DOCUMENTATION

ASSIGNMENT NUMBER 3

ORDERS MANAGEMENT

Jimborean Oana Elena

Group 30422

CONTENTS

1. Objectives. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .3
2. Analysis. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .. .3
3. Design. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .6
4. Implementation. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .9
5. Results. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .11
6. Conclusions. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .. .12
7. Bibliography. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .13
8. Objectives

•Main objective:

Design and implement an application for managing the client orders for a warehouse.

•Sub-objectives:

The application uses (minimally) the following classes:

-Model classes - represent the data models of the application (for example Order, Client, Product)

-Business Logic classes - contain the application logic

-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.

1. Problem analysis, modeling, scenarios, use cases

* General overview

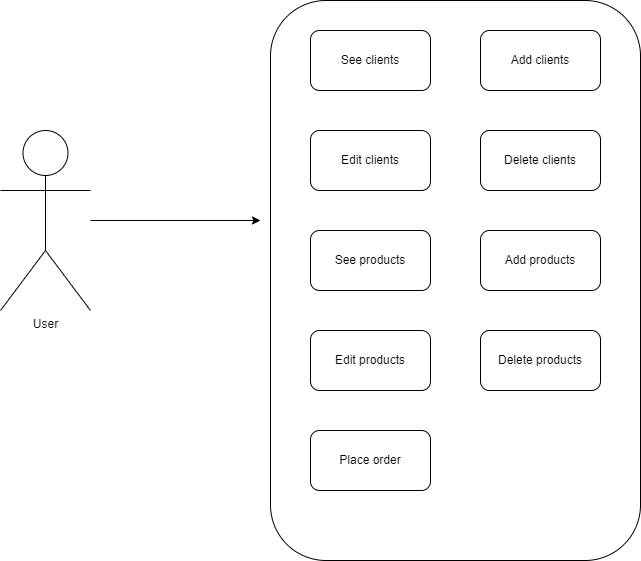
This application should be able to fulfil all the requirements in order to display, modify, and keep track of orders, clients and products. These are stored in a relational MySQL database, along with the information about the users which have access to the system. This way, all the data is easier to retrieve and access from different computers.

* Input and Output

When talking about the input in the application, the user can choose to manage 3 tables, Client, Product and Order. All tables have the options to Add (insert entry), Edit (modify entry) and Delete, and the user can introduce the values in the specific fields for any of the three different tables.

For example, for the Client table the user can introduce the name of the client, the address, phone number, the Email address. All fields have to obey the standard rules such as phone number has to have 10 digits (only digits), the name of the client can contain only letters and spaces, the email address has to contain the @ and the . symbols.

* Use cases



a) **Use Case**: See clients

**Primary Actor**: user

**Main Success Scenario**:

1.The user clicks on the “CLIENTS” button in the graphical user interface and sees the clients in a table.

b) **Use Case**: Edit clients

**Primary Actor**: user

**Main Success Scenario**:

1. The user clicks on the “CLIENTS” button in the graphical user interface and sees the clients in a table.

2. The user clicks on a row in the table to select a client.

3. The user clicks on the “EDIT CLIENT” button.

4. The user inserts new data into the fields.

5. The user clicks on the “SAVE” button.

Alternative Sequence 1: Client not selected

1. The user clicks on the “CLIENTS” button in the graphical user interface and sees the clients in a table.

2. The user clicks on the “EDIT CLIENT” button.

3. The application displays an error message.

4. The scenario returns to step 1.

Alternative Sequence 2: Incorrect data

1. The user clicks on the “CLIENTS” button in the graphical user interface and sees the clients in a table.

2. The user clicks on a row in the table to select a client.

3. The user clicks on the “EDIT CLIENT” button.

4. The user inserts incorrect data.

5. The user clicks on the “SAVE” button.

6. The application displays an error message.

7. The scenario returns to step 1.

c**) Use Case**: Delete clients

**Primary Actor**: user

**Main Success Scenario**:

1. The user clicks on the “CLIENTS” button in the graphical user interface and sees the clients in a table.

2. The user clicks on a row in the table to select a client.

3. The user clicks on the “DELETE CLIENT” button.

Alternative Sequence 1: Client not selected

1. The user clicks on the “CLIENTS” button in the graphical user interface and sees the clients in a table.

2. The user clicks on the “DELETE CLIENT” button.

3. The application displays an error message.

4. The scenario returns to step 1.

Alternative Sequence 2: The selected client has already placed an order

1. The user clicks on the “CLIENTS” button in the graphical user interface and sees the clients in a table.

2. The user clicks on a row in the table to select a client.

3. The user clicks on the “DELETE CLIENT” button

4. The application displays an error message.

5. The scenario returns to step 1.

d)**Use Case**: Add clients

**Primary Actor**: user

**Main Success Scenario**:

1. The user clicks on the “CLIENTS” button in the graphical user interface and sees the clients in a table.

2. The user clicks on the “ADD CLIENT” button.

3. The user inserts new data into the fields.

4. The user clicks on the “SAVE” button.

Alternative Sequence 1: Incorrect data

1. The user clicks on the “CLIENTS” button in the graphical user interface and sees the clients in a table.

2. The user clicks on the “ADD CLIENT” button.

3. The user inserts incorrect data.

4. The user clicks on the “SAVE” button.

5. The application displays an error message.

6. The scenario returns to step 1.

e) **Use Case**: See products

**Primary Actor**: user

**Main Success Scenario**:

1.The user clicks on the “PRODUCTS” button in the graphical user interface and sees the products in a table.

f) **Use Case**: Edit products

**Primary Actor**: user

**Main Success Scenario**:

1. The user clicks on the “PRODUCTS” button in the graphical user interface and sees the products in a table.

2. The user clicks on a row in the table to select a product.

3. The user clicks on the “EDIT PRODUCT” button.

4. The user inserts new data into the fields.

5. The user clicks on the “SAVE” button.

Alternative Sequence 1: Product not selected

1. The user clicks on the “PRODUCTS” button in the graphical user interface and sees the products in a table.

2. The user clicks on the “EDIT PRODUCT” button.

3. The application displays an error message.

4. The scenario returns to step 1.

Alternative Sequence 2: Incorrect data

1. The user clicks on the “PRODUCTS” button in the graphical user interface and

sees the products in a table.

2. The user clicks on a row in the table to select a product.

3. The user clicks on the “EDIT PRODUCT” button.

4. The user inserts incorrect data.

5. The user clicks on the “SAVE” button.

6. The application displays an error message.

7. The scenario returns to step 1.

g) **Use Case**: Delete products

**Primary Actor**: user

**Main Success Scenario**:

1. The user clicks on the “PRODUCTS” button in the graphical user interface and sees the products in a table.

2. The user clicks on a row in the table to select a product.

3. The user clicks on the “DELETE PRODUCT” button.

Alternative Sequence 1: Product not selected

1. The user clicks on the “PRODUCTS” button in the graphical user interface and

sees the products in a table.

2. The user clicks on the “DELETE PRODUCT” button.

3. The application displays an error message.

4. The scenario returns to step 1.

Alternative Sequence 2: An order was already placed having that product.

1. The user clicks on the “PRODUCTS” button in the graphical user interface and sees the products in a table.

2. The user clicks on a row in the table to select a product.

3. The user clicks on the “DELETE PRODUCT” button.

4. The application displays an error message.

5. The scenario returns to step 1.

h**) Use Case**: Add products

**Primary Actor**: user

**Main Success Scenario**:

1. The user clicks on the “PRODUCTS” button in the graphical user interface and sees the products in a table.

2. The user clicks on the “ADD PRODUCT” button.

3. The user inserts new data into the fields.

4. The user clicks on the “SAVE” button.

Alternative Sequence: Incorrect data

1. The user clicks on the “PRODUCTS” button in the graphical user interface and sees the products in a table.

2. The user clicks on the “ADD PRODUCT” button.

3. The user inserts incorrect data.

4.The user clicks on the “SAVE” button.

5.The application displays an error message.

6.The scenario returns to step 1.

i) **Use Case**: Place order

**Primary Actor**: user

**Main Success Scenario**:

1. The user clicks on the “ORDERS” button in the graphical user interface.

2. The user clicks on the combo box to select a client.

3. The user clicks on the combo box to select a product.

4. The user inserts a quantity.

5. The user clicks on the “ADD ITEM” button.

6. The user clicks on the “PLACE ORDER” button.

Alternative Sequence: Incorrect quantity format or quantity larger than stock

1. The user clicks on the “ORDERS” button in the graphical user interface.

2. The user clicks on the combo box to select a client.

3. The user clicks on the combo box to select a product.

4. The user inserts a wrong quantity.

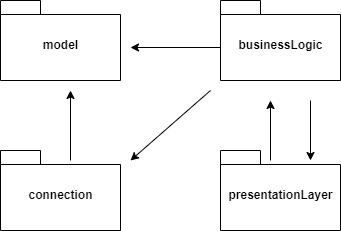
5. The user clicks on the “ADD ITEM” button.

6. The application displays an error message.

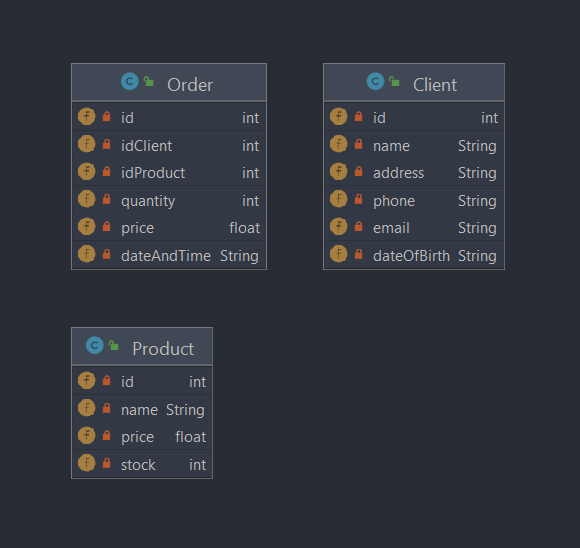
7. The scenario returns to step 1.

1. Design

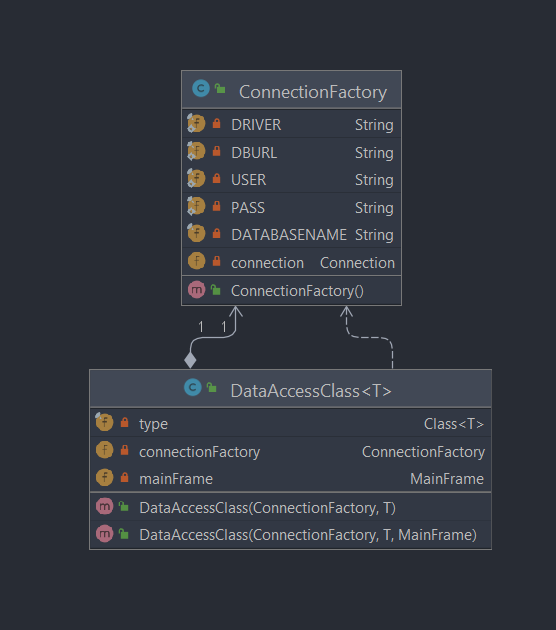
* Package Diagram



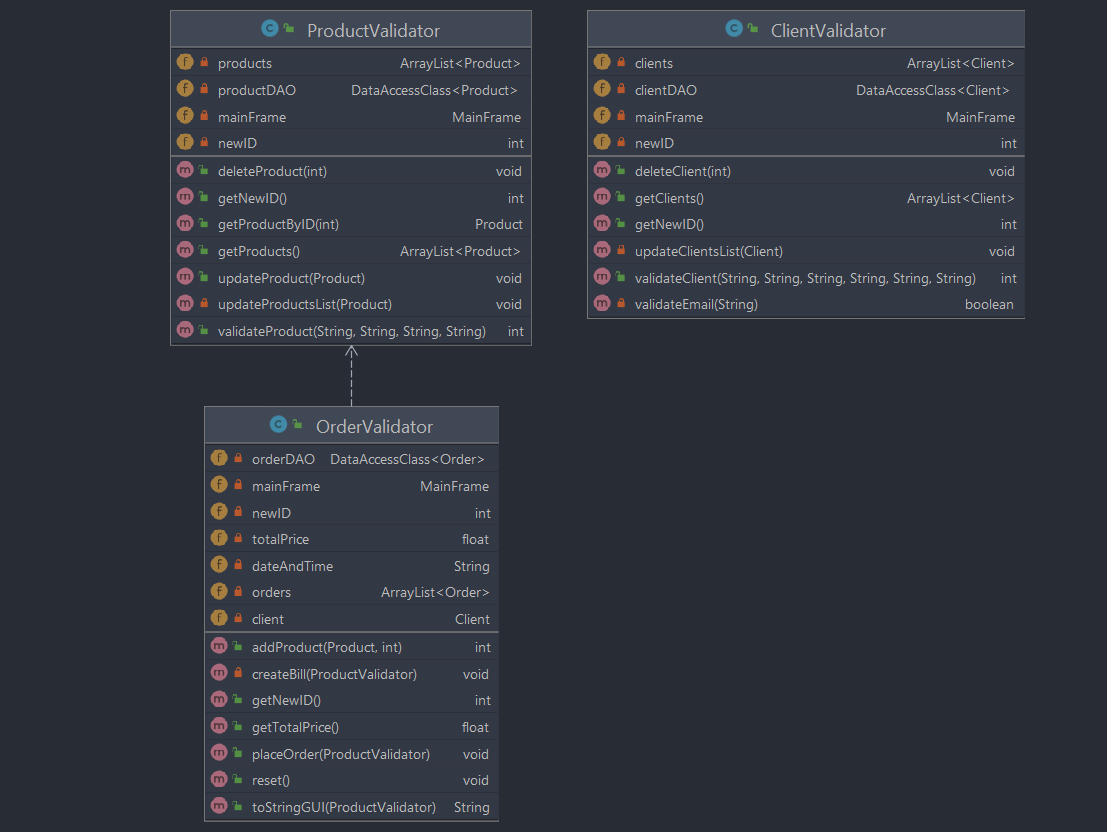
* UML diagram for classes from the Model package:



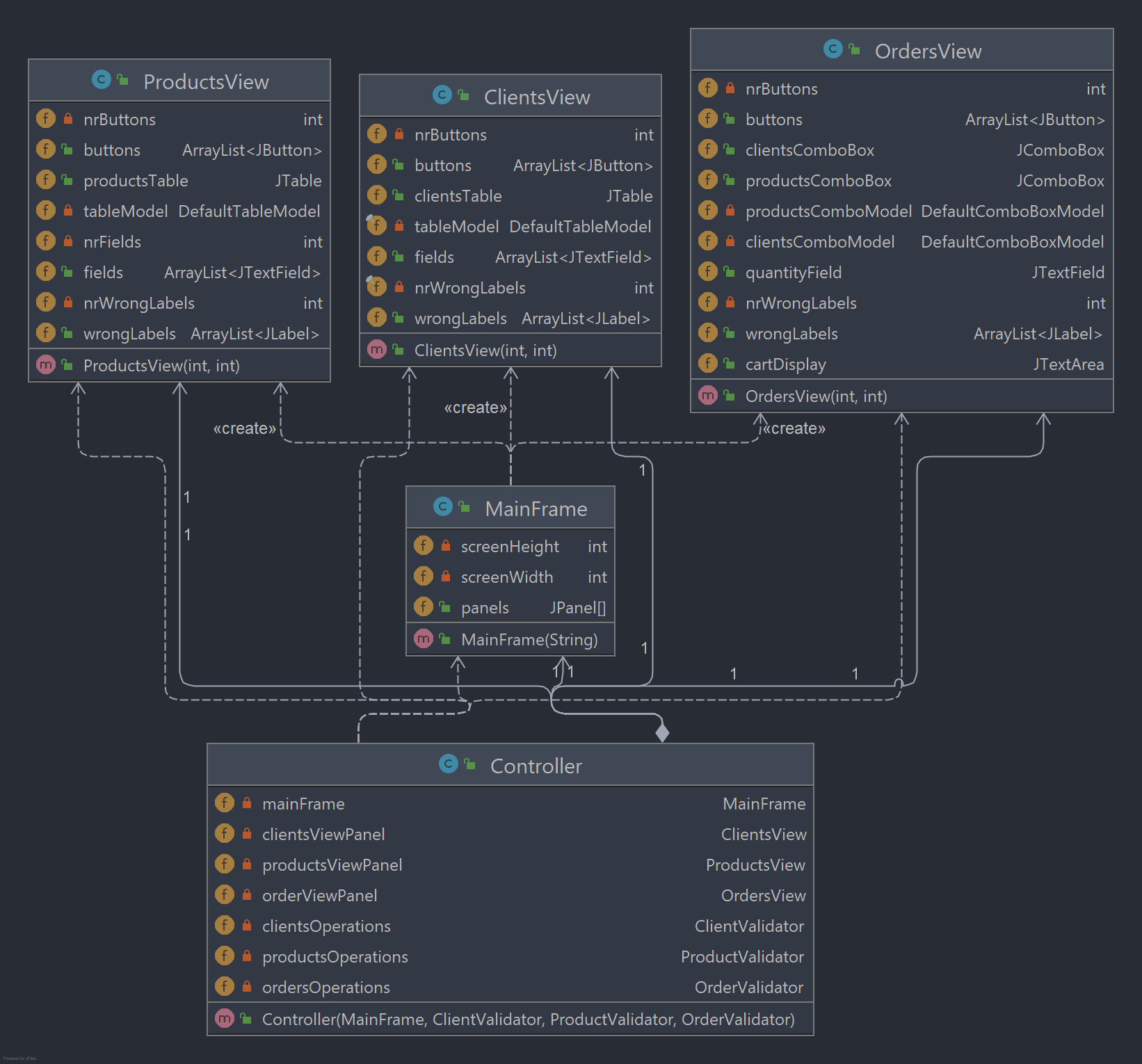
* UML diagram for classes from the Connection package:



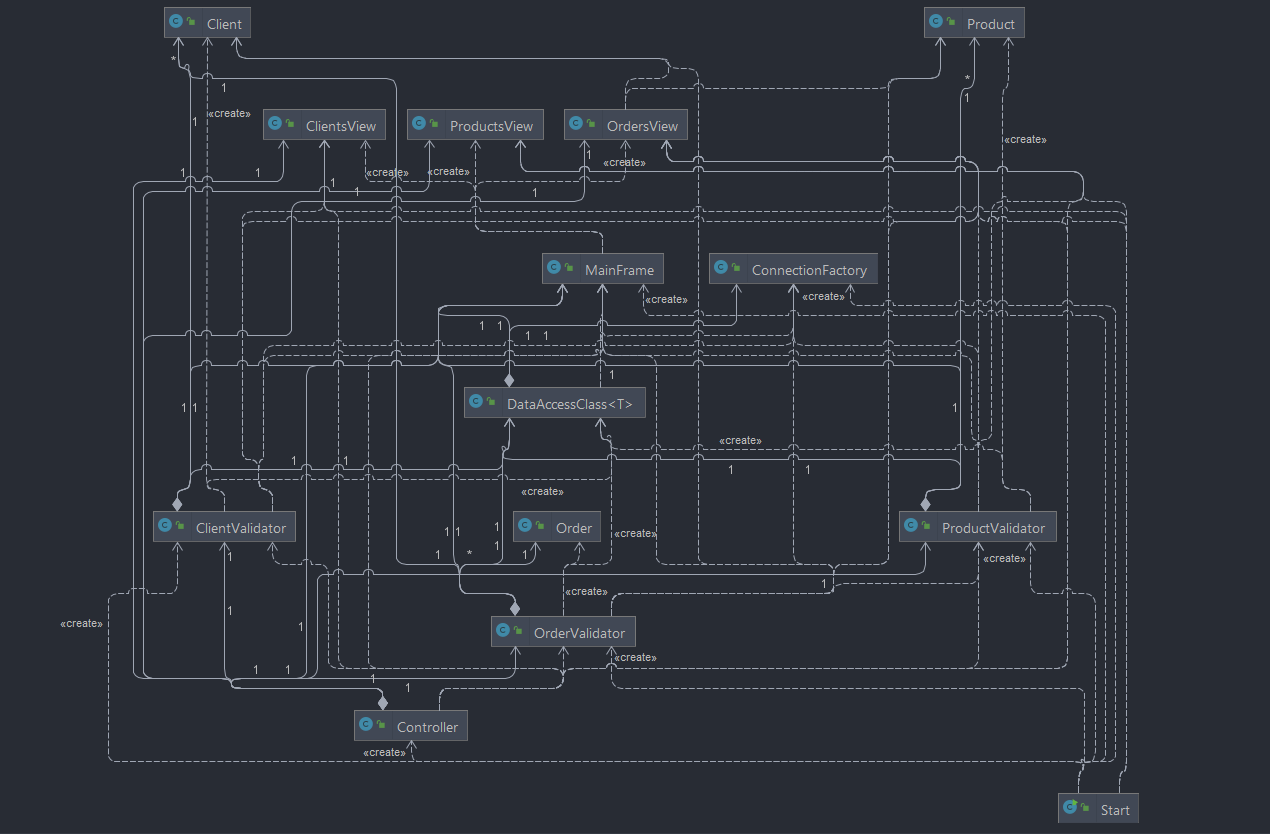
* UML diagram for classes from the BusinessLogic package:



* UML diagram for classes from the presentationLayer package:



* Class diagram:



1. Implementation

The application is divided into 4 packages: Model, businessLogic connection and presentationLayer.

1. Description of the Model package

*a.1. Client Class:* it is used to hold client’s data and it is mapped to a dataBase table. The toString method is used for viewing a client in the drop box when making an order, and the toStringBill method is used when printing the order information to the bill text file.

*a.2. Product Class:* Class Product is used to hold product’s data and it is mapped to a dataBase table.

*a.3. Order Class:* Class Order is used to hold order’s data and it is mapped to a dataBase table.

1. Description of the Connection package

*b.1 ConnectionFactory Class:* It is used for creating the connection with the database. The “CloseConnection” method attempts to close the established connection with the database. It returns 0 in case of success and -1 if an error occurred.

*b.2. DataAccessClass Class:* The class DataAccessClass is a generic class that uses reflection to create SQL queries for accessing the database.

In constructor, parameter connectionFactory is the object holding the connection to the database and parameter instance is an instance of class T used to store its type.

The “findAll” method returns an arraylist with all the entries in the table corresponding to class T. The “update” method attempts to update an entry in the database table which corresponds to the class T; parameter instance is the object whose data will be written in the table.

The “insert” method attempts to add an entry in the database table which corresponds to the class T.

The “deleteById” method attempts to delete an entry in the database table which corresponds to the class T.

The “findById”(public T findByID(int id)) method searches for an entry having the specified id in the table which corresponds to the class T. It returns object of type T if id is found, or null if not.

1. Description of the BusinessLogic package

*c.1 ClientValidator class:* it performs the operations on clients. It holds the clients in the clients’ field and uses clientDAO to communicate with the database. Field newID is the id that will be assigned to the next client that will be added. When changes are made, the clients table, database, and clients combo box are updated. If the input is correct, it edits/inserts the new client and makes the neccessary updates in the GUI and in the database, using the “updateClientList” method.

*c.2. ProductValidaotr Class:* it performs the operations on products. It holds the products in the products’ field and uses productDAO to communicate with the database. Field newID is the id that will be assigned to the next product that will be added.

*c.3. OrderValidator Class:* The “addProduct: method attempts to add a new product to the order. It returns 0 if the product was successfully added to the order, -1 if the quantity is larger than the available stock.

The “placeOrder” method is used when the user attempts to place an order. It sets the idClient and the dateAndTime of the orders, reduces the stock of the products, updates the database, creates the newID for the next order and creates a bill.

1. Description of the PresentationLayer Class

*d.1. ClientsView Class:* It extends JFrame. The “updateTable” method receives a list of objects and creates the header of the table by accessing the fields of the elements of data through reflection, and then adds entries in the table.

*d.2. ProductsView Class:* same as ClientsView class, but with products.

*d.3. OrdersView Class:* Class extends JPanel. The field cartDisplay is the place where the items that were already selected will apprear.

*d.4. Controller Class:* The “ButtonsListenerClients” Class implements ActionListener interface and handles user's request concerning clients, received from mainFrame's ClientsView panel. It calls methods of the clientValidator object and displays the appropriate messages into the GUI.

The “ButtonsListenerProducts” Class implements ActionListener interface and handles user's request concerning products, received from mainFrame's ProductsView panel. It calls methods of the productValidator object and displays the appropriate messages into the GUI.

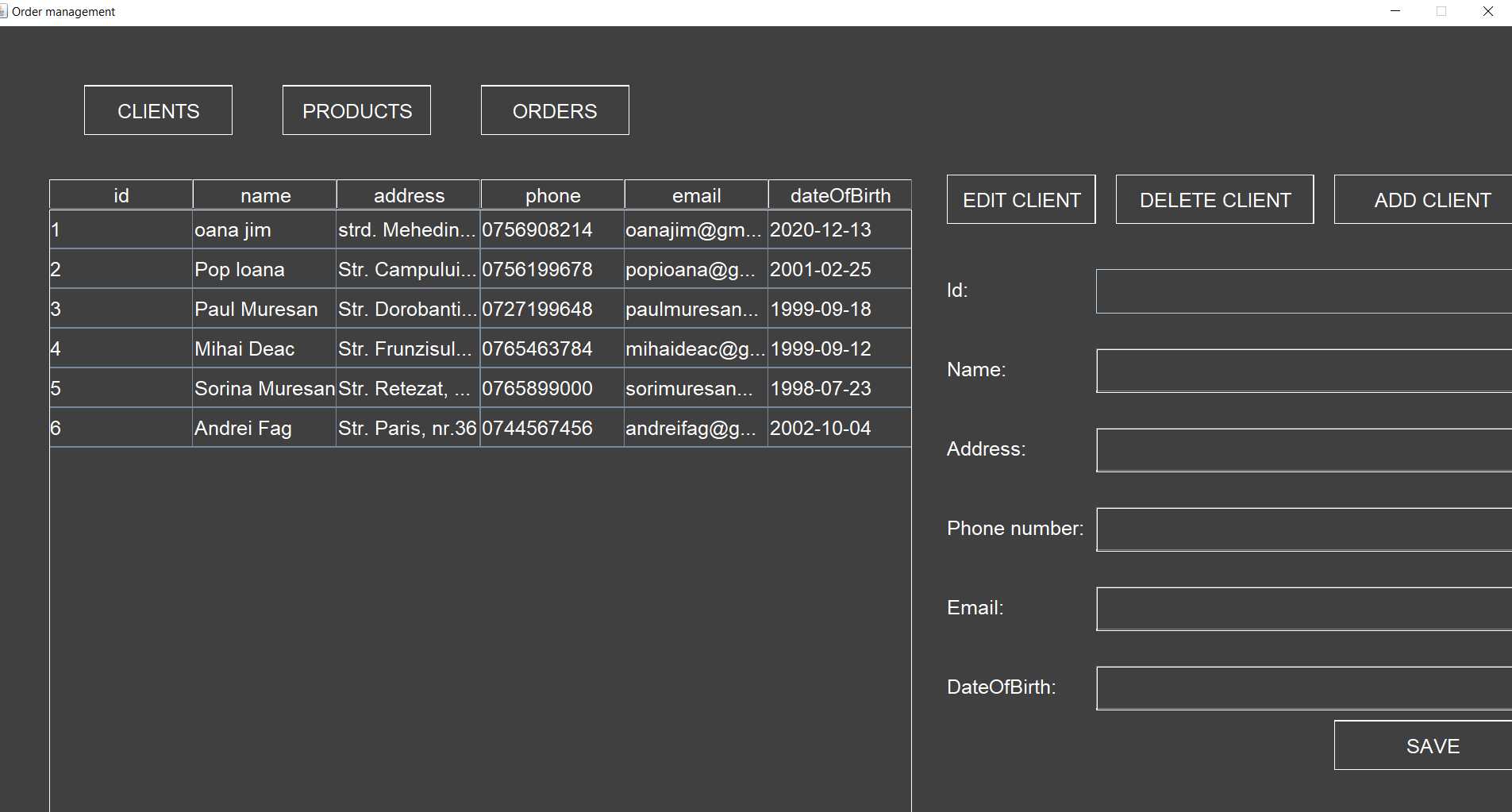
The “ButtonsListenerOrders” Class implements ActionListener interface and handles user's request concerning orders, received from mainFrame's OrdersView panel. It calls methods of the orderValidator object and displays the appropriate messages into the GUI.

The “ButtonsListenerOrders” Class implements ActionListener interface and handles user's request concerning orders, received from mainFrame's OrdersView panel. It calls methods of the orderValidator object and displays the appropriate messages into the GUI.

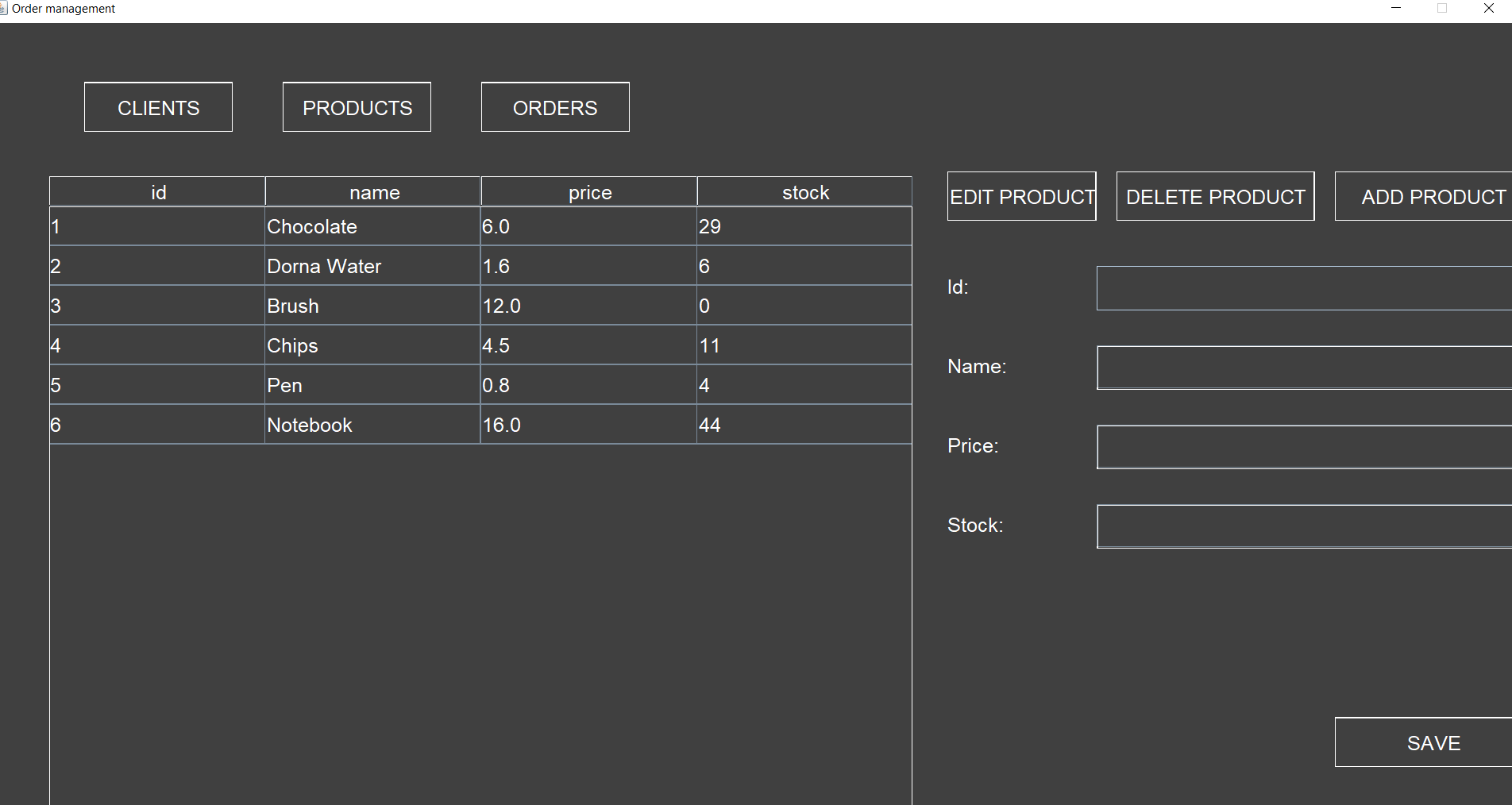
*d.5. MainFrame Class:* It extends JFrame. It Is the main window of the app and its contents change. It creates its 3 panels: ClientsView, ProductsView and OrdersView.

1. Results

GUI interface for Clients:

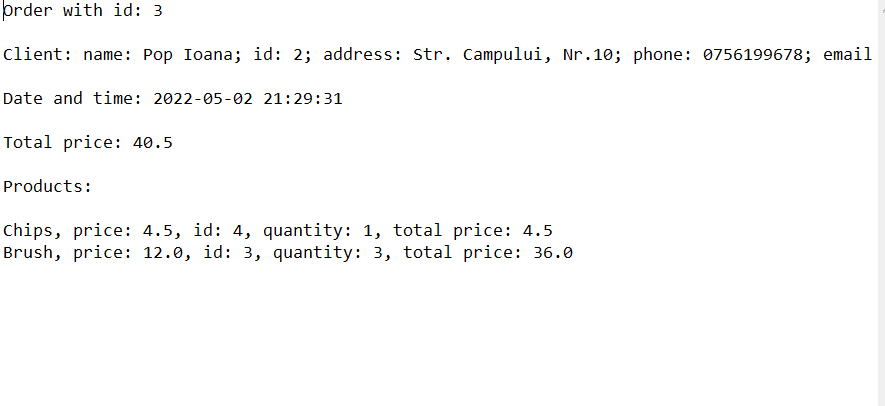


GUI interface for Products:



GUI interface for Orders:



The text file for the Bill:

1. Conclusions

What I have learned

- How to use maven dependency to make the connection to a database.

- How to use the Layered Architectural pattern.

- How to write a generic class.

- How to use reflection.

1. Bibliography

[https://app.diagrams.net/#](https://app.diagrams.net/)

<https://gitlab.com/utcn_dsrl/pt-layered-architecture>

<https://gitlab.com/utcn_dsrl/pt-reflection-example>

<https://stackoverflow.com/questions/27696238/sql-select-name-by-id/27696377?fbclid=IwAR1ZDDsAQPfwjlI_3MIi0_JfGhYLCPgJPhsOkUoNlxdp28QGmek20BzqfiM>