fw.write(LoggerManager.clientLeft(cp));//appends the
string to the file
                    fw.close();
                }
                catch(IOException ioe)
                {
                    System.err.println("IOException: " + ioe.getMessage());
                }
            }
        }
    }

    public synchronized double getAvgWaitingTime() {
        return Math.floor(avgWaitingTime * 100) / 100;
    }

    public ClientQueue getQueue1() {
        return queue1;
    }

    public Servicer getServicer1() {
        return servicer1;
    }

    public void setQueue1(ClientQueue queue1) {
        this.queue1 = queue1;
    }

    public void setServicer1(Servicer servicer1) {
        this.servicer1 = servicer1;
    }
}
```

This class offers the most important functionality for my application.

It implements the Runnable interface which allows the application to run a thread based on it.

It connects the servicers to their queues and runs the process of servicing.

G)Controller

```
public class Controller
{
    @FXML
    private Text statusf;

    @FXML
    private Text nrq1;

    @FXML
    private Text nrq2;

    @FXML
    private Text nrq3;

    @FXML
    private Text nrq4;

    @FXML
    private Text nrq5;

    @FXML
    private Text status1;

    @FXML
    private Text nr1;

    @FXML
    private Text status2;

    @FXML
    private Text nr2;

    @FXML
    private Text status3;

    @FXML
    private Text nr3;

    @FXML
    private Text status4;

    @FXML
    private Text nr4;

    @FXML
    private Text status5;

    @FXML
    private Text nr5;

    @FXML
```

```
    private TextField secOfSimulation;

    @FXML
    private Button startSimulation;
```

Controller is the class that controls the GUI.

It implements all the program functionality through the startSimulation() method which reacts to pressing the start button in the GUI.

2.3. Scenarios, use cases

The application is designed to allow the user to simulate 5 queues assigned to 5 servicers. By selecting the simulation duration then hitting start button the whole process of such a system starts:

-people arrive in a client pool

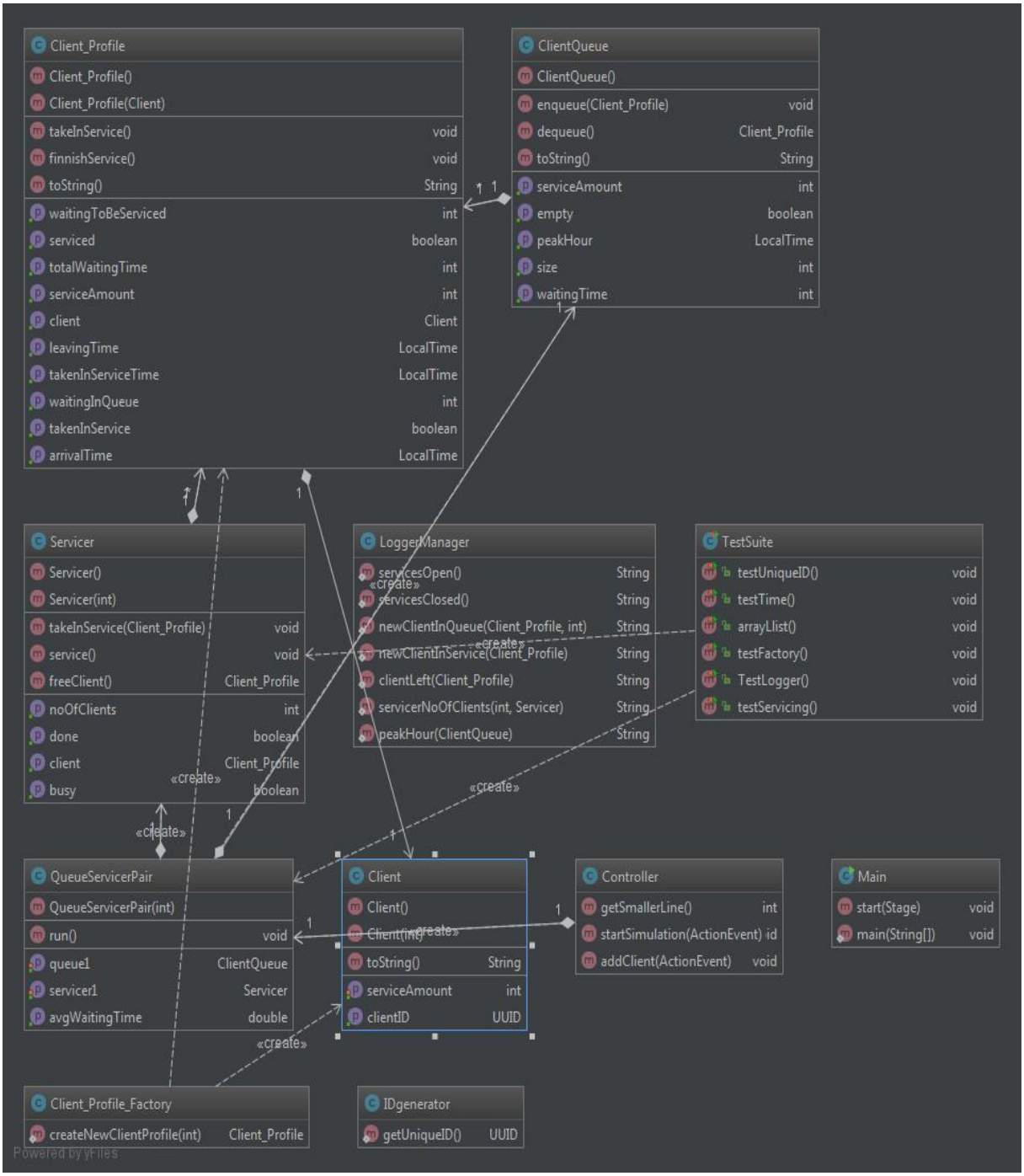
-from there a thread takes them and assigns them to the most short queue

-a servicer services the clients in the queue one at a time with a given speed(5 service amount units per sec)

-everything that happens in the system is carefully logged both in a log.txt file as well as in the application console but also in the GUI.

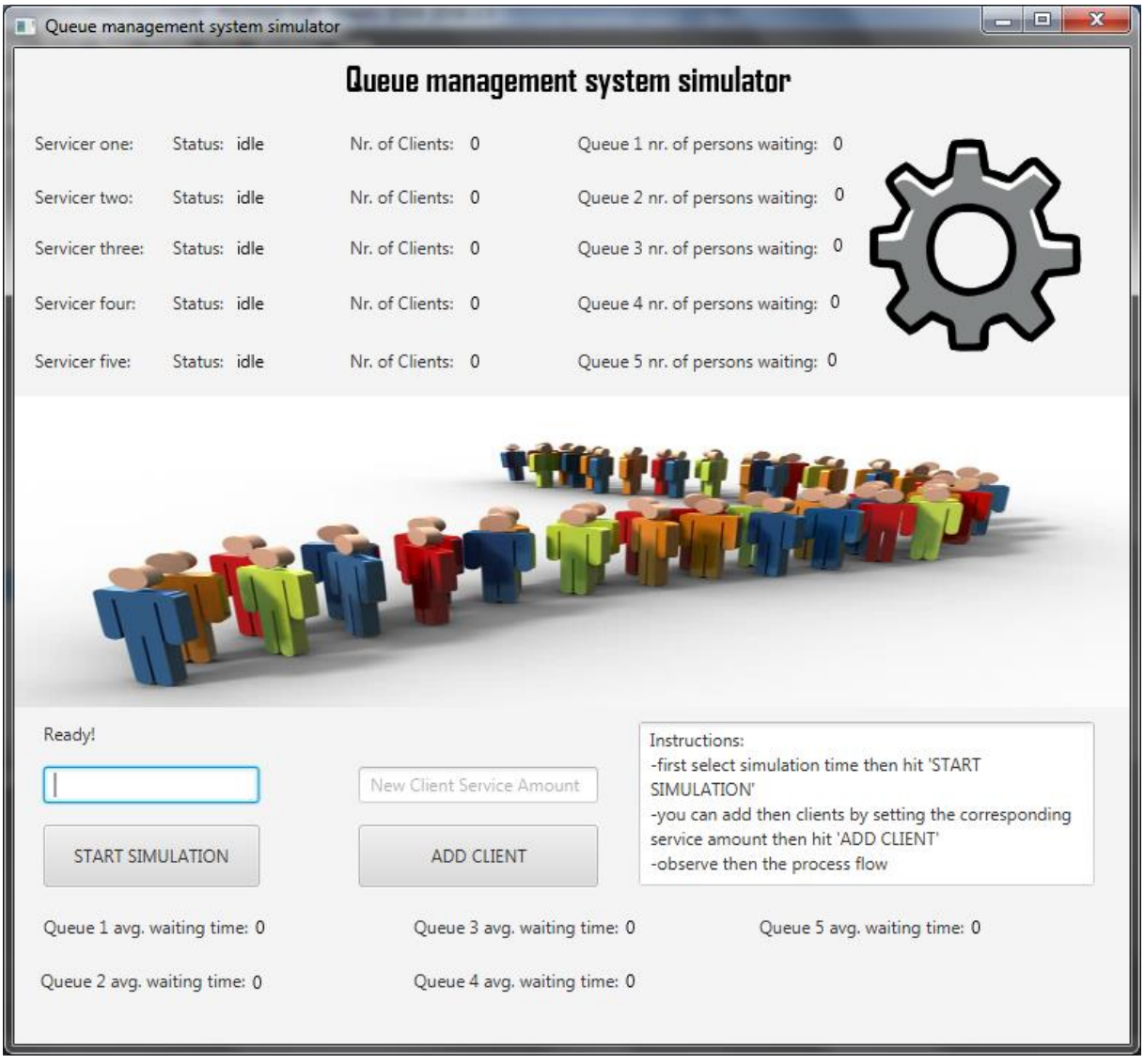
3. Design

3.1. UML diagram



3.2. Graphical User Interface (GUI)

The graphical user interface is designed to be easy to use, even for non-specialists.



Instructions are to be found in the bottom-right corner.

All sorts of details are displayed in order for the user to analyze the simulation.

ALL INPUT AND OUTPUT TIME UNITS ARE SECONDS!

A)Java fx

JavaFX is a software platform for creating and delivering desktop applications, as well as rich internet applications (RIAs) that can run across a wide variety of devices. JavaFX is intended to replace Swing as the standard GUI library for Java SE, but both will be included for the foreseeable future.[3] JavaFX has support for desktop computers and web browsers on Microsoft Windows, Linux, and macOS.

Before version 2.0 of JavaFX, developers used a statically typed, declarative language called JavaFX Script to build JavaFX applications. Because JavaFX Script was compiled to Java bytecode, programmers could also use Java code instead. JavaFX applications could run on any desktop that could run Java SE, on any browser that could run Java EE, or on any mobile phone that could run Java ME.

JavaFX 2.0 and later is implemented as a "native" Java library, and applications using JavaFX are written in "native" Java code. JavaFX Script has been scrapped by Oracle, but development is being continued in the Visage project.[4] JavaFX 2.x does not support the Solaris operating system or mobile phones; however, Oracle plans to integrate JavaFX to Java SE Embedded 8, and Java FX for ARM processors is in developer preview phase.[5]

On desktops, JavaFX supports Windows Vista, Windows 7, Windows 8, Windows 10, macOS and Linux operating systems.[6] Beginning with JavaFX 1.2, Oracle has released beta versions for OpenSolaris.[7] On mobile, JavaFX Mobile 1.x is capable of running on multiple mobile operating systems, including Symbian OS, Windows Mobile, and proprietary real-time operating systems.

B)Scene Builder

JavaFX Scene Builder is a visual layout tool that lets users quickly design JavaFX application user interfaces, without coding. Users can drag and drop UI components to a work area, modify their properties, apply style sheets, and the FXML code for the layout that they are creating is automatically generated in the background. The result is an FXML file that can then be combined with a Java project by binding the UI to the application's logic.

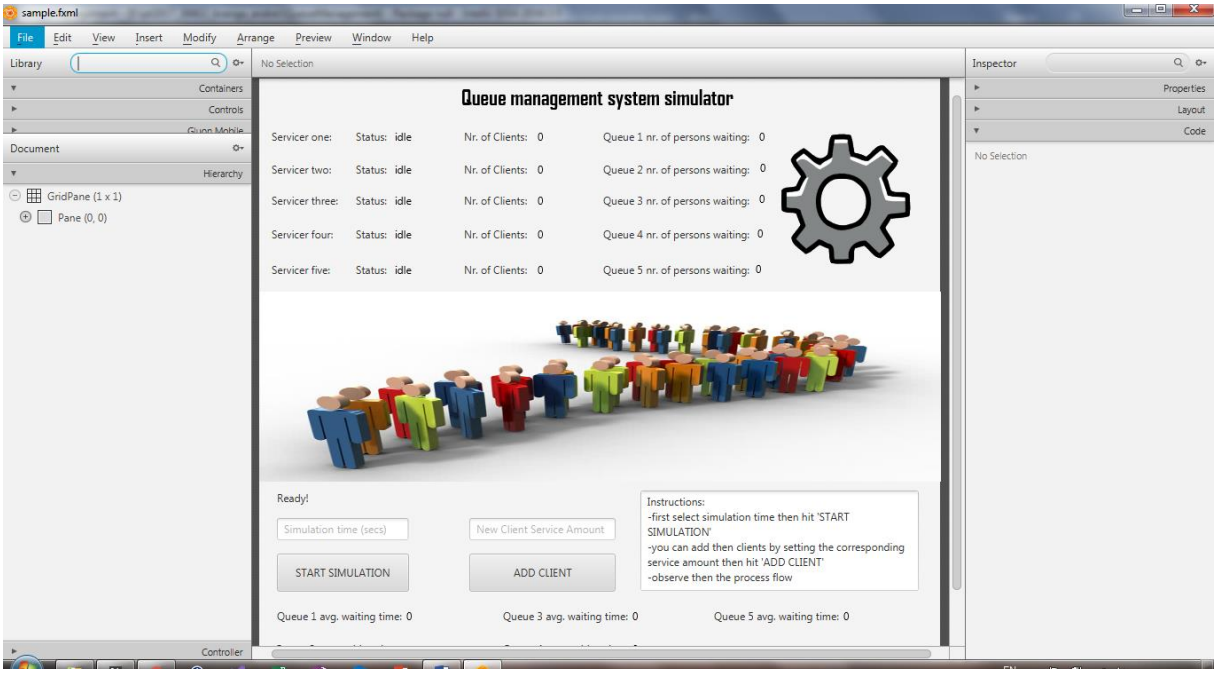
Scene Builder allows you to easily layout JavaFX UI controls, charts, shapes, and containers, so that you can quickly prototype user interfaces. Animations and effects can be applied seamlessly for more sophisticated UIs.

Scene Builder generates FXML, an XML-based markup language that enables users to define an application's user interface, separately from the application logic. You can also open and edit existing FXML files authored by other users.

Scene Builder can be used in combination with any Java IDE, but is more tightly integrated with NetBeans IDE. You can bind the UI to the source code that will handle the events and actions taken on each element through a simple process, run your application in NetBeans, and any changes to FXML in NetBeans will also reflect in your Scene Builder project.

Scene Builder is written as a JavaFX application, supported on Windows, Mac OS X and Linux. It is the perfect example of a full-fledge JavaFX desktop application. Scene Builder is packaged as a self contained application, which means it comes bundled with its own private copy of the JRE.

Example:



The resulted fxml:

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.control.Button?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.control.TextArea?>
<?import javafx.scene.control.TextField?>
<?import javafx.scene.image.Image?>
<?import javafx.scene.image.ImageView?>
<?import javafx.scene.layout.ColumnConstraints?>
<?import javafx.scene.layout.GridPane?>
<?import javafx.scene.layout.Pane?>
<?import javafx.scene.layout.RowConstraints?>
<?import javafx.scene.text.Font?>
<?import javafx.scene.text.Text?>

<GridPane alignment="center" hgap="10" minHeight="100.0" minWidth="100.0"
prefHeight="686.0" prefWidth="768.0" vgap="10"
xmlns="http://javafx.com/javafx/8.0.111"
xmlns:fx="http://javafx.com/fxml/1"
fx:controller="GUI_Controller.Controller">
   <columnConstraints>
      <ColumnConstraints />
   </columnConstraints>
   <rowConstraints>
      <RowConstraints />
   </rowConstraints>
   <children>
      <Pane prefHeight="696.0" prefWidth="768.0">
         <children>
            <Label fx:id="status" layoutX="20.0" layoutY="464.0"
```

```
prefHeight="17.0" prefWidth="381.0" text="Ready! " />
            <TextField fx:id="secOfSimulation" layoutX="20.0"
layoutY="495.0" promptText="Simulation time (secs)" />
            <Button fx:id="startSimulation" layoutX="20.0" layoutY="535.0"
mnemonicParsing="false" onAction="#startSimulation" prefHeight="43.0"
prefWidth="149.0" text="START SIMULATION" />
            <Button fx:id="addClient" layoutX="237.0" layoutY="535.0"
mnemonicParsing="false" onAction="#addClient" prefHeight="43.0"
prefWidth="165.0" text="ADD CLIENT" />
            <TextField fx:id="clientServiceAmount" layoutX="237.0"
layoutY="495.0" prefHeight="25.0" prefWidth="165.0" promptText="New Client
Service Amount" />
            <TextArea editable="false" layoutX="430.0" layoutY="464.0"
prefHeight="113.0" prefWidth="314.0" text="Instructions:&#10;-first select
simulation time then hit 'START SIMULATION'&#10;-you can add then clients
by setting the corresponding service amount then hit 'ADD CLIENT'&#10;-
observe then the process flow" wrapText="true" />
            <Label layoutX="14.0" layoutY="57.0" prefHeight="17.0"
prefWidth="93.0" text="Servicer one: " />
            <Label layoutX="109.0" layoutY="57.0" text="Status:" />
            <Label layoutX="231.0" layoutY="57.0" text="Nr. of Clients:" />
            <Text fx:id="status1" layoutX="153.0" layoutY="70.0"
strokeType="OUTSIDE" strokeWidth="0.0" text="idle"
wrappingWidth="78.13671875" />
            <Text fx:id="nr1" layoutX="314.0" layoutY="70.0"
strokeType="OUTSIDE" strokeWidth="0.0" text="0" wrappingWidth="35.09765625"
/>
            <Label layoutX="14.0" layoutY="94.0" prefHeight="17.0"
prefWidth="93.0" text="Servicer two: " />
            <Label layoutX="109.0" layoutY="94.0" text="Status:" />
            <Label layoutX="231.0" layoutY="94.0" text="Nr. of Clients:" />
            <Text fx:id="status2" layoutX="153.0" layoutY="107.0"
strokeType="OUTSIDE" strokeWidth="0.0" text="idle"
wrappingWidth="78.13671875" />
            <Text fx:id="nr2" layoutX="314.0" layoutY="107.0"
strokeType="OUTSIDE" strokeWidth="0.0" text="0" wrappingWidth="35.09765625"
/>
            <Label layoutX="14.0" layoutY="129.0" prefHeight="17.0"
prefWidth="93.0" text="Servicer three: " />
            <Label layoutX="109.0" layoutY="129.0" text="Status:" />
            <Label layoutX="231.0" layoutY="129.0" text="Nr. of Clients:"
/>
            <Text fx:id="status3" layoutX="153.0" layoutY="142.0"
strokeType="OUTSIDE" strokeWidth="0.0" text="idle"
wrappingWidth="78.13671875" />
            <Text fx:id="nr3" layoutX="314.0" layoutY="142.0"
strokeType="OUTSIDE" strokeWidth="0.0" text="0" wrappingWidth="35.09765625"
/>
            <Label layoutX="14.0" layoutY="167.0" prefHeight="17.0"
prefWidth="93.0" text="Servicer four: " />
            <Label layoutX="109.0" layoutY="167.0" text="Status:" />
            <Label layoutX="231.0" layoutY="167.0" text="Nr. of Clients:"
/>
            <Text fx:id="status4" layoutX="153.0" layoutY="180.0"
strokeType="OUTSIDE" strokeWidth="0.0" text="idle"
wrappingWidth="78.13671875" />
            <Text fx:id="nr4" layoutX="314.0" layoutY="180.0"
strokeType="OUTSIDE" strokeWidth="0.0" text="0" wrappingWidth="35.09765625"
/>
            <Label layoutX="14.0" layoutY="207.0" prefHeight="17.0"
prefWidth="93.0" text="Servicer five: " />
```

```
            <Label layoutX="109.0" layoutY="207.0" text="Status:" />
            <Label layoutX="231.0" layoutY="207.0" text="Nr. of Clients:"
/>
            <Text fx:id="status5" layoutX="153.0" layoutY="220.0"
strokeType="OUTSIDE" strokeWidth="0.0" text="idle"
wrappingWidth="78.13671875" />
            <Text fx:id="nr5" layoutX="314.0" layoutY="220.0"
strokeType="OUTSIDE" strokeWidth="0.0" text="0" wrappingWidth="35.09765625"
/>
            <Label layoutX="388.0" layoutY="57.0" text="Queue 1 nr. of
persons waiting:" />
            <Label layoutX="388.0" layoutY="94.0" text="Queue 2 nr. of
persons waiting:" />
            <Label layoutX="388.0" layoutY="129.0" text="Queue 3 nr. of
persons waiting:" />
            <Label layoutX="388.0" layoutY="167.0" text="Queue 4 nr. of
persons waiting:" />
            <Label layoutX="388.0" layoutY="207.0" prefHeight="0.0"
prefWidth="166.0" text="Queue 5 nr. of persons waiting:" />
            <Text fx:id="nrq1" layoutX="564.0" layoutY="70.0"
strokeType="OUTSIDE" strokeWidth="0.0" text="0" />
            <Text fx:id="nrq2" layoutX="565.0" layoutY="105.0"
strokeType="OUTSIDE" strokeWidth="0.0" text="0" />
            <Text fx:id="nrq3" layoutX="564.0" layoutY="140.0"
strokeType="OUTSIDE" strokeWidth="0.0" text="0" />
            <Text fx:id="nrq4" layoutX="562.0" layoutY="179.0"
strokeType="OUTSIDE" strokeWidth="0.0" text="0" />
            <Text fx:id="nrq5" layoutX="560.0" layoutY="219.0"
strokeType="OUTSIDE" strokeWidth="0.0" text="0" />
            <ImageView fitHeight="214.0" fitWidth="775.0" layoutX="-6.0"
layoutY="240.0" pickOnBounds="true">
               <image>
                  <Image url="@2.jpg" />
               </image>
            </ImageView>
            <Text layoutX="228.0" layoutY="30.0" strokeType="OUTSIDE"
strokeWidth="0.0" text="Queue management system simulator">
               <font>
                  <Font name="Agency FB Bold" size="24.0" />
               </font>
            </Text>
            <ImageView fitHeight="150.0" fitWidth="166.0" layoutX="587.0"
layoutY="61.0" pickOnBounds="true" preserveRatio="true">
               <image>
                  <Image url="@Gear.png" />
               </image>
            </ImageView>
            <Text fx:id="statusf" fill="#e10707" layoutX="256.0"
layoutY="434.0" strokeType="OUTSIDE" strokeWidth="0.0"
wrappingWidth="497.13671875">
               <font>
                  <Font name="System Bold" size="14.0" />
               </font>
            </Text>
            <Label fx:id="status6" layoutX="18.0" layoutY="634.0"
prefHeight="17.0" prefWidth="149.0" text="Queue 2 avg. waiting time: " />
            <Label fx:id="status61" layoutX="275.0" layoutY="597.0"
prefHeight="17.0" prefWidth="149.0" text="Queue 3 avg. waiting time: " />
            <Label fx:id="status62" layoutX="275.0" layoutY="634.0"
prefHeight="17.0" prefWidth="149.0" text="Queue 4 avg. waiting time: " />
            <Label fx:id="status621" layoutX="513.0" layoutY="597.0"
```

```
prefHeight="17.0" prefWidth="149.0" text="Queue 5 avg. waiting time: " />
            <Text fx:id="avg1" layoutX="166.0" layoutY="610.0"
strokeType="OUTSIDE" strokeWidth="0.0" text="0" />
            <Text fx:id="avg2" layoutX="164.0" layoutY="648.0"
strokeType="OUTSIDE" strokeWidth="0.0" text="0" />
            <Text fx:id="avg3" layoutX="421.0" layoutY="610.0"
strokeType="OUTSIDE" strokeWidth="0.0" text="0" wrappingWidth="54.46875" />
            <Text fx:id="avg4" layoutX="421.0" layoutY="647.0"
strokeType="OUTSIDE" strokeWidth="0.0" text="0" />
            <Text fx:id="avg5" layoutX="659.0" layoutY="610.0"
strokeType="OUTSIDE" strokeWidth="0.0" text="0" />
            <Label fx:id="status63" layoutX="20.0" layoutY="597.0"
prefHeight="17.0" prefWidth="149.0" text="Queue 1 avg. waiting time: " />
         </children>
      </Pane>
   </children>
</GridPane>
```

4. Conclusions

After this application development I have figured out the following:

- Every task is simpler if you break it into smaller pieces first
- Every part needs to be tested before joining the final configuration
- Some difficulties may appear which may seem easy to deal with but are not easy at all (ex: input string parsing)

4.1. Things I learned

I have learned to use javafx and SceneBuilder for developing better GUIs.

4.2. Future developments

There are lots of things that can be added to the application in the future, to improve the functionality, and also the graphical user interface.

Possible functionality improvements:

- catch all possible exceptions, and display a corresponding message (alert dialog, or result section).
- Integrate a logger in the GUI.
- Allow for a variable number of queues.
- Change the servicer's speed (service amount units per sec).

5. Bibliography

http://stackoverflow.com/

https://www.caveofprogramming.com/java-multithreading/java-multithreading-video-tutorial-part-2-basic-thread-communication.html