**Technical University of Cluj-Napoca**

**Fundamental Programming Techniques**

**Assignment 2**

**Queues Simulator**

**Cimpan Stefan Alexandru**

**Group 30423**

1. **Assignment objective**

**Main objective**

The main objective for this assignment is to design and implement a simulation application aiming to analyze queuing-based systems for determining and minimizing client’s waiting time.

**Sub-objectives**

* Analyze the problem and identify the requirements, described in chapter 2.
* Design the polynomial calculator, described in chapter 3.
* Implement the polynomial calculator, described in chapter 4.
* Test the polynomial calculator, described in chapter 5.

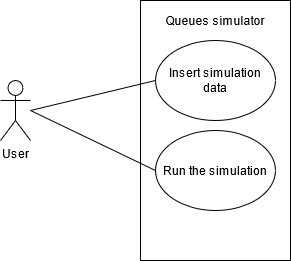
1. **Problem analysis, modeling, scenarios, use cases**

**Functional requirements**

* The application should generate randomly N clients based on the minimum and maximum arrival time and service time.
* Every queue should be implemented in a different thread.
* A log file should be generated after the simulation ends, containing the status of all the waiting clients and the queues.
* The application should display the average waiting time, average service time, and peak hour after the simulation ends.
* The application should have a user interface with two sections, the simulation setup in which the user inserts the needed data in order to run the simulation and the simulation section in which is presented for every second the current time, waiting clients, queues status and, when the simulation ends, to display the simulation results.

**Non-Functional requirements**

* The interface of the simulation should be intuitive and easy to use.



**Use case:** run the simulation

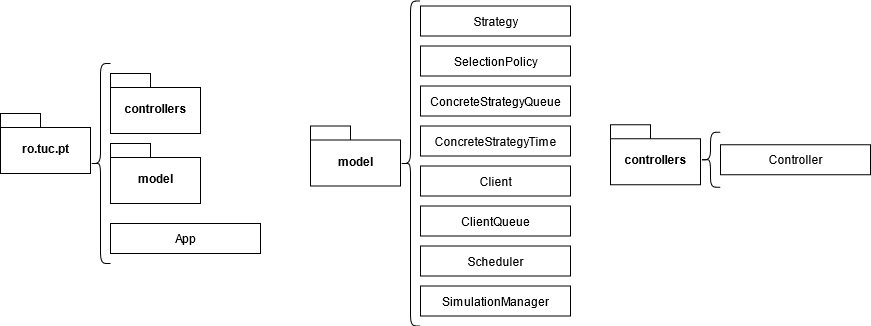
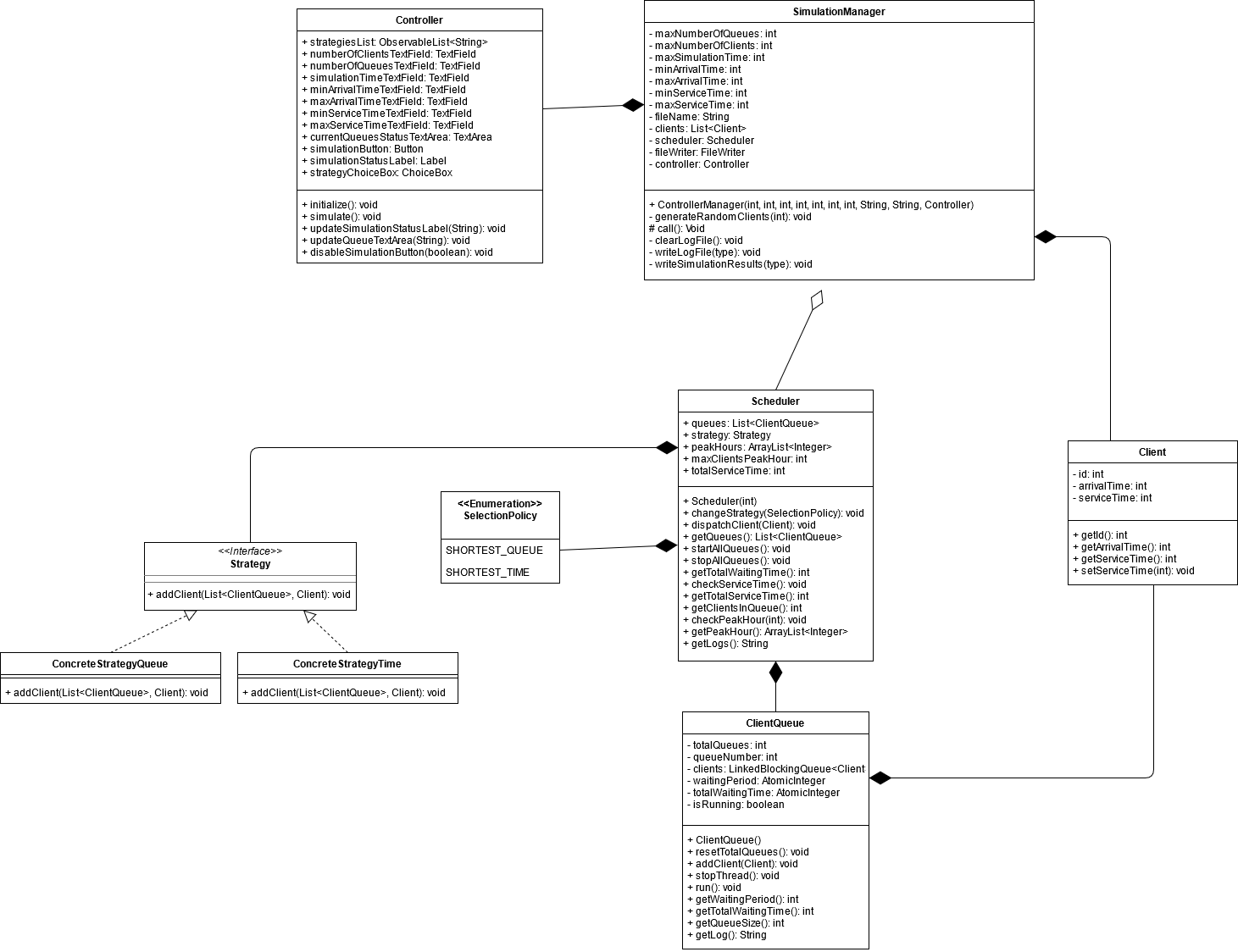
**Primary actor:** user

**Main Success Scenario:**

1. The user inserts the “Simulation setup” section the number of clients to be randomly generated, the number of queues, the maximum simulation time, the minimum and maximum service time, and arrival time for every client that will be generated.
2. The user selects the “Simulate” button in order to start the simulation.
3. The program generates the clients and starts the simulation. Every second in the simulation part of the user interface the status of the simulation will be shown with information about the current time of the simulation, the waiting clients, and the status of every queue.

**Alternative Sequence: Incorrect polynomials**

* The user inserts incorrectly the information for the simulation setup (e.g., the maximum arrival time to be smaller than the minimum arrival time, the maximum service time to be smaller than the minimum service time)
* The scenario returns to step 1.

1. **Design**

The application has the user interface and classes organized in packages using the Model-View-Controller architectural pattern. In the controllers package we have only the controller class, in the model package we have the Client, ClientQueue, ConcreteStrategyQueue, ConcreteStrategyTime, Scheduler, SimulationManager classes, the Strategy interface and the SelectionPolicy enum. For the view part, we have the View.fxml file in which all the elements of the interface are described. This file is saved in the resources folder. The user can select a strategy from two distinct ones. One strategy is based on the number of clients in a queue and in this strategy a new client is added in the queue that has the least number of clients. Another strategy is based on the total waiting time for every queue. In this strategy, a new client is added in the queue that has the minimum total waiting time.

1. **Implementation**

In the Client class we store all the information for every client, the id, arrival time, and service time. The methods in this class are for getting the id, arrival time, and service time and for setting the service time with a new one given as a parameter.

The ClientQueue represents a real queue implemented to be run as a separate thread in which we can add a new client, remove a client from the queue and generate a log string with the current state of the queue. All the clients in the queue are stored in linkedBlockingQueue in order to be thread safe. The waitingPeriod and totalWaitingTime are stored as AtomicIntegers and, in order to store the total number of queues, we have a static attribute, totalQueues, that is incremented every time we create a new ClientQueue and can be reseted by calling the static method resetTotalQueues. In order to keep the thread running we have a Boolean attribute isRunning that will be set to true by default. If we want to stop a ClientQueue thread, then we can call the stopThread method and this method will set the isRunning attribute to false. In the run method we have a while loop in which we pause the thread for a time equal to the front client’s service time. After that we remove the client in the front of the queue.

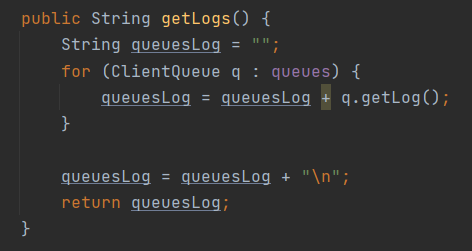
The getLog() method generates and returns a string based on the data of the called queue. Firstly, we declare a new empty string and we append the string “Queue” and the queue id and then we check if the queue has some clients. If the queue is empty, then we write in the string “closed”. If there are some clients in the queue, then we iterate through all the clients and generate a new string to be appended in the string that will be returned. The format for the string for every client is “(client\_id, arrival\_time, service\_time);”. After we generate the log string, we decrement the service time for the client in the front of the queue and we will decrement the waiting period of the queue.



The waiting period is initially set to zero and when we add a new client, we add to the current waiting period the newly added client’s service time. We only decrement this when after we build the log string of the current queue and if we have at least one client in the queue.

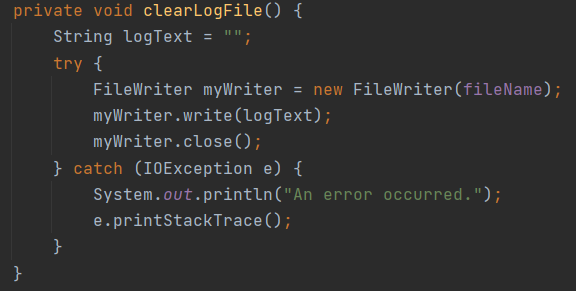
The Scheduler class takes care of all the queues and the result data after the simulation ends. The scheduler creates all the queues and sets the concrete strategy that will be used when we pick the queue in which a new client will be added. In order to start all the queues threads we need to call the startAllQueues method and when we want to stop all queues threads, we call the stopAllQueues method.

The getLogs() method will iterate through all the queues and append all the logs of every queue to a new empty string and then return the newly created string.

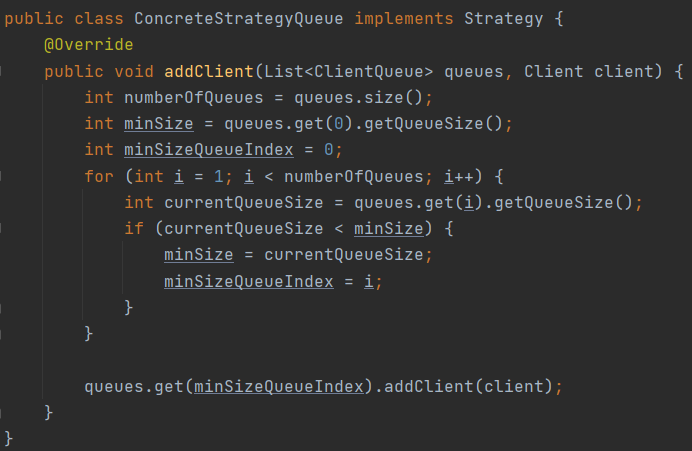
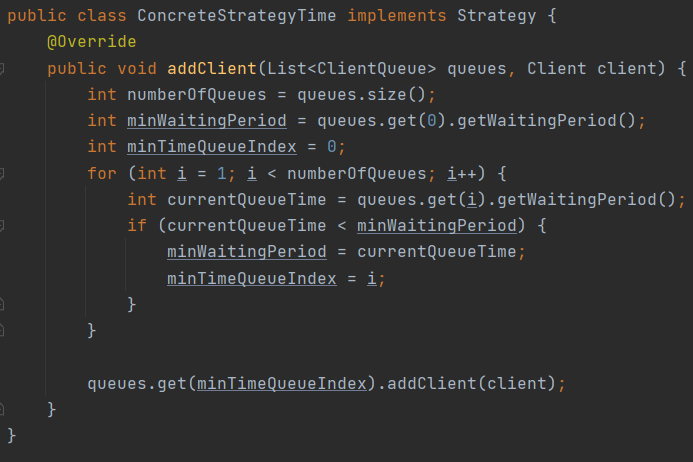


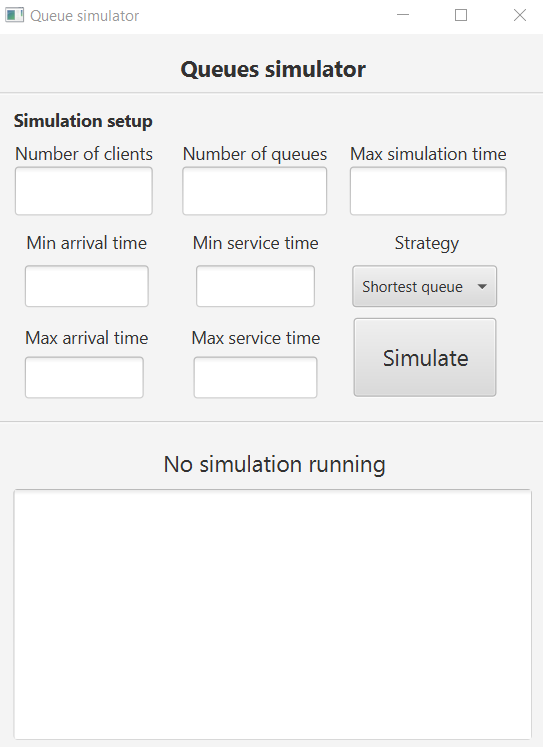
The SimulationManager class takes care of all the initial data that needs to be created and the running of the simulation. In order to update the user interface, we need to inherit the Task<Void> class that allows us to update the interface, which is running on the main thread of the application, from another thread. First, we set all the initial data given as parameters in the constructor, clear the log file, create a new Scheduler, and generate a list of random clients based on the minimum and maximum arrival time and service time. The list of clients is sorted in ascending order based on the client arrival time. Every second we check for clients that have the arrival time equal with the current simulation time and if we find them, we will use the scheduler’s dispatchClient method to add all clients that we found in a queue based on the selected strategy. Every client that we found that has the arrival time equal to the current simulation time will be removed from the clients list. After that, we will write in the log file the status of all the queues and the waiting clients. The simulation stops if we reached the maximum simulation time or if the client list and all the queues are empty. After the simulation stops, we first stop all the threads, then we write in the log file the results of the simulation and we will also display the results on the user interface with the message that the simulation has ended.

The writing in the file will be performed only by the SimulationManager class by using the FileWriter class. In the constructor of the SimulationManager we first delete the content of the log file and then we create a new FileWriter object that will append to the log file the simulation data every second. The SimulationManager class will write in the log file the status of the simulation before we pause the thread for one second and will write in the log file the simulation results after we processed all the clients or when we reached the maximum simulation time. We have three methods that will write in the log file, clearLogFile, writeLogFile and writeSimulationResults. In the clearLogFile method we create a new FileWriter and write an empty string in the log file in order to delete all the previous data. The writeLogFile method builds the string that will be written in the log file by generating a string for the current simulation time, appending the waiting clients if the clients list is not empty and append to the log string the generated logs from the scheduler. After building the log string the method writes it in the log file and then displays it in the text area on the user interface. The writeSimulationResults method computes the average waiting time, total service time, and peak hour and writes all of them in the log file and then on the user interface.



A client can be added to a queue based on 2 strategies: add the client to the shortest queue or add the client to the queue that has the lowest waiting period. This is implemented using the strategy pattern. We have the strategy interface in which we have the addClient method, a SelectionPolicy enum that stores out possible strategyes SHORTEST\_QUEUE or SHORTEST\_TIME, and two classes ConcreteStrategyQueue and ConcreteStrategyTime in which the addClient method is implemented.





The user interface is implemented using the javaFX framework. The interface and all its elements are described in the View.fxml file in the resource folder. The application is build using the Model-View-Controller architectural pattern. In the controller class we have the reference to all the elements in the interface and methods for changing the text of the simulation stats label, for changing the text of the text area in order to display the current status of the simulation, the results after the simulation finishes and the method that is called when we press the “Simulate” button. This method takes all the data needed for the setup of the simulation and creates a new SimulationManager task in order to have access to the application interface frame. The interface is split into two parts, the simulation setup and real-time simulation status.

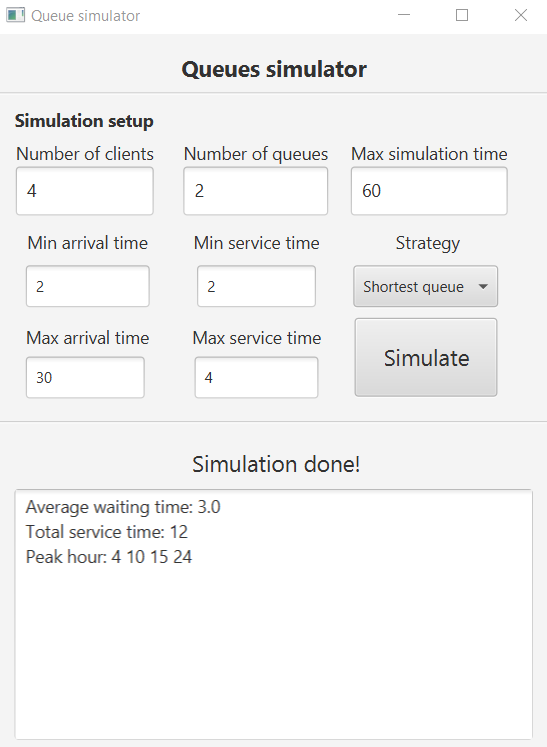
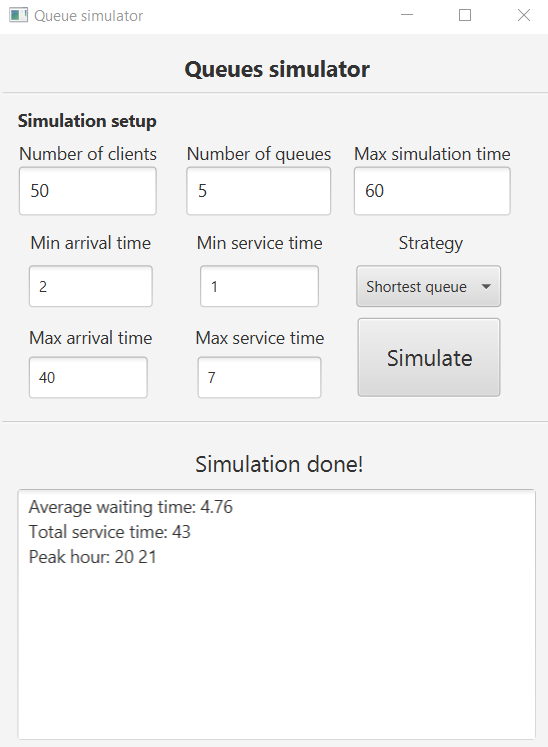
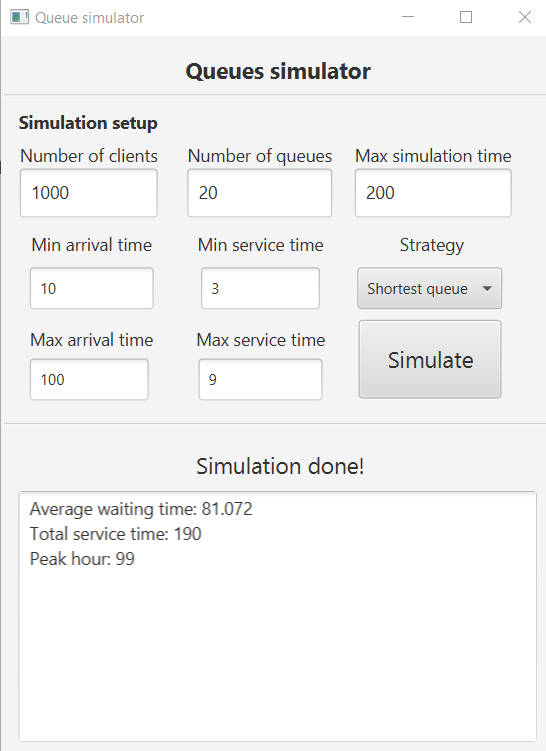
We first insert in the user interface the number of clients that we want to randomly generate, the number of queues, the maximum simulation time, the minimum and maximum arrival time and service time and we select from the drop-down menu the strategy of selecting a queue when we add a new client. After we provide all the needed data, we press the “Simulate” button and the simulation starts. The simulation status label will change to “Simulation running” and in the text area below we will see the real-time simulation status with all the queues, waiting clients and the current time. During the entire simulation, the “Simulate” button will be disabled and after the simulation ends the button will be enabled. After the simulation finishes, the simulation status label will change its text to “Simulation done!” and in the text area will appear the simulation results, the average waiting time, the average service time, and the peak hour.

In order to compile and run the program we need to run the following maven commands:

mvn clean:clean compiler:compile resources:resources javafx:run

These commands will clean the delete the output folder, recompile the program and create a new output folder, add the resource folder with the View.fxml file and the run the program.

1. **Results**

In order to test the application, all three tests provided in the assignment description file were run. For each test, a log file was generated with the name logTest1.txt, logTest2.txt, and logTest3.txt. The results of each test are presented in the pictures below.

1. **Conclusions**

By building this project I have learned how to work with threads and how to build a thread-safe application. I also learned how to create a multi-thread application that can display data on a user interface.

Possible future features for this application are for testing the program for random elements and for testing the user interface.

1. **Bibliography**
   * Fundamental Programming Techniques Course
   * Fundamental Programming Techniques Laboratory
   * https://docs.oracle.com