Technical University of Cluj-Napoca

Programming Techniques

Laboratory – Assignment 2

Queue Simulator

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

Laboratory Assistant:

Ing. Ciprian Stan

Student:

Enoiu Diana-Cristina

1. Assignment objective

Design and implement a simulation application aiming to analyze queuing based systems for determining and minimizing clients' waiting time.
Queues are commonly seen both in real world and in the models. 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 its "clients" are waiting in queues. 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 supplier. When a new server is added the waiting clients 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 clients 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 other clients in the queue and their service needs.
Input data:
- Minimum and maximum interval of arriving time between client,
- Minimum and maximum service time,
- Number of queues; - Simulation interval,
- Other information you may consider necessary,

Minimal output:
- Average of waiting time, service time and empty queue time for 1, 2 and 3 queues for the simulation interval and for a specified interval,
- Log of events and main system data,
- Queue evolution,
- Peak hour for the simulation interval.

2. Analysis

a) Problem analysis

This application should simulate clients waiting in a queue to receive a service (for example waiting at a supermarket). They have to wait in queues, queues are processing each client simultaneously, just like in real world. To process queues in parallel we have to use threads. Threads will allow us to process things concurrently, thus, they can be used in the simulation of our queues. We can also compute the average waiting time (the time spent by each client in the queue) or the peak hour based on our simulation.

b) Modeling

The clients are generated randomly, each one having its ID, an arrival time and their own service time (the time it will take to process their task, once they get on top of the queue).
The user can set in a setup window the following:

- The number clients that have to be generated,
- The maximum number of queues that will be used in the simulation,
- The minimum and maximum arrival time for each client,
- The minimum and maximum service time for each client,
- Also, the time interval in which the simulation will take place.

The user can now go to the simulation scene, where it can actually start the simulation. While the simulation is running it will display the current second of the simulation, the clients that are in the waiting queue (they will be placed on one of the available queues once they arrive), the actual queues with its corresponding clients that wait there. When the simulation stops, it will display the average waiting time, average service time and the peak hour (when most clients were in queue).

Consider the following input data for the application:

•N=4 clients

•Q = 2 queues

•$t_{simulation}{}^{MAX}$=60, a 60 seconds simulation interval

•[2, 30] -the bounds for the client parameters, respectively a minimum and maximum arrival time, meaning that clients will go to the queues from second 2 up to second 30.

•[2, 4] -the bounds for the service time, meaning that a client has a minimum time to wait in front of the queue of 2 seconds and a maximum time of 4 seconds.

Using this input data, a set of 4 clients are generated random, each client ibeing defined by the following tuple: (IDi, $tarrival$,$tservice$).

A number of Q threads will be launched to process in parallel the clients. Another thread will be launched to hold the simulation time $tsimulation$ and distribute each client into the queue with the smallest waiting time when $tarrivali \geq tsimulation$. The log of events is saved in a .txt file which contains the status of the pool of waiting clients and the queues as the simulation time $tsimulation$goes from 0 to $tsimulationMAX$.

The data is displayed in this form:

Time 3

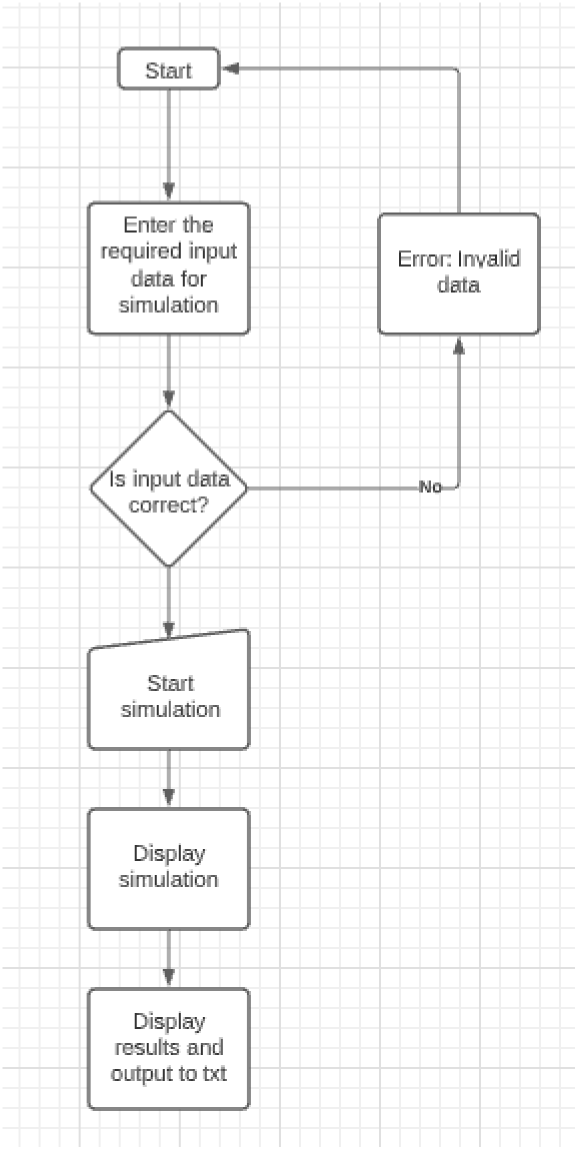
Waiting clients: (3,4,3); (4,10,2)

Queue 1: (1,2,1);

Queue 2: (2,3,3);

c) Different scenarios and use cases

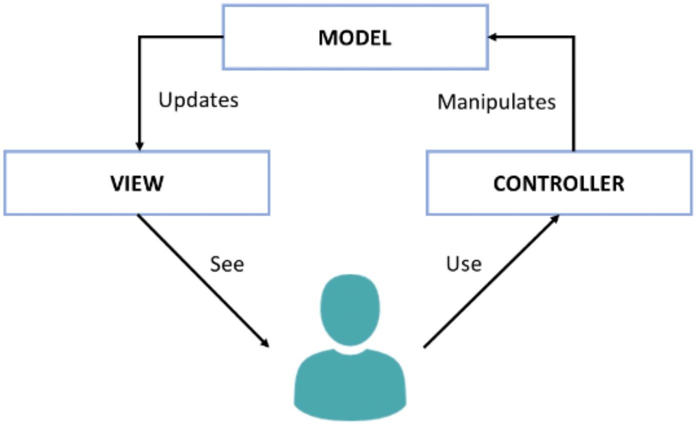
A use case is a methodology used in system analysis to identify, clarify, and organize system requirements. The use case is made up of a set of possible sequences of interactions between systems and users in a particular environment and related to a particular goal.
The use cases are corelated with the steps the user makes when using the application.
The application should have a user friendly interface in order for the user to understand faster what are the steps that are needed to be performed in order to achieve the desired result.
For this application, the flow-chart from below is describing the way the application is expected to behave, based on the user inputs.



3. Design and Implementation

a) *Design decisions*

This application uses an object-oriented programming design. The application has a graphical user interface made in JavaFX, with the help of Scene Builder.
For the design of this application it was used, also, an architectural pattern, the MVC pattern, (Model-View-Controller pattern), which is used to separate application's concerns.
The Model–View–Controller is a software design pattern commonly used for developing user interfaces that divides the related program logic into three interconnected elements.



This application' classes have, thus, been split into four main packages:

- The application package which contains the Main class;
- The model package which has the Task, Server used in the modelling of the program, Strategy interface used for selecteing the strategy and the Scheduler.
- The view package containing .fxml, .css files used for displaying the actual interface;
- The controller package, which manages the view and model packages, contains the SetupController class for the setup.fxml and the SimulationController for the simulation.fxml. It also contains the Simulation Manger, which will actually be a thread that executes the simulation.

For this application we must use multi-threading. Multithreading is a Java feature that allows concurrent execution of two or more parts of a program for maximum utilization of CPU. Each part of such program is called a thread. So, threads are light-weight processes within a process.

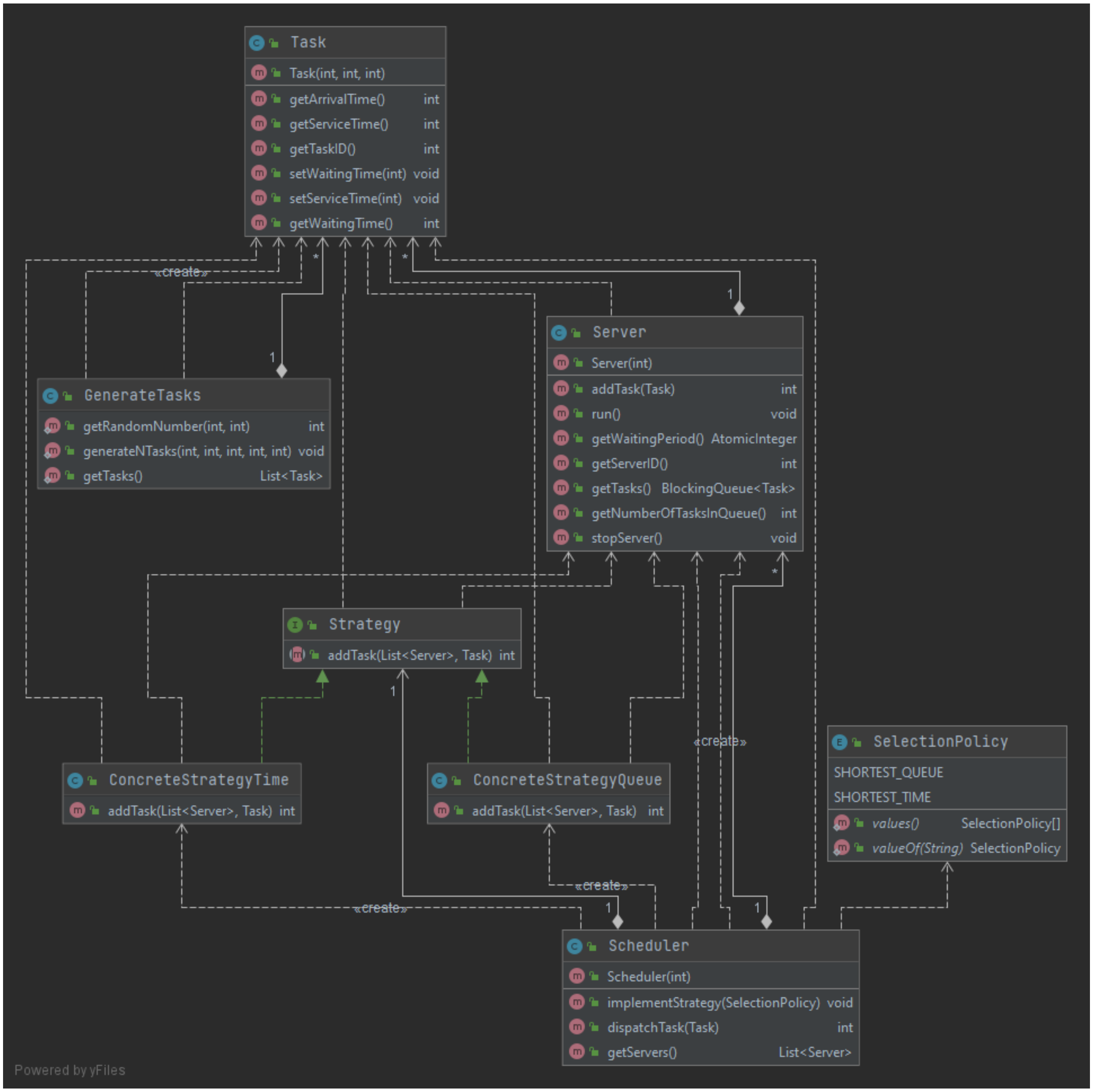
For the design of this application it was used also a Strategy pattern in order to decide where to place the current task in a queue in order to obtain a minimal waiting time. The shortest time policy will add the task where it finds a queue with the minimal waiting time. The shortest queue policy will add the task where it finds a queue with the smallest number of tasks in the queue.

A scheduler is also used in order to place the task to a queue according to the implemented strategy.

b) *Data structures*

For this application the data structures that were used are primitive data types (integers, doubles), non-primitive data types (Strings), more complex data types such as ArrayLists, BlockingQueue and AtomicInteger to assure thread safety and newly created objects such as Task and Server.
We have used threads for the execution of the queues in parallel.

c) *UML class diagram*



d) *Class design*

Because of the MVC architecture, this program consists of 4 parts:

1) **The Model** – contains the logic of the application

Task Class
Used to simulate the client object. This class has an taskID, arrivalTime, serviceTime and waitingTime as fields.
Constructor:

- *public Task(int clientID, int arrivalTime, int serviceTime)*
  //the constructor that initializes the task with its corresponding ID, arrival time, service time and sets the waiting time to zero(initially).

Methods:

- *just some getters and setters.*

Server Class
This class implements a Runnable interface as it will run a thread. Is modeled using BolckingQueue for the list of tasks and AtomicInteger for the waiting period to assure thread safety.
Constructor:

- *public Server(int ID);* //to initialize the BlockingQueue, AtomicInteger and also the server ID.

Methods:

- *public int addTask(Task newTask);*
  //adds a new task to the BlockingQueue tasks, sets the new WaitingTime
- *public void run();*
  //to run the thread, where each task is peeked at the top of the queue to see if it is done, otherwise to decrease the service time
- *public void stopServer();*
  //to stop the thread execution

Scheduler Class
This class sends tasks to servers according to the established strategy.
Constructor:

- *public Scheduler(int noOfServers);*
  //initialize the scheduler, instantiate the new servers and also starts the threads.

Methods:

- *public void implementStrategy(SelectionPolicy policy);*
- *//*selects the corresponding policy to implement the strategy
- *public int dispatchTask(Task t);*
- *//*place the task on the server according to the strategy

Strategy Interface
Interface used to choose the policy to distribute the clients. The classes that implement this interface are ConcreteStrategyTime and ConcreteStrategyQueue. Those are selected with the help of the enum class SelectionPolicy.
Methods:

- *int addTask(List<Server> servers, Task t);*
- //add the task to the queue according to the strategy, will return the waiting time for that task

GenerateTasks Class
This class is responsible for generating the N clients that are received from the user. The arrival time, and service time are selected randomly form the interval values given by the user. In the end the generated tasks are sorted by the arrival time.
Methods:

- *public static void generateNTasks;*
  //for generating the N tasks

2) **The View**– contains the graphical interface, the user interface.

   This package contains the .fxml, .css files used in the displaying and the designing of the graphical user interface.
   For this application, Scene Builder was used in order to be able to develop much faster an user-friendly interface, and with a nice-looking design without having advanced knowledge with .fxml files.
   This package contains the setup.fxml the scene where the user will enter the data, and also the simulation.fxml the scene where the user can see the simulation.

3) **The Control**–manages the model and the view packages

   This is a very important package because it acts on both model and view. It controls the data flow into model object and updates the view whenever data changes.

   SetupController class – controls the setup.fxml
   The controller responsible for getting the input data from the user and loading the simulation scene after the go to simulation button was pressed.
   Methods:
   - *public void initializeSimulation(ActionEvent event);*
   //receives an request from the user when the mouse click on the go to simulation button, it receives the data that was entered by the user and the it loads the simulation scene
   - *private void loadSimulationScene(ActionEvent event);*
   //loads the simulation scene and displays it

   SimulationController class – controls the simulation.fxml
   This controller is responsible for starting the simulation from the simulation manager instance, then it displays the data of the simulation in the GUI and also prints it to a .txt file.

Methods:
- *public void initData(SimulationManager simulationManager);*

//gets the simulationManager instance

- *public void printSimulationUpdates(String time, String waitingQueue, String queues, String waitingQueueForSim, String queuesForSim);*

//prints the simulation data to the GUI and to txt file

- *public void printResults(float averageWaitingTime, float averageServiceTime, int peakHour);*

//prints the results of the simulation those being the average waiting time, the average service time and the peak hour

SimulationManager class – controls the simualtion

This controller is responsible for the actual execution of the simulation as it also implements a Runnable. When the thread runs, in the beginning it also starts the servers. Then for each second it gets the tasks in the waiting queue that arrived (has the same arrival time as the current time), then it calls the scheduler to dispatch the tasks. Once a task is dispatched it gets deleted, if there are no task to dispatch the waiting queue remains unmodified. Then the GUI has to be updated at each second the make it a real-time simulation. Once the time limit is reached, the simulation stops but before that it displays the results of the simulation (average waiting time, the average service time and the peak hour).

Constructor:
- *public SimulationManager(int timeLimit, int noOfTasks, int minArrivalTime, int maxArrivalTime, int minServiceTime, int maxServiceTime, int noOfServers, SelectionPolicy selectionPolicy) ;*

//stores the data from the user

//also generates the random tasks and stores them

Methods:
- *private void startServers() ;*

//method that initialize the scheduler and starts the servers

- *public void run();*

//runs the SimulationManager thread

- *private void computeStrings(String time);*

//constructs the necessary strings in order to be displayed in the GUI and txt

- *private float computeAverageServiceTime ();*

//calculates the average service time

- *private int helpComputePeakHour();*

//helps at the finding of the peak hour

- *private void updateGUI(String time, String waitingQueue, String queues, String waitingQueueForSim, String queuesForSim) {;*

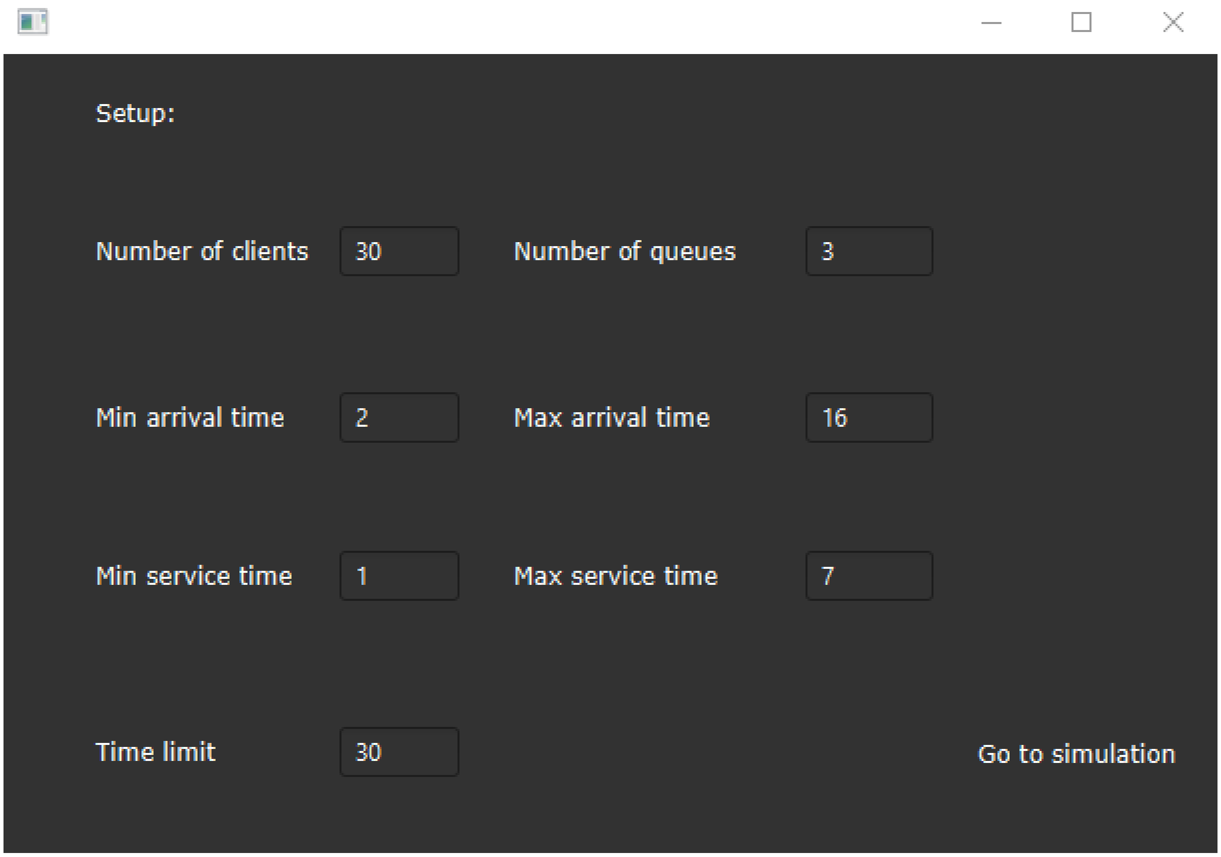
//updets the GUI with the corresponding strings

4) The Application – contains the Main class which runs the main() method in order to “turn on” the application, and also set the necessary stage and scene for the Graphical-User Interface.

e) Packages

Main packages:
App
Model
View
Controller

f) User interface



Setup:

Number of clients 30

Number of queues 3

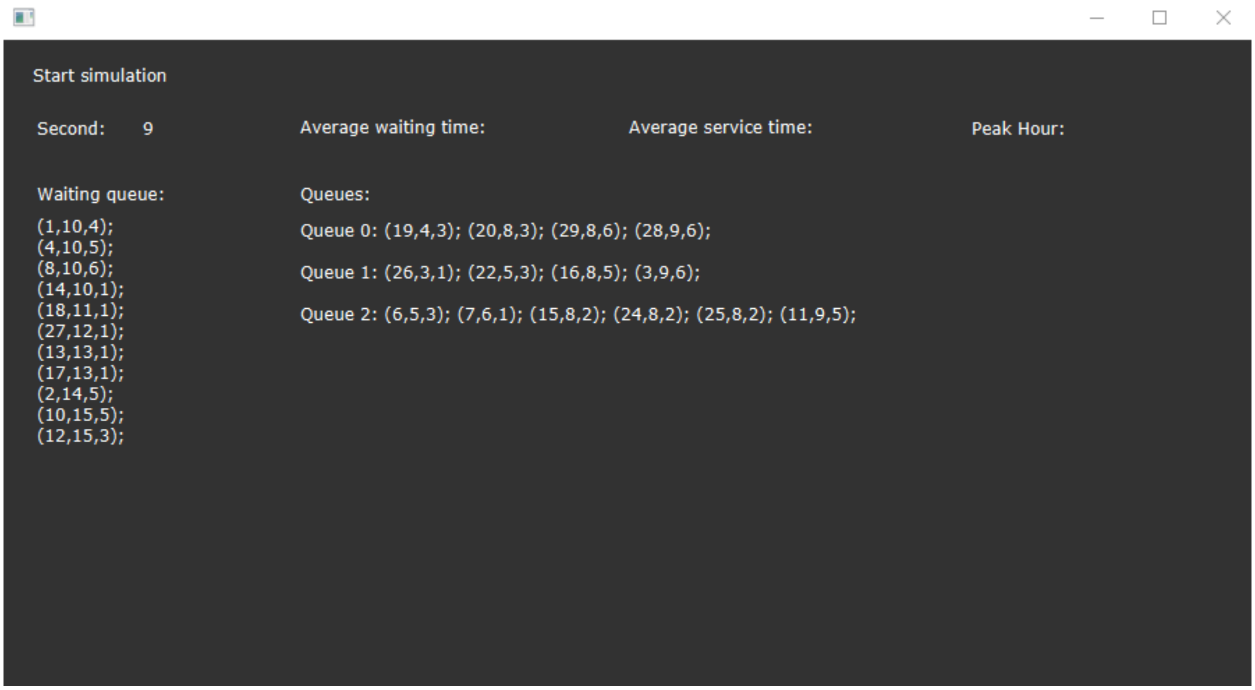
Min arrival time 2

Max arrival time 16

Min service time 1

Max service time 7

Time limit 30



```
Second:    9        Average waiting time:        Average service time:        Peak Hour:

Waiting queue:      Queues:
(1,10,4);           Queue 0: (19,4,3); (20,8,3); (29,8,6); (28,9,6);
(4,10,5);
(8,10,6);           Queue 1: (26,3,1); (22,5,3); (16,8,5); (3,9,6);
(14,10,1);
(18,11,1);          Queue 2: (6,5,3); (7,6,1); (15,8,2); (24,8,2); (25,8,2); (11,9,5);
(27,12,1);
(13,13,1);
(17,13,1);
(2,14,5);
(10,15,5);
(12,15,3);
```

4. Conclusions

This assignment helped me understand the importance of the multi-threading in the developing of a software application. Multithreading allows concurrent execution of two or more parts of a program, this being useful in the simulation of our queues.
This assignment also was an introduction in building a project with Gradle, and it helped me familiarize with the building of a user interface with JavaFX.
The using of the MVC architectural was also important for me to understand because it is the basic structure which most web applications, mobile apps and desktop programs are built on.

5. Bibliography

http://stackoverflow.com
https://openjfx.io/openjfx-docs/#gradle
https://ro.wikipedia.org/wiki/Model-view-controller
https://app.diagrams.net/
https://en.wikipedia.org/wiki/Strategy_pattern
Programming Techniques – Lectures of prof. Ioan SALOMIE