RUSU CARLA MARIA
GROUP 30421

Queue Simulator

RUSU CARLA MARIA
GROUP 30421

Objective

"Design and implement a simulation application aiming to analyse queuing based systems for determining and minimizing clients' waiting time."

OBJECTIVE	ITEM	DETAILS
MAIN OBJECTIVE	Queue simulator	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.
SECONDARY OBJECTIVES	Server storage	An object of type Queue stores a BlockingQueue of Clients; The BlockingQueue assures the synchronization of the data throughout the thread's lifetime; It provides easy update (add or remove clients), keeps track of important attributes in order to calculate statistics and is responsible for creating a unique thread for itself, as well as storing the information to be displayed.
	Task storage	An object of type Client stores the necessary input information for a specific client (task) and provides sorting capabilities (method compareTo).
	Simulation manager	The simulation manager is responsible for running the simulation within the given parameters (input data) and for providing the required output; it serves in a way as the main thread, providing the means to follow the evolution of the queues in real time;
	Random client generator	The method generateNRandomClients() within the simulation manager creates a number of clients whose attributes are within input intervals; it also sorts the resulting list thus facilitating the entering of the queues in an ordered, intuitive manner (first come, first serve, or FIFO);
	Task scheduler	The scheduler is the queue and thread creator, as well as the manager of the queues' states; it also deals with the assignation of a client to the correct queue, assuring optimal dispersion with regard to the smallest waiting time among the queues;
	User interface	The user interface is composed of the SImulatorFrame and the Controller; it provides an intuitive input and output system in order to offer the required information and much flexibility to the simulator.
	Input parsing and validation	The input is parsed and validated in the Controller, at the moment the simulation start is attempted; some helpful dialog messages are given to orient the user in case of an invalid input;

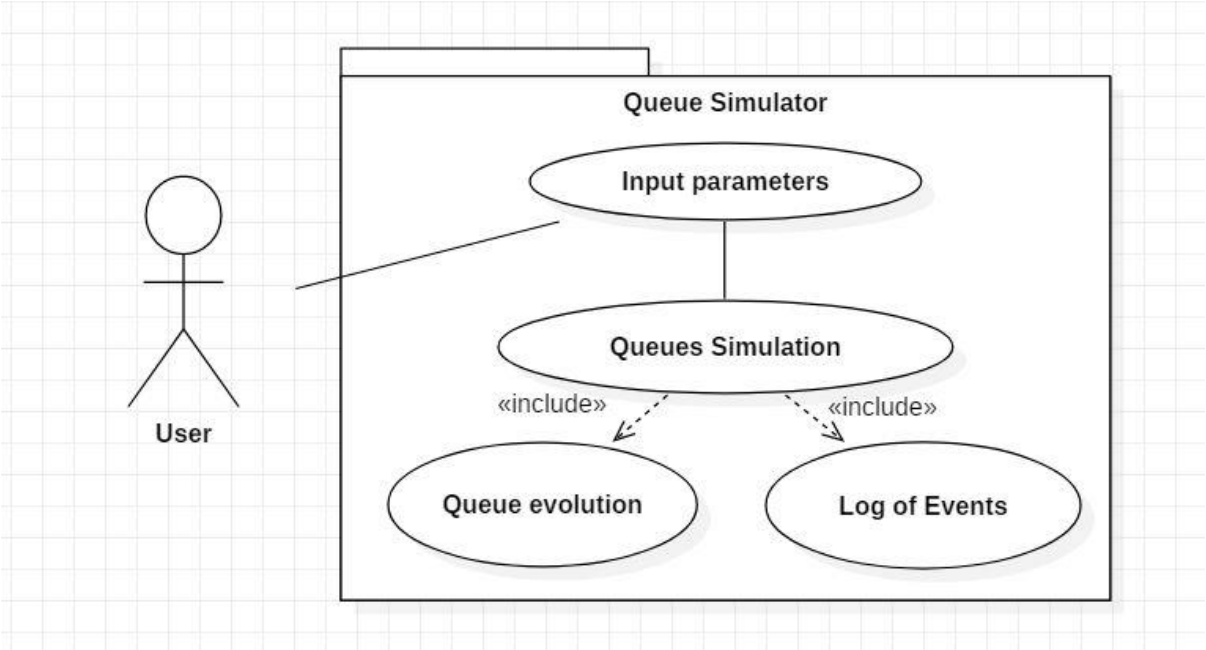
RUSU CARLA MARIA
GROUP 30421

Problem analysis

- Input the necessary data:
    a) Minimum and maximum interval of arriving time between customers;
    b) Minimum and maximum client processing time;
    c) Number of queues;
    d) Number of clients;
    e) Simulation interval (time in seconds up to which simulation is run);
- Pressing the start button will begin the execution of the application
- The processing of the inputs is responsible for validating, parsing and converting the Strings into integers that are stored in the corresponding attributes. This is done by the application, with no action required from the user.

  With the use of regular expressions and String comparison, the Strings are validated: no empty or non-numeric inputs are allowed.

- The results will be shown in the text field areas labelled "Log of events:" and "Queue evolution:" as well as the label "Peak time is ?" and several dialog boxes.



RUSU CARLA MARIA
GROUP 30421

- Main success scenario:
    1. User introduces the input parameters and selects clicks the start button
    2. The simulator validates, parses and converts the input
    3. The results are computed in real time using multiple threads;
    4. The result is displayed on the text areas: a real time queue evolution and a comprehensive log of events are given;
- Alternative (error) sequences:

    a) Blank input

    1. User leaves some inputs blank (empty) and selects start
    2. The app tries to validate the input and fails
    3. An error message dialog box is issued stating that the fields cannot remain empty;
    4. The scenario returns to first step

    b) Non-numeric input

    1. User introduces non-numeric data and selects start

    2. The app tries to validate the input and fails

    3. An error message dialog box is issued stating that the input only accepts numeric forms;

    4.The scenario returns to step 1

    c) Incorrect intervals values

    1. User introduces data and selects start

    2. The app tries to validate the input and fails

    3. An error message dialog box is issued stating that the input only accepts mathematically correct intervals;

    4.The scenario returns to step 1

RUSU CARLA MARIA
GROUP 30421

Design

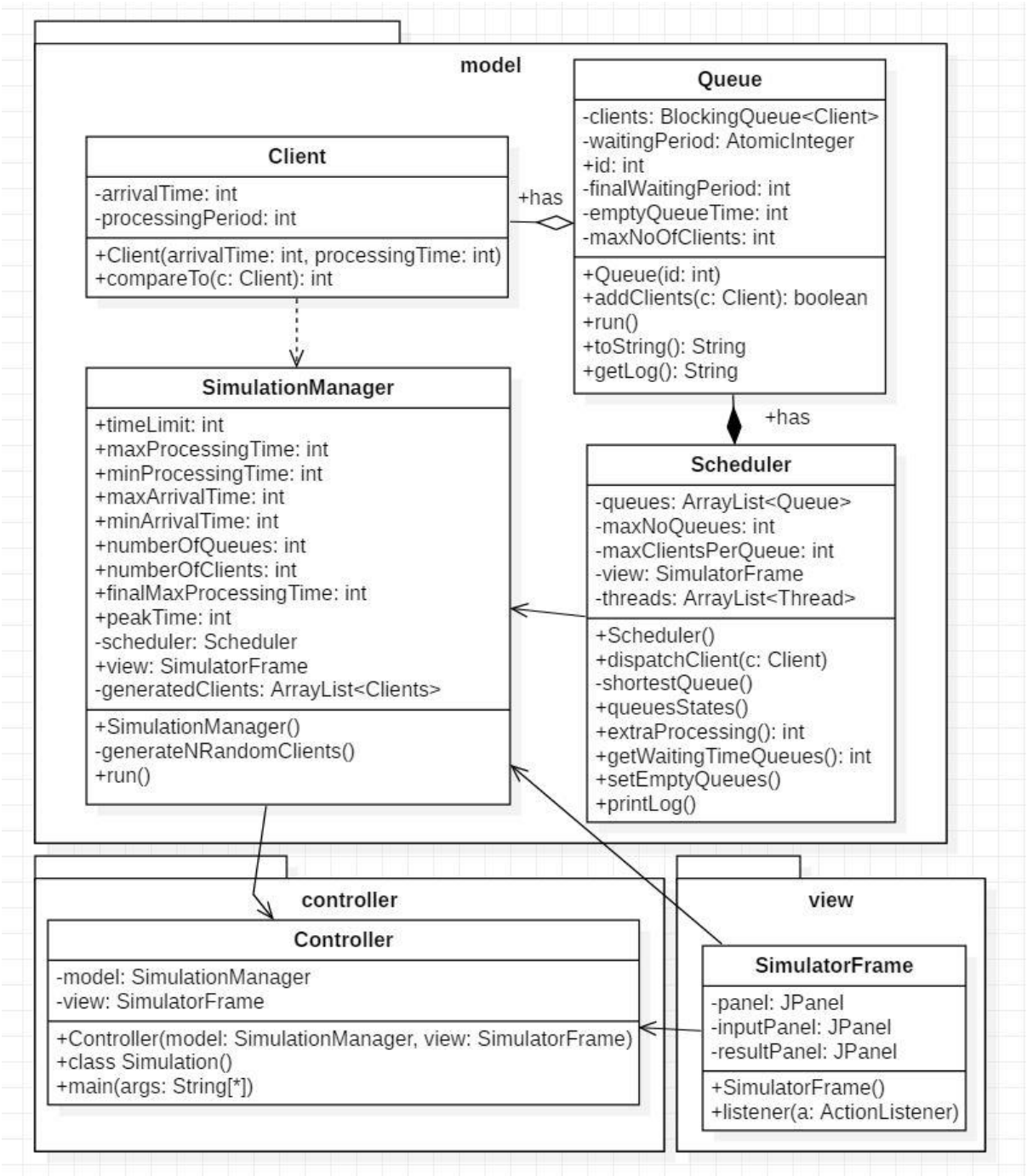
The application is made up of three MVC packages: model (contains the client, queue, scheduler, simulation manager models), view (contains the silmulator frame with the graphical elements) and controller (with the class controller, which maps the action to the button and transfers the input into the system).

- ***Client class***
  Implements a task (the client); has arrivalTime, processing time and an id.
- ***Queue class***
  Contains zero, one or more Clients, thus has a BlockingQueue of Clients; Stores the queue's current waiting period as an AtomicInteger to better deal with threads; contains other useful attributes and an id; has the means to add clients to the queue; overrides run(), toString() and offers the means to get a log of itself;
- ***SimulationManager class***
  Has all the given input parameters as attributes; makes use of a Scheduler to dispatch the clients to their respective queues; links the system with the output through the view attribute; generates a list of random clients within the necessary parameters; overrides run() thus managing all the other threads and making possible the real-time parallel queue evolutions
- ***Scheduler class***
  Contains a list of queues and threads as well as a view to communicate with the GUI; dispatches the clients to queues based on shortest waiting period; offers a comprehensive log of the queues' states;
- ***SimulatorFrame***
  Contains the elements of graphical user interface; has containers (panels) which group the inputs and the results respectively; has components: pairs of labels and text fields for the inputs and for the two text areas(log and queue evolution);
- ***Controller***
  Contains the necessary method to deal with the clicking of the start button; contains methods to validate, parse and convert the input. Contains the main method where the necessary objects are instantiated;

The UML diagram displayed below offers a complete understanding of the relationship between the classes. Inner classes and interface implementations are represented as well.

Algorithmically, the implementation of the threads is intuitive, as simple as possible and straightforward.

RUSU CARLA MARIA
GROUP 30421



RUSU CARLA MARIA
GROUP 30421

Implementation

A detailed explanation of the implemented classes and algorithms is given below.

- ***Client class***
  The class Client provides the arrivalTime, processingTime and id necessary for the model. Getters and setters are defined for these attributes as well as a constructor.
- ***Queue class***
  The class implements the runnable interface for the threads: each object of this type will represent a thread. It manages thread access to resources using the BlockingQueue of Clients and the AtomicInteger representing the waiting period; other necessary attributes are also defined: id, finalWaitingPeriod, finalProcessingPeriod, emptyQueueTime and maxNoOfClients
    a) ***Constructor Queue():*** initialises the blocking queue and the other attributes.
    b) ***addClient():*** takes a Client as an argument and returns a Boolean representing the status of the addition of the client to the queue. If the queue is not yet full, the addition is made and the corresponding attributes are computed
    c) ***run():*** overrides run(); takes out clients every time a processing period is finished;
    d) ***toString():*** overrides toString(); gives the current states of the object
    e) ***getLog():*** stores the average waiting time, service time and empty queue time;
- ***Scheduler class***
  Has a list of queues and threads; is responsible for their management: client dispatching and information about the queues at any given moment; communicates with the view to provide output;
    a) Scheduler(): contructor initialises queues and threads and starts threads
    b) dispatchClient(): sends argument client to the shortest queue based on waiting time; gets current state of queue
    c) shortestQueue(): saves the queue with the shortest waiting period
    d) queuesStates(): updates the result text areas
    e) extraProcessing(): sets the timeLimit to a new value
    f) getWaitingTimeQueues(): makes a total out of the waiting times of each queue
    g) setEmptyQueues(): updates the time needed to empty queues;
    h) printLog(): sets the dialog boxes with the statistics
- ***SimulationManager class***
  The class implements the runnable interface for the threads: an object of this type represents a sort of main thread that manages the other ones. It stores all the input parameters and creates some other ones for the output; makes use of a scheduler, a view and a list of randomly generated clients;
    a) ***Constructor SimulartionManager():*** initialises the attributes, generates the clients.

b) ***generateNRandomClients():*** generates a list of clients sorted based on their arrivalTime; updates log of events
c) ***run():*** overrides run(); takes out clients out of the generated list and dispatches them to a queue every second that coincides with the arrival time of a certain client; computes peak time and the maximum processing time; updates log and queue evolution; sets the empty time of queues;

- ***Controller class***
  Has a SImulationManager object and a view to facilitate the communication between the two
  a) ***Controller():*** the constructor initialises the attributes
  b) ***Inner class Simulation():*** deals with the action to be performed when the simulation is started; parses the input and provides corresponding error messages; provides the output by calling start() to the main thread
  c) ***Main():*** instantiates the basic components to run the application
- ***SimulatorFrame class***

  Extends JFrame, multiple labels and text fields to fetch the input and two text areas to write the output

  - ***SimulatorFrame():*** the constructor initialises all the components and containers and assures a user-friendly interface

RUSU CARLA MARIA
GROUP 30421

Results

The results for a mock-test with the following inputs are provided:

- Maximum arrival time: 5
- Minimum arrival time: 0
- Maximum processing time: 3
- Minimum processing time: 2
- Number of clients: 10
- Number of queues: 3
- Simulation interval: 10

Log of events:

Clients with arrival time and processing time generated: 0-3 | 2-2 | 2-3 | 2-2 | 3-2 | 3-3 | 4-3 | 4-3 | 4-3 | 5-2 |

Queue 0 contains no clients.

Queue 1 contains no clients.

Queue 2 contains no clients.

Queue 0 adds client 1

1

Queue 0 contains clients: 1

Queue 1 contains no clients.

Queue 2 contains no clients.

2

Queue 0 contains clients: 1

Queue 1 contains no clients.

Queue 2 contains no clients.

Queue 1 adds client 2

Queue 2 adds client 3

Queue 1 adds client 4

3

Queue 0 contains no clients.

Queue 1 contains clients: 2 4

Queue 2 contains clients: 3

Queue 0 adds client 5

Queue 0 adds client 6

4

Queue 0 contains clients: 5 6

RUSU CARLA MARIA
GROUP 30421

Queue 1 contains clients: 4

Queue 2 contains clients: 3

Queue 1 adds client 7

Queue 2 adds client 8

Queue 0 adds client 9

5

Queue 0 contains clients: 6 9

Queue 1 contains clients: 4 7

Queue 2 contains clients: 8

Queue 2 adds client 10

6

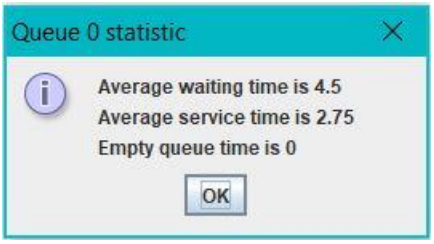
Queue 0 contains clients: 6 9

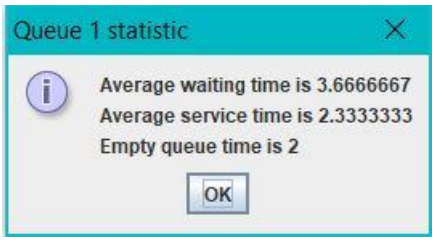
Queue 1 contains clients: 7

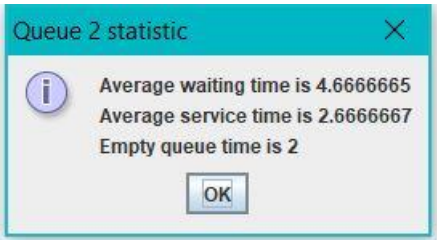
Queue 2 contains clients: 8 10

7

Peak time: 5







Conclusions

To conclude, the application meets the requirements: has an OOP design, is organized in packets, has classes of at most 200 lines, most methods have under 30 lines and respects Java naming conventions. An intuitive graphical user interface that displays real-time queue evolution is provided, as well as a log of events, a random client generator and more than one threads.

The realization of this project has helped me in learning to work with threads and further developed my OOP knowledge regarding the possible structure of application and its capabilities.

Further developments are a possibility: more varied input parameters, more operations such as pausing the simulation and unpausing it at any time, developing the GUI, etc. To finalise, the project was a great gateway towards understanding the workings of threads and their possible utility.

Bibliography

- http://users.utcluj.ro/~igiosan/teaching_poo.html
- http://www.java2s.com/Code/Java/Swing-JFC/AnexampleofJListwithaDefaultListModel.htm
- http://users.utcluj.ro/~cviorica/PT2019/
- http://coned.utcluj.ro/~marcel99/PT2019/Tema%202/