 **Universitatea Tehnică Cluj Napoca**

**Facultatea de Automatică și Calculatoare**

**Programming Techniques**

**Homework #02**

**Queues Mangement**

**Ardelean Octavian Ioan**

**Group 30421**

**1.Project Objective and Description**

**Requirement**: Design and implement a simulation application aiming to analyze queuing based systems for determining and minimizing clients’ waiting time.

The purpose of this project is to implement a standalone application that simulates the running of up to 3 queues using multiple threads.

**Description:**

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 "customer" to wait before receiving a "service". The management of queue based systems is interested in minimizing the time amount its "customers" 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 customers will be evenly distributed to all current available queues.

The system should simulate a series of customers 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. 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 customers – when they show up and how much service they need. The finish time depends on the number of queues, the number of other customers in the queue and the service needs of those other customers.

**Use Case Model:**

Step1: Identifying the actors

Since the problem has very few specifications, we will implement a simple application with no need for maintenance whatsoever, therefore the only actor in this case is the user conducting the simulation

Step 2 : Identifying the use cases

A use case represents the specification of a sequence of actions, including variants, which the system can execute, interacting with the actors.

Since in our case we only have one actor, we will next list the ways in which he/she can interact with the system.

User:

-input minimum and maximum service time

- input minimum and maximum client arrival time

-input the simulation time

-control which queues are opened or not

Since the use case produces an observable result for the user, we can easily say that the user is a **primary actor**.

Step 2 : Identifying the use cases

A use case represents the specification of a sequence of actions, including variants, which the system can execute, interacting with the actors.

Since in our case we only have one actor, we will next list the ways in which he/she can interact with the system.

User:

-input the minimum arrival time and maximum arrival time between customers(the result will be a randomly generated value between these limits)

-input the minimum serve time and maximum serve time for the customers(also a random value)

-select the simulation interval(in hours)

-before or after starting the simulation, control which of the three queues are open/closed.

Since the use case produces an observable result for the user, we can easily say that the user is a **primary actor**.

Moreover, we can only identify one use case for this kind of approach problem, which is: The user performs an operation on the inserted polynomials, printing results and properties of that last executed operation.

Step 3: Textual description of the use cases

The use cases must have a beginning and end clearly identifiable. Possible

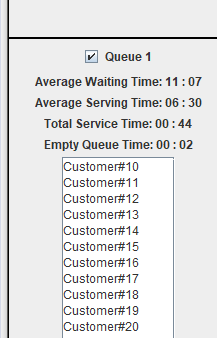
variants must be also specified, like a success scenario, alternative sequences, meanwhile it's tried to arrange the descriptions in a sequential manner, for better understanding purposes.

A scenario represents a particular succession of sequences which is executed from the very beginning to the end of a se case. It can be used to show an interaction or an execution of an instance of a use case.

**Scenarios**

A scenario represents a particular succession of sequences which is executed from the very beginning to the end of a se case. It can be used to show an interaction or an execution of an instance of a use case.

Scenario : We perform the simulation using 1 queue



We observe how the queue processes the customers one by one and updates the stats about it in real time.

b) Flow of events

Pre-conditions: The input provided by the user must follow the specified format. This means that the min and max time intervals must be positive numbers. Moreover, the difference between the upper limit and the lower limit must be a positive number, corresponding to hours and minutes or minutes and seconds (depending on the case).

Main success scenario:

|  |  |
| --- | --- |
| 1. The user introduces the time intervals | 2.The system checks the input  3.The system waits for the simulation interval |
| 4.The user selects the simulation interval | 5.The system checks if the selected simulation interval is a valid interval. |
| 6.The user starts the simulation | 7.The system starts the simulation and if everything was ok until here, it starts displaying the output in real simulation time. |
| 7.The user further interacts with the system by enabling/disabling the checkboxes that open/close several queues. | 8.After the simulation is complete, the system outputs the peak hour with the number of customers for the previously ended simulation. |

Error sequences:

E1) The error that can appear happens after pressing the "Start" button while not having a correctly formated input (i.e. the time intervals are not positive numbers, or the values for the minutes and seconds contain also letters or any other characters introduced accidentally.

E2) Closing all queues when there still are clients. The application will not know where to put the rest of the clients.

c) UI Requirements

The I/O mechanism available to the user must be:

-a working machine (PC, MAC, etc.)

-a keyboard to be able to input the time intervals

-a mouse to be able to press the "Start" button

-a monitor to be able to see the results

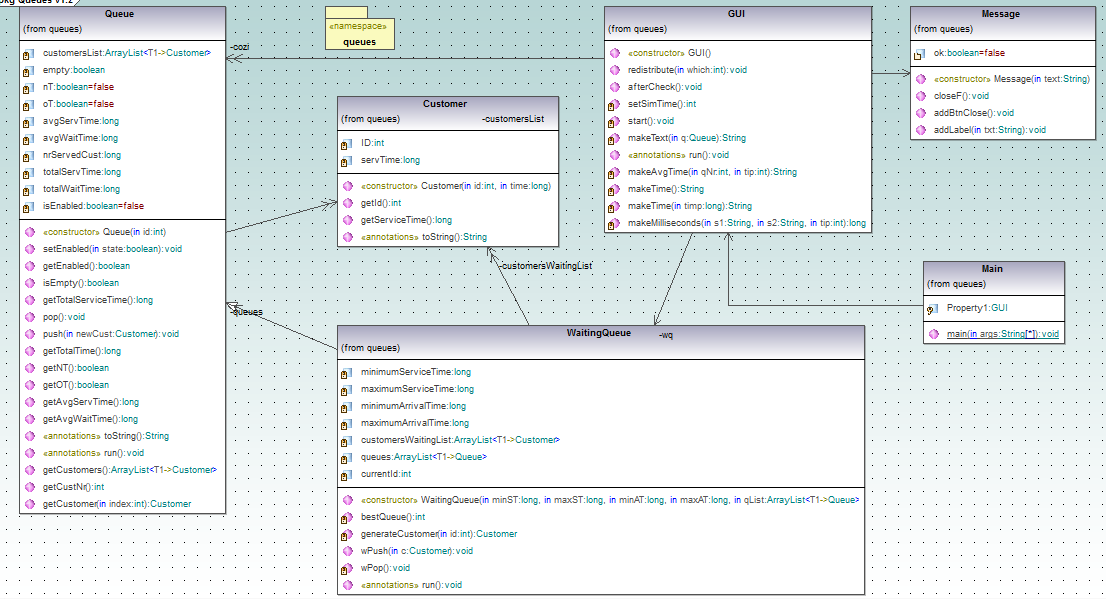
-an operating system, in order to be able to run the application.

**3.Projection**

**3.1 Class Diagram**

In this section, we will present the classes used to design this application and the relationships between them in an UML diagram. The diagram was designed Altova UMoldel software.

Even if at first sight this design seems to be too simplified since it only contains 5 entities, at a closer look we realize we just don't need more, the specific set of requirements being implemented easier and faster this way, without loosing any of its functionality.



**3.2 Class designing**

In order to gain a closer-to-real-time simulation, the entities used will implement the Runnable class or extend the Thread class, since we will need to create a thread for each queue

By breaking the solution to this problem into levels, we realize that at the lowest level we will need to have entities like Customer and Queue. The Customer entity will contain information about each customer, such as his/her ID and the service time he/she will need to wait.

The Queue entity will hold information about each queue independently, such as the number of customers that are waiting in queue at a given moment of time, attributes that will enable/disable this queue. This entity will dirrectly create instances of the Customer entity and place them in the queues.

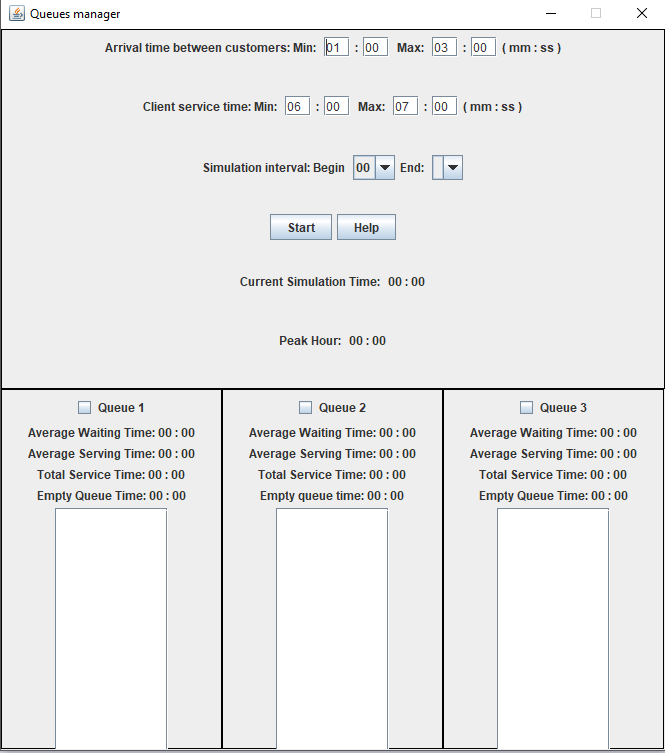
The WaitingQueue entity will handle the processing of the customers, such as placing them at the most efficient queue, generate customers based on the interval of arrival between them and place them in a waiting list until it computes the best available queue to place them.

The GUI entity has multiple tasks to manage, first of all being to create the Grafical User Interface that will make the connection between the user and the application. After this, it will process the input and create instances of the other classes in order to display results in the output Frame.

The Main entity has only one purpose, being to create an instance of the GUI class and to adjust the frame.

**3.3 Graphical user interface (G.U.I.)**

The main frame will look like this:



First of all the GUI contains 4 text fields for inputing the minimum and maximum arrival time between customers in minutes and seconds (we assumed, for the purpose of this simulation we won't need a time between customers more than one hour). Then, we have another 4 text fields for inputing the minimum and maximum service time for each customer, also in minutes and seconds.

Then we have 2 combo boxes that allow the user to select the simulation interval. While implementing these two combo boxes a major input problem has been avoided (that is the case when the user enters a lower limit greater than the upper limit in the simulation interval time). So, after the user selects the start hour of simulation from the first combo box, the second combo box enables only hours greater than the previously selected one.

The "Start" button which starts the simulation and all the processing behind the GUI is enabled.

Labels for displaying real simulation time information, such as the Current simulation time, the Peak Hour for the simulation(this only after the simulation has finished), the average waiting time, average serve time, total service time and empty queue time for each of the queues.

The interactive part is realized through the three checkboxes, which, when enabled mean that the corresponding queue is open. When a checkbox is unchecked, the customers that were waiting in the corresponding queue are re-distributed to the other opened queues.

Three text areas that show the queue evolution in real simulation time.

**4. Implementation**

The main objects from the problem domain, which are modeled in the solution domain are the customers, the queues and the GUI.

1.Customer

This class represents the abstraction of the real-world concept of a customer queueing for a service. The attributes of this class are the unique ID identifying a customer and the servTime that we will randomly generate later.

The constructor sets the attributes to the corresponding values passed as a parameter( the id and the serv time).

2.Queue

This class represents the abstraction of the real-world concept of a queue consisted of customers. In order to bring the simulation even closer to the reality, this class will extend the Thread class, meaning that we will create a separate thread for each queue. This class has a higher number of attributes since we will need to know information about each of the three queues in real time of simulation. Hence, we identify the attributes:  
- empty; - boolean attribute, returns true if the queue is empty, false otherwise

- full ; - boolean attribute, returns true if the queue is full, false otherwise

- customersList ; - an arrayList of customers currently in queue

- id ; - integer attribute, represents the id of the queue

- nrServedCust ; - integer attribute, represents the current number of served customers

- avgServTime ; - integer attribute, represents the average service time for a queue

- avgWaitTime ; - integer attribute, represents the average wait time for a queue

- totalServTime ; - integer attribute, represents the total service time for a queue

- totalWaitTime ; - integer attribute, represents the total wait time for a queue

- isEnabled ; - boolean attribute, is true if the queue is enabled with the checkbox from the GUI class

The constructor of this class only take as a parameter the id of the queue and initializes the attributes to default values.

3.WaitingQueue

Class that extends the Queue class, to inherit it's methods and to implement some extra methods needeed for the correct processing of the customers, will implement another queue, in which the customers are placed until the processing of the current queues is done, such as computing the best queue for each customer. Once this is done, the customer is poped from this list and pushed into the best avaialable queue.

Has the following attributes:

-currentId; - integer attribute, generates an ID for each customer;

-customersWaitingList; - the list containing the customers waiting to be processed, until the best available queue is computed

-maximumArrivalTime; - max value of the arrival time, needed to compute the interval

-maximumServiceTime; - similar

-minimumArrivalTime; -similar

-minimumServiceTime; -similar

-queues; - arrayList of Queues having the size 3.

The constructor of the class takes as parameters the time between customers, the service time interval to generate the customers and the arrayList of queues.

4.GUI

This class has the main priority since it has multiple roles. First of all, it creates the frame that will contain our simulation, will start the threads for each queues and for the waiting list of customers. In order to perform all these operations, this class needs to extend the JFrame class and to implement the Runnable class.

Class implements the following methods:

-redistribution(int which); - this method has the purpose of redistributing the customers from a recently closed queue to the other avaiable ones. We will do this by removing the customers from the queue and inserting them into the waiting list.

-afterCheck() - method which handles the enabling and disabling of the checkboxes which control the queues.

-makeAvgTime(int qNr, int tip) - is a method which computes the virtual simulation time from miliseconds into a format of virtual minutes and seconds to be displayed in the corresponding labels. The corresponding type is given by the int variable.

-makemilliseconds(String s1, String s2, int tip) - method that processes the input from the text boxes, converting those strings into milliseconds to be further processed.

-setSimTime() - method that takes the information entered by the user and converts it into milliseconds.

-start() - method which processes the input provided by the user and validates it.

-run() - method that implements the code that will be executed when an instance of GUI is created. In our case, this will be the task of updating the labels from the frame and check if the simulation is over in order to display the peak hour and the number of customers in queues at that time. In order to update the swing components we create a small timer which updates the current time label and handles the execution of the code, once it is updated.

5.Message

This class has the role of creating the frame for the help, placing the close button and implement the listener for it. Unfortunetly the implementation of the help file is unfinished so I won’t go into further detail.

6.Main

The purpose of this class is to create a direct instance of the GUI class, that consists of creating the frame, set it to visible and arrange it to the required dimension.

**6. Conclusions and further development**

**Conclusions:**

An application that simulates the evolution of a queue management system (i.e. a shop or supermarket). The user has the posibility to control the functionality of the queues, to open/close them in real simulation time. Real time results are updated and displayed in the corresponding labels.

**Further development:**

Some of the functionalities of this application can be extended further:

Introduce a help file

To give the user the option to pause, resume and start a new simulation.

A graphical representation for the queue evolution.

Implement a simulation with more than three queues.

**7.Bibliography**

* <https://labacal.utcluj.ro/moodle/> and other UTCN courses for Java basics
* <https://stackoverflow.com/> for different times when I was stuck
* <https://www.youtube.com/user/caveofprogramming> for multi thread implementation
* various internet forums