Order Management System

NAME: Deac Denisa Bianca

CONTENTS TABLE

1. Objective

Main objective

The main objective of this assignment is to design and implement an order management system for processing client orders for a warehouse. The application is designed according to the layered architecture pattern and uses relational database to store the needed information about the clients, products, and orders.

Sub-objectives

To create this application, the requirements of the problem should be analyzed and identified. The design and the implementation should be done based on these aspects.

In order to test the problem, several input examples will be introduced in the graphical user interface and the modified result can be check either in the database or in the table displayed on the window.

2. Problem analysis, modelling, scenarios, use cases

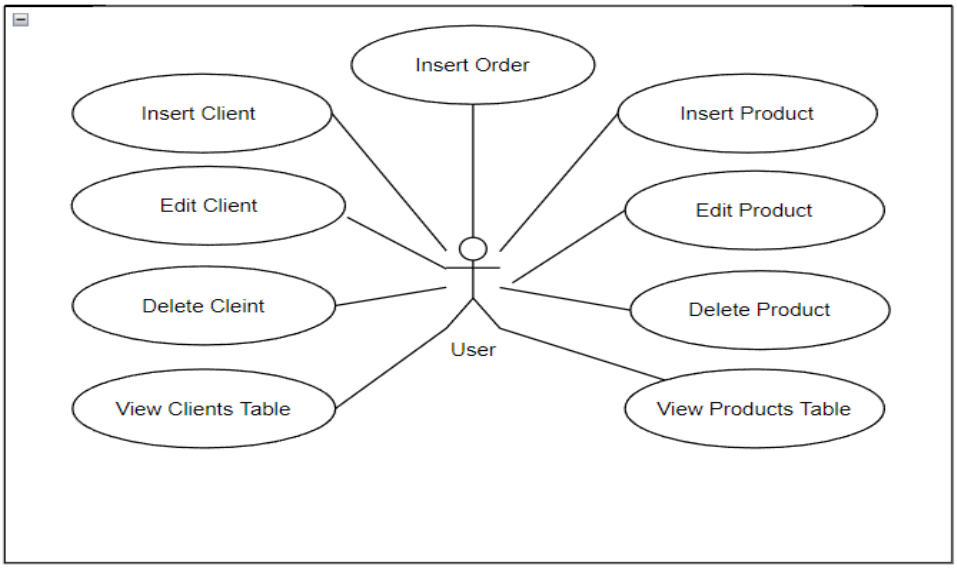
Problem analysis

The order management system should give to the user the possibility to perform a different set of operations on the Clients, Products and Orders on the database. The tables have the options defined as the CRUD operations (read representing visualization): for clients and products, insertions, updates and deletion are the possible operations that can be performed, while for orders only the insertion is possible. Beside them, the visualization of the content of the tables are possible. After each of these operations, the modifications made to the database should be visible.

The operations will be done through a user-friendly graphical interface from which the user will choose the set of operations: for clients, products or orders. After that, depending on his choice, a new window will replace the previous one with the necessary fields to input the data that will be used to perform the requested operation.

Scenarios and use cases

For the Client and Product interface, the options are the same: to insert, delete, update, or visualize clients/products. Only the fields differ since the input data for a client are the name, address and email, while for a product are name, price, and quantity.



FOR CLIENTS

Use Case 1: insert client

Primary Actor: user

Main Success Scenario:

1. The user inserts input data for the: name of the client, address, email
2. The user clicks on the "insert" button
3. The application validates the data and displays a message if the client was successfully introduced
4. The user can click on the "back" button to work on another type of entity or to choose another operation

Alternative Sequence: Empty fields

1. The user left one or many fields empty
2. Dialog error message will be displayed:
   - ➔ For missing name
   - ➔ For missing address
   - ➔ For missing email
3. Exit the dialog error message and returns to step 1 for a successful scenario

Alternative Sequence: Invalid input data

1. The user inserts invalid input for the client's information
2. Dialog error message will be displayed:
   - ➔ For incorrect name: does not contain only alphabetic characters
   - ➔ For incorrect address: does not contain only alphanumerical characters
   - ➔ For incorrect email: does not contain the format @yahoo.com or @gmail.com
3. Exit the dialog error message and returns to step 1 1 for a successful scenario

Use Case 2: edit client

For editing a client, the scenario works the same but has additionally, it also has the "id" field which will be checked the same way to be integer value.

Use Case 3: delete client

Primary Actor: user

Main Success Scenario:

1. The user inserts input data for the id of the client
2. The user clicks on the "delete" button
3. The application validates the data and displays a dialog message if the client was successfully introduced
4. The user can click on the "back" button to work on another type of entity or to choose another operation

Alternative Sequence: Empty fields

1. The user left the id field empty
2. Dialog error message will be displayed:
   - ➔ For missing id
3. Exit the dialog error message and returns to step 1 for a successful scenario

Alternative Sequence: Invalid input data

1. The user inserts invalid input for the client's information
2. Dialog error message will be displayed:
   - ➔ For incorrect id: is not integer value
3. Exit the dialog error message and returns to step 1 1 for a successful scenario

Use case 4: view clients

Main Success Scenario: The user presses the "View" button, and the table will appear below

FOR PRODUCTS

Use Case 1: insert product

Primary Actor: user

Main Success Scenario:

5. The user inserts input data for the: name of the product, quantity, price
6. The user clicks on the "insert" button
7. The application validates the data and displays a message if the product was successfully introduced
8. The user can click on the "back" button to work on another type of entity or to choose another operation

Alternative Sequence: Empty fields

4. The user left one or many fields empty
5. Dialog error message will be displayed:
   - ➔ For missing name
   - ➔ For missing quantity
   - ➔ For missing price
6. Exit the dialog error message and returns to step 1 for a successful scenario

Alternative Sequence: Invalid input data

1. The user inserts invalid input for the client's information
2. Dialog error message will be displayed:
   - ➔ For incorrect name: does not contain only alphabetic characters
   - ➔ For incorrect quantity is not integer
   - ➔ For incorrect price: is not double
3. Exit the dialog error message and returns to step 1 for a successful scenario

Use Case 2: edit product

For editing a product, the scenario works the same but has additionally it also has the "id" field which will be checked the same way to be integer value.

Use Case 3: delete product

Primary Actor: user

Main Success Scenario:

5. The user inserts input data for the id of the product
6. The user clicks on the "delete" button
7. The application validates the data and displays a dialog message if the product was successfully introduced
8. The user can click on the "back" button to work on another type of entity or to choose another operation

Alternative Sequence: Empty fields

4. The user left the id field empty
5. Dialog error message will be displayed:
   - ➔ For missing id
6. Exit the dialog error message and returns to step 1 for a successful scenario

Alternative Sequence: Invalid input data

4. The user inserts invalid input for the client's information
5. Dialog error message will be displayed:
   - ➔ For incorrect id: is not integer value
6. Exit the dialog error message and returns to step 1 for a successful scenario

Use case 4: view products

Main Success Scenario: The user presses the “View” button, and the table will appear below

Functional requirements:

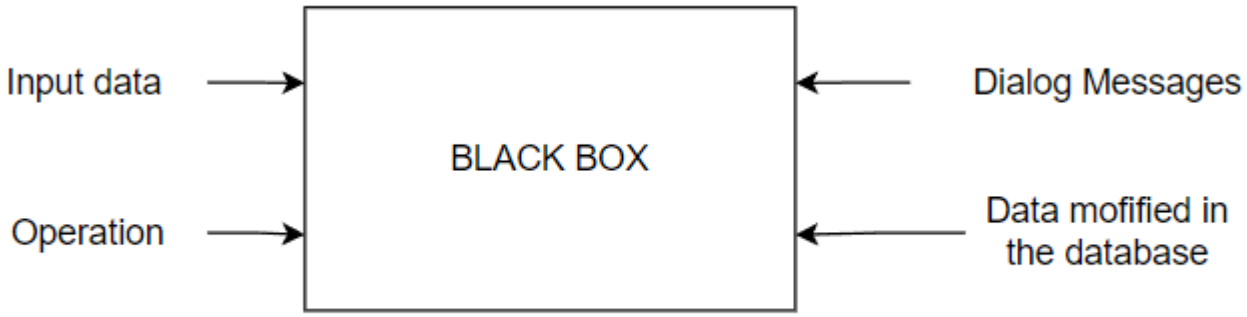
1. The application should allow the user to choose the category of operations
2. The application should allow the user to input data for inserting, editing and deleting and visualizing clients and products
3. The application should allow the user to input data for inserting orders
4. The application should generate a bill for every order created

Non-Functional requirements:

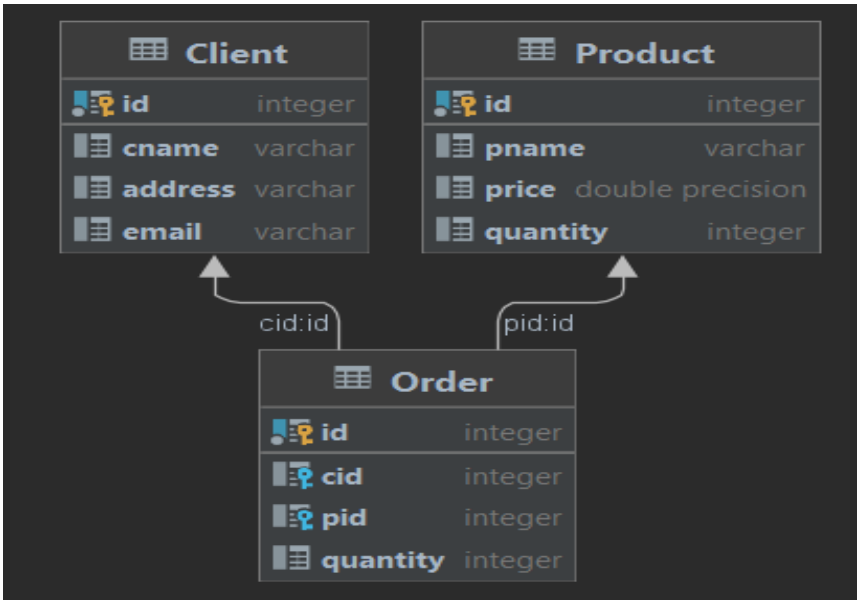
1. The application should be intuitive for the user
2. The application should be easy to use by the user
3. The application should inform the user in case of any errors or input mistakes

3. Design

Overall system design consists in receiving as input an operation to pe perform and some data for the fileds.

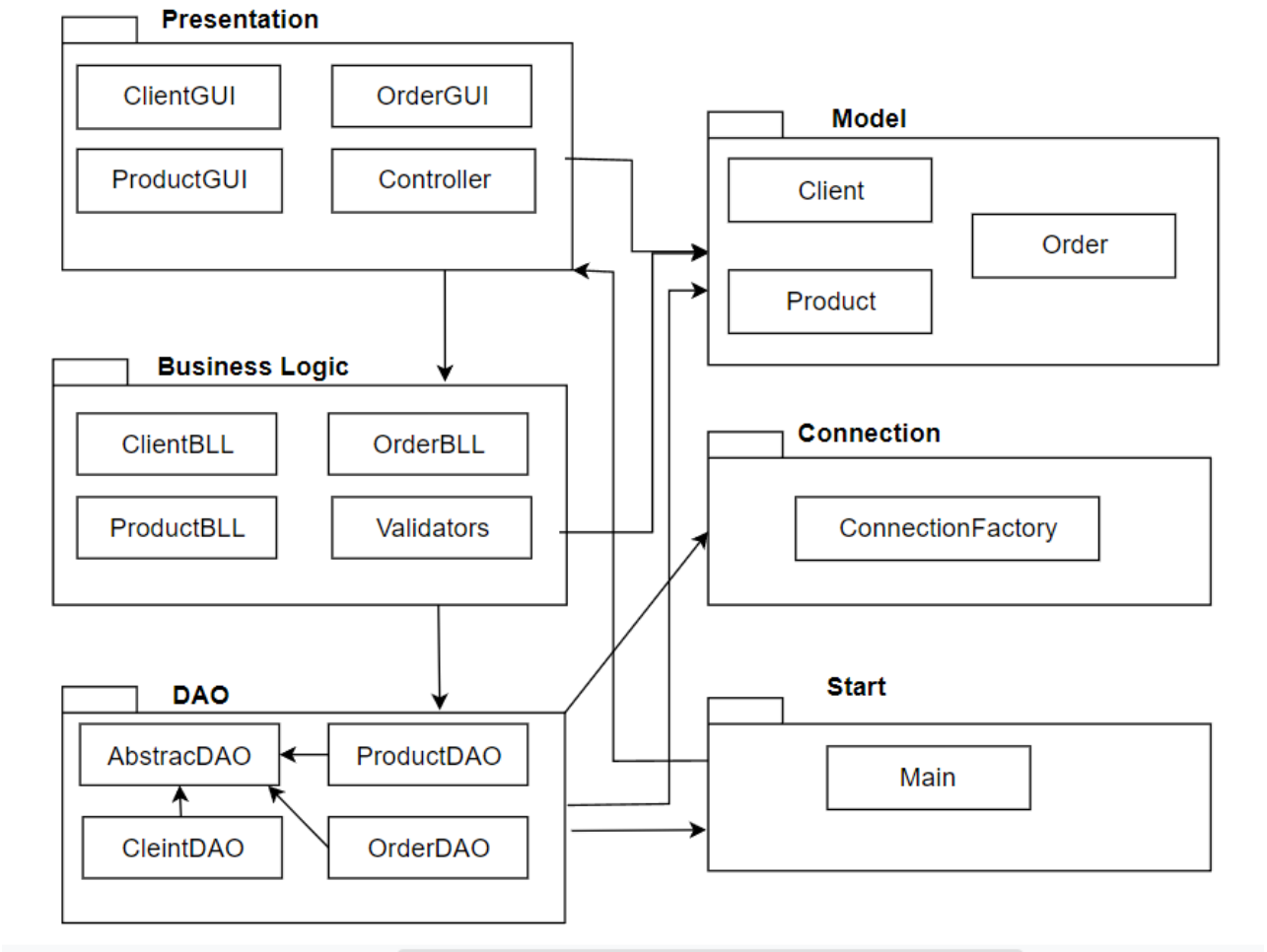


My application uses a relational database, which I designed, to store the data about the clients, products, and orders. All the information is stored in tables which have the same attributes as the entities in the Model package.



Java packages give a lot of help in organizing multiple modules and grouping together related classes and interfaces. My application is designed according to the Layered Architecture Pattern for a better organization. Everything is divided between 6 packages:

- **Model:** contains the classes representing the entities I am working on: Product, Client, and Order
- **Connection**: established the connection with the data base and performs the close operations
- **Presentation**: contains all the graphical interface classes and the controller that manipulates the data from the interface and add the listeners to the buttons
- **DAO**: represents all the data access operations having the methods that perform the CRUD operations
- **BusinessLogic**: contains the classes that validate the data and performs the DAO operations in a logical manner only if all the conditions are true
- **Start**: having the Main class on which the application starts



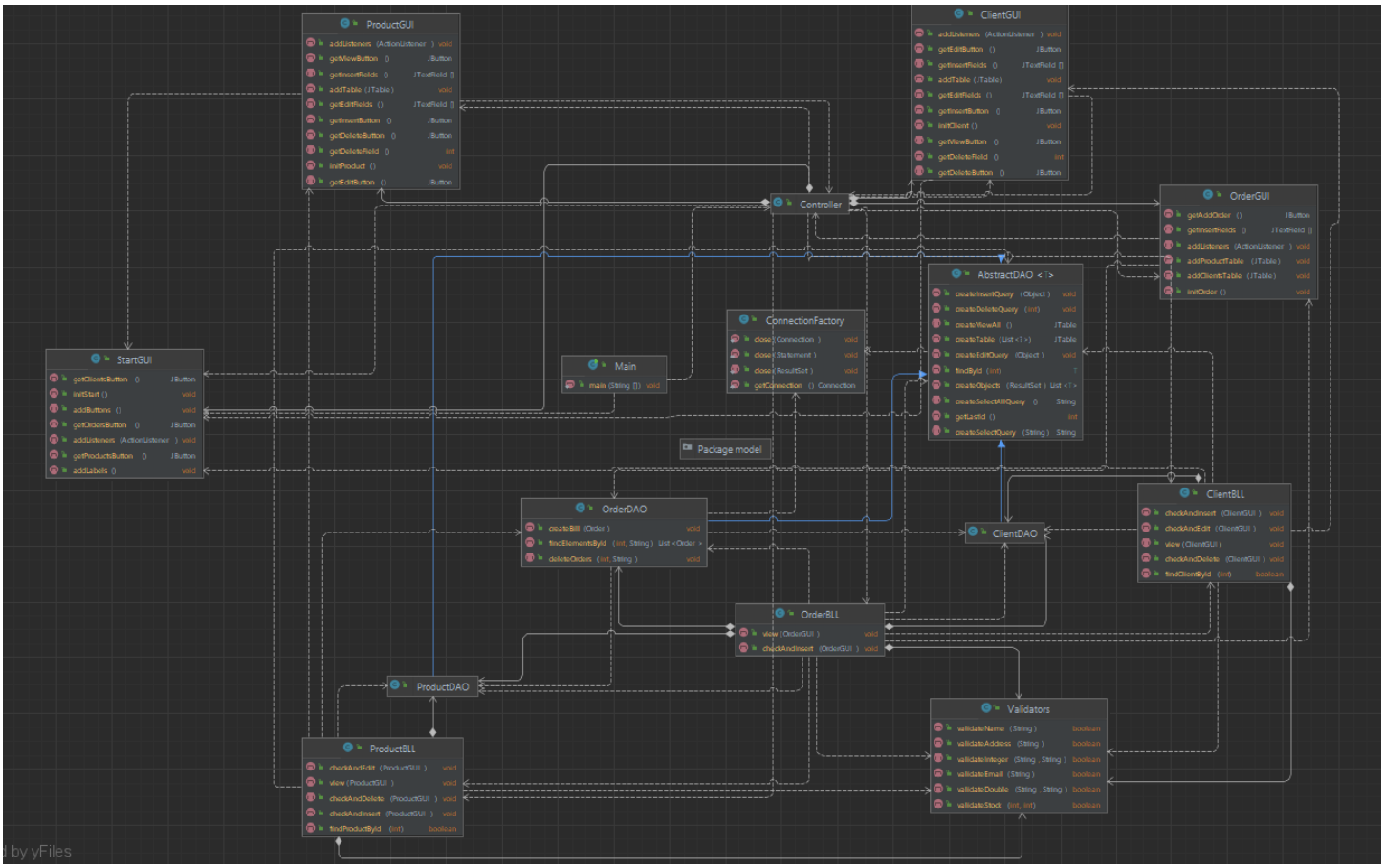
Classes diagram

The concept behind designing this order management system is the following: the user will choose what type of operations he wants to perform: for Clients, Products, or Orders. After that, the input data will be written in the given fields and the operation corresponding to them will be performed by clicking on the respective button.

The following image illustrates the dependencies between classes and shows the methods existent in each of them.

The data structure mainly used is Array List. Reflection Techniques represent an important part of the way my CRUD operations are implemented (further explanation in the Implementation Chapter). Also, an external library is used for creating a pdf, iText, and another one for the database connection to work.

Patter matching is also a used technic that was really helpful for verifying the input data.



7. Implementation

MODEL CLASSES

The management application is based on 3 main entities which are represented in the model package, Client, Product, Order. All 3classes contain attributes, constructors, getters and setters.

```
public class Client {
    private int id;
    private String cname;
    private String address;
    private String email;
```

```
public class Product {
    private int id;
    private String pname;
    private  double price;
    private int quantity;
```

```
public class Order {
    private int id;
    private int cid;
    private int pid;
    private int quantity;
```

CONNECTIO FACTORY

Using reflection, ConnectionFactory class establishes the link between the application and the database. The connection to the DB will be placed in a Singleton object. The class contains methods for creating a connection, getting an active connection, and closing a connection, a Statement or a ResultSet.

```java
public class ConnectionFactory {


    private static final Logger LOGGER = Logger.getLogger(ConnectionFactory.class.getName());
    private static final String DRIVER = "org.postgresql.Driver";
    private static final String DBURL = "jdbc:postgresql://localhost:5432/assignment3";
    private static final String USER = "postgres";
    private static final String PASS = "dontworrybehappy";
    private static ConnectionFactory singleInstance = new ConnectionFactory();
```

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

DATA ACCESS OPERATIONS CLASSES

The main important class in the DAO package si represented by the AbstractDAO which is a generic class that implements all the CRUD operations by creating querys and executing them. The constructor of the class obtains the object of the generic type T. The data is manipulated through reflection tehniques and the connection with the database is done by using the class mentioned before, ConnectionFactory.

This class contains methods for: finding an element by its index, selecting elements depdnding on a given field, returining all the data from the table, getting the next available id, creating object from resultSet. Beside them, the most important operations are implemented here, the CRUD ones: insert(create) object, view all (read), edit(update), delete.

```java
    public int getLastId(){
        StringBuilder query = new StringBuilder();
        String table = type.getSimpleName();
        Field field = type.getDeclaredFields()[0];
        field.setAccessible(true);

        query.append("select MAX(");
        query.append(field.getName()+") as id from \""+ table + "\"");

        Connection connection = null;
        Statement statement = null;
        ResultSet resultSet = null;
        try {
            connection = ConnectionFactory.getConnection();
            statement = connection.createStatement();
            resultSet = statement.executeQuery(String.valueOf(query));

            while(resultSet.next()) {
                int index = resultSet.getInt( columnLabel: "id");
                return index+1;
            }

        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            ConnectionFactory.close(resultSet);
            ConnectionFactory.close(statement);
            ConnectionFactory.close(connection);
        }
        return 0;
    }
```

```java
public JTable createViewAll(){
    Connection con = ConnectionFactory.getConnection();
    String query = createSelectAllQuery();
    Statement statement = null;
    ResultSet resultSet = null;
    List<T> objList = new ArrayList<T>();

    try {
        statement = con.createStatement();
        resultSet = statement.executeQuery(query);
        objList = createObjects(resultSet);

        return createTable(objList);
    } catch (SQLException e) {
        e.printStackTrace();
    } finally {
        ConnectionFactory.close(resultSet);
        ConnectionFactory.close(statement);
        ConnectionFactory.close(con);
    }
    return null;
}
```

The other classes in this package are extending the AbstractDAO. ClientDAO and ProductDAO are only using the methods inherited from the generic class. Only OrderDAO contains also another specific methods: createBill, findElementsByID and deleteOrders.

CreateBill creates for each order a pdf with the details of the order, total price, information about the client. It uses iText library for creating the pdf.

```
bill.append("Order number "+ order.getId()+"\n\nCLIENT DETAILS:\n");
bill.append("Name: "+client.getCname()+"\n"+"Address: "+client.getAddress()+"\nEmail: "+client.getEmail()+"\n\n");
bill.append("ORDER DETAILS:\n");
bill.append("Product: "+product.getPname()+"\nProduct Price: "+ product.getPrice()+"\nQuantity: "+order.getQuantity());
bill.append("\n\nTotal price: "+product.getPrice()*order.getQuantity());

try {
    String billName = "bill"+order.getId()+".pdf";

    Document document = new Document();
    PdfWriter.getInstance(document, new FileOutputStream(billName));

    document.open();

    document.add(new Paragraph( string: "Order number "+ order.getId()+"\n\nCLIENT DETAILS:\n"));
    document.add(new Paragraph( string: "Name: "+client.getCname()+"\n"+"Address: "+client.getAddress()+"\nEmail: "+client.getEmail()+"\n\n"));
    document.add(new Paragraph( string: "ORDER DETAILS:\n"));
    document.add(new Paragraph( string: "Product: "+product.getPname()+"\nProduct Price: "+ product.getPrice()+"\nQuantity: "+order.getQuantity
    document.add(new Paragraph( string: "\n\nTotal price: "+product.getPrice()*order.getQuantity() ));
    document.close();
```

The method deleteOrders will delete all the orders in which a given client appears if the client is deleted. It uses the other method, findElementsById which returns the list with all the orders of that type.

```
public void deleteOrders( int cid, String field){
    List<Order> orders = findElementsById( cid, field);

    if(orders != null) {
       for (Order o : orders) {
          createDeleteQuery(o.getId());
       }
    }
}
```

BUSINESS LOGIC CLASSES

The package business logic contains the classes that perform the logical operations: it executed the CRUD operations if and only if all the input data is validated.

The class Validators contain all the methods used to verify is a certain string has the necessary form to be the requested attribute. There are methods for checking if a string is integer, double, name (only alphabetic characters), address (alphanumeric) or email (contains @yahoo.com or @gmail.com). They use pattern matching.

```java
public boolean validateInteger( String nr, String  categ){
    if(nr .equals("")){
        JOptionPane.showMessageDialog( parentComponent: null,  message: categ+" missing",  title: "INTEGER ERROR", JOptionPane.ERROR_MESSAGE);
        return false;
    }
    if(!nr.matches( regex: "[0-9]+")){
        JOptionPane.showMessageDialog( parentComponent: null,  message: categ+" is not integer",  title: "INTEGER ERROR", JOptionPane.ERROR_MESSAGE);
        return false;
    }
    return true;
}
```

```java
public boolean validateAddress(String address){
    if (address .equals("")) {
        JOptionPane.showMessageDialog( parentComponent: null,  message: "Address missing",  title: "ADDRESS ERROR", JOptionPane.ERROR_MESSAGE);
        return false;
    }
    if (!address.matches( regex: "[a-zA-Z0-9\s]+") ){
        JOptionPane.showMessageDialog( parentComponent: null,  message: "Not a valid address",  title: "ADDRESS ERROR", JOptionPane.ERROR_MESSAGE);
        return false;
    }
    return true;
}
```

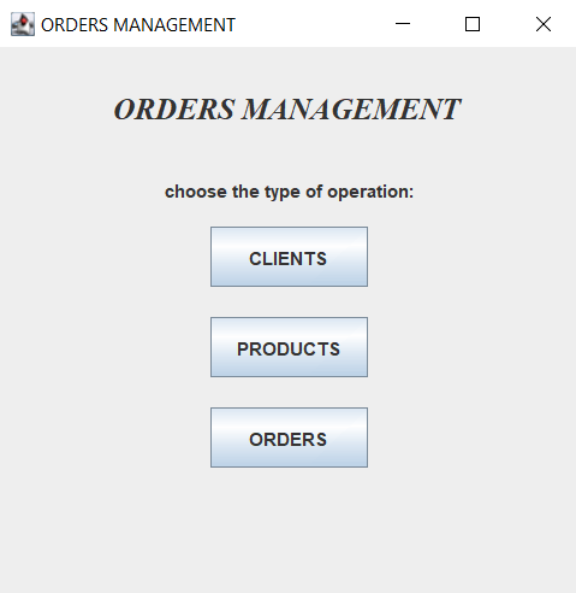
The other 3 classes, CLientBLL, ProductBLL and OrderBLL take input data from GUI, validates it and then do the CRUD operations.

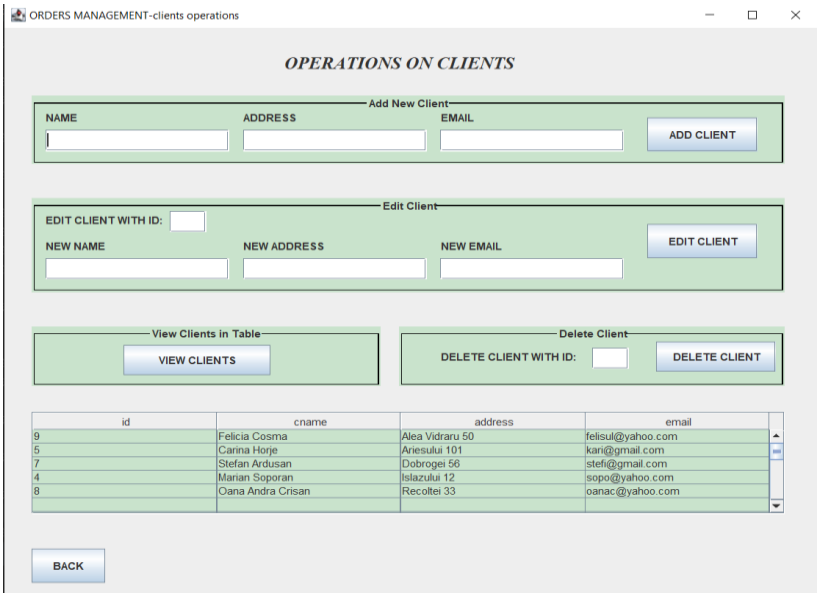
```java
public void checkAndDelete(ClientGUI clientGUI){
    int id = clientGUI.getDeleteField();
    if(findClientById(id)) {
        //delete all orders made by this client
        StringBuilder sb = new StringBuilder();
        sb.append("SELECT * FROM \"Order\" where cid ="+ id);

        OrderDAO orderDAO = new OrderDAO();
        orderDAO.deleteOrders(id,  field: "cid");
        clientDAO.createDeleteQuery(id);
    }
}
```

PRESENTER CLASSES

In the presenter package are implemented all the GUI classes, one class for each window: Start, Client, Product, Order. All the buttons, fields and lables are added here.

The graphical user interface is intuitive and uses dialog messages for succesful case and also for errors.



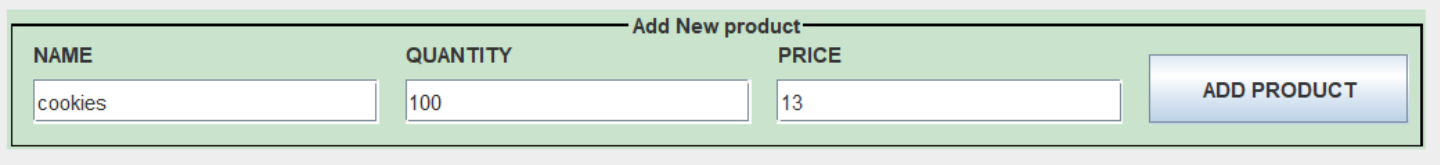


The Controller is also integrated in this package since the action listeners for the buttons are added from there. For the operation buttons, the BLLs are instantiated for the methods corespondent to each button to be executed.

```
private void addListenerProduct() {

    if (okProduct == 1) {

        productGUI.addListeners(e -> {
            ProductBLL productBLL = new ProductBLL();
            if (e.getSource() == productGUI.getInsertButton()) {
                productBLL.checkAndInsert(productGUI);
            }
            else if (e.getSource() == productGUI.getEditButton()) {
                productBLL.checkAndEdit(productGUI);
            } else if (e.getSource() == productGUI.getDeleteButton())
               productBLL.checkAndDelete(productGUI);
            } else if (e.getSource() == productGUI.getViewButton()) {
               productBLL.view(productGUI);
            }
        });
    }
}
```

8. Results

To check the functionality of the application, the user interface is used, and a set of data can be inserted.



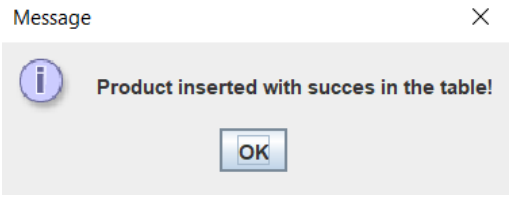


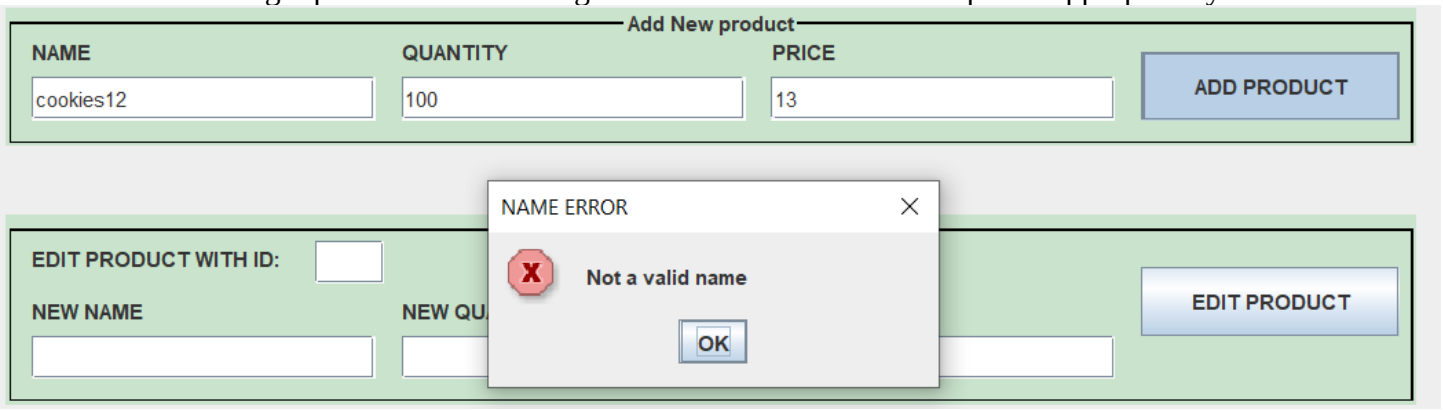
Table before insertion:

id	pname	price	quantity
11	book	20.0	10
4	drops	20.0	50
12	eyeliner	9.989999771118164	80
10	pen	12.5	50
9	iphone	1000.0	18

Table after insertion is shown bellow. As it can be seen, the data was modified in the database.

id	pname	price	quantity
11	book	20.0	10
4	drops	20.0	50
12	eyeliner	9.989999771118164	80
10	pen	12.5	50
9	iphone	1000.0	18
13	cookies	13.0	100

Wrong input data can also be given to see if the interface responds appropriately.



9. Conclusion

This project was interesting, a good exercise for a future job on this domain. I find the interaction between the program and the user challenging but a pleasant way of coding and working.

After this assignment I learned about Reflection Techniques which proved to be extremely useful in this kind of work. Beside this, databases integrated din this kind of management systems are crucial and applicable in many kinds of application. I was impressed by the applicability of these tools and I will fore sure use them in the future.

10. Bibliography

- https://www.jrebel.com/blog/java-regular-expressions-cheat-sheet
- https://www.baeldung.com/java-pdf-creation
- https://docs.oracle.com/javase/7/docs/api/java/sql/ResultSet.html