**Homework 2**

Programming techniques

**Queues**

*Student:* Florin – Catalin Ungureanu

*Grupa:* 30424

*Profesor:* Ioan Salomie

*Advisor :* Cristina Pop

*Cuprins*

1. Main Objective .................................................................................... 1

2. Problem analysis, Modeling, Scenarios.................... 1

*2.1. Problem analysis........................................................................ 1*

*2.2. Modeling ..................................................................................... 2*

*2.3. Scenarios ...................................................................................... 5*

3. Design ................................................................................................ 5

*3.1. UML diagram ......................................................................... 6*

*3.2. Class design .................................................................. 7*

*3.3. Algorithm .................................................................................. 10*

*3.4. User interface .................................................................... 11*

4. Implementare si testare .......................................................................... 12

5. Rezultate, concluzii si dezvoltări ulterioare ........................................... 14

6. Bibliografie ............................................................................................ 14

1. **Main objective**

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

Description : Queues are commonly used to model real world domains. 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 their "clients" are waiting in queues before they are served. 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 service supplier. When a new server is added the waiting customers 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 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 clients – when they show up and how much service they need. The finish time depends on the number of queues, the number of clients in the queue and their service needs.

* 1. **Problem analysis**

To implement the application we chose the Model-View-Controller (MVC) design pattern. The Model-View-Controller design concept, in the specialized and used MVC documentation, emerged as a need to transpose traditional data management methods into "virtual" environments, more specifically to ease user mode of operation respecting the same basic principles and differentiating through the tools used. However, the basic principles of the concept are easy to understand, but the details are complex enough to launch many debates and meet contradictory implementations.

The principle underpinning the Model-View-Controller concept is the sharing of responsibilities. In a based application designed to meet this concept, the model part will only work with the state of the application and its logic, will not count how this state is or will be represented to the user, or how it interacts with the application. Also, the "view" part is concerned only with creating the user interface based on the data and especially the changes of its status received from the "model". It does not matter the logic of the application or how the "input" process takes place, but only the correct representation of the current state of the model. Finally, the "controller" deals with the translation of user actions into "updates" to the model, and it is not important what the "update" model will use.

The model represents the hard-programming part, the logic of the application. He is responsible for actions such as data operations and their rescue. The model thus represents a collection of classes that will perform certain functions, unrelated to the actions of the user. They will be the basis for any function that your application will need to fulfill.

The controller is the link between the model and the view, between the logic of the application and the actions of the user. Depending on the user's requirements, the controller will perform a specific function specifically defined for the user application section. The function will use the model to process the data and send the results to the view, which will update the data, according to the actions of the user. In this package we included the classes: ClickListener, Controller and TableCellListener.

1. **Design**

To implement the project, I used the Eclipse development environment and the Java programming language. The problem has been divided into classes that will be presented below. The steps to solve the problem were to create the UML diagram underlying the project, and then to build the required classes and methods.

For a better understanding of the class structure we chose to divide them into several packages, these being: model, view, controller and main.

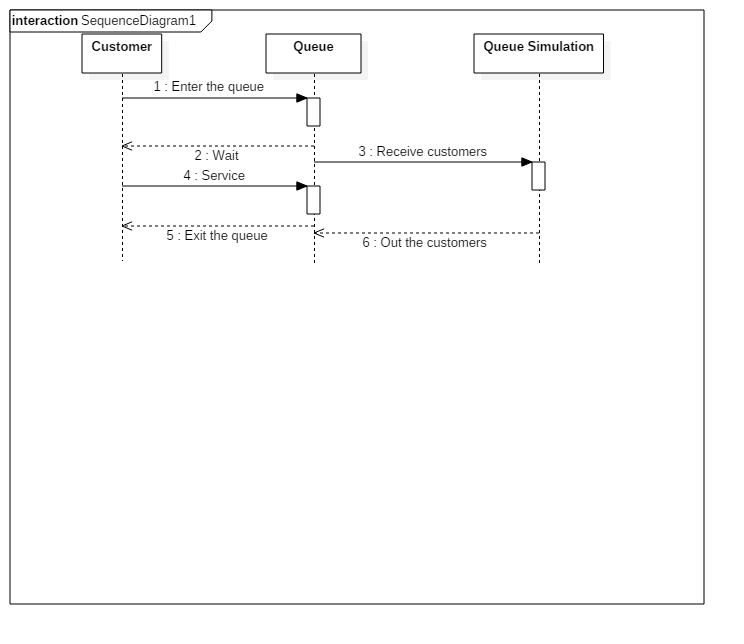
The problem lies in creating an algorithm for dividing customers at queues that may exist at a time. The algorithm makes this queue location so that the wait time of a client is minimal. At the end, the number of items sold, the number of clients served by a particular house, the average waiting time for each home, and the average wait time of the clients will be calculated. In order to determine the queue to which a customer is assigned, the time required to empty is calculated for each queue. The customer will be seated at the queue that has the shortest time to empty.

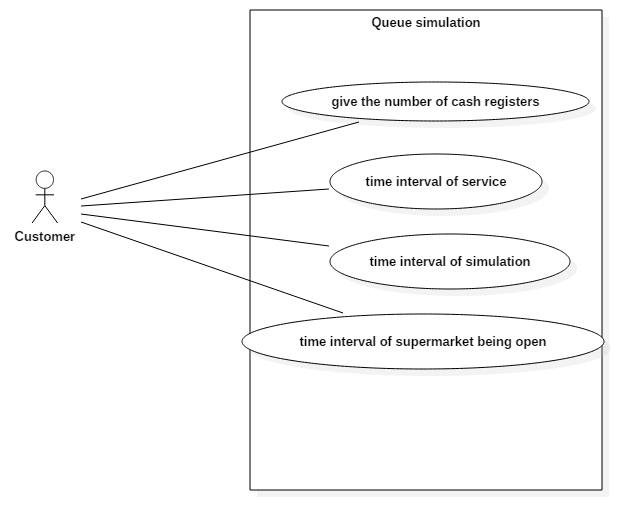
For each client the corresponding id will be shown, this id being unique to the client, the id generated timer used in the java language, as well as the house that serves the client.

This application should be as relevant as a real case, so the arrival of the customers will be generated between the time the store is open. At the time of closure, customers cease to defend themselves, and all that remains to do is to serve customers who are still waiting.

The thread concept defines the smallest processing unit that can be programmed to run by the operating system. It is used in programming to streamline program execution by executing distinct code portions in parallel within the same process. Sometimes, however, these portions of code that constitute the body of the threads are not completely independent, and at certain times of execution, it may happen that a thread has to wait for the execution of some other thread instructions to continue the execution of its own instructions. This technique by which a thread waits for the execution of other threads before continuing its execution is called thread synchronization.

Sequence diagram





1. **What I have learned**

Following the implementation of this theme, a new experience has been gained in programming, by using the thread concept, which is beneficial to any programmer, has recapitulated graphical interface elements starting with the use of buttons, JTextField, JTextArea and I Learned to use JScrollPane to look at all project output data. A difficult project with regard to the use of threads, the rest of the program containing little elements learned so far.

**5. Further Developments**

The application can be modified to add customers to simulate overcrowding at a given time, the queue client's placement algorithm might be modified to take into account the cumulative waiting time of customers who are already in the queue. It could create more compelling animations for clients, for example, to see more fluently how the client advances in the queue and when the circle gradually decays.

**6.Bibliography**

1. OOP courses – Marius Joldos
2. [www.stackoverflow.com](http://www.stackoverflow.com)
3. <http://docs.oracle.com/javase/tutorial/>
4. Thinking In Java –Bruce Eckel