Programming Techniques
HOMEWORK 2

Student: Tudorica Andrei
Group: 30421

0. Contents

1. Objective of the homework

Design and implement a simulation application aiming to analyze queuing based systems for determining and minimizing clients' waiting time.

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;

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. Problem Analysis

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.

Simulations are very important because, with a minimum amount of resources, they allow us to model a real life situation. Technology has advanced so much that, nowadays, we can simulate almost everything in a way that is almost, if not equally, as good as the reality. So, with the aid of simulations, we can avoid a lot of problems that could occur in real life situations.

3. Modelling

The process of modelling is defined as the process of modularising a big problem into smaller ones, that are easier to understand and debug. This also helps making an abstract idea more clear. In software development, modelling is essential in order to build an application that has a strong background structure.

Our problem, as stated above in the “Problem Statement” paragraph, includes queues that we have to manage, optimize and simulate for a given time interval, with the input data bounds being given.

Real life queues could be modelled quite easily, using the data structure with the same name. They work on the F I F O principle, which is, first in, first out. We can implement these methods in our application

and by doing so, the modelling of our queues becomes much more simple.

Because, as we all know, queues evolve in parallel, the best way of implementing them is using multithreading. Multithreading is the ability of a program or an operating system process to manage its use by more than one user at a time and to even manage multiple requests by the same user without having to have multiple copies of the programming running in the computer. Each user request for a program or system service (and here a user can also be another program) is kept track of as a thread with a separate identity. As programs work on behalf of the initial request for that thread and are interrupted by other requests, the status of work on behalf of that thread is kept track of until the work is completed.

The primary function of multithreading is to run multiple tasks at the same time. In a Java program, these tasks are represented as threads and have a separate execution path. Also, handling of multithreaded Java programs is easy because you can decide the sequence in which execution of Java threads take place. Some of the advantages of multithreading are: enhanced performance by decreased development time, simplified and streamlined program coding, improvised GUI responsiveness, simultaneous and parallelized occurrence of tasks, better use of cache storage by utilization of resources, decreased cost of maintenance, better use of CPU resource. Multithreading does not only provide you with benefits, it has its disadvantages too. For example the debugging and testing process is very complex, it involves overhead switching of context, increased potential for deadlock occurrence, increased difficulty level in writing a program and unpredictable results.

4. Use cases

The developed software application is a basic one, therefore it only simulates the queues, and optimizes them in a greedy way. The app is useful though for organizing simple queues, like, for example, the ones

in a supermarket, giving the clients directions for the queue with the shortest waiting time.

4.1. Scenarios

Let's consider an example of how the application works;

IMPORTANT NOTES:

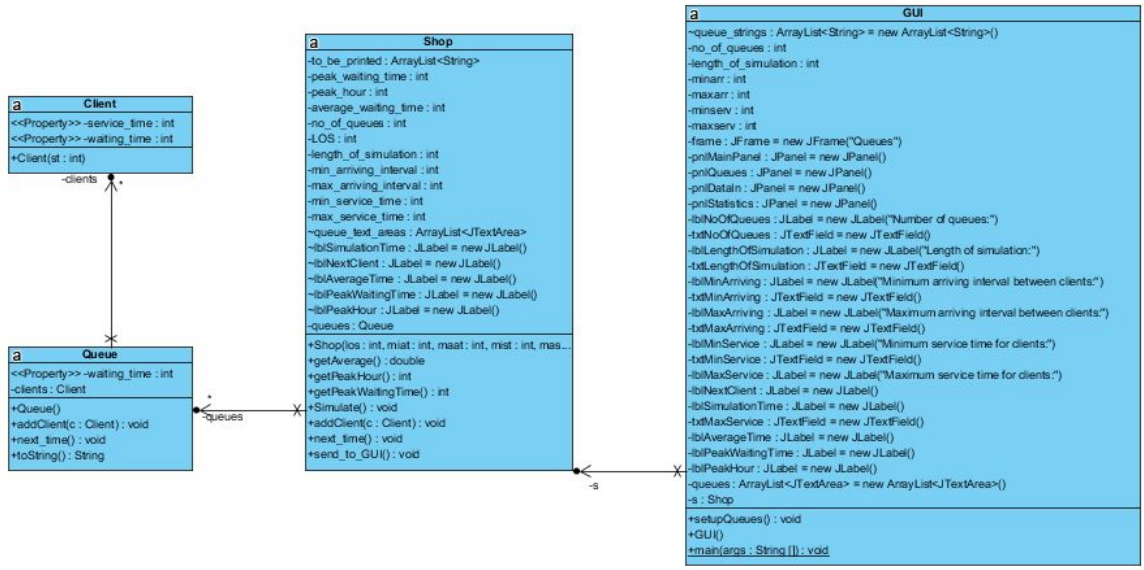
- The input text boxes may only contain integer numbers.
- The input data introduced in the "Minimum arriving interval between clients:" text box must be smaller than the input data introduced in the "Maximum arriving interval between clients:" text box.
- The input data introduced in the "Minimum service time for clients:" text box must be smaller than the input data introduced in the "Maximum service time for clients:" text box.
- The input data introduced in the "Number of queues:" text box must be greater than zero.
- There is no input data validation method, so any data that is not respecting the rules will make the application stop.

Steps for the performing a simulation, for 50 seconds, a shop that has 4 serving queues, each taking clients and serving them in between 5 and 10 seconds (the clients arrive one after another, with a delay between them of 5 to 10 seconds):

1. The app is launched .
2. The user introduces the number of seconds the simulation should last in the first text box, in our case 50 (see the rules for introducing the input data in the program above).
3. The user introduces the number of queues in the shop, in our case 4 (see the rules for introducing the input data in the program above).

4. The user introduces the the minimum arriving interval between two clients, in our case 5 (see the rules for introducing the input data in the program above).
5. The user introduces the the maximum arriving interval between two clients, in our case 10 (see the rules for introducing the input data in the program above).
6. The user introduces the the minimum service time for a client, in our case 5 (see the rules for introducing the input data in the program above).
7. The user introduces the the maximum service time for a client, in our case 10 (see the rules for introducing the input data in the program above).
8. The user clicks on the first button of the user interface, the one that has the word “Simulate” written on it.
9. The input panel will disappear.
10. The queue panel will appear, having the numbers of columns specified by the input text box.
11. Random clients will be generated at random delays, between the ones specified in the "Minimum arriving interval between clients: " and "Maximum arriving interval between clients:" text boxes, having random service times between the values specified in the "Minimum service time for clients:" and "Maximum service time for clients: " .
12. Above the queue panel there are two dynamically updated labels, one showing the time remaining until the next client comes, and one showing how much time there is left from the simulation.
13. Under the queue panel there are three dynamically updated labels: one presenting the Peak waiting time of the simulation (the longest time a client had to wait), one presenting the peak hour of the simulation (the second when the peak waiting time was met) and one showing the average waiting time of all the queues.

5. Design



5.1. The Client class

```
1  public class Client {
2      private int service_time, waiting_time;
3
4      public Client(int st) {...
8
9      public int getService_time() {...
12
13     public int getWaiting_time() {...
16
17     public void setService_time(int st) {...
20
21     public void setWaiting_time(int wt) {...
24 }
```

The Client class contains the necessary information regarding each client, that is the service time it needs, stored as an int and the time he needs to wait to be served, also stored in an int. The only constructor of the class is the one that receives the service time for the client to be created. The class also contains getters and setters for the private variables service_time and waiting_time.

5.2. The Queue class

The Queue class contains a list of Client elements, being the clients that have joined that queue. The Queue also has a waiting_time variable which is the time that a new client would have to wait if he joins this Queue. A new instance of this class is created with no arguments, as it is visible in the constructor. the class contains 4 methods. The first one is a getter for the waiting time of the list. The second one is a function that a client to the ArrayList of Clients in the queue. This method also sets the waiting_time of the client and adds the service time of the new added client to the waiting time of the queue. The third method computes what happens to the current queue when a second passes. It decreases the waiting time of the entire queue and of every element in the queue, all these happening in a separate thread. This way every queue is evolved in parallel with the other queues, simulating the queues as real as possible. The third method and the last one of this class returns a printable and user friendly version of a Queue, returning a string built by the times remaining for each client until he gets to the head of the queue, and the time until the element that is in the head of the queue is served and can leave. It also informs about the total wait time of the queue.

5.3. The Shop class

```
1  import java.util.ArrayList;
6
7  public class Shop {
8      private ArrayList<Queue> queues;
9      private ArrayList<String> to_be_printed;
10     private int peak_waiting_time, peak_hour, average_waiting_time, no_of_queues;
11     private int LOS, length_of_simulation, min_arriving_interval, max_arriving_interval, min_service_time,
12             max_service_time;
13     ArrayList<JTextArea> queue_text_areas;
14     JLabel lblSimulationTime = new JLabel();
15     JLabel lblNextClient = new JLabel();
16     JLabel lblAverageTime = new JLabel();
17     JLabel lblPeakWaitingTime = new JLabel();
18     JLabel lblPeakHour = new JLabel();
19
20     public Shop(int los, int miat, int maat, int mist, int mast, int no_of_queues, ArrayList<String> where_to_print,
47
48     public double getAverage() {
51
52     public int getPeakHour() {
55
56     public int getPeakWaitingTime() {
59
60     public void Simulate() throws InterruptedException {
104
105    public void addClient(Client c) {
119
120    public void next_time() {
140
141    public void send_to_GUI() {
150
151 }
```

The Shop class is the one managing the set of queues, that is why it contains an array list of Queue objects. In the set of variables there can be found the statistical variable for the peak waiting time, peak hour of the simulation, the average waiting time of the simulation. The filter variables are also something definitory for this class. Those are: the length of the simulation, the number of queues, the minimum arriving time between clients, the maximum arriving time between two clients, the minimum service time for a client, the maximum serving time for a client. A couple of visual elements are defined, as the class also prints some elements directly to the graphic user interface, with every second that passes from the simulation. These are the text boxes boxes for the remaining simulation time, for the time remaining until the next client arrives, for the average waiting time of all the queues in the shop for the entire time of the simulation, for the peak hour and for the peak waiting time of the simulation, and an array list of text areas for each queue.

The class contains seven methods: a constructor, get average,get peak waiting time, Simulate, add client,next time, send to gui. They will be described in what follows:

- The constructor receives 13 arguments: the length of the simulation, the number of queues, the minimum arriving time between clients, the maximum arriving time between two clients,

the minimum service time for a client, the maximum serving time for a client, the text boxes boxes for the remaining simulation time, for the time remaining until the next client arrives, for the average waiting time of all the queues in the shop for the entire time of the simulation, for the peak hour and for the peak waiting time of the simulation, and an array list of text areas for each queue. Every variable of the class is initialized using these arguments.

- The get average method returns the average waiting time of the simulation, being a getter for that variable.
- The get peak hour method returns the peak hour of the simulation, being a getter for that variable.
- The get peak waiting time method returns the peak waiting time of the simulation , being a getter for that variable.
- The Simulate method, that extends InterruptedException, is the core of the application. First of all, it runs in a parallel thread with the Graphical user interface and its main purpose is to manage the queues and their evolution. It uses a local variable to compute the amount of time remaining until the next client arrives. That variable decreases second by second and when it is zero, a new random time between the minimum arriving time between two clients and the maximum arriving time between two clients and then creates a random integer between the minimal service time for a client and the maximal service time for a client and creates a new client with that service time, using the previously described constructor in section 5.1. The method also computes the average waiting time for each second to compute the total average time of the simulation. Then calls the next time function which runs the queues for one second. It calls the send to GUI function that prints the stats from this second of run. The last but not the least, it sleeps the thread for 1 second to simulate the real time.
- The add client method adds a client to the shop queue. This method is intended to add the client at the queue with the least waiting time, thus making the simulation more efficient. By iterating through all queues, the minimum waiting time of a queue at that given moment is stored in a variable, and at the end, the client is

added in that particular queue. What is more, this method also computes the peak hour of the simulation. To be more precise, it computes the moment of the simulation with the greatest peak.

- The next_time() method is used to start a thread for each queue we have in the application. An array list stores these threads. For each queue in the simulation, the application starts its corresponding thread, and then joins them back. This happens at each second of the simulation.
- The send_to_GUI() method is used to display the queues and information about them onto the graphical user interface, so that the user can see in an interactive way the evolution of the queues. The information that is displayed, is updated at each second, so the person using the application can clearly see the evolution of the queues and important information about them.

5.4. The GUI class

```
1  import java.util.ArrayList;
7
8  public class GUI extends Frame
9  {
10     ArrayList<String> queue_strings=new ArrayList<String>();
11     private int no_of_queues,length_of_simulation,minarr,maxarr,minserv,maxserv;
12     private JFrame frame = new JFrame("Queues");
13     private JPanel pnlMainPanel = new JPanel();
14     private JPanel pnlQueues = new JPanel();
15     private JPanel pnlDataIn = new JPanel();
16     private JPanel pnlStatistics = new JPanel();
17     private JLabel lblNoOfQueues=new JLabel("Number of queues:");
18     private JTextField txtNoOfQueues=new JTextField();
19     private JLabel lblLengthOfSimulation=new JLabel("Length of simulation:");
20     private JTextField txtLengthOfSimulation=new JTextField();
21     private JLabel lblMinArriving=new JLabel("Minimum arriving interval between clients:");
22     private JTextField txtMinArriving=new JTextField();
23     private JLabel lblMaxArriving=new JLabel("Maximum arriving interval between clients:");
24     private JTextField txtMaxArriving=new JTextField();
25     private JLabel lblMinService=new JLabel("Minimum service time for clients:");
26     private JTextField txtMinService=new JTextField();
27     private JLabel lblMaxService=new JLabel("Maximum service time for clients:");
28     private JLabel lblNextClient=new JLabel();
29     private JLabel lblSimulationTime=new JLabel();
30     private JTextField txtMaxService=new JTextField();
31     private JLabel lblAverageTime=new JLabel();
32     private JLabel lblPeakWaitingTime=new JLabel();
33     private JLabel lblPeakHour=new JLabel();
34     private ArrayList<JTextArea> queues=new ArrayList<JTextArea>();
35     private Shop s;
36
37     public void setupQueues()
46
47     public GUI() {
132    public static void main(String[] args) throws InterruptedException{
133        new GUI();
134    }
135 }
136
```

The GUI class has the sole purpose to create the graphical user interface in such a way that it is as user friendly as possible. Although it is rather simplistic, it clearly indicates to the user what should be done, what parameters should be introduced, and how do the queues evolve during time. The user introduces in the text fields, the parameters indicated in the labels above. After pressing the ‘ Simulate ’ button, the application converts what the user types in the text fields, into integers, and stores them in variables. With the help of these variables, the constructor is called, having them as parameters. Also when the ‘Simulate ’ button is pressed, the input panel will disappear and the queue panel appears displaying the number of text fields equal to the number of queues given in the input data phase. After that, second by second, the text fields are updated with the new data concerning each queue.
The structure of the graphical user interface is as follows: there is a frame that contains a panel, main panel. The main panel contains 3 other panels, the data in panel, the queues panel and the statistics panel. The data in panel contains the text boxes and the labels for the first phase of the simulation, the data introduction. The queue panel contains the labels with the timers for clients arrival and remaining simulation time. The statistics panel contains the labels with the average waiting time , peak hour and peak waiting time. All the labels in the graphical user interface are dynamically updated every second in order to offer the user the real feeling of the queue management system.

5.5. Logger

All the events encountered in the simulation process can be reviewed in the log file created using Logger from java util.

6. Results

The result of the development of this application, that lasted for two weeks together with writing the documentation, is a software easy to use that makes it easy for the user to perform a couple of simulations on a virtual queue management system. This tool can be, of course, improved in several ways that will be described in the next section.

7. Further development

A couple features can be modified or added in order to improve the functionality of the application:

- A better graphical user interface
- A simpler way of introducing the data and more simulation filters
- Better optimization algorithms for queue management

8.Conclusions

During the development of this project i learned and improved of several skills:

- A better use of Eclipse
- A better use of the Java coding language
- A better understanding and usage of the object oriented programming paradigms
- More organized and clear code
- A better use of thread operations
- I learned how to create and add elements to a log file using Logger from java util

9. Bibliography

1. Java threads tutorial http://www.javaworld.com/article/2077138/java-concurrency/introduction-to-java-threads.html
2. Java general documentation https://docs.oracle.com/javase/7/docs/api/
3. Lecture and laboratory guidelines http://www.coned.utcluj.ro/~salomie/PT_Lic/3_Lab/
4. Word counter https://wordcounter.net/