

Technical University of Cluj-Napoca

Faculty of Automation and Computer Science

2nd Semester 2018-2019

Programming Techniques

Homework 2

Student: Elekes Lukacs-Roland

Group: 30424

Supervising teacher: Viorica Chifu

Contents

1. Objectives	3
2. Problem Analysis, Modelling, Scenarios, Use Cases	3
2.1. Problem Analysis	3
2.2. Modelling	3
2.3. Scenarios, Use Cases.....	4
3. Design	6
3.1. Class Design and UML Diagram.....	6
3.2. Packages and Relationships.....	7
3.3. Algorithms and Data Structures.....	8
3.4. Graphical User Interface	8
4. Implementation	9
4.1. Class Client.....	9
4.2. Class Queue	10
4.3. Class Scheduler.....	11
4.4. Class Manager	12
4.5. Class GUI	14
5. Usage and Testing.....	14
6. Conclusions	15
7. Bibliography	16

1. Objectives

The main objective is to design and implement a simulation application aiming to analyze queuing based systems for determining and minimizing clients' waiting time.

In order to achieve the main objective, we have to take in consideration the following secondary objectives, implementation of:

- Design a task (client) (2.2., 3.1.)
- Design a server (queue) (2.2, 3.1., 4.2.)
- Create threads (2.2, 3.)
- Design a system that manages tasks and servers (2.2, 3.1, 4.4.)
- Design a graphical user interface where the simulation can be visualized (3.4., 4.5.)

2. Problem Analysis, Modelling, Scenarios, Use Cases

2.1. Problem Analysis

A simulation application may be useful in almost every situation when a process, or something continuous has to be modelled. By using a simulation, a process can be represented and visualized. This certain system may be helpful analyzing real life scenarios where queues exist, a good example would be a supermarket. It is also good for providing an efficient way to choose a queue from the client's point of view, or by deciding how many servers should be open at a time from the supermarket's point of view, etc. .

Using a graphical user interface, the visualization becomes more concrete and easier to analyze.

2.2. Modelling

For such a system it is necessary to have some basic elements, like Client and a Queue. In order to simulate this process, we need clients who arriving at certain points in time and queues, where

they can go to be served. There is also needed an entity which handles the simulation, generates clients, decides how to dispatch clients, starts queues, calculates statistics and others.

Thus, a Client must have at least two important attributes which are the arrival time and the time needed for their service. Two more attributes were also added, an ID for the client (for the better visualization) and a finish time (which is useful in calculation of statistics).

The Queue is an entity that contains clients and serves them.

The simulation manager is an entity that has access to every component in this simulation system in order to maintain the events.

More information is discussed at the Design and Implementation chapters.

2.3. Scenarios, Use Cases

Scenarios includes the simulation of this real life situation. Data is introduced, and the system starts the simulation with the information provided by the user.

The inputs data is the following:

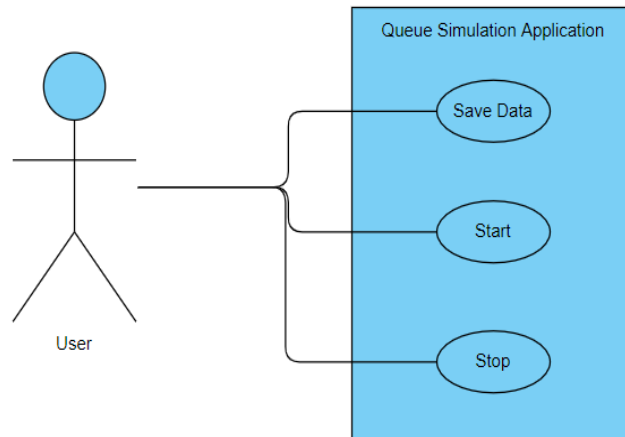
- Minimum and maximum interval of arriving time between two clients
- Minimum and maximum interval for service time
- Number of queues
- Number of clients
- Simulation length (time units)

All these data are integers and it is assumed that they are correctly introduced. It is the user's responsibility at this point to introduce valid information for a correct functioning of the system.

After the parameters are introduced in the corresponding fields, user should save the data and only after that start the simulation (otherwise some default values are taken and simulation starts with those parameters).

As an output, the user can see the arrival of clients, evolution of queues and at the end of the simulation some statistics, like average waiting time, clients served or peak hour.

The use case diagram is the following:



Use case title: Starting the Simulation

Actor: User

Main success scenario:

1. User introduces data
2. User saves data by pressing the button "Save Data"
3. User starts simulation by pressing the button "Start"
4. Simulation starts
5. Simulation ends and statistics are shown

Alternative sequence:

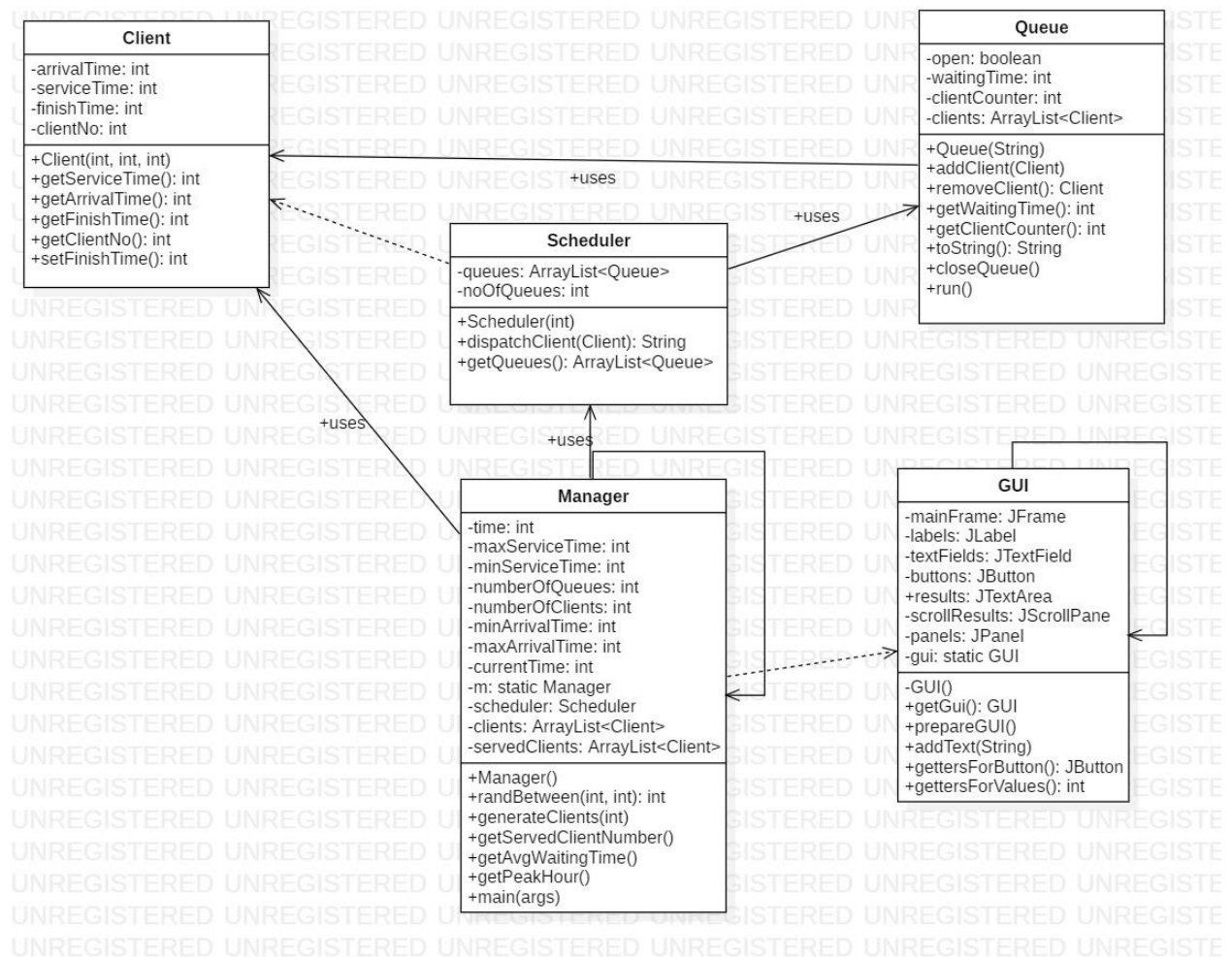
1. User introduces data
2. User saves data by pressing the button "Save Data"
3. User starts simulation by pressing the button "Start"
4. Simulation starts
5. User stops simulation by pressing the button "Stop"
6. Simulation ends and statistics are shown

3. Design

3.1. Class Design and UML Diagram

The design of this assignment is based on using and handling threads. There are also ideas and approximations taken from the Model View Controller architectural pattern, because the interaction between the user and the system is done through a graphical user interface, but for a simpler implementation the Model and View are not completely separated.

The first steps were to design the entities like Client or Queue, which was not difficult since their complexity is not high. After these classes, designing the structure of threads and the system where they are working was a key step. For this classes named Scheduler and Manager were used.



In the class Queue a thread was implemented, which is simple, a client was removed from the queue being the client who is served, the thread was not Runnable state for client's service time and the waiting time at the was decremented.

The Scheduler class is responsible for creating the threads (queues). It has another important role, which is working as a dispatcher of the clients that arrive. I chose to use the strategy based on choosing the smallest waiting time from all the queues, so the Scheduler puts every client to the corresponding place.

The Manager class is responsible for starting all the threads from the servers (queues) and for providing a thread where clients arrive at arbitrary moments of time. For this, there are other tasks that this class does, like generating random clients, taking information from the graphical user interface and start the simulation with the data taken from there (from this point of view it can be associated to the controller part of the Model View Controller pattern). This class implements action listeners for buttons and after the simulation ends it also calculates some statistics that are displayed on the graphical interface.

For the design of the graphical user interface I chose to use the Singleton Pattern, thus the class GUI is a singleton class, which means that is a class that can have only on object (an instance of the class) at a time. In this way, the user interface is reachable from any point.

3.2. Packages and Relationships

This assignment is structured similarly to a project that uses the Model View Controller pattern, however, we cannot tell that this project is based on this pattern (the causes were discussed before).

In the package named **model** I have all the components that are parts of the simulation, that are for preparing the simulation. Here belongs the classes Client, Queue and Scheduler.

In the package named **view** the singleton class GUI is placed, which is responsible for the graphical representation.

To the package named **controller** belongs the Manager class, which is connected to the GUI class and to entities from the model package. Also it controls the whole system.

The relationships existing among the classes are mostly associations and aggregations, but there exists also dependencies.

Some important relationships to mention are the Scheduler Queue relationship, where the Scheduler has queues, or the association between Queue and Client, which is in fact an aggregation. An association to mention is the relationship between Manager and Client.

3.3. Algorithms and Data Structures

The algorithm used by a Queue is the following: take a client, serve (put to sleep the thread) and decrement the waiting time.

The algorithm used by the Scheduler is finding the index of the queue which has the smallest waiting time. The algorithm iterates through the list of queues, find the minimum waiting time and saves the index of that queue. After the desired queue is found, the new client is inserted there. In the same time the finish time of a client can be calculated since now we know the waiting time of the queue (finish time = arrival + service + waiting time).

The Manager class uses algorithms for generating clients, running the thread and calculating statistics. When a client is generated, a random arrival time after the last client arrived is generated, so we take the last client's arrival time and sum with the random number generated in the interval (given by the user).

In the thread the algorithm is the following, take the first client from the generated list of clients, if the arrival time is equal with the current time, he is inserted in one of the queues, print the state of queues and increment the current time. After the current time reaches the simulation time, the simulation is over, and statistics are displayed.

Every queue displays the number of clients he served, the average waiting time is calculated and the peak hour is displayed.

For the average waiting time, every served client's waiting time is summed and calculated an arithmetical average.

For the peak hour, the clients arrived in the last hour is counted, if a change of hour is detected, a comparison is made to the current and last count of clients.

Scheduler, Manager and Queue use the data structure ArrayList.

3.4. Graphical User Interface

The graphical user interface provides an easy handling of the system for the user. It has some labels (for information), text fields (for introducing data and output of the result) and buttons.

The buttons are representative since they have a label describing shortly the operation it performs.

Since the main objective of this assignment was the designing of the system and working with threads I did not spend much time for creating a nice GUI and I did not use tools for it. As a main future development this has to be mentioned.

Queue Simulation

Arriving time between two customers: 2 4 Queues No.: 3

Interval for service time: 5 6 Clients No.: 50 Simulation time: 100

Save Data Start Stop

4. Implementation

4.1. Class Client

The class Client represents and implements a client. As attributes it has: arrivalTime, serviceTime, clientNo and finishTime, all of them being integers.

In the constructor of the class, values for arrival time, service time and client ID are assigned.

The class has getters for every attribute and has a method that sets the finish time using the following formula: finish time = arrival time + service time + waiting time (obtained from the queue).

4.2. Class Queue

The class Queue represents and implements a queue, extends class Thread. The attributes of this class are the following:

- private boolean open – set true as default, describes if the queue runs or not
- private int waitingTime – the waiting time of the queue, updated when clients arrive or leave
- private int clientCounter – for counting the number of clients served at the queue
- private ArrayList<Client> clients – represents the clients that are waiting in the queue
- There is an attribute for logging (global logger).

Most of the methods are synchronized, because they act on shared resources. There exist methods for inserting and removing clients.

The run() method is overridden from the Thread class, where a client is served (the algorithm was discussed before).

There also is a toString() method, which returns a representation of the queue object in String format showing the server's name, number of clients are in and the waiting time.

Examples:

```
public synchronized Client removeClient() throws InterruptedException{
    while(this.clients.size() == 0) {
        wait();
    }
    Client c = this.clients.get(0);
    this.clients.remove(0);
    notifyAll();
    return c;
}

public void run() {
    Client current;
    while(open) {

        try {
            current = removeClient();
            sleep(current.getServiceTime()*1000);
            this.waitingTime -= current.getServiceTime();

            LOGGER.log(Level.INFO,"Client " + current.getClientNo() +
" was served at queue " + getName() + " leaves at " + current.getFinishTime() + "\n"
);
            System.out.println("Client " + current.getClientNo() + "
was served at queue " + getName() + " leaves at " + current.getFinishTime());
        }
    }
}
```

```

        GUI.getGui().addText("\nClient " + current.getClientNo() +
" was served at queue " + getName() + " leaves at " + current.getFinishTime() +
"\n");
    } catch (InterruptedException e) {
        //e.printStackTrace();
    }
}

}

public String toString() {
    String result = "";
    result = "" + this.clients.size() + " clients in the queue. Waiting
time: " + this.waitingTime + "\n";
    return result;
}

```

4.3. Class Scheduler

Scheduler class is responsible for dispatching clients and initializes the servers.

As attributes it has an integer representing the number of queues (private int noOfQueues) and an ArrayList of queues (private ArrayList<Queue> queues). Also an attribute can be found for logging.

In the constructor the queues are initialized. It has a method for dispatching clients, where the best queue is found (according to the time strategy).

Examples:

```

public String dispatchClient(Client c) {
    int index = 0;
    try {
        int min = this.queues.get(0).getWaitingTime();
        for(int i = 1; i<this.noOfQueues; i++) {
            int aux = queues.get(i).getWaitingTime();
            if(aux < min) {
                min = aux;
                index = i;
            }
        }

        c.setFinishTime(min); // set finish time for the client;
        queues.get(index).addClient(c);
        System.out.println("Client " + c.getClientNo() + " goes to queue
" + index);
        LOGGER.log(Level.INFO, "Client " + c.getClientNo() + " goes to
queue " + index + "\n" );
    }
}

```

```

    }
    catch (InterruptedException e) {
    }
    String result = "";
    result = "Client " + c.getClientNo() + " goes to queue " + index +
"\n\n";
    return result;
}

```

4.4. Class Manager

Class Manager is a class that controls the threads and generates the random clients.

The attributes are:

```

private int time = 100;
private int maxServiceTime = 10;
private int minServiceTime = 5;
private int numberOfQueues = 3;
private int numberOfClients = 100;
private int minArrivalTime = 2;
private int maxArrivalTime = 4;
private int currentTime = 0;
private static Manager m;
private Scheduler scheduler;
private ArrayList<Client> clients;
private ArrayList<Client> servedClients;

```

The names are representative, for some attributes default values are given (mostly used in testing phases, but I left them there, "nu strica"). An attribute for logging can be found.

In the constructor, actionListeners are added to the gui's buttons, and all attributes are instantiated. The different servers are started. By constructing an object, the simulation starts automatically, because it has as an attribute a static instance of himself.

There are other methods, like generateClients(int n) which generates n random clients, or methods for calculating statistics, printing and logging.

Method getAvgWaitingTime() calculates and prints the average waiting time after the simulation is finished, getPeakHour() calculates and prints the peak hour.

Method run() in this class is the base of the simulation, since it works in real time and shows the evolution of the stack, gets new clients and inserts them in queues, and handles all the servers, in addition it lists the statistics.

The Manager class contains the main function for starting the application.

Some examples:

```
public double getAvgWaitingTime() {
    double avg = 0;
    if(!this.servedClients.isEmpty()) {
        for(Client c : this.servedClients) {
            avg += (c.getFinishTime() - c.getArrivalTime());
        }
        avg /= this.servedClients.size();
    }
    DecimalFormat df = new DecimalFormat();
    df.setMaximumFractionDigits(2);
    GUI.getGui().addText("/nAverage waiting time: " + df.format(avg) + "
minutes.\n");
    LOGGER.log(Level.INFO, "Average waiting time: " + df.format(avg) + "
minutes.\n");
    return avg;
}

private void generateClients(int n) {
    int arrival;
    int service;
    arrival = randBetween(this.minArrivalTime, this.maxArrivalTime);
    service = randBetween(this.minServiceTime, this.maxServiceTime);
    Client c = new Client(arrival, service, 0);
    this.clients.add(c);
    for(int i = 1; i < numberOfClients; i++) {
        arrival = clients.get(i-1).getArrivalTime() +
randBetween(this.minArrivalTime, this.maxArrivalTime);
        service = randBetween(this.minServiceTime, this.maxServiceTime);
        clients.add(new Client(arrival, service, i));
    }
}

public void run() {
    while(currentTime < time) {
        try {
            System.out.println(currentTime);
            Client c = clients.get(0);

            if(c.getArrivalTime() == currentTime) {

                String result = scheduler.dispatchClient(c);
                GUI.getGui().addText(result);

                servedClients.add(c);
                clients.remove(0);

                for(Queue q : this.scheduler.getQueues()) {
```

```

        GUI.getGui().addText(q.getName() + ": " +
q.toString());
    }

    }
    sleep(1000);
    currentTime++;
} catch (InterruptedException e) {
}
}
System.out.println();
getServedClientNumber();
getAvgWaitingTime();
getPeakHour();
}

```

(Logging and printing instruction are in the documentation from this example of code.)

4.5. Class GUI

Class GUI is a Singleton Class, implements the graphical user interface of the application.

As attributes it has a long list of Swing components like a frame, labels, text fields, buttons, panels, a text area and a scroll.

The layout of the frame is a GridLayout, components are arranged and put in panels (having FlowLayout) and the panels are added to the frame. All these are done in the method prepareGUI() called in the constructor. There also exists some getter methods and a method for checking the inputs' number format.

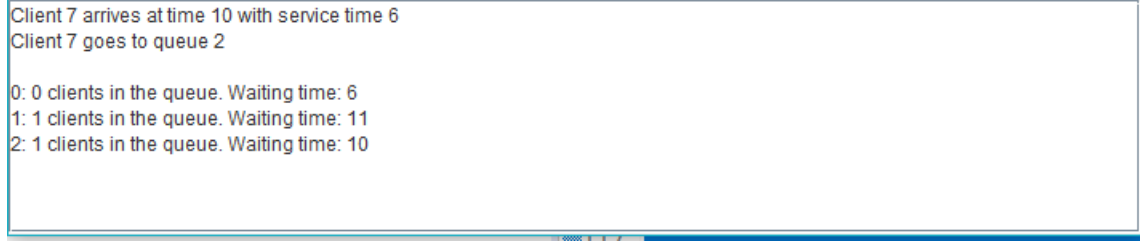
5. Usage and Testing

The user has to data, he must save the data by pressing the button “Save Data”, then he can start the simulation by pressing “Start”. If the user wants to stop the simulation before it reaches the final time, user can press “Stop”, and the simulation is finished. For a new simulation, the application has to be reopened.

It is assumed, that the data introduced is correct from a logical point of view.

If the data introduced is not a number, a dialog window appears and warns the user.


During the simulation, the user can see events and the state of queues:



Client 7 arrives at time 10 with service time 6
Client 7 goes to queue 2

0: 0 clients in the queue. Waiting time: 6
1: 1 clients in the queue. Waiting time: 11
2: 1 clients in the queue. Waiting time: 10

User can see the clients waiting in the queue (the client being served is not counted) and the waiting time. If the simulation reaches the final time, the last state of the queues is displayed, the statistics are displayed, and the simulation finishes by serving all the clients that were already waiting, new clients does not arrive anymore.



Client 71 was served at queue 1 leaves at 149
At 0 31 clients were served.
At 1 23 clients were served.
At 2 19 clients were served.

Average waiting time: 4.77 minutes.

Peak hour was the 1. hour since queues were opened. 29 clients were in total.

The application was tested with different values for the parameters to find and optimize the functioning of the system.

6. Conclusions

During the development of this assignment, some research was needed for understanding the concepts of concurrent programming. After this assignment I can say that I am more familiar with creating and using threads.

A newly learned concept was the Singleton pattern used to create only one object of a class.

As a conclusion, a window builder or a drag and drop tool would help a lot in designing a nice graphical user interface and save time.

Future developments:

- More work on the graphical interface
- Queues representing by animations
- Other statistics can be easily calculated
- Implementation of more strategies (smallest queue strategy)
- More precise dialog windows

7. Bibliography

- <http://users.utcluj.ro/~cviorica/PT2019/>
- <https://www.geeksforgeeks.org/killing-threads-in-java/>
- https://www.tutorialspoint.com/design_pattern/singleton_pattern.htm
- <https://www.geeksforgeeks.org/logging-in-java/>
- https://www.tutorialspoint.com/java/java_multithreading.htm
- <https://docs.oracle.com/javase/tutorial/essential/concurrency/runthread.html>
- <https://stackoverflow.com/questions/1102891/how-to-check-if-a-string-is-numeric-in-java>