Sabou Cezara
Computer Science, year 2
group 30421

Queue Simulation System

1.Objectives

1.1Main objective

The main objective of this project is to create a Java Application that could provide the user the ability to generate a queue simulation, in which clients are created and each assigned to a certain queue. The user should only insert some time intervals : minimum and maximum arrival time between clients, minimum and maximum time required for each client to finish being served in the queue(letting the server scan each item and pay for it), the maximum number of queues used and the duration of the simulation. By pressing the "Start" button, the application should be able to give the correct result in the Graphical User Interface

1.2 Secondary objectives

The secondary objectives derive from the main objective:

1. The ability to use multiple threads and create thread safe interface methods;
2. Being able to convert the given input into corresponding adequate candidates for processing;
3. Creating ArrayBlockingQueues that contain pseudo-randomly generated Clients;
4. Implementing a GUI(Graphical User Interface)

2. Problem Analysis

Queue simulation represents a day to day activity that everyone has at least once experienced. Therefore, the best option would be to create a GUI that not only is easy to use, but also offers correctness and precision.

I will describe the functionality of the system in the following paragraphs.

Before working with Queues we have to at first obtain them, by letting the user introduce the intervals for creating the random service time for each client, and also the random arrival time between clients.

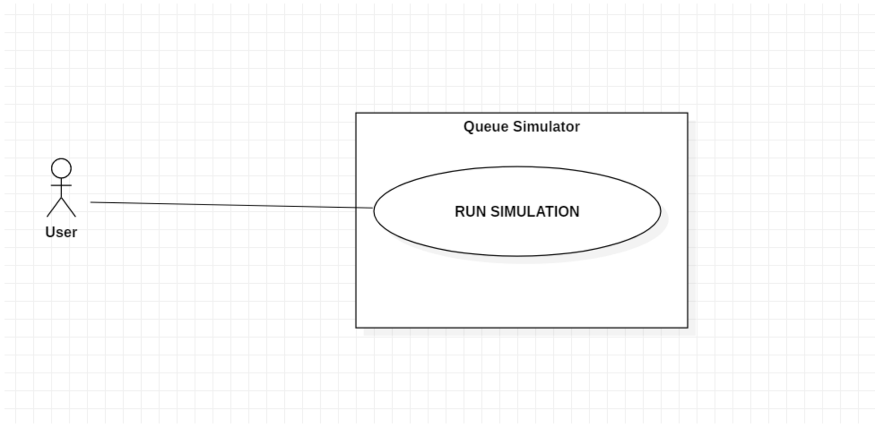
The queues actually are arrayBlockingQueues which can work in classes that extend "Thread". This special kind of queues is used mainly because it is thread safe and because accessing the items(clients) inside them is very easy and fast.

The clients that are the objects from the queues, are randomly created in the ClientFactory class: they are given a unique id, and random service times and arrival times. After they are created, they are added to the BlockingQueue that links the ClientFactory and the Task Manager.

Sabou Cezara
Computer Science, year 2
group 30421

The Task Manager's job is to simply get the clients from the Client Factory and assign them each to a queue that is active and has the smallest waiting time possible. Given the fact that Task manager also uses a Blocking Queue object, it will also extend the Thread Class.

The Queue Simulation acts as the controller : it gets the input from the GUI and sends it to the Task Manager which starts the queue simulation through the display service. The Display Service Class is another thread that as long as there are elements in the queues and the simulation is not over, it will continue to display them. This class also is part of the controller as it is used by the Queue Simulation class in the overridden run method.



The use case diagram presents :

Use case title: Queue simulation System

- Summary: this case allows the users to simulate queue evolution
- Preconditions: intervals are correctly inserted as integer values that correspond to the necessities of the application
- Main success scenario:
  - User gives the correct time intervals
  - User presses start button
  - Threads start successfully without any exceptions
  - The queues start showing on the assigned textFields
  - The average waiting time is displayed in the assigned text field and correctly calculated
  - The queues empty themselves after the simulation is over and the ClientFactory thread has stopped
    - Alternative sequences:
  - The input is not valid
  - The number of queues is higher than 10

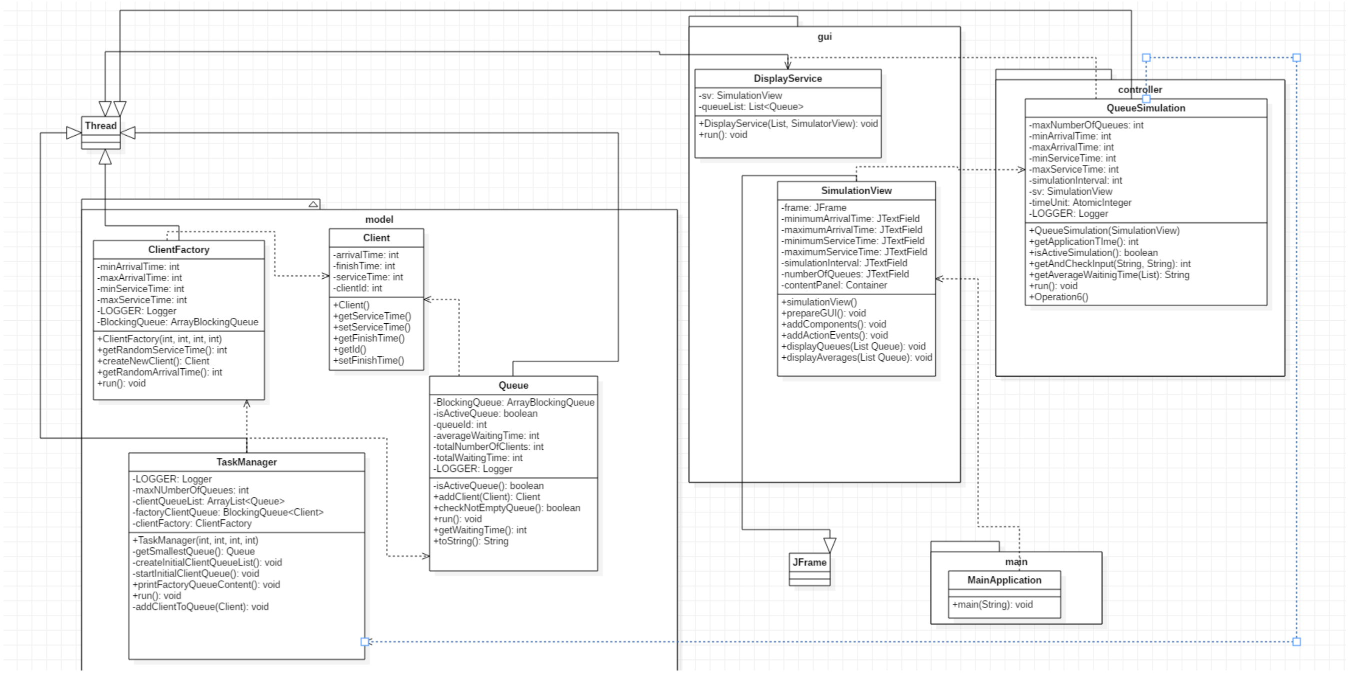
Sabou Cezara
Computer Science, year 2
group 30421

3.Design

The structure of the application is based on 5 different packages, some of them being structured in a MVC pattern:

1. The Model is the one that contains the following classes: Client, Client Factory, Queue, Task Manager .
2. The View is responsible for directly interacting with the user, but also helping the controller pick the correct commands adapted to the user's needs.
3. The Controller links the model and the View, managing the interactions.
4. The exception class was simply created to group the exceptions and separate them from the rest of the project.
5. The Main class is meant to only initiate the application, and it is not as important.

I created this UML diagram in order to show the main relationships of the application and to accentuate the MVC pattern structure.


Thread
model
ClientFactory
-minArrivalTime: int
-maxArrivalTime: int
-minServiceTime: int
-maxServiceTime: int
-LOGGER: Logger
-BlockingQueue: ArrayBlockingQueue
+ClientFactory(int, int, int, int)
+getRandomServiceTime(): int
+createNewClient(): Client
+getRandomArrivalTime(): int
+run(): void
Client
-arrivalTime: int
-finishTime: int
-serviceTime: int
-clientId: int
+Client()
+getServiceTime()
+setServiceTime()
+getFinishTime()
+getId()
+setFinishTime()
Queue
-BlockingQueue: ArrayBlockingQueue
-isActiveQueue: boolean
-queueId: int
-averageWaitingTime: int
-totalNumberOfClients: int
-totalWaitingTime: int
-LOGGER: Logger
-isActiveQueue(): boolean
+addClient(Client): Client
+checkNotEmptyQueue(): boolean
+run(): void
+getWaitingTime(): int
+toString(): String
TaskManager
-LOGGER: Logger
-maxNUmberOfQueues: int
-clientQueueList: ArrayList<Queue>
-factoryClientQueue: BlockingQueue<Client>
-clientFactory: ClientFactory
+TaskManager(int, int, int, int)
-getSmallestQueue(): Queue
-createInitialClientQueueList(): void
-startInitialClientQueue(): void
+printFactoryQueueContent(): void
+run(): void
-addClientToQueue(Client): void
gui
DisplayService
-sv: SimulationView
-queueList: List<Queue>
+DisplayService(List, SimulatorView): void
+run(): void
SimulationView
-frame: JFrame
-minimumArrivalTime: JTextField
-maximumArrivalTime: JTextField
-minimumServiceTime: JTextField
-maximumServiceTime: JTextField
-simulationInterval: JTextField
-numberOfQueues: JTextField
-contentPanel: Container
+simulationView()
+prepareGUI(): void
+addComponents(): void
+addActionEvents(): void
+displayQueues(List Queue): void
+displayAverages(List Queue): void
controller
QueueSimulation
-maxNumberOfQueues: int
-minArrivalTime: int
-maxArrivalTime: int
-minServiceTime: int
-maxServiceTime: int
-simulationInterval: int
-sv: SimulationView
-timeUnit: AtomicInteger
-LOGGER: Logger
+QueueSimulation(SimulationView)
+getApplicationTIme(): int
+isActiveSimulation(): boolean
+getAndCheckInput(String, String): int
+getAverageWaitinigTime(List): String
+run(): void
+Operation6()
JFrame
main
MainApplication
+main(String): void

4.Implementation

In the Model package:

4.1 Client

The "Client" class is used to create all the clients that will further populate the Queues. It is a simple class that holds the following attributes that represent the times: arrivalTime – the real application time given as parameter, at the time the client has been created; finishTime – the real application time when the client has finished its service time and it was taken out of the queue; serviceTime – time randomly generated given in the QUI as input; clientId – unique id set when client was created.

The class also holds the corresponding getters and setters.

4.2 Client Factory

The "ClientFactory" class has as its attributes the corresponding times given as input, it also has a blocking queue that will hold all the newly generated clients, it also extends the Thread class.

The constructor sets the times necessary for creating the random service time in the getRandomServiceTime method and for the arrival time for the getRandomArrivalTime method.

The createNewClient method takes the randomly obtained times and creates a new Client with a unique id and adds it to the blocking queue.

The run method is used to continuously add randomly generated clients to the blocking queue, as long as the simulation is still active.

4.3 Queue

This class is the main object of our application and its purpose is simple and straightforward: it takes clients and it waits as long as their service time needs, calculating the average waiting time as well.

It contains as an important attribute the blocking queue, which is used because it is easy to access and also used specifically for threads, given the fact that the class actually extends the thread Class.

The class also has a method that computes the average waiting time as the sum of all the service times given by the clients and client number.

The run method runs as long as the queue still has something to process(waiting clients) and it simply calculates the waiting time while it prints some logger messages that say whenever a new client has been added. After adding the client, it will wait/ sleep as long as its service time lasts, that is, as long as the client needs to process its products.

4.4 Task Manager

The Task Manager is meant to take the blocking queue from the client factory and assign each client the queue with the smallest waiting time. Given the fact that the class uses blocking queues, it means that it will implicitly need to extend the Thread class. This class has some methods that create all the queues given by the maximum number of queues from the input, initializes them and starts them using start().

The run method continuously extracts a client and assigns it to a queue with smallest waiting time by using the addClientToQueue method.

In the GUI package:

4.5 Display Service

The Display Service class extends the Thread class and it takes the queueList created in the Task Manager. It will display the evolution of the queues by using the displayQueues method used in the simulation view, and also displays the average waiting time using the displayAverages method also declared in the simulation view.

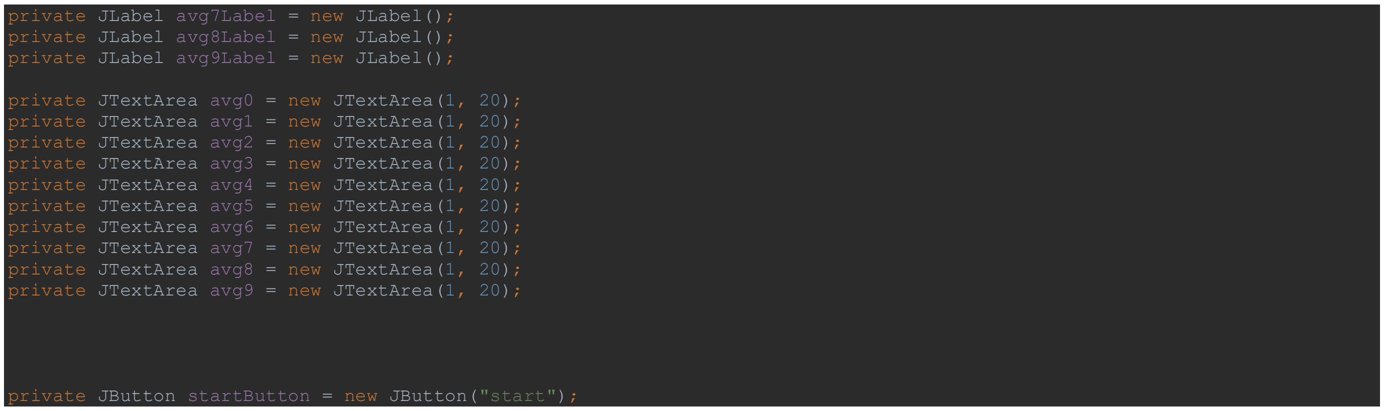
4.6 Simulation View

This class is the way the user interacts with the application. This is where all the labels and textfiwelds are implemented :

```
                      private JFrame frame = new JFrame("QueueSimulation");
private JPanel northPanel = new JPanel();
private JPanel centerPanel = new JPanel();
private JPanel southPanel = new JPanel();
private JLabel minimumArrivalTimeLabel = new JLabel();
private JLabel maximumArrivalTimeLabel = new JLabel();
private JLabel minimumServiceTimeLabel = new JLabel();
private JLabel maximumServiceTimeLabel = new JLabel();
private JLabel simulationIntervalLabel = new JLabel();
private JLabel numberOfQueuesLabel = new JLabel();
private JTextField minimumArrivalTime = new JFormattedTextField();
private JTextField maximumArrivalTime = new JFormattedTextField();
private JTextField minimumServiceTime = new JFormattedTextField();
private JTextField maximumServiceTime = new JFormattedTextField();
private JTextField simulationInterval = new JFormattedTextField();
private JTextField numberOfQueues = new JFormattedTextField();
private Container contentPanel;

private JLabel ta0Label = new JLabel();
private JLabel ta1Label = new JLabel();
private JLabel ta2Label = new JLabel();
private JLabel ta3Label = new JLabel();
private JLabel ta4Label = new JLabel();
private JLabel ta5Label = new JLabel();
private JLabel ta6Label = new JLabel();
private JLabel ta7Label = new JLabel();
private JLabel ta8Label = new JLabel();
private JLabel ta9Label = new JLabel();
private JTextArea ta0 = new JTextArea(1, 30);
private JTextArea ta1 = new JTextArea(1, 30);
private JTextArea ta2 = new JTextArea(1, 30);
private JTextArea ta3 = new JTextArea(1, 30);
private JTextArea ta4 = new JTextArea(1, 30);
private JTextArea ta5 = new JTextArea(1, 30);
private JTextArea ta6 = new JTextArea(1, 30);
private JTextArea ta7 = new JTextArea(1, 30);
private JTextArea ta8 = new JTextArea(1, 30);
private JTextArea ta9 = new JTextArea(1, 30);

private JLabel avg0Label = new JLabel();
private JLabel avg1Label = new JLabel();
private JLabel avg2Label = new JLabel();
private JLabel avg3Label = new JLabel();
private JLabel avg4Label = new JLabel();
private JLabel avg5Label = new JLabel();
private JLabel avg6Label = new JLabel();
```



```
private JLabel avg7Label = new JLabel();
private JLabel avg8Label = new JLabel();
private JLabel avg9Label = new JLabel();

private JTextArea avg0 = new JTextArea(1, 20);
private JTextArea avg1 = new JTextArea(1, 20);
private JTextArea avg2 = new JTextArea(1, 20);
private JTextArea avg3 = new JTextArea(1, 20);
private JTextArea avg4 = new JTextArea(1, 20);
private JTextArea avg5 = new JTextArea(1, 20);
private JTextArea avg6 = new JTextArea(1, 20);
private JTextArea avg7 = new JTextArea(1, 20);
private JTextArea avg8 = new JTextArea(1, 20);
private JTextArea avg9 = new JTextArea(1, 20);




private JButton startButton = new JButton("start");
```

There were 10 text areas created that are not editable and they display the live queue evolution, as well as the average times created.

Whenever the start button is pressed, the queue simulation starts, and this way the whole application is started.

In the exception package:

4.7 InvalidInputexception

This method generates an exception if the input is not valid. And is able to display the error message costumized.

In the controller package

4.8 Queue Simulation

The queue simulation class is by far the most important one. It links the model with the gui.

It holds as attributes the corresponding times, that are taken from the simulation view :

- int minArrivalTime;
- int maxArrivalTime;
- int minServiceTime;
- int maxServiceTime;
- int simulationInterval;
- int maxNumberOfQueues.

It also creates the timeUnit : which is used to get the real application time;

It has a Logger that displays the start of the simulation.

The getAndCheckInput method takes the Strings from the textFields, tests the correctness of the input values and returns the result as an integer.

The run method initializes the display service and the task manager, that is, the queue simulation is the one which starts the whole application. It also increments the real application time given by the time unit.

5.Results

The results show the average waiting times for each queue and they show a successful simulation of the queue evolution during the simulation interval. The application is successful and it does not crash and the outputs are unique and generated in the corresponding random intervals.

6.Conclusions

In conclusion, the application uses OOP concepts and it serves as a Queue Simulation system. A Graphical User Interface is created and easy to use, in order to offer the users an easy access to queue processing and to give back fast and correct results.

By developing this application I gained experience in trying to divide the classes into the Model View Controller pattern, understanding its functionalities and obtaining a clearer perspective of the project. I became more familiar to the BlockingQueue objects and also linking the methods designed by me to the front end. I learned how to operate on threads and generate random time intervals.

I am aware that the application is not yet in its final form, and there are some adjustments that need to be done. The input conversion could be improved, as well as the interface. I also believe that there are more efficient ways in which I could implement the methods that compute the peak time. I also would add other methods and options for the user such as: increasing or decreasing the time interval in which the clients are created and dynamically creating the queues .

7.Reference

https://www.tutorialspoint.com/java/index.htm https://www.jetbrains.com/help/idea/maven-support.html

https://examples.javacodegeeks.com/desktop-java/swing/java-swing-layout-example/

https://docs.oracle.com/javase/tutorial/uiswing/events/intro.html