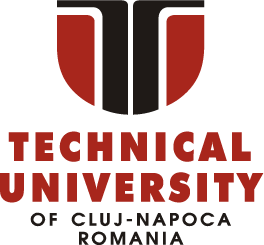
Technical University of Cluj-Napoca

Programming Techniques

Laboratory Assignment Two Documentation

Queues Simulator

**

Teacher: prof. Dr. Ioan Salomie

Laboratory teacher: Dr. Cristina Bianca Pop

Student: Nemeș Mihnea-Andrei

Group: E\_30424

Table of Contents

[1.Assignment Objective 3](#_Toc69174023)

[2.Problem analysis, modelling, scenarios, use cases 3](#_Toc69174024)

[3.Design 4](#_Toc69174025)

[4.Implementation 8](#_Toc69174026)

[5.Results 10](#_Toc69174027)

[6.Conclusions 10](#_Toc69174028)

[7.Bibliography 11](#_Toc69174029)

# 1.Assignment Objective

Design and implement a simulation application aiming to analyze queue based systems for determining and minimizing clients’ waiting time. The implemented application should simulate in a simulation time defined before, a series of N clients arriving for service entering Q queues who wait, get served, and leave the queues.

# 2.Problem analysis, modelling, scenarios, use cases

a. Problem Analysis

Queues are a common appearance in day to day lives, defined as a line or sequence of people or vehicles awaiting their turn to be attended to or to proceed. In computing, a queue is a list of data items, commands, etc. stored so as to be retrievable in a definite order, usually the order of insertion. In our case, the main objective of a queue is to provide a temporary place for a “client” to wait before receiving a “service”. The management of queue-based systems is interested in minimizing the amount of time the clients wait before being able to receive the service. There are multiple ways in which a servicer can achieve this objective, some more cost efficient than others. The most obvious solution would be increasing the number of queues, but this costs a lot of money and implies an increase in the needed space for providing services (in our case processors).

In computer science, a thread of execution is the smallest sequence of programmed instructions that can be managed independently by a scheduler, which is typically a part of the operation system. The implementation of threads and processes differs between operation systems, but in most cases, a thread is a component of a process. Multiple threads can exist within one process, executing concurrently and sharing resources such as memory, while different processes do not share these resources. In particular, the threads of a process share its executable code and the values of its dynamically allocated variables and non-thread-local global variables at any given time.

b. Modelling

The goal of this project is for the user to be able to simulate the way in which queues behave in real life, for instance the queues formed at a supermarket by inserting in the text fields of the interface:

* Simulation Interval (*tsimulation Max*)
* Number of Clients (N)
* Number of Queues (Q)
* Minimum and Maximum arrival time (tarrivalMIN <= tarrival <= tarrivalMAX)
* Minimum and Maximum service time (tserviceMIN <= tservice <=tserviceMAX)

After the user introduces the input, presses start and the program checks its validity, the results will be displayed in real time in the TextArea present in the GUI. There the user can follow the evolution of the queues during the simulation time of the program. If any one of the inputs is wrongly imputed the program will throw an error message and will not start the simulation until the mistake is rectified.

During each simulation a text file is created containing the evolution of the program as seen on the screen with the addition of some additional information representing simulation results such as Average waiting time, Average service time and Peak hour.

c. Scenarios and Use cases

A Use case is a list of actions or event steps typically defining the interactions between a role (known in UML as an actor) and a system to achieve a goal. The actor can be either an external system or a human.

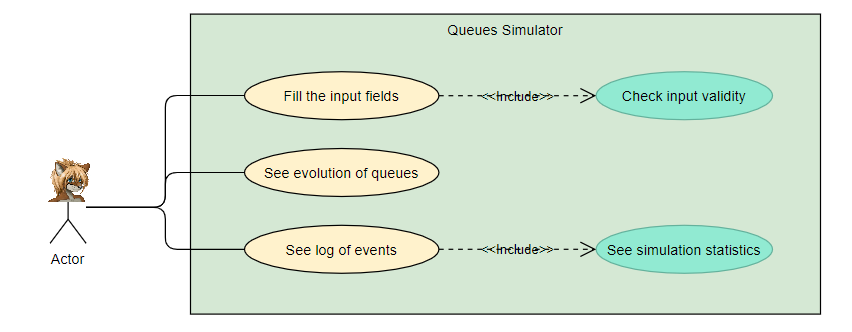
The use cases are in strong correlation with the steps that the user has to follow, this is one of the reasons why my user interface design was made in a friendly manner.

Use case: Start Simulation

Primary actor: the user

Main: Success Scenario: The user inserts all the required fields -> click the button “Start Simulation” -> all the data he introduced was correct -> the application shows the real time evolution of the queues in the interface -> the application generates a text file containing the evolution of the queues + additional information regarding statistics

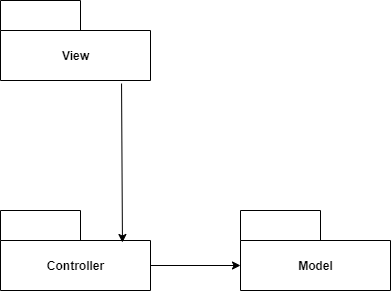
Alternative: Incorrect data: The user inserts all the required fields -> click the button “Start Simulation” -> the introduced data is not correct (introduced data is not numbers, or the min arrival/service times are bigger than the max arrival/service times) -> the application throws an error message representing the field that was not introduced correctly. This scenario leads the application to return to the initial state, waiting for the user to introduce the input parameters.

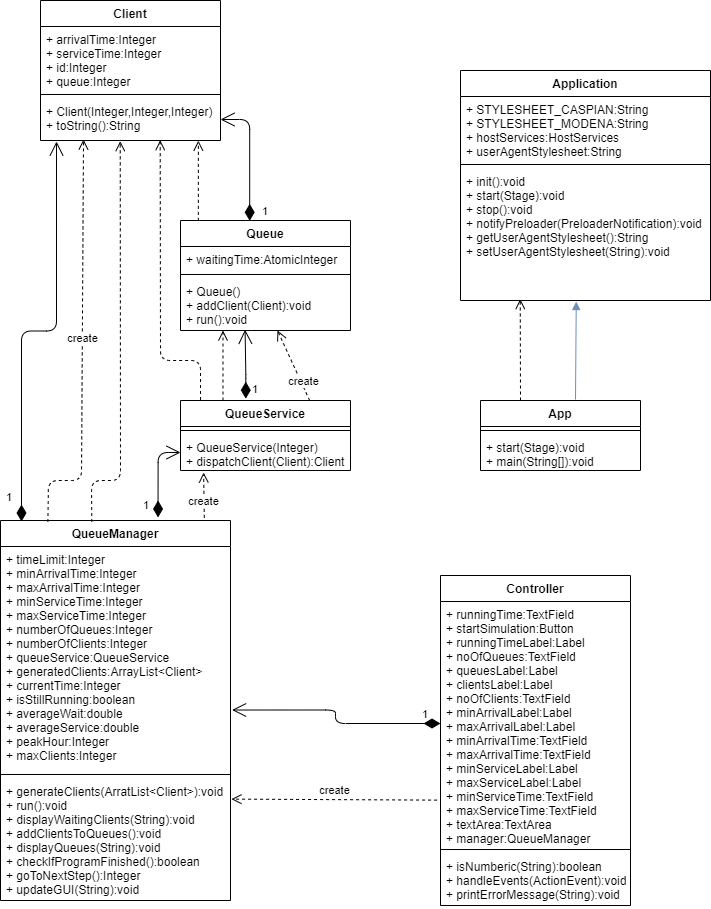


# 3.Design

a. UML Diagrams

1. Package Diagram



2. Class Diagram

b. Data Structures

The data structures used in my project are mostly primitive data types such as Integer, boolean, double or String, but also some complex data types, such as ArrayList and BlockingQueue. Aside from data structures from java, I have also used objects I have created myself: Client, Queue and QueueManager.

The decision to use ArrayList was made because it is an easy data type to work with and much more efficient than using vectors. As for BlockingQueues, their use is justified because in this project we are working with Threads and thus we need to make sure they are safe during the execution of the project. BlockingQueue also implements Runnable, making it a class that is able to run a thread.

c. Packages

Java Packages are used in order to organize modules and group classes and interfaces that are related. In Object-Oriented-Programming(OOP) development, the most encountered architectural pattern is the Model-View-Controller(MVC). It is considered by many developers as a useful pattern that allows the reuse of object code and it helps to reduce the time to develop apps with graphical user interfaces.

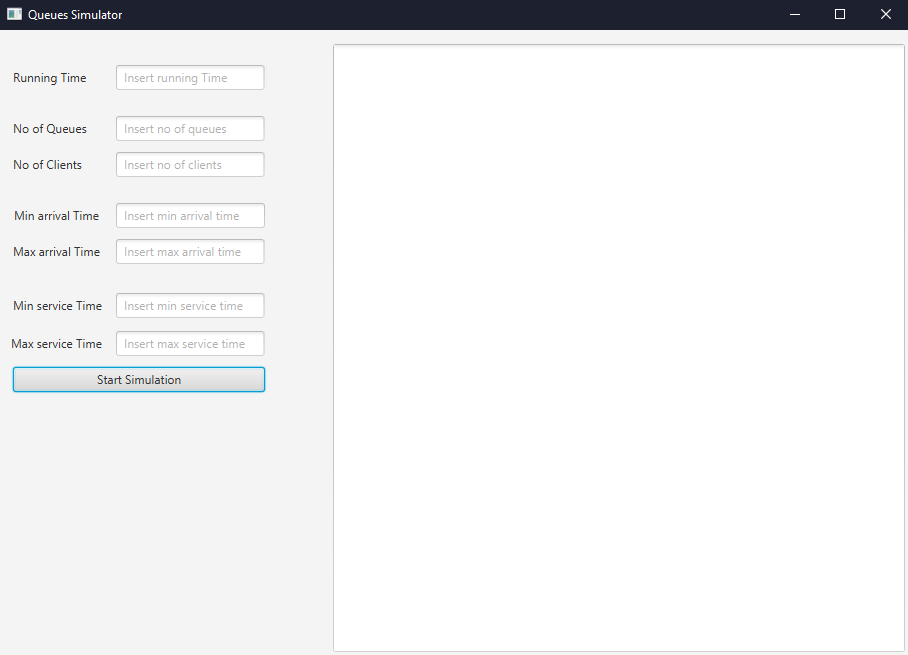
The MVC pattern is comprised of:

* Model- represents the logical structure of of the data in a software application and it does not contain information about the GUI. In my case, in this package I have the Client, Queue, QueueManager, QueueService classes.
* View- represents a collection of classes that represent the elements that are present in the user interfaces(for example buttons, textfields, labels, textareas etc.). In my case, in this package I only have an fxml file that specifies the layout of my graphical interface elements.
* Controller- represents the mediator between Model and View. In this package I have the Controller class, which implements the functionalities of the elements present in the user interface,

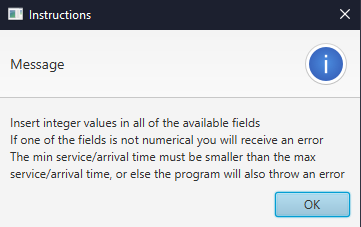
Inside this package I also have a class App that runs the GUI.

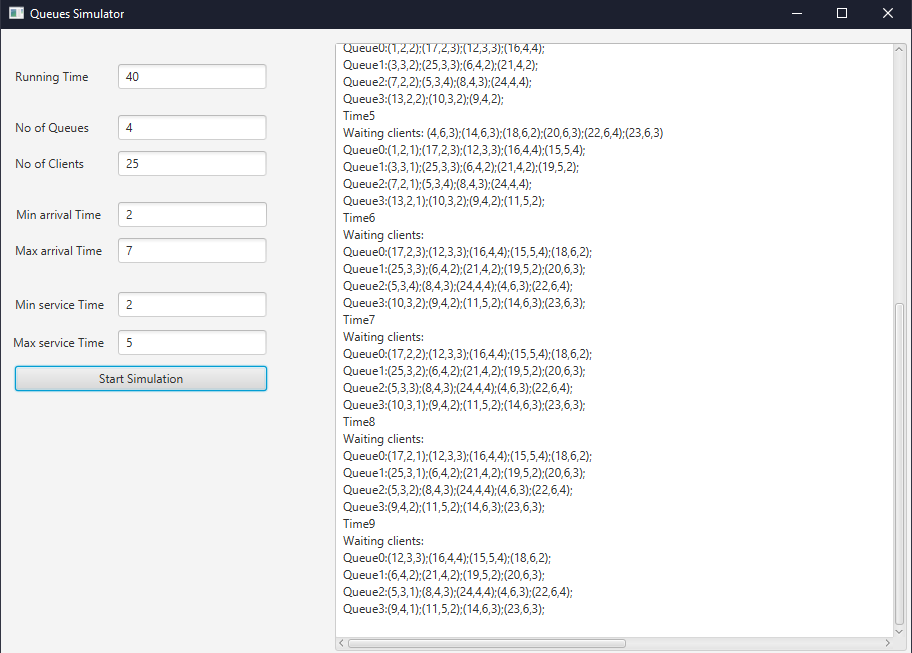
d. Graphical User Interface

The interface represents the link between the software application and the user, and it must be friendly and easy to use, so that the user does not encounter any confusion when trying to run the program.



When the user first opens the application, an information message appears on the screen informing the user of the rules when wanting to run the program. This information covers the type of numbers needed in the fields, what happens when one of the fields is not a numerical value and what happens when a min field has greater value than a max field, be it the service or arrival time.





This picture presents the GUI during a random simulation time in the running of the program, where it displays the current simulation time, the clients waiting to enter a queue, and the state of every queue. When the simulation is over, the TextArea in the GUI will display a message saying the simulation is done, and additional information regarding statistics is written in the text file.

# 4.Implementation

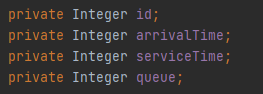
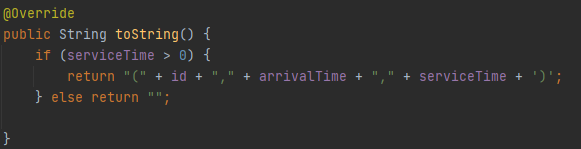
Class Design

Classes are based on the idea that a problem can be decomposed into smaller problems that are easier to deal with. My program was realized following the MVC architectural pattern, so my app has 3 main parts:

Model package:

a. Client

The data fields implemented in this class are id, arrivalTime, serviceTime, queue, all 3 being Integer. Getters and Setters are implemented for all these fields. The only other method implemented inside the Client class is a toString() method, used for displaying a client inside a queue/inside the waiting clients list.

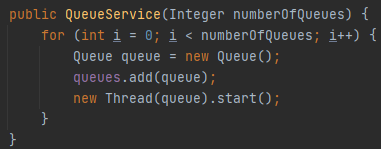
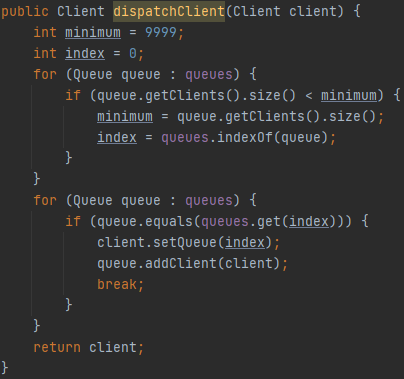
b. Queue

The data fields implemented in this class are clients of BlockingQueue<Client> type and waitingTime of type AtomicInteger. This class implements Getters and Setters for both these fields, an empty constructor, an addClient() method which receives a client as a parameter and adds it to the list of clients of the queue. The only other method is the run() method which implements the removal of clients when their service time reaches 0, meaning they have completely received their service.

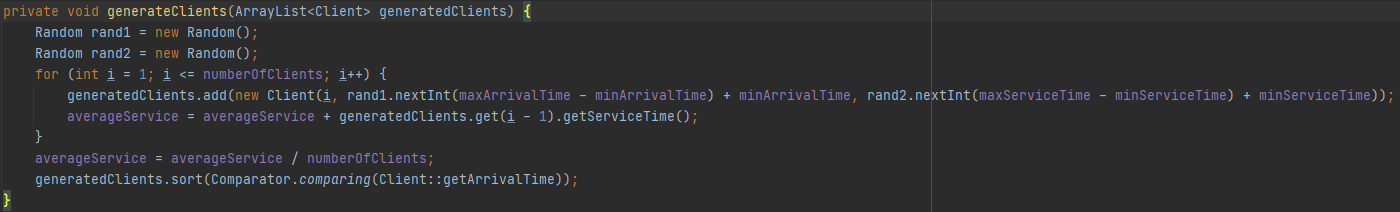
c. QueueService

This class implements the addition of clients to a queue based on the strategy that a client should enter the queue with the smallest number of clients currently in the queue. The only other method is the constructor of the class, which receives as parameter a number representing the number of queues and starts the respective thread for each queue. The QueueService class also has a getter for the queues.

d. QueueManager

This is the “most important” class of my project, as it constructs and directly the execution of the project. This class implements a method named generateClients() that gets as parameter an empty list of clients, generatedClients and using the java class Random generates for each client id (1 <= id <= noOfClients) a random arrival time and service time, between the values given in the GUI. The clients are then sorted in increasing order based on the arrival time.



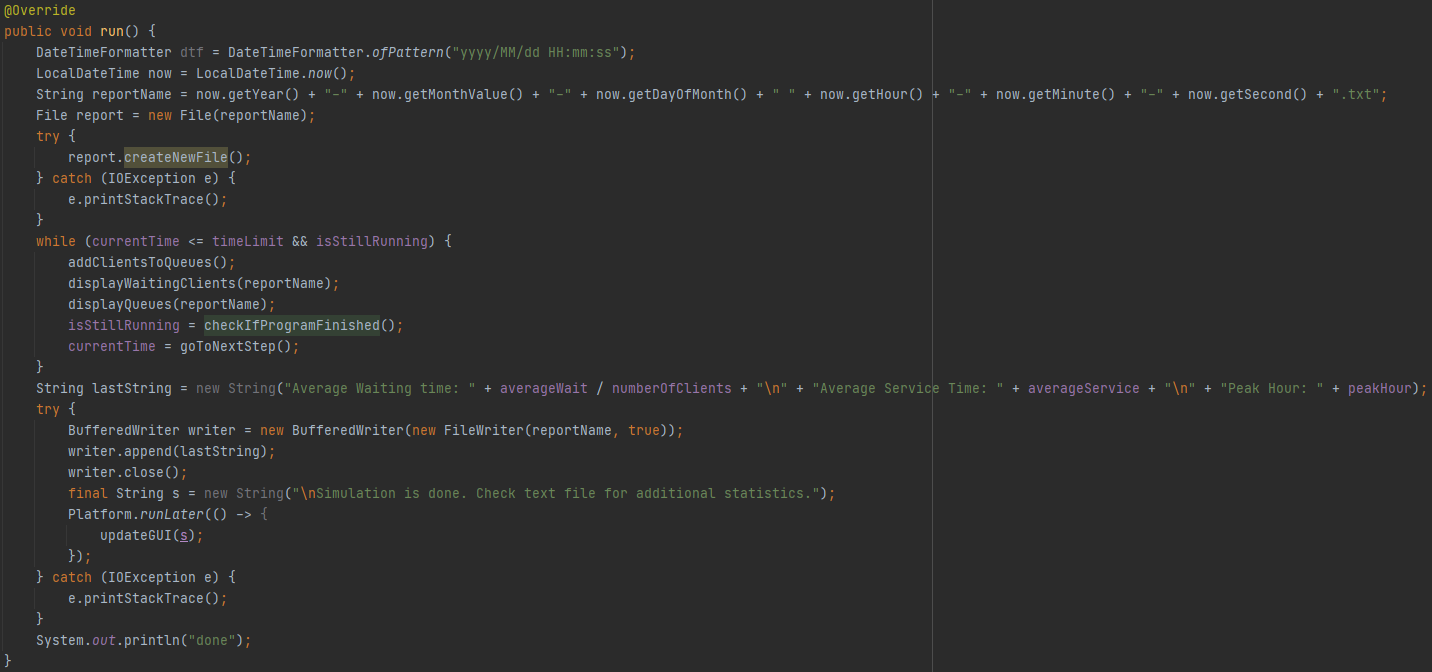
The “main” method of the QueueManager class is the run method in which all the operations are done. For writing in a file I have used the Java Class File and BufferedWriter. For each simulation of my program, it creates a new file according to the pattern y-m-d h-m-s. The main part of this method is the while that coordinates, the execution of the program using code written in different methods in order to follow the required limits for method length.

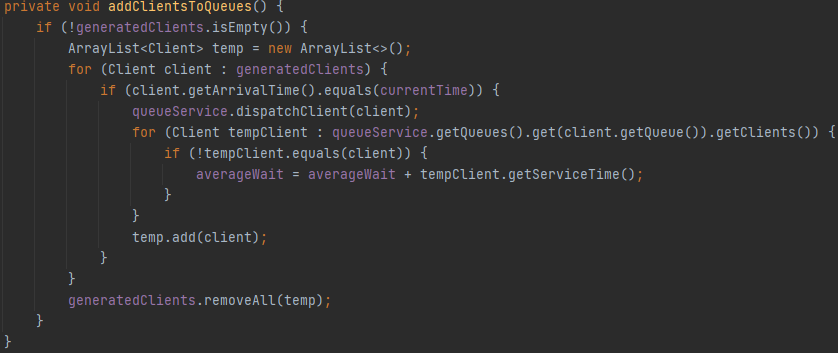
The method addClientsToQueues() checks if there are any clients left waiting to be placed into a queue. In case the list is not empty the program goes through the waiting clients list client by client and checks whether there are any clients that have the arrival time equal to the current time. In case it finds a client that matches that condition, it uses the dispatchClient method to find the best queue to enter the client in. Inside this method the program also calculates the time the client will have to wait in the queue until he will be the head of the queue and adds that time to the averageWait field.

The method displayWaitingClients() is only used to display on the GUI and in the log file the list of clients waiting to be placed in a queue. The method displayQueues() is also only used to display on the GUI and in the log file every queue with their respective clients currently in the queue at the current simulation time. This method also decrements the service time of the heads of each queue, in order for the program to nicely display the service time of clients at each simulation time. Moreover, the method checks if this current time represents the peak hour of the simulation, meaning the time when the queues have the maximum number of clients in them.

The method checkIfProgramFinished() checks whether the program has finished simulating. Determining the finish time of the simulation is a matter of checking whether there are no waiting clients that want to enter queues and all the queues are closed. The result of this method is stored in the boolean isStillRunning field, that appears in the while condition.

The method goToNextStep() increments the current time and puts the thread to sleep for one second.





# 5.Results

For this project I have run the three tests that were given in the assignment resources. These tests are in the files Test1, Test2 and Test3 respectively, the files containing the results of the simulation in real time and the statistics given at the end of the simulation, average wait, average service, and peak hour. Due to the fact that arrival time and service time are chosen randomly for each client, the results of these tests will be somewhat different from one simulation to another, as well as the statistics of the simulation. For example, during one simulation, most clients can have a service time can be mostly centered around 3 and in another simulation the common service time could be centered around 9.

# 6.Conclusions

This assignment offers a good opportunity to remember and apply the OOP paradigms and concepts. Also, because I have never worked with threads before, this project was a nice opportunity to learn how to coordinate the execution of the threaded program. The fact that the proposed project dealt with a real life situation was a good approach because it made it easier to consider the implementation looking at the real life and programming in parallel. What I can take from this project is to not panic when not knowing something I have to use in my work, but to research on my own and try to understand how different things work. Dividing the problem into smaller ones is also very important because projects can get very overwhelming with the number of things a program does.

The project is not perfect by far, huge improvements can be made, like synchronizing threads to ensure that they do not clash with each other. The methods can also be improved to make sure they don’t cause multiple computations to happen in a “clock cycle” thus degrading the output of the program. Also, the GUI could be designed to be friendlier and have a nicer look.

# 7.Bibliography

* <https://www.baeldung.com/java-start-thread>
* Programming Techniques – Lecture Slides by prof. Ioan Salomie
* Assignment 2 Support Presentation – Programming Techniques Teams
* https://openjfx.io/openjfx-docs -for configuring the javafx
* <https://www.jetbrains.com/help/idea/opening-fxml-files-in-javafx-scene-builder.html>
* <https://app.diagrams.net/> - diagrams
* <https://docs.oracle.com/javafx/2/threads/jfxpub-threads.htm> - working with threads and concurrency