**Cristian Flaviu - Remus**

**Group 30423**

# Homework3

## 1) Objectives :

The main purpose of this program is to implement a simulation application for processing customer orders for a warehouse.

In order to do that we had find a way to store information about the Clients (name, age ,address ,id) , Products(product Name, Quantity ,Product ID , Price) and Orders(ID Client, ID Product ,Quantity Ordered, Order ID).

Relational databases are used to store the products, the clients and the orders.

The secondary objectives are:

a) **Designing Model Classes:** represent the data models of the application(Client , Product , Orders )
b) **Designing Business Logic classes:** these classes will contain the application logic
c) **Designing Data Access classes:** these classes will contain the access to the database
d) **Implementing the Graphic User Interface (GUI).** Final step where we make a simple GUI for the user to interact with.

2) **Problem analysis, scenarios and use cases :**
The program expects interaction with a user in order to introduce date and specify the flow of instruction. The user can choose what operation she/he wants to perform(inserting, deleting, updating ) the current tables.

3)

a) Use case “Delete Client”
The user have to open the program and choose from the combobox situated right under the button DeleteClient the Id of the Client he/she wants to delete from the table. After selecting the Client the user has to press the button DeleteClient and the Client will be deleted.
b) Use case “DeleteProduct”
The user have to open the program and choose from the combobox situated right under the button DeleteProduct the Id of the Product he/she wants to delete from the table. After selecting the Product the user has to press the button DeleteClient and the Client will be deleted.

c) Use case “UpdateClient”

The user have to open the program and choose from the combobox situated right under the button UpdateClient the Id of the Client he/she wants to edit from the table. After selecting the Client the user has to press the button UpdateClient and a new windows will be opened. The user has to introduce the new value for the attribute of the client(Name, Address, Email, Age) and right after that she/he has to press the button update in order for the data to be updated.

d) Use case “UpdateProduct”

The user have to open the program and choose from the combobox situated right under the button UpdateProduct the Id of the Product he/she wants to edit from the table. After selecting the Product the user has to press the button UpdateClient and a new window will be opened from where the user has to introduce the new value for the attribute of the Product(Name, Quantity, Price) and right after that she/he has to press the button update in order for the data to be updated.

e) Use case “New Order”

The user have to open the program and press the button NewOrder. A new window will open and the user has to choose an IdClient , an IdProduct and a quantity. After that she/he has to press the button NewOrder.

f) Use case “NewProduct”

The user have to open the program and press the button NewProduct. A new window will open and the user has to choose a Name , a quantity and a price for the new product. After that she/he has to press the button NewProduct.

g) Use case “New Client”

The user have to open the program and press the button |NewClient. A new window will be opened and the use has to introduce the data required and right after that to press the button Save.

Alternative scenarios :

The user does not respect the specified format (he has introduced other characters than numbers,etc). In this case the system won’t produce any output and an error message will be displayed.

## 4) Design (design decisions, UML diagrams, data structures, class design, interfaces, relationships, packages, algorithms, user interface)

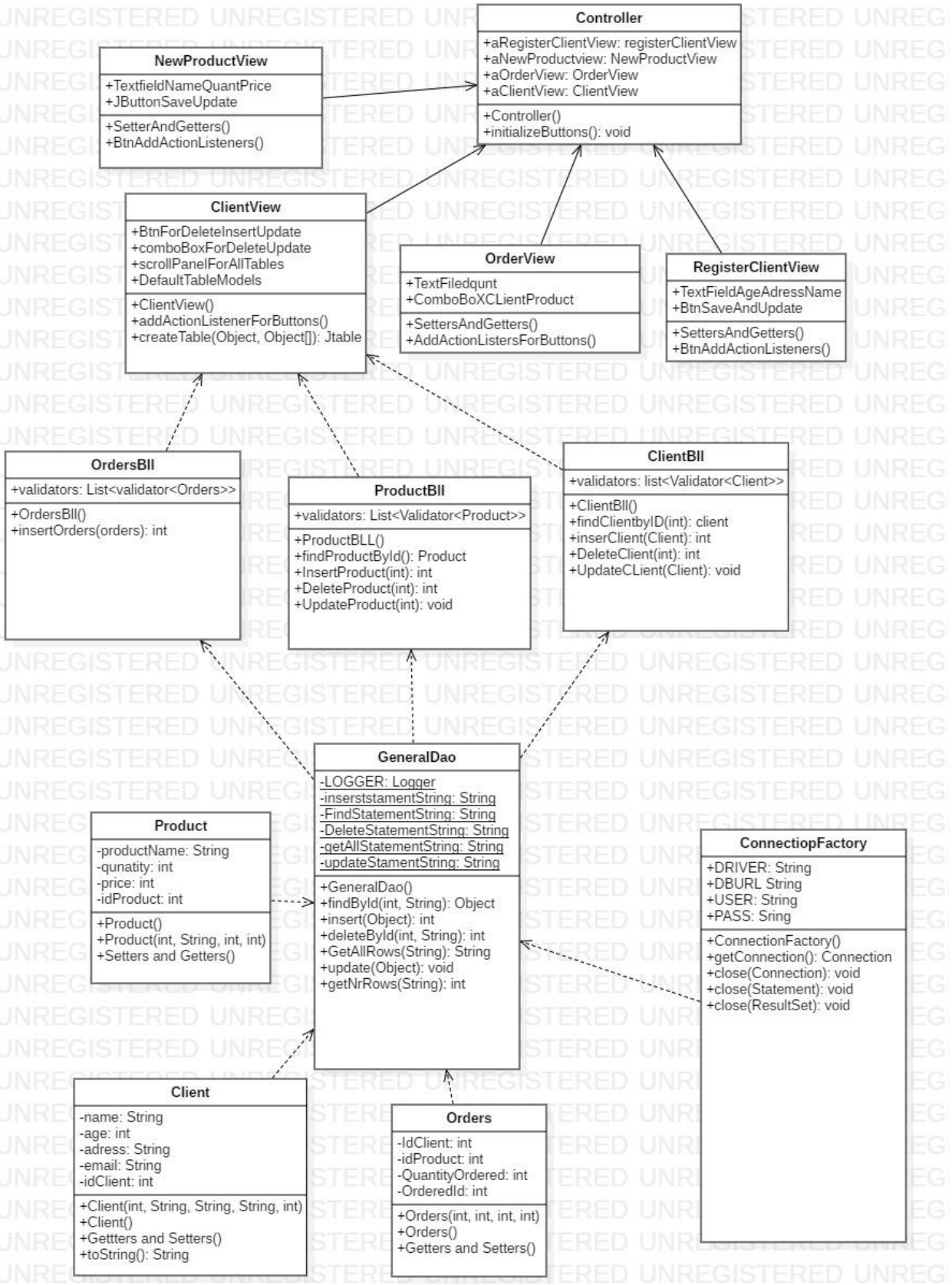
Firstly, we must design a design all model classes that we need to store the information(Client, Product, Orders).

After that we have to design the classes which contains the access to the database(ConnectionFactory) and the classes which cantains the data application logic(GeneralDao, ClientBLL, ProductBll,OrderBll).

The final step will represent designing the GUI. It simply displays the required information in different JScrollPanes.

There are 4 packages :

- **models** which holds the following classes: Client ,Orders and Product
- **dao** which holds the following classes: GeneralDao
- connection which holds the following classes : ConnectionFactory
- **bll** which hold the following classes: StudentBll, ClientBll, OrdersBll
- **validator** which holds the following classes: ClientAgeValidator ,OrdersProductValidator, OrdersQuantityValidator, ProductProceValidator
- **view** which hold the following classes : ClientView(main View), Controller, NewProductView, OrderView, RegisterClientView


Controller
+aRegisterClientView: registerClientView
+aNewProductview: NewProductView
+aOrderView: OrderView
+aClientView: ClientView
+Controller()
+initializeButtons(): void
NewProductView
+TextfieldNameQuantPrice
+JButtonSaveUpdate
+SetterAndGetters()
+BtnAddActionListeners()
ClientView
+BtnForDeleteInsertUpdate
+comboBoxForDeleteUpdate
+scrollPanelForAllTables
+DefaultTableModels
+ClientView()
+addActionListenerForButtons()
+createTable(Object, Object[]): Jtable
OrderView
+TextFiledqunt
+ComboBoXCLientProduct
+SettersAndGetters()
+AddActionListersForButtons()
RegisterClientView
+TextFieldAgeAdressName
+BtnSaveAndUpdate
+SettersAndGetters()
+BtnAddActionListeners()
OrdersBll
+validators: List<validator<Orders>>
+OrdersBll()
+insertOrders(orders): int
ProductBll
+validators: List<Validator<Product>>
+ProductBLL()
+findProductById(): Product
+InsertProduct(int): int
+DeleteProduct(int): int
+UpdateProduct(int): void
ClientBll
+validators: list<Validator<Client>>
+ClientBll()
+findClientbyID(int): client
+inserClient(Client): int
+DeleteClient(int): int
+UpdateCLient(Client): void
GeneralDao
-LOGGER: Logger
-inserststamentString: String
-FindStatementString: String
-DeleteStatementString: String
-getAllStatementString: String
-updateStamentString: String
+GeneralDao()
+findById(int, String): Object
+insert(Object): int
+deleteById(int, String): int
+GetAllRows(String): String
+update(Object): void
+getNrRows(String): int
Product
-productName: String
-qunatity: int
-price: int
-idProduct: int
+Product()
+Product(int, String, int, int)
+Setters and Getters()
ConnectiopFactory
+DRIVER: String
+DBURL String
+USER: String
+PASS: Sring
+ConnectionFactory()
+getConnection(): Connection
+close(Connection): void
+close(Statement): void
+close(ResultSet): void
Client
-name: String
-age: int
-adress: String
-email: String
-idClient: int
+Client(int, String, String, String, int)
+Client()
+Gettters and Setters()
+toString(): String
Orders
-IdClient: int
-idProduct: int
-QuantityOrdered: int
-OrderedId: int
+Orders(int, int, int, int)
+Orders()
+Getters and Setters()

# 5) Implementation

a) Client

```
private String name;
private int age;
private String adress;
private String email;
private int idClient;
```

The value of the `idClient` will be an interger autogenared by the mysql and the rest of the attributes will be set by the user.

b) Product

```
private String productName;
private int quantity;
private int price;
private int idProduct;
```

The value of the `idProduct` will be an interger autogenared by the mysql and the rest of the attributes will be set by the user.

c) `Orders`

```
private int IdClient;
private  int IdProduct;
private int quantityOrdered;
private int OrderId;
```

The value of the `OrderId` will be an interger autogenared by the mysql and the rest of the attributes will be set by the user.

d) ConnectionFactory

```
private static final Logger LOGGER = Logger.getLogger(ConnectionFactory.class.getName());
        private static final String DRIVER = "com.mysql.cj.jdbc.Driver";
        private static final String DBURL = "jdbc:mysql://localhost:3306/sys";
        private static final String USER = "root";
        private static final String PASS = "rectilinie1";

private Connection createConnection() {
    Connection connection = null;
    try {
    connection = DriverManager.getConnection(DBURL, USER, PASS);
    } catch (SQLException e) {
    LOGGER.log(Level.WARNING, "An error occured while trying to connect to the database");
    e.printStackTrace();
    }
    return connection;
        }
```

This method will create a connection to mysql by using the driver, user and password mentioned before.

```
        public static void close(Connection connection) {
        if (connection != null) {
        try {
        connection.close();
        } catch (SQLException e) {
        LOGGER.log(Level.WARNING, "An error occured while trying to close the connection");
        }
        }
                }
```

This method will close the Connection Connect and in case of error a suggestive message will be printed.

e) General Dao

```
    private static String insertStatementString = "INSERT INTO ";
    private static String findStatementString="SELECT * FROM";
    private static String deleteStatementString = "DELETE FROM ";
    private static String getAllStratementString = "SELECT * FROM ";
    private static String updateStatementString = "UPDATE ";

    public static int insert(Object aObject) {
    Connection dbConnection = ConnectionFactory.getConnection();
    int insertedId = -1;
    PreparedStatement insertStatement = null;
    insertStatementString = "INSERT INTO ";
    insertStatementString = insertStatementString.concat(aObject.getClass().getSimpleName() + " (");
    java.lang.reflect.Field[] attributes = aObject.getClass().getDeclaredFields();
    for (int i = 0; i < attributes.length - 1; i++) {
    System.out.println("ATTRIBUTE NAME: " + attributes[i].getName());
    insertStatementString = insertStatementString.concat("`" + attributes[i].getName() + "`,");

    }
    insertStatementString = insertStatementString.substring(0, insertStatementString.length() - 1)
    .concat(") VALUES(");

    for (java.lang.reflect.Field field : attributes) {
    insertStatementString = insertStatementString.concat("?,");

    }
    insertStatementString = insertStatementString.substring(0, insertStatementString.length() - 1 -
    2).concat(")");

    System.out.println(insertStatementString);

    try {
    insertStatement = dbConnection.prepareStatement(insertStatementString,
    Statement.RETURN_GENERATED_KEYS);
    if (aObject.getClass().getSimpleName().equals("Client")) {
    System.out.println("here is a client");

    insertStatement.setString(1, ((Client) aObject).getName());
    insertStatement.setInt(2, ((Client) aObject).getAge());
    insertStatement.setString(3, ((Client) aObject).getAdress());
    insertStatement.setString(4, ((Client) aObject).getEmail());

    }

    else {
    if (aObject.getClass().getSimpleName().equals("Product")) {
    System.out.println("here is a Product");

    insertStatement.setString(1, ((Product) aObject).getProductName());
    insertStatement.setInt(2, ((Product) aObject).getQuantity());
    insertStatement.setInt(3, ((Product) aObject).getPrice());

    } else {
    System.out.println("here is a order");

    if (aObject.getClass().getSimpleName().equals("Orders")) {
    insertStatement.setInt(1, ((Orders) aObject).getIdClient());
                                        insertStatement.setInt(2, ((Orders)
aObject).getIdProduct());
                                        insertStatement.setInt(3, ((Orders)
aObject).getquantityOrdered());
                                    }
                                }
```

```
                }
                insertStatement.executeUpdate();

                ResultSet rs = insertStatement.getGeneratedKeys();
                if (rs.next()) {
                        insertedId = rs.getInt(1);
                }
        } catch (SQLException e) {
                LOGGER.log(Level.WARNING, "StudentDAO:insert " + e.getMessage());
        } finally {
                ConnectionFactory.close(insertStatement);
                ConnectionFactory.close(dbConnection);
        }
        return insertedId;

        }
```

This method will insert an object into the table. Firstly, it generates the insertStamentString by concatenating to the string the ClassName of the object and its attributes .After that it checks what type of object is (Client, Product or Orders) and will modify the the insertStamentString according to what type of object is .The method will return the auto generated id.

```
public static int deleteByID(int id, String ClassName) {
                Connection dbConnection = ConnectionFactory.getConnection();
                PreparedStatement findStatement = null;
                deleteStatementString = "DELETE FROM ";
                deleteStatementString = deleteStatementString.concat(ClassName + " Where");

                if (ClassName.equals("Client")) {
                        System.out.println("trying to delete a client");
                        deleteStatementString = deleteStatementString.concat(" idClient=?");
                } else {
                        if (ClassName.equals("Product"))

                        {
                                deleteStatementString      =      deleteStatementString.concat("
idProduct=?");
                        }
                        else
                        {
                                if(ClassName.equals("Orders"))
                                {
                                        deleteStatementString   =   deleteStatementString.concat("
OrderId=?");
                                }
                        }
                }
                System.out.println(deleteStatementString);

                try {

                        findStatement = dbConnection.prepareStatement(deleteStatementString);
                        findStatement.setInt(1, id);
                        return findStatement.executeUpdate();

                } catch (SQLException e) {
                        System.out.println(e.getMessage());
                } finally {
                        ConnectionFactory.close(findStatement);
                        ConnectionFactory.close(dbConnection);
                }

                return -1;

        }
```

This method get as an input the id of the object that you want to deleted and as a String the type of the data you want to delete(Client, Product, Orders).Firstly it generates the query for the deletion by adding the attributes of the class mentioned as parameter and right after that it will execute the query.

```
public static String[][] getAllRows(String ClassName) {
		Connection connection = ConnectionFactory.getConnection();
		PreparedStatement findStatement = null;
		ResultSet rs = null;
		ResultSetMetaData rsmd = null;
		String[][] Data = new String[100][100];
		int row = 0;
		getAllStratementString = "SELECT * FROM ";
		getAllStratementString = getAllStratementString.concat(ClassName);
		System.out.println(getAllStratementString);
		try {
			findStatement = connection.prepareStatement(getAllStratementString);
			rs = findStatement.executeQuery();
			rsmd = (ResultSetMetaData) rs.getMetaData();

			while (rs.next()) {
				for (int i = 0; i < rsmd.getColumnCount(); i++) {

					Data[row][i] = rs.getObject(i + 1).toString();

				}
				row++;

			}

		} catch (SQLException e) {
			System.out.println(e.getMessage());
		} finally {
			ConnectionFactory.close(rs);
			ConnectionFactory.close(findStatement);
			ConnectionFactory.close(connection);
		}
		return Data;
	}
```

This method get as input the Name of the Class/Table from where you want to copy your data and will return a matrix of type String with all the data from the Table specified as parameter by iterating through all row of the table;

f) ClientView

```
public JTable createTable(Object aObject, Object[][] bObjects) {
		JTable table = null;

		java.lang.reflect.Field[] attributes = aObject.getClass().getDeclaredFields();
		String[] columnNames = new String[attributes.length];

		for (int i = 0; i < attributes.length; i++) {
			System.out.println("ATTRIBUTE NAME: " + attributes[i].getName());
			columnNames[i] = attributes[i].getName();
		}
		String aux=columnNames[columnNames.length-1];
		for (int i = attributes.length-1; i >0; i--) {

			columnNames[i] = columnNames[i-1];



		}
```

```
                columnNames[0]=attributes[attributes.length-1].getName();


                if (aObject.getClass().getSimpleName().equals("Client")) {
                        modelClient = new DefaultTableModel((String[][]) bObjects, columnNames);
                        table = new JTable(modelClient);
                        table.setModel(modelClient);
                } else {
                        if (aObject.getClass().getSimpleName().equals("Product")) {
                                modelProduct = new DefaultTableModel((String[][]) bObjects, columnNames);
                                table = new JTable(modelProduct);
                                table.setModel(modelProduct);
                        }
                        else
                        {
                                if(aObject.getClass().getSimpleName().equals("Orders"))
                                {modelOrders = new DefaultTableModel((String[][]) bObjects, columnNames);
                                table = new JTable(modelOrders);
                                table.setModel(modelOrders);

                                }
                        }

                }

                table.setBounds(109, 121, 299, 162);

                return table;

        }
```

This method will generates a JTable which will have as columns the attributes of the Object aObject and will store the data from the matrix of Objects bObjects.

```
        public void addRowClient(Object object) {

                if (object.getClass().getSimpleName().equals("Client")) {
                        System.out.println("add a client");
                        ClientBll aBll = new ClientBll();
                        int a=aBll.insertClient((Client) object);

                        comboBocDelete.addItem(a+"");
                        comboBoxUpdateClient.addItem(a+"");
                        while (modelClient.getRowCount() > 0) {
                                modelClient.removeRow(0);
                        }

                        String[][] aStrings = GeneralDao.getAllRows(object.getClass().getSimpleName());
                        int row = GeneralDao.getNrRows(object.getClass().getSimpleName());
                        for (int i = 0; i < row; i++) {
                                modelClient.addRow(aStrings[i]);
                        }

                } else {
                        if (object.getClass().getSimpleName().equals("Product")) {

                                System.out.println("add a Product");
                                ProductBll aBll = new ProductBll();
                                int a=aBll.insertProduct((Product) object);
                                comboBoxProduct.addItem(a+"");
                                comboBoxUpdateProduct.addItem(a+"");
                                while (modelProduct.getRowCount() > 0) {
                                        modelProduct.removeRow(0);
                                }

                                String[][]                              aStrings                              =
GeneralDao.getAllRows(object.getClass().getSimpleName());
                                int row = GeneralDao.getNrRows(object.getClass().getSimpleName());
```

```
                        for (int i = 0; i < row; i++) {
                                modelProduct.addRow(aStrings[i]);
                        }
                }
                else
                {
                        if (object.getClass().getSimpleName().equals("Orders"))
                        {
                                System.out.println("add a Order");
                                OrdersBll aBll = new OrdersBll();
                                aBll.insertOrders((Orders) object);

                                while (modelOrders.getRowCount() > 0) {
                                        modelOrders.removeRow(0);
                                }

                                String[][] aStrings =
GeneralDao.getAllRows(object.getClass().getSimpleName());
                                int row =
GeneralDao.getNrRows(object.getClass().getSimpleName());
                                for (int i = 0; i < row; i++) {
                                        modelOrders.addRow(aStrings[i]);
                                }
                        }
                }

        }

    }
```

This method gets as parameter an object and it will add that object to an table according with that type of object is(Client, Product, Orders)

## 6) Results

There were no tests done on this Project

## 7) Conclusion

This homework brought a nice view of how to make a connection to mysql by using java and also how to use reflection. I believe I learned more about this by doing this practical homework than only by reading on the subjects. Further improvements of the code can be done making a cleaner GUI and by "cleaning" the code a little bit.

## 8) Biography

Mostly what was presented in Tema3_HW3_indication at the lab .

https://stackoverflow.com