Order Management

Student: Suciu Mihai

Group: E_30422

Cuprins

1. Objective

The requirement of this project is :

Consider an application Order Management for processing customer orders for a warehouse. Relational databases are used to store the products, the clients and the orders. Furthermore, the application uses (minimally) the following classes:

- Model classes - represent the data models of the application (for example Order, Customer, Product)
- Business Logic classes - contain the application logic (for example Order Processing, Warehoused, Client Admin)
- Presentation classes – classes that contain the graphical user interface
- Data access classes - classes that contain the access to the database

Other classes and packages can be added to implement the full functionality of the application.

a. Analyze the application domain, determine the structure and behavior of its classes and draw an extended UML class diagram

b. Implement the application classes. Use javadoc for documenting classes.

c. Use reflection techniques to create a method that receives a list of objects and generates the header of the table by extracting through reflection the object properties and then populates the table with the values of the elements from the list. JTable createTable(List

d. Implement a system of utility programs for reporting such as: under-stock, totals, filters, etc.

The main objective of this homework was to design and implement an application Order Management for processing customer order for a warehouse where relational databases are used to store the products, the clients and the orders.

This problem can be split as follows:

- Creating the Client, Product and Order classes.
- Creating the Databases in MySQL with the following tables : Client, Product and Comenzi
- Creating the link between Eclipse and MySQL
- Creating data access operation for each class, to insert ,update and delete information in Data bases
- Creating the graphical interface

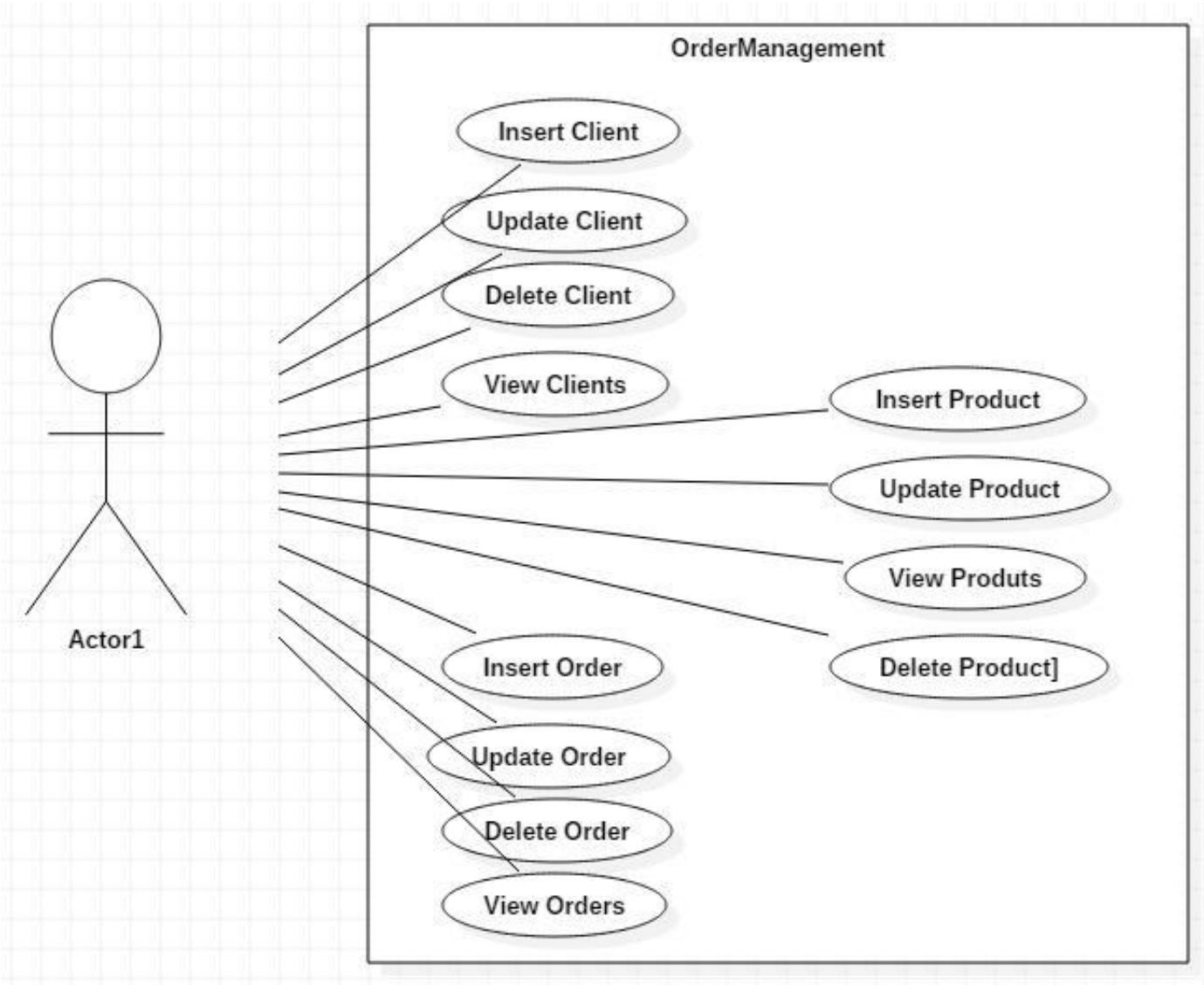
2. Problem analysis, use case diagram, assumptions

2.1 Problem analysis

- For this implementation I used design pattern Model-View-Controller (MVC). **Model–view–controller** (**MVC**) is an architectural pattern commonly used for developing user interfaces that divides an application into three interconnected parts. This is done to separate internal representations of information from the ways information is presented to and accepted from the user. The MVC design pattern decouples these major components allowing for efficient code reuse and parallel development.
- The *model* is the central component of the pattern. It expresses the application's behavior in terms of the problem domain independent of the user interface. It directly manages the data, logic and rules of the application.
- A *view* can be any output representation of information, such as a chart or a diagram. Multiple views of the same information are possible, such as a bar chart for management and a tabular view for accountants.
- The third part or section, the *controller*, accepts input and converts it to commands for the model or view.
- The model is responsible for managing the data of the application. It receives user input from the controller.
- The view means presentation of the model in a format.
- The controller responds to the user input and performs interactions on the data model objects. The controller receives the input, optionally validates it and then passes the input to the model.

2.2 Use Case diagram

The application will contain a graphical interface, where the actor client will be able to introduce the information, and after it to perform the insertion, update and delete the data in MySQL Data Bases.



CASE I

Actor: Client

Case: **without** error

Insert one Client in Data Bases

a) The client writes input data: ClientID , Name, Mail and address.
   For example:
   ClientID: 2
   Name : Mihai
   Mail : test@gmail.com
   Address : Grove Street east coast 9
b) Press “Insert Client”
c) Press “View clients” to check if the Client was added

CASE II

Actor: Client

Case: **without** error

Update one Client in Data Bases

a) The client writes input data: ClientID , Name, Mail and address.
   ID MUST BE THE SAME AS THE CLIENT YOU WANT TO CHANGE
   ! For example:
   ClientID: 2
   Name : Mihai
   Mail : mailChanged@gmail.com
   Address : Grove Street east coast 9
b) Press “Update Client”.
c) Press “View clients” to check if the Client was updated.

CASE III

Actor: Client

Case: **without** error

Delete one Client from Data Bases

a) The client writes the client's id.
   For example:
   ClientID: 2
b) Press "Delete Client.
c) Press "View clients" to check if the Client was updated.

d)

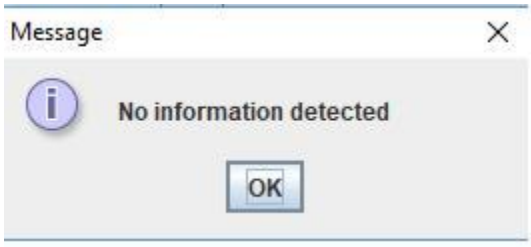
CASE IV

Actor: Client

Case: **with** error

Insert one Client in Data Bases

a) You don't insert anything
b) Press "Insert Client"
c) You will receive this



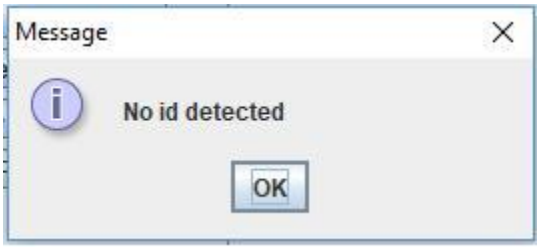
CASE V

Actor: Client

Case: **with** error

Delete one Client in Data Bases

a) You don't insert an id
b) Press "Delete Client"
c) You will receive this



2.3 Assumptions

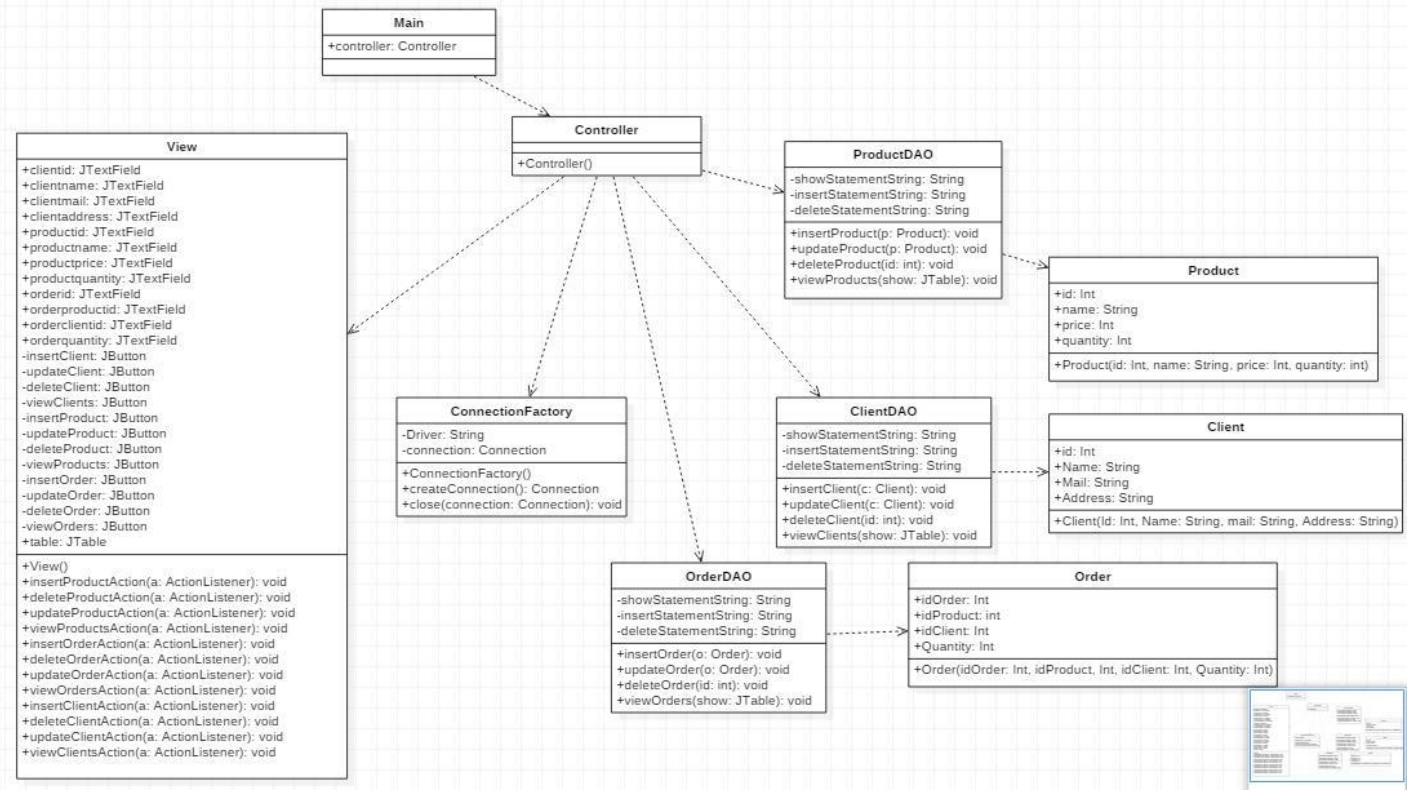
The client will know that do not request to display one empty databases.

The client will not introduce the same id as another object from databases.

The client will not press insert buttons without write any data in text fields.

3. Design

3.1 UML Diagram



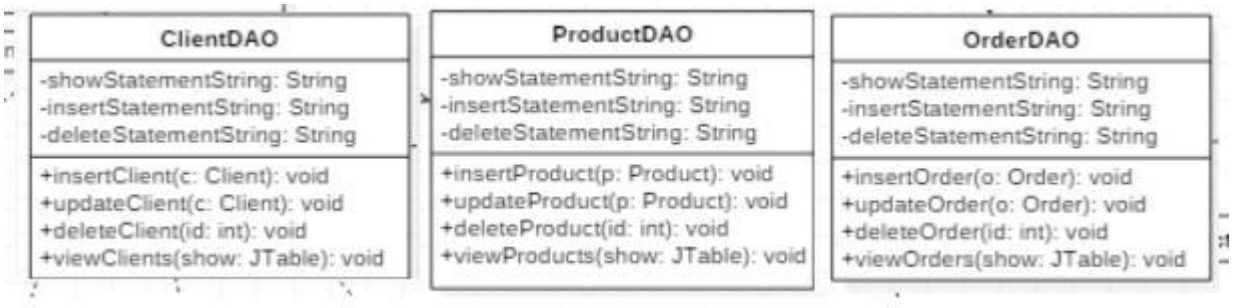
(uml Diagram for this project)

3.2 Class design

The projection of this application started with Client, Product and Order classes. I will represent below each class with their attributes and operations. Here we have four attributes and one constructor. The same in the Product and Order classes.

(uml diagrams for Client, Order and Product classes)

The second step I implemented three classes: ClientDAO, ProductDAO and OrderDAO where DAO = data access operation.



(uml diagrams for ClientDAO, OrderDAO and ProductDAO classes)

Here I have 3 attributes which help me to access the data bases like :

“INSERT INTO client(idClient, nameClient, mail, address) “ + “values(?, ?, ?, ?)” and four methods to make the following actions: insert, update, delete (with data bases) and show the tables in the graphical interface.

The next class and the most important is the Connection class. Here I have one string and one connection, one constructor and two methods, to create and close the connection.

(uml diagrams for ConnectionFactory and Controller classes)

Last class is Controller. Here are linked all classes and control the buttons actions.

The Main class contains an attribute of type Controller which construct the application

3.3 Graphical Interface

I will present below the View class with its attributes and methods.

The first 12 text fields are used for input data, 4 for each class : Client, Product and Order. Here you need to write: for client -id, name, mail, address, for Product – id, name, price, quantity and for Order – id, Product id, Client id and the quantity you want to buy.

The next 12 buttons are used to help you to access the data bases. Here you can insert one client, one product or one order. If you missed something you can modify with the Update buttons. After it if you don't want one of client / product or order, you can delete it at any time. Last 4 buttons are used to check the tables and show it in right of the graphical interface. The last argument here is the table which help me to show you something when you press View buttons.

Here I have some methods which create the action listener for each button.

4. Implementation

Classes & important methods

First class is Client. Here I have 4 attributes , one constructor and getters and setters for each attribute.

```
public Client(int id, String name, String mail, String address) {
                this.id = id;
                this.name = name;
                this.mail = mail;
                this.address = address;
}
```

Like this one I have 2 more classes with the same constructor, getters and setters : Product and Order. I will show below only the attributes.

For Product:

```
public int id;
public String name;
public int price;
public int quantity;
```

For Order:

```
public int  idOrder;
public int  idProduct;
public int  idClient;
public int  quantity;
```

The next classes that I implanted are for data access information: ClientDAO, ProductDAO and OrderDAO. First, in this classes I have 3 strings which help me to insert/update and delete data.

```
private final static String showStatementString = "SELECT * FROM client";

private final static String insertStatementString = "INSERT INTO client (idClient, nameClient,
mail, address)"+ "values(?, ?, ?, ?)";

private final static String deleteStatementString = "DELETE FROM client WHERE idClient=?";
```

More than these 3 strings, I have some methods like this one to insert data in tables.

```
public static void insertClient (Client c) {
            Connection dbConnection = ConnectionFactory.getConnection();
            PreparedStatement insertStatement = null;
            try {
                  insertStatement =
dbConnection.prepareStatement(insertStatementString);
                  insertStatement.setInt(1, c.getId());
                  insertStatement.setString(2, c.getName());
                  insertStatement.setString(3, c.getMail());
                  insertStatement.setString(4, c.getAddress());
```

```
                insertStatement.execute();
                ConnectionFactory.close(insertStatement);
                System.out.println("Client added");
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
```

The most important think in this method is “ insertStatement.execute()”. Here I tell the databases to insert my information. Like this one is the product insertion. One different mode to insert is at Order. Here I have one Boolean variable which is true when the quantity inserted is lower then quantity of the product selected.

```
                if(o.getQuantity()>quantity)
                        ok=false;

                if(ok) {
                        insertStatement.execute();
                        JOptionPane.showMessageDialog(view, "Added");
                }
                else {
                        JOptionPane.showMessageDialog(view, "Not enough quantity");
                        }
```

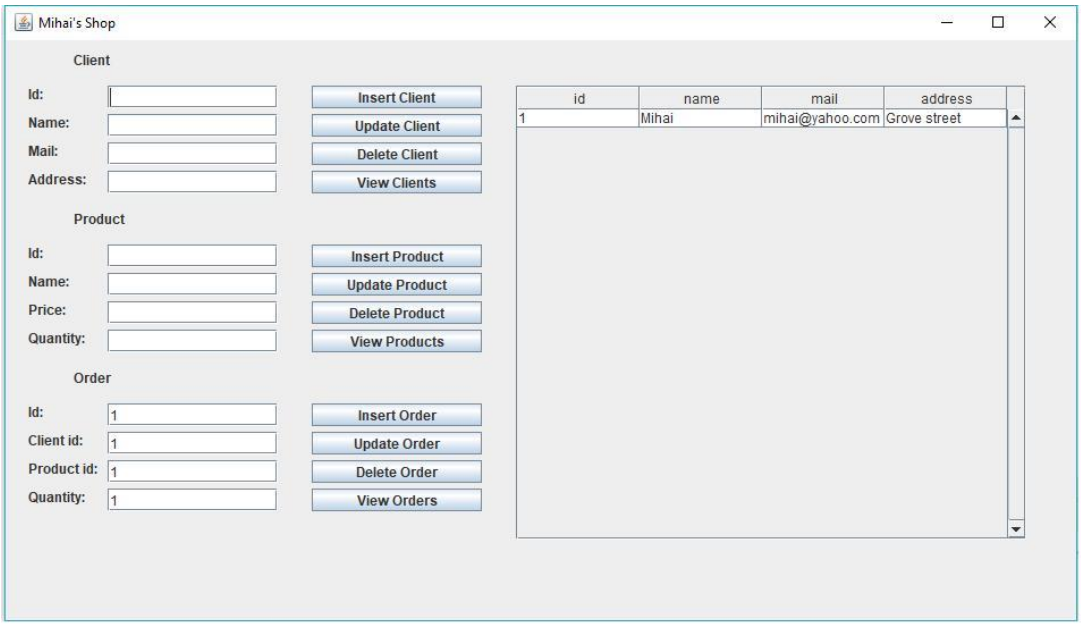
The “brain” class of this project is Connection Factory. First, here I have one String

```
            private static final String DRIVER = "com.mysql.jdbc.Driver";
```

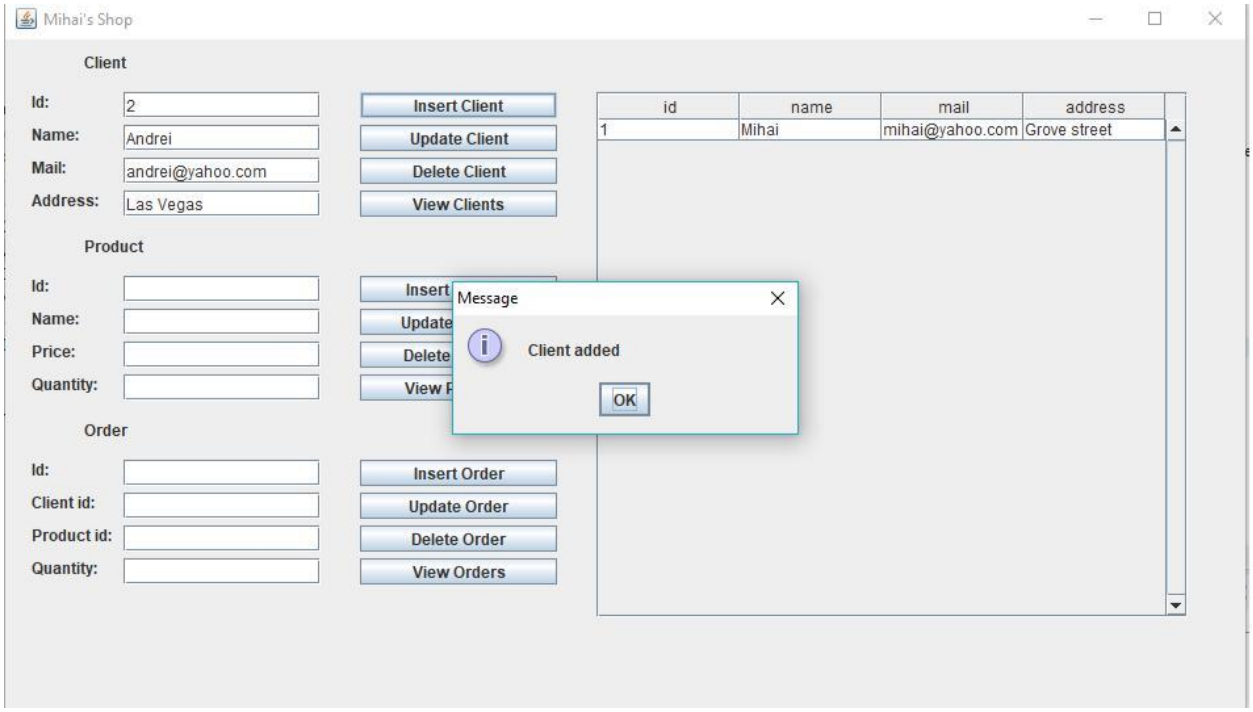
which help me to finish the connection. Which is here:

```
            public static Connection createConnection()
                        { try {
                        connection = (Connection) DriverManager.getConnection(
            "jdbc:mysql://localhost:3306/hw3?autoReconnect=true&useSSL=false",
"root", "pass");
                        } catch (SQLException e) {
                e.printStackTrace();
                        }
                if (connection != null)
                        System.out.println("Connection created");
                else
                        System.out.println("Connection error");
                return connection;
        }
```

5. Results



You need to write the input data, press “Insert Client” and after it you will see this message:



For example, id: 2 , Name: Andrei, mail: andrei@yahoo.com and the address: Las Vegas.

6. Conclusions

During the design of this project I improved my abilities in interfaces. Before this I learned how to use the concept of data bases and the connection between this one and java and this is the most important think for every programmer. This project was hard just from the point of view of connection with data bases, the rest contain only previously learned items.

Further development: improve the graphical interface, introduces some imagines of products, and more error messages.

7.Bibliography

- http://www.java2s.com/Code/Java/Swing-JFC/ScrollableLabel.htm
- https://www.ntu.edu.sg/home/ehchua/programming/java/JDBC_Basic.html
- https://www.youtube.com/watch?v=CrHZIW9pnnQ