Technical University of Cluj-Napoca

Faculty of Automation and Computer Science

Programming Techniques

Homework Assignment 2

Queue simulation

Student: Eric-Andrei Dumea

Group: 30423

Supervising Teacher: Cristina Bianca Pop PhD.

# Table of contents

1 . Problem specification . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3

2 . Problem analysis, modelling, scenarios, use-cases . . . . . . . . .3

2.1 Problem analysis, modelling . . . . . . . . . . . . . . . . . . . . . 3

2.2 Scenarios, use cases . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4

3 . Design . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5

3.1 Diagram . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5

3.2 Classes Design . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5

3.3 Packages & Interfaces . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8

3.4 User Interface . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8

4. Implementation & testing . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8

5. Results . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10

6. Conclusions, further development, self-taught knowledge . . . . . . . . . . 10

7. Bibliography . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 11

# Problem specification

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

## Short 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.

# Problem analysis, modelling, scenarios, use cases

## Problem analysis, modelling

For this application I used the paradigms of Object Oriented Programming in Java. Object-oriented programming (OOP) refers to a type of computer programming (software design) in which programmers define not only the data type of a data structure, but also the types of operations (functions) that can be applied to the data structure. In this way, the data structure becomes an object that includes both data and functions. In addition, programmers can create relationships between one object and another. For example, objects can inherit characteristics from other objects. The main concepts of Object Oriented Programming are Encapsulation, Abstraction, Inheritance and Polymorphism.

*Inheritance* provides a powerful and natural mechanism for organizing and structuring your software. Classes inherit state and behavior from their superclasses, and you can derive one class from another using the simple syntax provided by the Java programming language.

*An interface* is a contract between a class and the outside world. When a class implements an interface, it promises to provide the behavior published by that interface.

*A class* is a blueprint or prototype from which objects are created.

Polymorphism is the ability of an object to take on many forms. The most common use of polymorphism in OOP occurs when a parent class reference is used to refer to a child class object.

Encapsulation in Java is a mechanism of wrapping the data (variables) and code acting on the data (methods) together as a single unit. In encapsulation, the variables of a class will be hidden from other classes, and can be accessed only through the methods of their current class.

Java is a multi-threaded programming language which means we can develop multi-threaded program using Java. A multi-threaded program contains two or more parts that can run concurrently and each part can handle a different task at the same time making optimal use of the available resources specially when your computer has multiple CPUs.

By definition, multitasking is when multiple processes share common processing resources such as a CPU. Multi-threading extends the idea of multitasking into applications where you can subdivide specific operations within a single application into individual threads. Each of the threads can run in parallel. The OS divides processing time not only among different applications, but also among each thread within an application.

Multi-threading enables you to write in a way where multiple activities can proceed concurrently in the same program.

An application like this can be used by big retail companies, in order to find the best balance between number of cashiers and waiting times, thus offering the customers a better experience with less financial effort. Also, this system can be implemented for many frameworks, like airports, fast food chains, or simpler said, every single business that has a queue included.

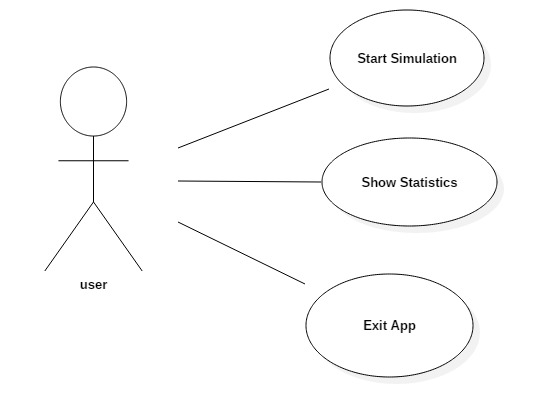
Queues are existential since the dawn of time. People will always form a queue when they need the same thing. This app simulates the behavior of some queues in time, a natural, human behavior.

Based on the above, the main entity of this project was the queue, which is modeled on the framework of Customers (hence the class). Also, I used multithreading programming, hence, I needed a scheduler class, in my case, Model. Also the GUI (Graphical Unit Inteface) was implemented in its separate class, and a class Tester is used to have the main() method .

This application should simulate a series of clients arriving at the queue, entering a queue, waiting, being served and finally leaving the queue .

## Scenarios, use cases

Use case diagrams are usually referred to as behavior diagrams used to describe a set of actions (use cases) that some system or systems (subject) should or can perform in collaboration with one or more external users of the system (actors). Each use case should provide some observable and valuable result to the actors or other stakeholders of the system.



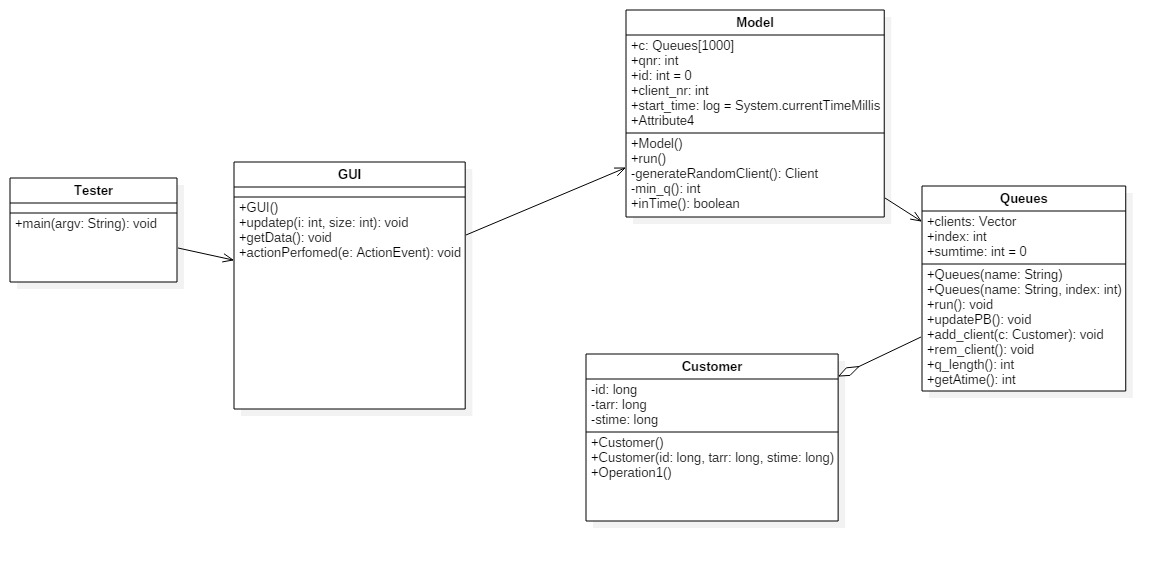
The program will have a basic input interface with buttons and some textboxes. One must provide a valid input, so there will be no non- positive values in the textboxes. Output is interactive, the progress of the queues being tracked by some progress bars, a common concept for almost any basic personal computer user, being present in every Operating System architecture. Also, a log will be shown, tracking the progress in text, showing the time stamps of the clients when enqueuing and dequeuing.

# 3.Design

## 3.1 UML Class Diagram

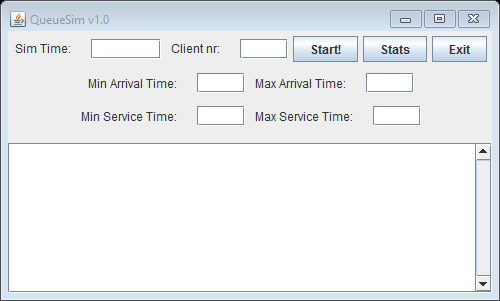
In software engineering, a class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects.

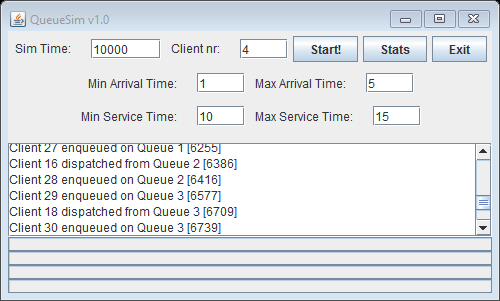
The class diagram is the main building block of object-oriented modelling. It is used both for general conceptual modelling of the systematics of the application, and for detailed modelling translating the models into programming code



## 3.2 Classes design

1.GUI Class

The “View” class implements the interaction with the user, also known as GUI or Graphical Interface Unit. The window that pops up at runtime:

After inserting the desired simulation values, the window looks like this:

The Tester class contains the main method, *public static void main(String[] args),* which creates a new instance of the GUI class, thus calling the constructor, followed by the f*.setVisible(true)* instruction, making the GUI visible. The constructor builds up the GUI part, using the javax.swing framework. Here, we have a JFrame called *appFrame* that is actually the window that pops on the screen, the canvas for the other visual components. Also, I used a BoxLayout layout JPanel as a container for the other components. I divided the components into blocks ( divide et impera ), assembling them in the container. I used some smaller JPanels using FlowLayout to achieve this.

The GUI class implements the *ActionListener* interface, thus it contains the *public void actionPerformed(ActionEvent e)* method. This method specifies the behaviour of the application at different stimuli (e.g. – Start button has been pressed).

When the “Stats” button is pressed, a new JFrame will pop up. Closing this window will just hide it, the app will still run.

Also, the Tester, as the name states, contains developer’s information, in the form of the *testing(. . .)* method. It contains hardcoded strings of polynomials that helped at the debugging. (now commented)

2. Customer class

This class is the nucleus of the application, because every Queue will be splitted into Customers. We have only three fields, **private** **int** t\_arr **private** **int** t\_dep **private** **int** stime which contain the exponent and the coefficient of the monomial. We have two constructors for this class, one without parameters, designed specifically for the auxiliaries needed in development, which makes all the fields null, and one with parameters, that instantiates the fields.

Also, there was a need of setters and getters for both fields, as they are required in every single operation .

1. Queue class

If the Customer class is the nucleus of the project, the Queues class is the atom of this application. It contains three fields: a vector of customers, an index needed for the progressbars, and sum\_time, which is needed for the average times. I needed a method to act like a getter for the size of the vector.also, I have some getters for the fields in this class.

This class extends the class Thread, which means that the method run() had to be implemented. This method contains vital synchronization elements, and an infinite while-loop that calls another method in this class, rem\_customer(). This method removes the first customer in the list, while there are clients in it and prints the log in the text area on the GUI and on the std\_input.

Also, there is the constructor which instantiates the fields.

Also, I used an inline-comparator definition for sorting the arr

1. Model class

This class contains the methods to do operations on queues. Also it contains many time-related fields, a queue vector and the queue number and queue number.

This class extends the Thread class, thus, it needs to implement the run() method. In this run() method, we add the clients into their suiting queues. The queue index at which a new client is to be added is computed in the min\_q() method.

The method generateRandomClient() returns a client with random arrival time, from the given input, and a random serving time, also given at the input.

The method inTime() returns true if and only if the simulation time is not exceeded.

## 3.3 Packages and interfaces

A Java package is a mechanism for organizing Java classes into namespaces. Java packages can be stored in compressed files called JAR files, allowing classes to download faster as a group rather than one at a time. Programmers also typically use packages to organize classes belonging to the same category or providing similar functionality.

In this application, I used:

* javax.swing.\* , java.awt.\* and java.util.\* packages mainly for the GUI part

Interfaces used: ActionListener implemented by the Graphical Unit Interface class ( GUI ).

## 3.4 User Interface

As I’ve previously mentioned, the user interface is pretty much basic and easily-understood, even by nonfamiliarized people. When running the application, a window will open and it will provide to the user the possibility of giving inputs to choose the parameters of the simulation. This window is constructed in the GUI class using some predefined classes and instructions.

Also, the queue progress is shown on some progress bars, such that every user, no matter how unexperienced is, can track the evolution od them.

## 3.5 Algorithms

Choosing from all the methods implemented in this program, I consider the run() method from class Model the most complex one, as it controls every move made by the application. It acts like a control unit for everything, doing most of the computations required.

*public void run(){*

*try{*

*int i=0; // index care parcurge*

*while( i<client\_nr&&inTime()){*

*i++;*

*Customer cl=this.generateRandomClient();*

*int min\_index=min\_q();*

*System.out.println("Client "+id+" enqueued on Queue "+ (min\_index+1)+" ["+cl.getTarr()%1000000+"]");*

*GUI.ta.append("Client "+id+" enqueued on Queue "+ (min\_index+1)+" ["+cl.getTarr()%1000000+"]\n");*

*this.wtime+=cl.getStime();*

*c[min\_index].add\_client(cl);*

*sleep(160);*

*}*

*//System.exit(0);*

*}*

# 4. Implementation and testing

## Implementation

The implementation process began with the class diagram and some sketches of how the program should work. After considering the values needed as an input and the requirements of the program, as well as the final output, the actual implementation of the classes followed.

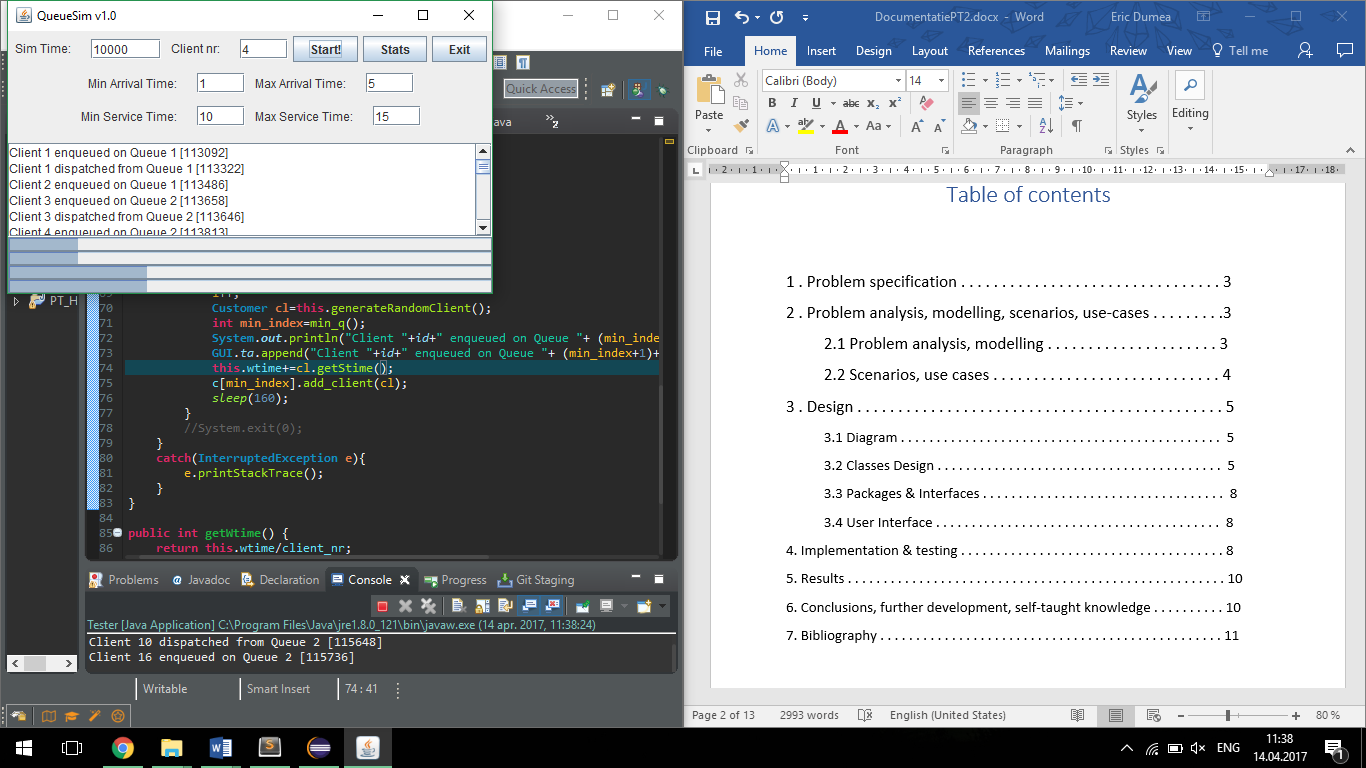
As an input condition, all the input parameters should be non-negative numbers. Also, the start button should be pressed for the app in order to produce and run the app .

After the parameters were correctly introduced, the user presses “start” and the application begins its execution. A model thread is started, and the model starts a thread for each queue, inserting and removing clients.

The operation that raised many difficulties, was the thread synchronization, using sleep(), notifyAll() and wait() methods. Also, updating the GUI from the threads was not that much of an easy job, involving static fields in every class.

# 5. Results

The user interface provides, even to the users with little knowledge in this subject, the possibility of simulating queue evaluation without any knowledge of statistics and drawing due to the fact it is easy and very simple to use. The results are displayed in an easy-to-understand form, and the Exit button add some flexibility to this application. Also, the stats button allows the user to see statistics about the evolution of the queues, and enables them to view the results of the simulation, after it has ended. The window while running:



# 6. Conclusions, further development, gained knowledge

In the end, being able to implement something so dynamically and simulating such an often encountered situation of everyday life while combining everything with the concepts and rules of Object-oriented programming was a challenge with a self-rewarding result. As a further development, the first thing that should be a better exception system, especially at the input.

The problem specification presented itself as an interesting subject with many possibilities. Considering that it was the first program that used threads, the process of making was mostly based on a trial-error method and a lot of additional research which proved to be successful in the end.

Updating the graphical user interface in real time was also a challenge that needed extra attention and studying. Working with threads turned out to impose a lot of additional concepts and ways of implementation on top of the basic Object-oriented paradigms.

There is still a lot of space for improvements in order to make the program more efficient. Some of the algorithms could be optimized and maybe implemented in a different way. Some constrains can be imposed on the input, meaning that more exceptions can be handled with and there can be created more statistics for the simulation. Moreover, the animation can be done using some figures and more advanced graphic. The graphical user interface can be improved. There could be added some colors or given a different layout of the components in the frame.

This project/assignment taught me, as a first thing, to do a proper GUI using code only, to use Junit and to do a proper documentation (as in following a structure and using English extensively). Also, the main benefit of this project was the refresh on the Java coding skills.

# 7. Bibliography

* <http://coned.utcluj.ro/~salomie/PT_Lic/3_Lab/HW2_Tema2/Tema2_HW2_Barem.pdf>
* <https://stackoverflow.com> – for some quick answers to basic questions
* <http://users.utcluj.ro/~jim/OOPE> and the moodle Lab session guides of Mr. Joldos.
* <https://docs.oracle.com/javase/tutorial/java/concepts/>
* <https://en.wikipedia.org/wiki/Class_diagram>
* <https://www.tutorialspoint.com/java/java_multithreading.htm>
* <http://coned.utcluj.ro/~salomie/PT_Lic/3_Lab/HW2_Tema2/Tema2_HW2.pdf>
* <http://courses.caveofprogramming.com/>
* <http://inf.ucv.ro/documents/tudori/laborator8_53.pdf>
* <http://www.uml-diagrams.org/use-case-diagrams.html>