

Technical University of Cluj-Napoca
Faculty of Automation and Computers
Department of computers

Programming Techniques

HW 2

Student: Peter Ioan-Vlad

Group: 30421

Contents

1. Problem Specification.....	3
2. Example of working	4
3. Design.....	5
3.1 Relational Diagram.....	
3.2 Class Definitions	
3.3 The sequence diagram	
3.4 Multithreading	
3.5 Classes Design.....	
3.5.1 GUI.....	
3.5.2 Server	
3.5.3 Client.....	
3.5.4 Shop.....	
3.5.5 Main.....	
3.6 Packages and Interfaces	
3.7 User Interface	
4. Using and testing the application	
5. Conclusions	

1. Problem specification

The objective of this project was to design and implement a simulation of queuing based system for determining and minimizing client's waiting time.

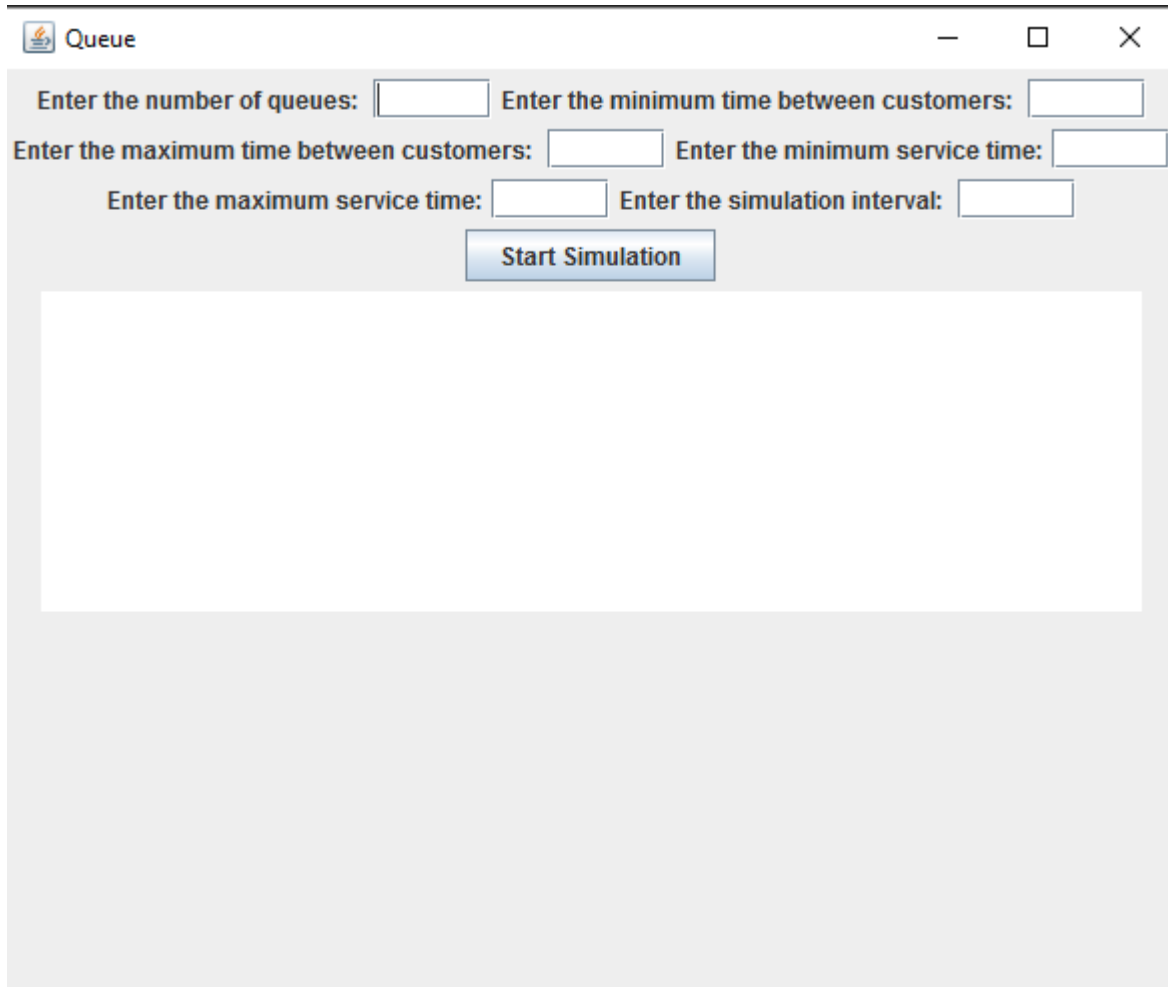
Queues are commonly used to model real world domains. The main objective of a queue is to provide a place for a "client" to wait before receiving a "service". The management of queue based systems is interested in minimizing the time amount their "clients" are waiting in queues before they are served. 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.

The input data would be the minimum and maximum interval of arriving time between customers, minimum and maximum service time, the number of queues, the simulation interval.

The output data would be the average waiting time for each queue and

2.Example of Working



The screenshot shows a window titled "Queue" with standard window controls (minimize, maximize, close). Inside the window, there are five input fields for user configuration:

- Enter the number of queues:
- Enter the minimum time between customers:
- Enter the maximum time between customers:
- Enter the minimum service time:
- Enter the maximum service time:
- Enter the simulation interval:

Below these fields is a blue button labeled "Start Simulation". At the bottom of the window is a large, empty white rectangular area intended for the simulation output.

The user will fill all the blank spaces after each sentence that describes what each input means. After this he will press the “Start Simulation” button and in the text box below will be a representation of each queue followed by a number of “*” symbols that will represent each client.

3.2 Class Definitions

```
*Server.java  *Client.java  Shop.java  *MainClass.java  *GUI.java X
1  package Queues;
2
3  import java.awt.event.ActionEvent;
12
13  public class GUI extends JPanel {
14
15      private static final long serialVersionUID = 1L;
16
17      private int minIntervalTime;
18      private int maxTimeInterval;
19      private int minServiceTime;
20      private int maxServiceTime;
21      private int simInterval;
22      private int nrOfQueues;
23      JLabel lblNrOfQueues = new JLabel("Enter the number of queues: ");
24      JTextField textNrOfQueues = new JTextField(5);
25      JLabel lblIntervalMin = new JLabel("Enter the minimum time between customers: ");
26      JTextField textMinTimeInterval = new JTextField(5);
27      JLabel lblIntervalMax = new JLabel("Enter the maximum time between customers: ");
28      JTextField textMaxTimeInterval = new JTextField(5);
29      JLabel lblServiceMin = new JLabel("Enter the minimum service time:");
30      JTextField txtMinServiceTime = new JTextField(5);
31      JLabel lblServiceMax = new JLabel("Enter the maximum service time:");
32      JTextField txtMaxServiceTime = new JTextField(5);
33      JLabel lblSimInterval = new JLabel("Enter the simulation interval: ");
34      JTextField txtsimulationInterval = new JTextField(5);
35
36      JTextArea txtQueues = new JTextArea(10,50);
37
38      private Shop shop;
39
40      public int getNrOfQueues(){
41          return this.nrOfQueues;
42      }
43
44      public GUI() {}
90
91
92      public void updateTxtQueues()
96
97  }
```

The class definition of the GUI class.

```

1 package Queues;
2 import java.util.ArrayList;
3
4 public class Shop extends Thread{
5
6     private int minTimeInterval;
7     private int maxTimeInterval;
8     private int minServiceTime;
9     private int maxServiceTime;
10    private int simInterval;
11    private int nrOfQueues;
12
13    private ArrayList<Server> servers = new ArrayList<Server>();
14
15    private long initialTime;
16
17    private GUI gui;
18
19    public long getTime(){
20    }
21
22    public Server selectServer(){
23    }
24
25    public Shop(int nrOfQueues, int simInterval, int minIntervalTime, int maxTimeInterval, int minServiceTime, int maxServiceTime, GUI gui){
26    }
27
28    public String listQueue(){
29    }
30
31    public void run(){
32    }
33
34    public void updateTextGUI()
35    {
36        gui.updateTxtQueues();
37    }
38 }

```

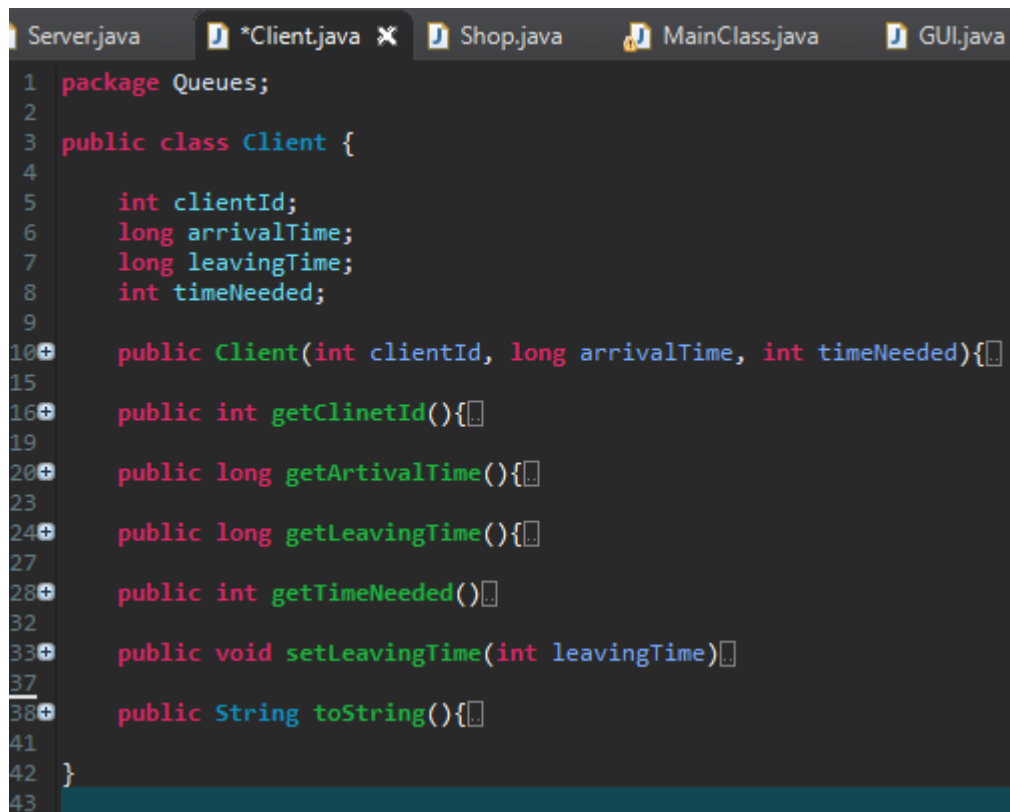
The class definition of the Shop class.

```

1 package Queues;
2 import java.util.Vector;
3
4 public class Server extends Thread{
5
6     private int serverId;
7     private Vector<Client> clients = new Vector();
8
9     private Shop shop;
10
11    public Server(int id, Shop s){
12    }
13
14    public void setId(int id){
15    }
16
17    public int getServerId(){
18    }
19
20    public void run(){
21    }
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37    public synchronized void addClient(Client client) throws InterruptedException {
38    }
39
40
41
42
43
44    public synchronized void delClient() throws InterruptedException{
45    }
46
47
48
49
50
51
52
53
54
55
56
57
58    public int getNrClients(){
59    }
60
61
62
63
64
65
66
67
68 }
69

```

The class definition of the Server class.

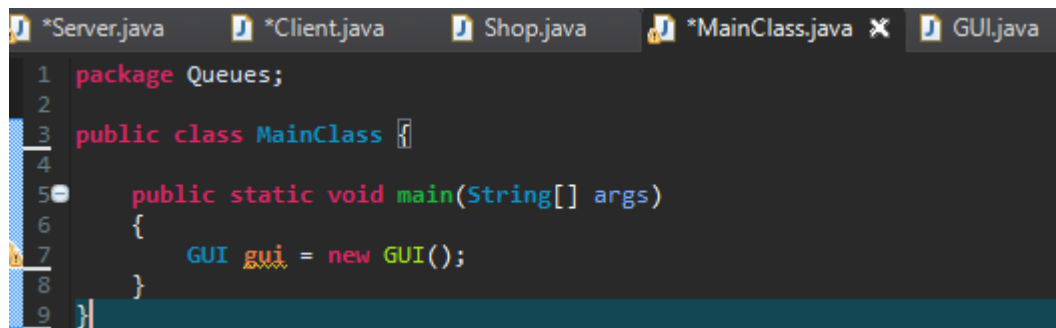


```

1 package Queues;
2
3 public class Client {
4
5     int clientId;
6     long arrivalTime;
7     long leavingTime;
8     int timeNeeded;
9
10    public Client(int clientId, long arrivalTime, int timeNeeded){
11
12    }
13
14    public int getClientId(){
15
16    }
17
18    public long getArrivalTime(){
19
20    }
21
22    public long getLeavingTime(){
23
24    }
25
26    public int getTimeNeeded(){
27
28    }
29
30    public void setLeavingTime(int leavingTime){
31
32    }
33
34    public String toString(){
35
36    }
37
38    }
39
40
41
42
43

```

The class definition of the Client class.



```

1 package Queues;
2
3 public class MainClass {
4
5     public static void main(String[] args)
6     {
7         GUI gui = new GUI();
8     }
9 }

```

The class definition of the MainClass.

3.4 Multithreading

Multithreading is the ability of a central processing unit or a single core in multi-core processor to execute multiple processes or threads concurrently, appropriately supported by the operating system.

Multithreading in Java is a process of executing multiple threads simultaneously. Thread is basically a lightweight sub-process, a smallest unit of processing.

In Java, a Thread is essentially the Object that represents one piece of work. When you start your application and it starts to run, Java has created a Thread and this Thread is what will carry to work that your application is meant to do. One thread can only do one task at a time.

A thread can be in one of the following states:

NEW – A thread that has not yet started is in this state.

RUNNABLE – A thread executing in the Java virtual machine is in this state.

BLOCKED – A thread that is blocked waiting for a monitor lock is in this state.

WAITING – A thread that is waiting indefinitely for another thread to perform a particular action is in this state.

TIMED_WAITING – A thread that is waiting for another thread to perform an action for up to a specified waiting time is in this state.

TERMINATED – A thread that has exited is in this state.

A thread can be in only one state at a given point in time.

3.5 Classes Design

The classes that are use are the following: Server, Client, Shop, MainClass, GUI.

3.5.1 GUI Class

In this class the interface for the project is made. It imports the Queues package some libraries are imported (`java.awt.event.ActionEvent`, `java.awt.event.ActionListener`, `javax.swing.JButton`, `javax.swing.JFrame`, `javax.swing.JLabel`, `javax.swing.JPanel`, `javax.swing.JTextArea`, `javax.swing.JTextField`).

`ActionEvent` is a class, `e` is an instance of that class. You can use it to call it's methods / properties.

`ActionListener` are event handlers to implement. You implement an action listener to define what should be done when a user performs certain operation.

`JButton` can be configured, and to some degree controlled by Actions. Using an Action with a button has many benefits beyond directly configuring a button.

`JFrame`, a frame, implemented as an instance of the frame `JFrame` class, is a window that has decorations such as a border, a title, and supports button components that close or iconify the window. Applications with GUI usually include at least one frame.

A `JLabel` object can display either text, an image, or both. You can specify where in the label's display area the label's contents are aligned by setting the vertical and horizontal alignment. By default, labels are vertically centered in their display area. Text-only labels are leading edge aligned, by default; image-only labels are horizontally centered, by default.

The `JPanel` class provides general-purpose containers for lightweight components. By default, panels do not add colors to anything except their own background; however, you can easily add borders to them and otherwise customize their painting.

A JTextArea is a multi-line area that displays plain text. It is intended to be a lightweight component that provides source compatibility with the java.awt.TextAreaclass where it can reasonably do so.

JTextField is intended to be source-compatible with java.awt.TextField where it is reasonable to do so. This component has capabilities not found in the java.awt.TextField class. The superclass should be consulted for additional capabilities.

The first line on the GUI class that extends the JPanel class is defining the serialVersionUID which is a universal version identifier for a Serializable class. Deserialization uses this number to ensure that a loaded class corresponds exactly to a serialized object. If no match is found, then an InvalidClassClassException is thrown. On the web I read that it is not mandatory to add, but should be added if possible.

The next lines of code are declarations of the private parameters (minimum and maximum interval of arriving time between customers, minimum and maximum service time, number of queues and the simulation interval) that our program relies on. They are private and to get them there are use some getters.

The next things in the code are the declarations of the labels (JLabel) and text fields(JTextField) that are used to create the input boxes so that we can introduce the essential data for our program and a text area (JTextArea) created in order to display on it the repartition of people in the queues. All the cariables have specific names, so that is very easy to know what kind are them. For example lblNrOfQueues is the lable that tell us what will be done with our input, txtNrOfQueus is the text field in which we will introduce the value and finally nrOfQueues is the value that has been converted from a string to an integer.

In the graphical user interface we have the Start Simulation button that starts our simulation after the inputs had been given. When the button is pressed the inputs that are strings are parsed to integers and after their conversion an object of type shop is created with the new variables.

At the final of the class the size of the frame is set(600x500) and the visible mode of the frame is set to true.

3.5.2 Server Class

The Server class has the role of the person at cash register. This class has a vector that keeps all the clients that are in the queue.

The class addClient throws an InterruptedException. Thrown when a thread is waiting, sleeping or otherwise occupied and the thread is interrupted, either before or during the activity. Occasionally a method may wish to test whether the current thread has been interrupted, and if so, to immediately throw this exception. The function adds a client to the vector initialize in the beginning.

In this class there is also system out that prints in “real time” when each client is added in queue and it when it is served to the corresponding server. This is the log of events.

The function delete client deletes the client that is at the index 0, this being the first person in the queue and prints in the log of events when it is served.

3.5.3 Client Class

The clients that waits ant the queue has an id an arrival time a time needed(the time he will spend with the server), a constructor is made in the beginning of this class.

After the constructor there are several getters and a setter. Those being getClientId, getArrivalTime, getLeavingTime, getTimeNedded and the setter setLeavingTime.

After those there is a toString method that makes a string of this type:

```
"Client "+clientId+" arrival time "+arrivalTime+" time needed "+timeNeeded
```

3.5.4 Shop Class

The Shop class extends the Thread class that refers to multithreading which is when two or more tasks executing concurrently within a single program. A thread is an independent path of execution within a program. Many threads can run concurrently within a program. Every thread in Java is created and controlled by the `java.lang.Thread` class.

The time used in this project is the current time accessed by

```
System.currentTimeMillis()/1000 - initialTime;
```

it is divided by 1000 so that the program will work with seconds, not milliseconds. The `java.lang.System.currentTimeMillis()` method returns the current time in milliseconds. The unit of time of the return value is a millisecond, the granularity of the value depends on the underlying operating system and may be larger.

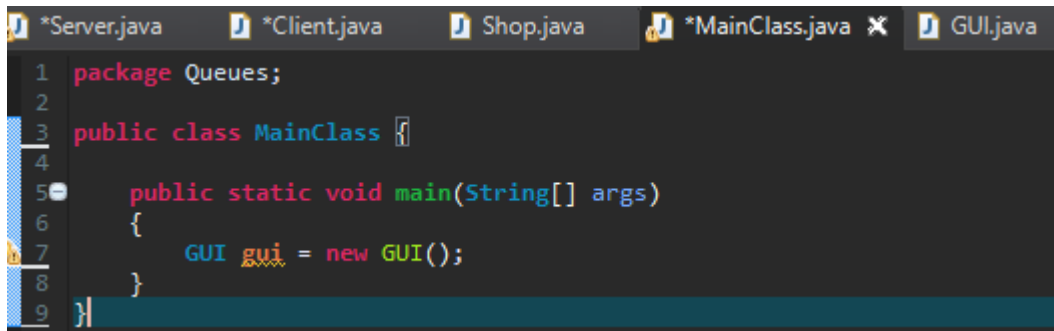
The `selectServer` return the queue that has the least clients waiting so when a new client is generated we will know where to put the client to reduce his or hers waiting time.

The public Shop gets the time in seconds the `minTimeInterval`, `maxTimeInterval`, `minServiceTime`, `maxServiceTime`, `simInterval`, `nrOfQueues` and the gui.

`ListQueue` is used to display in the graphical user interface each server with series of “*” that represent each client waiting in line. The stars appear and disappear when the client that represent the stars is put in line or removed after it has been served.

`Run()` will pass through all the queues and start each thread.

3.5.5 Main Class



```
1 package Queues;
2
3 public class MainClass {
4
5     public static void main(String[] args)
6     {
7         GUI gui = new GUI();
8     }
9 }
```

In the main class only the GUI class will be called.

4. Using and testing the application

The user will have to open the project and after that the graphical user interface window will appear. He will have to fill up the blank spaces with the numbers taking in consideration the description before each box that describes what each value that he is introducing is meaning. After introducing the parameters below the button each queue is represented as a series of symbols (“*”).

Also in the console of the IDE he is using we will be able to see what happens in real time, when each client is getting in line and when the clients are served.

5. Conclusions

This project made me learn the multithreading programming which I found to be very useful but a bit tricky to learn. I felt that this project made my coding skills better and learned a lot of new thing about this programming language and the objective programming. There are lots of things that can be improved and I am sure that in the near future I will be able to make such improvements in future projects.

6. References

<http://users.utcluj.ro/~jim/OOPE/>

<http://docs.oracle.com/javase/7/docs/api/overview-summary.html>

<http://docs.oracle.com/javase/1.5.0/docs/tooldocs/windows/javadoc.html>

<http://stackoverflow.com/>