Ian Chelaru
Technical University of Cluj-Napoca
2nd year Computer Science
Programming Techniques 2017/2018

Assignment 2 – Specification

1. Assignment's objective

The main purpose of the assignment is to design and implement a simulation application aiming to analyze queuing based systems for determining and minimizing clients' waiting time. The application is written in java programming language. The integrated development environment (IDE) used is Eclipse.

Secondary objectives:

- use threads in order to get a simulation in real time
- create a graphical user interface (GUI) for an easier interaction between the user and the application

2. Features

The application has only one use-case which consists in doing the simulation setup and running the simulation. The simulation consists in generating a number of random clients. Each of them enters an available queue, waits until he will be the first one in its queue, is served and then leaves the queue.

The simulation takes place in real time (1 second in the application is equal to 1 second in the real world).

When running the application an initial frame ("Simulation Setup") shows on the screen (Figure 1). The user has to introduce the input data by filling up every text field. In order for the application to work properly, it is assumed that each input value represents a strictly positive integer number.

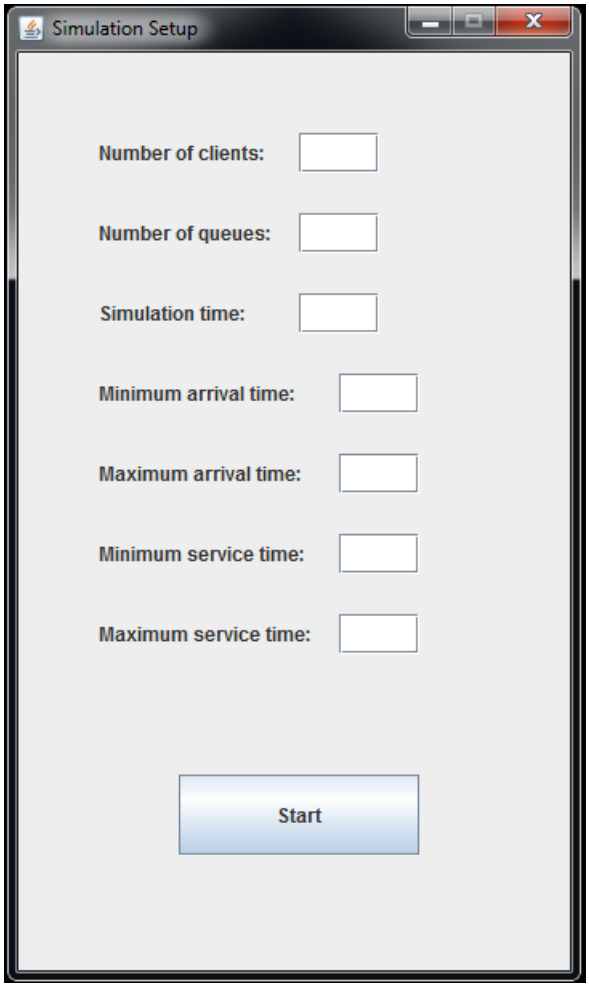

Figure 1 – Simulation Setup (GUI)

The significance of every text field is described below:

- "Number of clients": represents how many random clients will be generated at the start of the simulation;
- "Number of queues": represents how many queues will be available for the clients from the total number of 10 queues. Thus, an extra assumption has to be made: the number of queues is an integer number between 1 and 10 (inclusively);
- "Simulation time": represents how many seconds the simulation will lasts;
- "Minimum arrival time" and "Maximum arrival time" represents the bounds of each client's arrival time;
- "Minimum service time" and "Maximum service time" represents the bounds of each client's service time.

To start a new simulation, the user has to press the “Start” button. A new frame (“Log of Events”) will show on the screen (Figure 2). The “Log of Events” frame will present useful information about the simulation. On the left side of this frame the queue evolutions will be displayed. After each second, the queues may change (clients may enter and/or leave the queues). At each second the state of the queues will be displayed (not matter if they changed or not). On the right side of the “Log of Events” frame the events will be displayed. At each second of the simulation time, the frame will display the clients which leaved or entered a queue. At the end of the simulation the frame returns to the user the average waiting time and the peak time (Figure 3). The waiting time for a client is the amount of seconds he was part of a queue. The peak time represents that time of the simulation when the total number of clients presented at the queues was the highest.

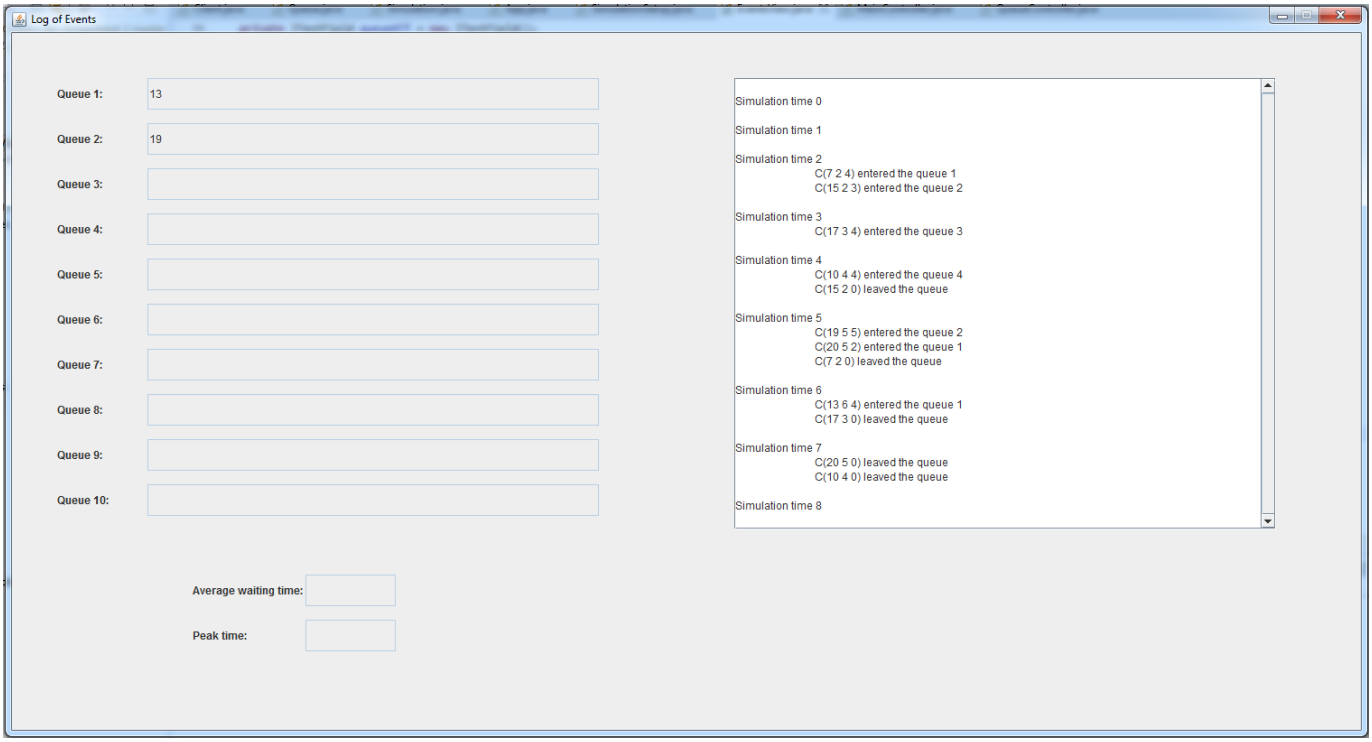

Figure 2 – Log of Events (GUI)

The initial frame “Simulation Setup” will remain open after the simulation begins. In this way the user can run multiple simulations in the same time only by pressing the “Start” button.

The text fields remain unchanged after a simulation is launched. Thus, the user can start a new simulation with the same parameters without rewriting the input data. However, the parameters can be changed and run multiple simulations with different characteristics.

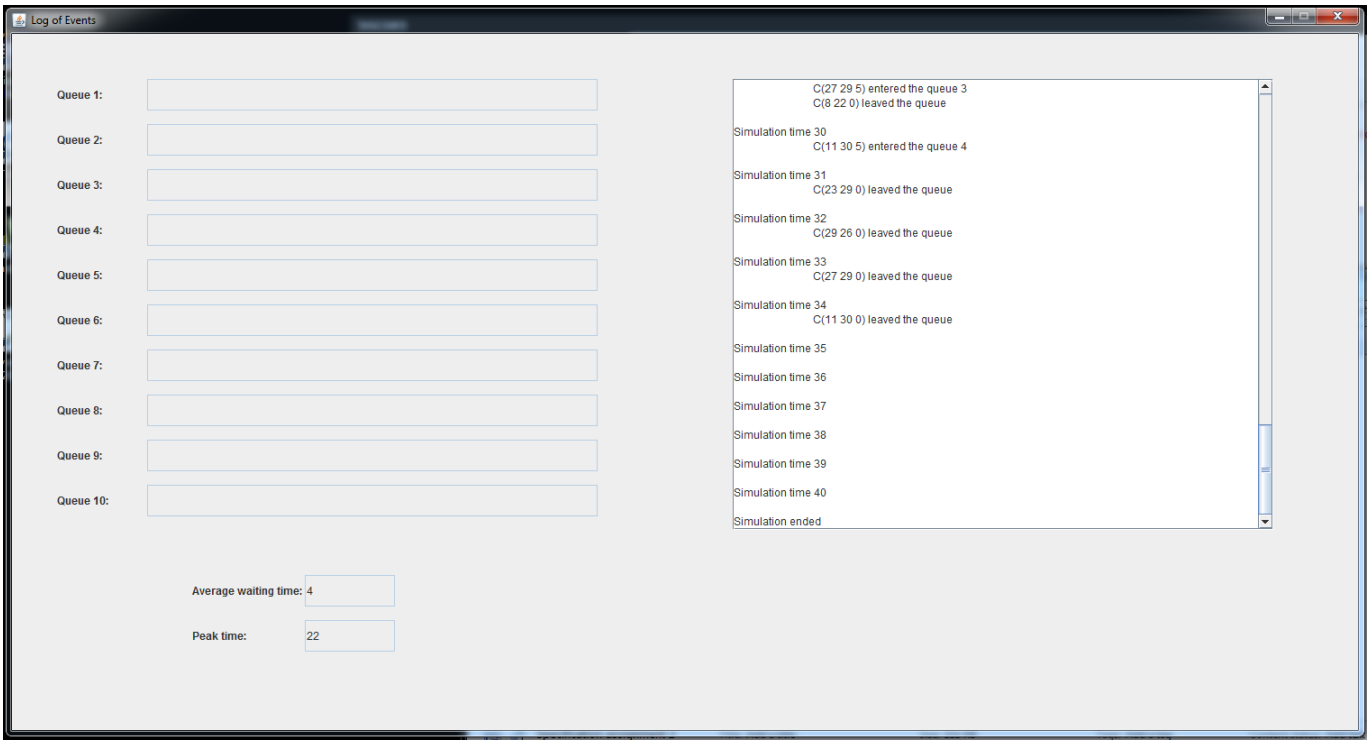

Figure 3 – End of simulation

In order to close one simulation, the user has to press the "x" button from the upper right corner of the corresponding "Log of Events" frame.

In order to close the application the user has to press the "x" button from the upper right corner of the unique frame "Simulation Setup". By doing this, implicitly, every simulation will be closed.

3. Design

In order to design the system, the MVC (model – view – controller) pattern is used. This pattern splits the application into three parts. The model consists in implementing classes for the objects used to solve the problem. These classes have the purpose to manage the input data and to implement the methods which solve the application requirements. The view creates the GUIs which help the user to communicate with the application. The controller transmits data between the model and the view and sets the behavior of these two. The model and the view do not communicate directly. They communicate only through the controller.

The project is structured on three packages: models, controllers and views (Figure 4). Each one contains the classes which handle the corresponding part of the application.

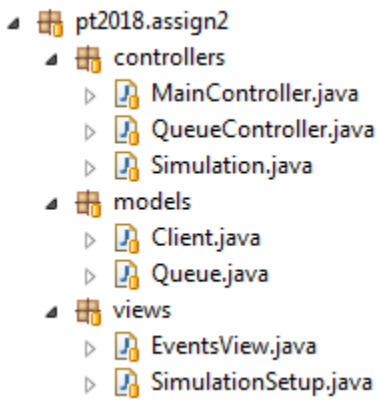

Figure 4 - Packages

Model

The application simulates a system based on clients and queues. Therefore, these two entities are simulated by two classes: Client and Queue. Instances of these two classes are used in order to build a realistic simulation.

Each client object has a unique id which differentiates the object by other clients (like in real world where every person has a unique id). Every client object has an arrival time and a service time. The arrival time represents the moment when the client shows up and enters a queue (for example, if the arrival time is 5, then, after 5 seconds from the start of the simulation, the client will enter one of the queues). The service time represents the amount of seconds the client needs to be served. The service time does not count the time the client has to wait in order to start being served. For instance, if a client “a” enters a queue and there are some other clients before him in the respective queue, then the client “a” has to wait for the other clients to be served. The service time represents how many seconds are between the moment when the client reached the start of the queue and the moment he can leave the queue.

Each queue object is determined by the list of clients which wait at the respective queue. The queue objects respect the FIFO principle (First In First Out). So the first client that enters a queue will be the first one which leaves that queue.

Due to the fact that a queue has more clients there is an aggregation relationship between class Client and class Queue.

View

The application has two classes which implement the view part: “SimulationSetup” and “EventsView”. These two create the frames that are visible to the user. Two classes are needed because they have different responsibilities. The “SimulationSetup” creates the “Simulation Setup” frame and has the purpose of taking the input parameters in order for the simulation to be created by the controller. The “EventsView” creates the “Log of Events” frame which has the purpose of displaying the state of the model at each second of the simulation (it outputs the simulation).

Controller

In order for the application to simulate in real time, threads have to be created. A thread is like a smaller process. Like processes, more threads can be executed in the same time (multi-threading). Thus, for each queue a corresponding thread will be created. All the queues’ threads will execute the same instructions, but every queue will be processed independently. A thread which runs the entire system is also needed.

There are three classes that implement this part of the application: “Simulation”, “QueueController” and “MainController”.

The “QueueController” and the “Simulation” implement the “Runnable” interface. They have to override the void run() method, which is the method that contains the sequence of instructions which the created threads will execute.

The purpose of the “QueueController” is to handle the threads created to simulate the queue behavior. It also have to communicate information about the state of the queue to the “EventsView” class (and this is the reason why I choose to put this class in the controller package).

The “Simulation” class creates the simulation and manages other classes. It generates the clients and queues needed to simulate the system, it starts the threads for each queue and runs the thread of the entire system. It also displays information about the state of the system to the “EventsView” class.

The “MainController” has the purpose of taking the input data given by the user, creates the simulation and starts the simulation’s thread.

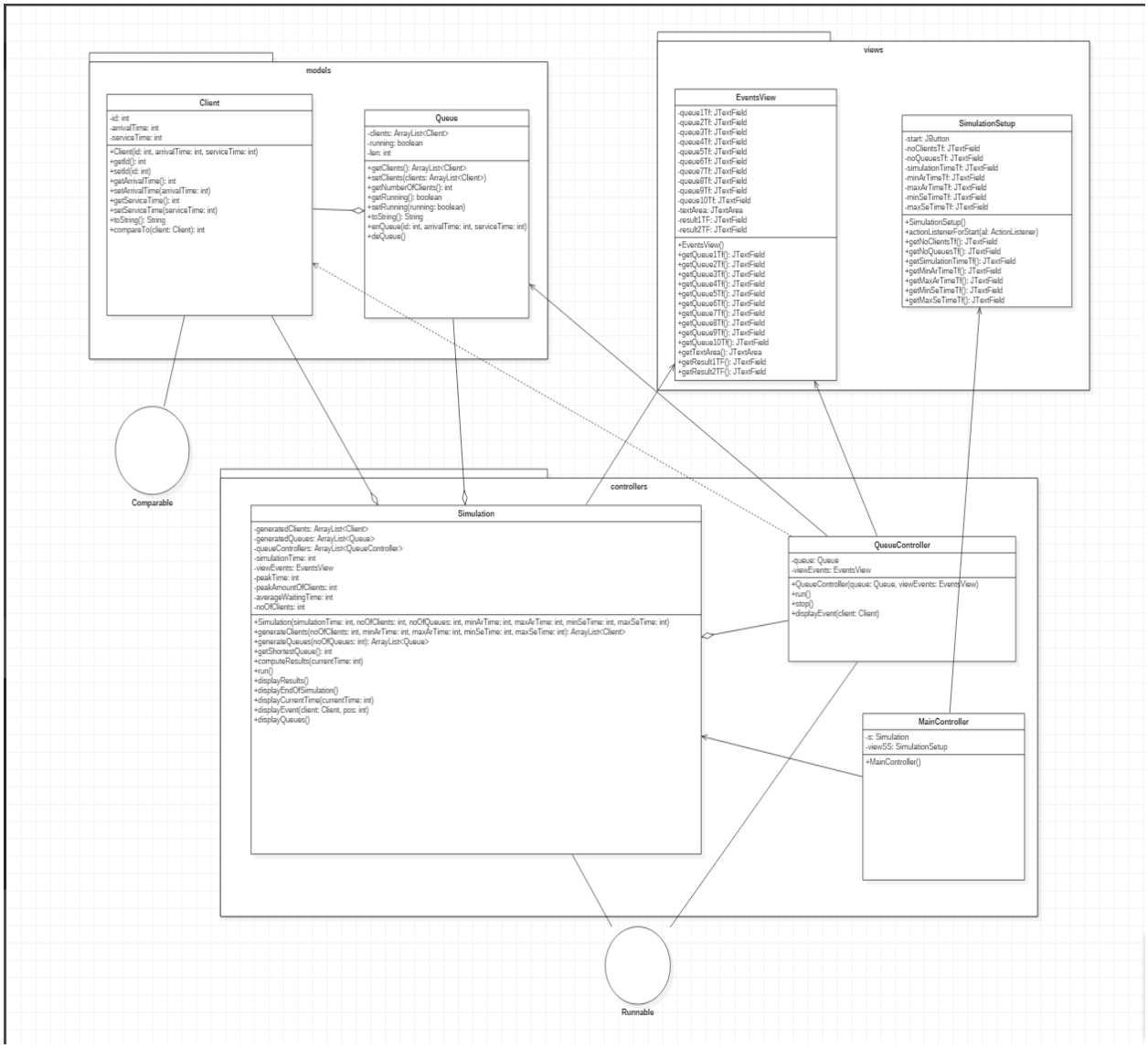

Figure 5 – UML Diagram

Data structures

The ArrayList class, which is a “resizable-array implementation of the List interface” (javadoc) from the java.util package, is used to handle all the lists (arrays) used in the application (clients, queues, queue controllers).

4. Implementation

Javadoc was used in order to specify each class with its attributes and main methods.

5. Conclusions

What I learn from the assignment:

- what threads are
- how to start, run and stop a thread
- how to handle multithreading
- how to sort a collection using the Comparable interface
- how to use a JTextArea and a JScrollPane

Improvements:

- open queues during the simulation, not only at the start of it
- return other information which could be important for the user
- the possibility of having more than 10 queues