

**ORDER MANAGEMENT SYSTEM**

Student: Francesca Dițulescu

Teacher: Viorica Chifu

Table of Contents

Assignment Objective . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3

Main Objective . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3

Sub-objectives . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .3

Problem analysis, modeling, scenarios, use cases . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .3

Problem analysis . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .3

Solution . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .3

Requirements . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3

Use cases . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4

Design . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .8

Overall System Design . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8

Division into packages . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10

Division into classes . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .10

UML Class Diagram . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .11

GUI Design . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .12

Implementation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 15

Conclusions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .17

Bibliography . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .17

ASSIGNMENT OBJECTIVE

MAIN OBJECTIVE

The main objective of this project is to design and implement an order management application with a dedicated graphical user interface through which the user can perform different operations (create, view, update, delete) on 3 different categories – clients, products and orders.

SUB-OBJECTIVES

* Analyze the problem and identify the requirements
* Design the management system application
* Implement the management system application
* Test the management system application

PROBLEM ANALYSIS

The purpose of this project is to solve the following problem: keeping track of logistics on paper is really difficult, especially when you have to update the information (on clients, products and orders).

SOLUTION

As a solution, this project provides a faster and more interactive application that can help the user (the owner of the warehouse) to manage its clients, products and orders easier.

REQUIREMENTS

Functional requirements

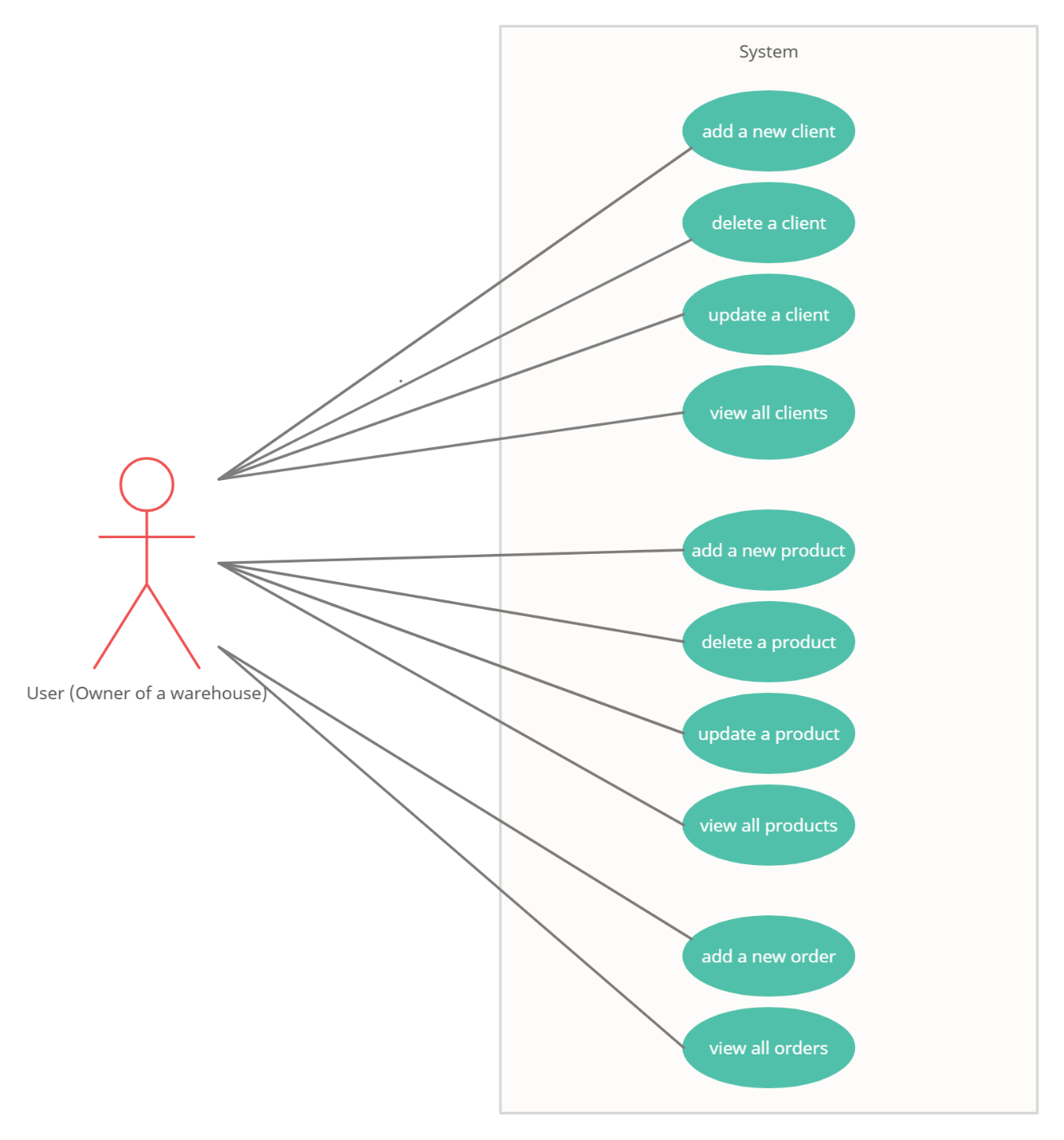
* The order management system should allow the user to insert data.
* The order management system should allow the user to select which operation he/she wants to perform.
* The order management system should be able to insert a client/product/order.
* The order management system should be able to delete a client/product.
* The order management system should be able to list all the clients/products/orders.
* The order management system should be able to update the information regarding a client/product.

Non-functional requirements

* The order management system should be intuitive and easy to use by the user.
* The order management system should have a nice and pleasant graphical user interface.

USE CASES

The Use Case Diagram:



1. Add a new client

Primary actor: user

Main success scenario:

1. The user selects the *clients* button from the main window.
2. The user inserts the data about the new client (id, name, address, email, age) in the graphical user interface.
3. The user selects the *insert* button.
4. The order management system performs the insertion and displays a message saying that the insertion was successful.

Alternative sequence:

1. The user introduces an invalid input (name/email/age).
2. The scenario returns to step 1.
3. The user introduces an already existing ID for the new client.
4. The scenario returns to step 1.

2. Delete a client

Primary actor: user

Main success scenario:

1. The user selects the *clients* button from the main window.
2. The user inserts the ID of the client he/she wants to delete.
3. The user selects the *delete* button.
4. The order management system performs the delete operation and displays a message saying that the delete operation was successful.

Alternative sequence:

1. The user introduces a non-existing ID (there is no client with this ID assigned in the table).
2. The scenario returns to step 1.

3. Update a client

Primary actor: user

Main success scenario:

1. The user selects the *clients* button from the main window.
2. The user inserts the new data (address, age) and the ID of the existing client he/she wants to update.
3. The user selects the *update* button.
4. The order management system performs the update operation and displays a message saying that the update operation was successful.

Alternative sequence:

1. The user introduces a non-existing ID for the client.
2. The scenario returns to step 1.
3. The user introduces an invalid input (address/age).
4. The scenario returns to step 1.

4. View all clients

Primary actor: user

Main success scenario:

1. The user selects the *clients* button from the main window.
2. The user selects the *view* button.
3. The order management system performs the view operation and displays the table with all the clients in the graphical user interface.

Alternative sequence - None

5. Add a new product

Primary actor: user

Main success scenario:

1. The user selects the *products* button from the main window.
2. The user introduces the data about the new product (id, name, price, quantity) in the graphical user interface.
3. The user selects the *insert* button.
4. The order management system performs the insertion and displays a message saying the insertion was successful.

Alternative sequence:

1. The user introduces an invalid input (name/price/quantity)
2. The scenario returns to step 1.
3. The user introduces an already existing ID for the product he/she wants to insert
4. The scenario returns to step 1.

6. Delete a product

Primary actor: user

Main success scenario:

1. The user selects the *products* button from the main window.
2. The user introduces the ID of the product he/she wants to delete.
3. The user selects the *delete* button.
4. The order management system performs the delete operation and displays a message saying that the delete operation was successful.

Alternative sequence:

1. The user introduces a non-existing ID.
2. The scenario returns to step 1.

7. Update a product

Primary actor: user

Main success scenario:

1. The user selects the *products* button from the main window.
2. The user introduces the new data (price, quantity) and the ID of the product he/she wants to update.
3. The user selects the *update* button.
4. The order management system performs the update operation and displays a message saying the update operation was successful.

Alternative sequence:

1. The user introduces an invalid input (price/quantity)
2. The scenario returns to step 1.
3. The user introduces a non-existing ID.
4. The scenario returns to step 1.

8. View all products

Primary actor: user

Main success scenario:

1. The user selects the *products* button from the main window.
2. The user selects the *view* button.
3. The order management system performs the view operation and displays the table with all the products in the graphical user interface.

Alternative sequence - None

9. Add an order

Primary actor: user

Main success scenario:

1. The user selects the *orders* button from the main window.
2. The user selects an ID from the existing clients.
3. The user selects an ID from the existing products.
4. The user inserts the quantity for the order.
5. The user selects the *create* button.
6. The order management system performs the insertion and displays a message saying that the insertion was successful.

Alternative sequence:

1. The user introduces a quantity that is greater than the one available.
2. A message informing that there is not enough stock is shown on the screen.
3. The scenario returns to step 1.
4. The user introduces an already existing ID for the order
5. The scenario returns to step 1.

10. View all orders

Primary actor: user

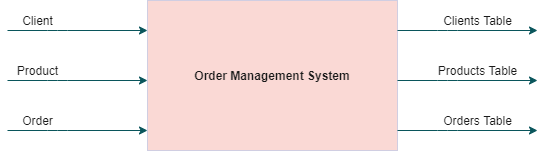
Main success scenario:

1. The user selects the *orders* button from the main window.
2. The user selects the *view* button.
3. The order management system performs the view operation and displays the table with all the orders in the graphical user interface.

Alternative sequence – None

DESIGN

Level 1: Overall System Design

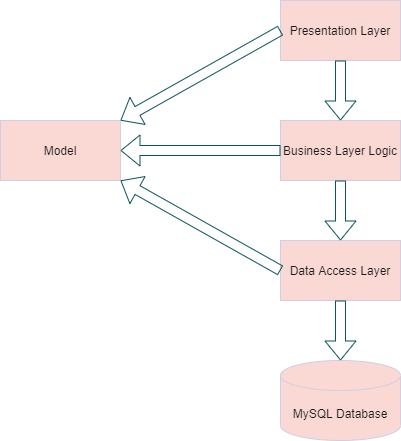


Level 2: Division into sub-systems/packages

LAYERED ARCHITECTURE

The architecture used for this project is the layered architecture style. By using this style, the components/modules that have the same functionalities are organized into horizontal layers, meaning that each layer has a specific role in the application. Each layer uses the functions provided by the one below and so on.

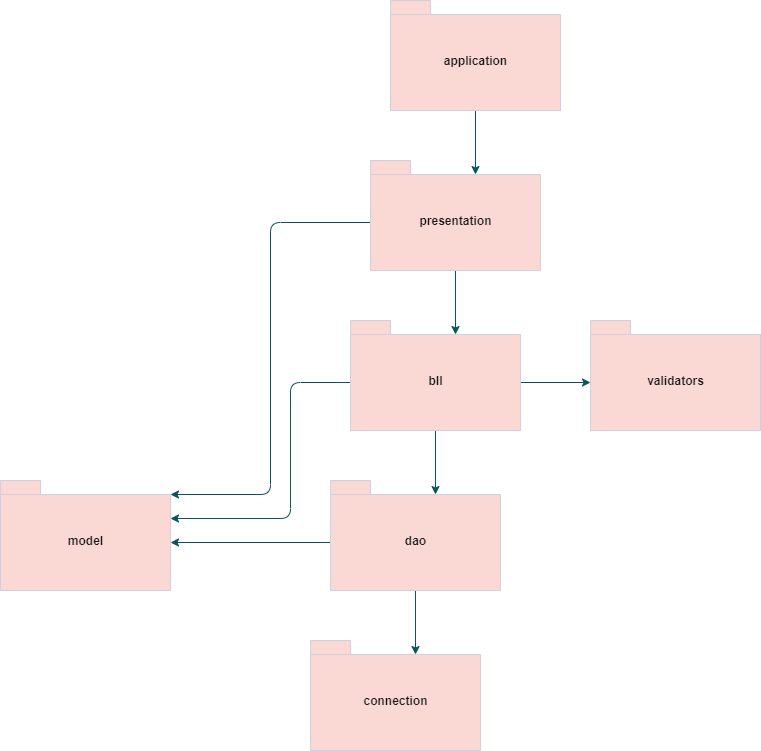
The visual representation of the layered architecture:



When the user interacts with the application, the very first thing he/she sees is the Presentation Layer (the graphical user interface). After that, the Business Layer Logic takes care of the execution of the operations that the user wants to perform on the application (it coordinates the application, processes commands and makes logical decisions and evaluations). Since the Business Layer Logic requires data to work with, it will get the respective data from the Data Access Layer who is responsible for the connection to the database server and for storing and retrieving data from the database to the user.

The big advantage when it comes to the layered architecture is that it provides a clear abstract view of the system as a whole while also providing enough detail to understand the roles and responsibilities of individual layers and the relationship between them.

My package diagram is the following:



Level 3: Division into classes

a) MODEL PACKAGE – it contains the main classes that reflects the tables from the database – Client, Product and Order.

The Client class reflects the Client table from the database and contains information about the client, each attribute in the class representing a field in the client table, i.e. id, name, address, email and age.

The Product class reflects the Product table from the database and contains information about the product, each attribute in the class representing a field in the product table, i.e. id, name, price and quantity.

The Order class reflects the Order table from the database and contains information about the order, each attribute in the class representing a field in the order table, i.e. id, the corresponding client’s id, the corresponding product’s id and the quantity.

b) PRESENTATION PACKAGE – it contains the classes representing the graphical user interface for the application and the controller class.

The AppMainWindow class is composed of a JFrame with 3 buttons, each of them leading to their respective frame (the *Clients* button will open the ClientWindow frame, the *Products* button will open the ProductWindow frame and the *Orders* button will open the OrderWindow frame).

Except for the AppMainWindow, all the other frames (ClientWindow, ProductWindow and OrderWindow) have the same structure: they are composed of a JFrame with 3 panels:

- one with the text fields where the user introduces his/her input data

- one where the table with all the information stored in the database is shown to the user

- one with the buttons that performs the operations on the tables

The Controller class deals with initializing the buttons from all the windows and implementing the actions that have to be performed after clicking one of the buttons.

c) BUSSINESS LOGIC LAYER (BLL) PACKAGE – it contains the classes which deal with the logic behind each operation (insert, update, delete).

ClientBLL takes care of insertion, deletion and updating of a client. It uses a ClientDAO object in order to get access to the result returned after executing one of these 3 operations.

ProductBLL takes care of insertion, deletion and updating of a product. It uses a ProductDAO object in order to get access to the result returned after executing one of these 3 operations.

OrderBLL takes care of creating an order. It also has a method which prints the bill of the last inserted order in the table.

All the three BLLs use from the AbstractDAO class a function which searches for an element from a table bu the id.

d) VALIDATORS PACKAGE – it contains a validator interface, three validators for each model (Client, Product, Order) and an exception.

The three validators are used to verify if the data which is to be inserted in the database is valid (for e.g. we check if the name of the client/product is composed of only a – Z characters or if the product’s quantity is a positive integer)

The exception is used to signal that the quantity selected when making an order is greater that the stock available in the product section.

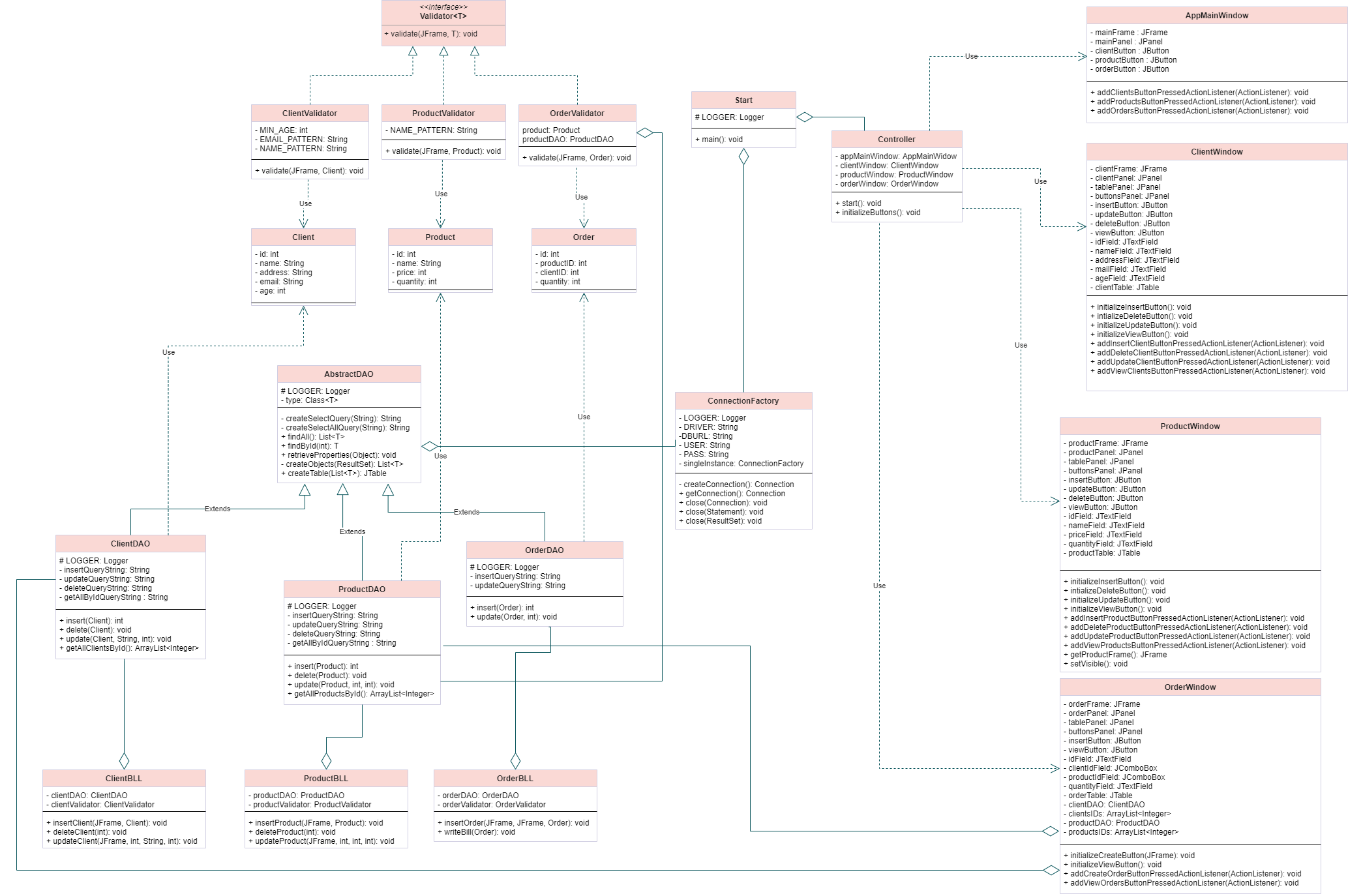
e) DATA ACCESS OBJECT (DAO) PACKAGE – here are all the classes which executes different query statements on the database’s tables.

AbstracDAO class contains functions that can be applied on any object directly. It has three methods: one finds an element from a table by its id, one finds all the elements from a table and one returns as a JTable the table with all the information.

f) CONNECTION PACKAGE – it contains methods useful for connecting to a database server.

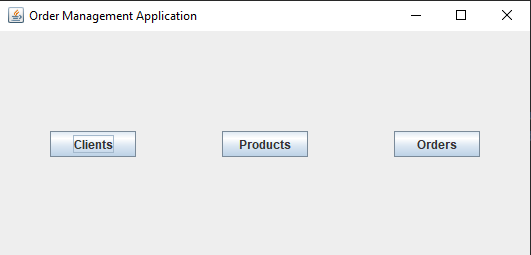
g) APPLICATION PACKAGE – it contains the main method which starts the application.

UML CLASS DIAGRAM



GUI DESIGN

MAIN WINDOW – it contains 3 buttons which will lead the user to the section he/she wants to execute an operation

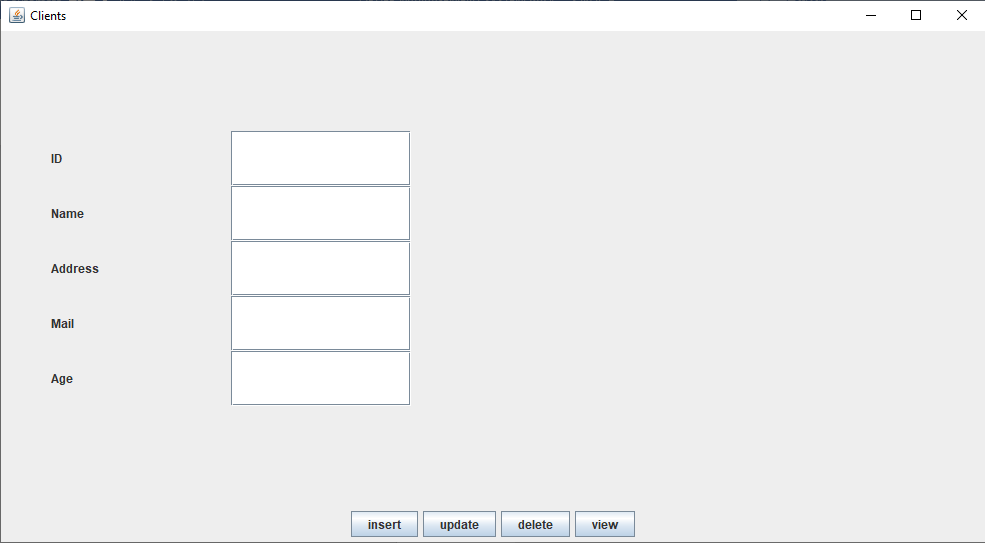


After pressing one of the buttons, another window (frame) will open and appear on the screen, making it possible for the user to open at the same time all the three windows and see all the data from all the tables at the same time.

The Client and the Product windows have the same format; they are composed of three panels – one for the text fields (each of them according to what data the entries require when inserting into the database), one for the table view (after listing all clients/products) and the button panel which contains 4 buttons – *insert*, *delete*, *update* and *view*.

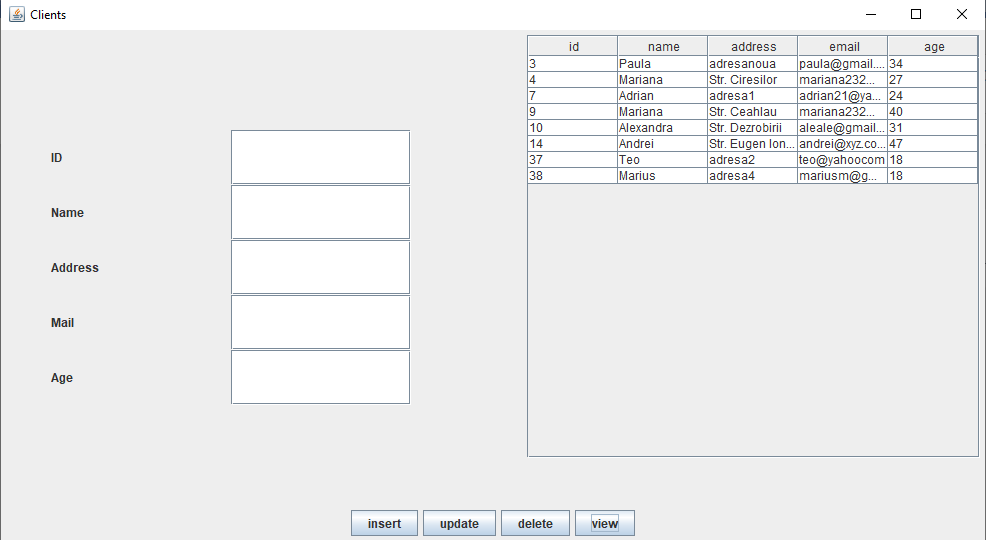
CLIENTS WINDOW – the same for PRODUCTS WINDOW

(after pressing *Clients* button on the main window)



CLIENTS WINDOW

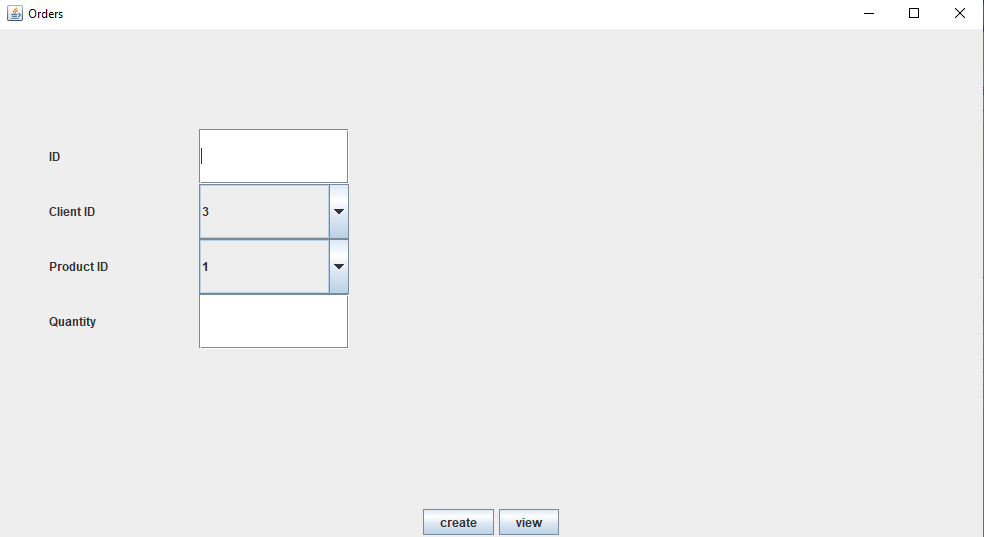
(after pressing the view button in order to also see how the table is placed in the GUI)



The Order window is similar to the other two. The only differences are that the text fields’ panel contains two JComboBoxes which allows the user to select an existing client and product in order to create the order and that it only has two buttons – create and view.

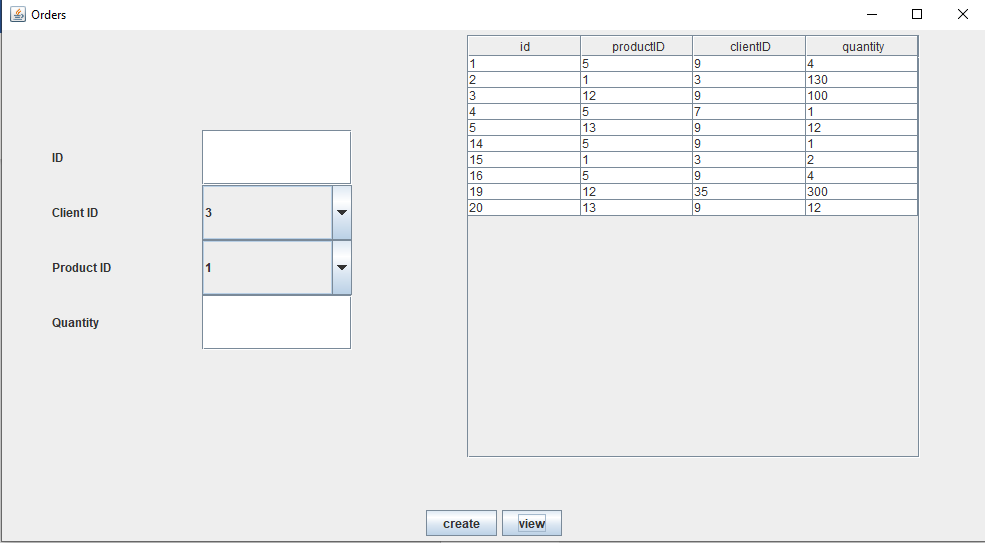
ORDERS WINDOW

(after pressing *Orders* button on the main window)



ORDERS WINDOW

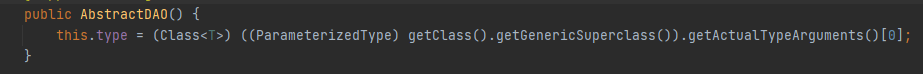
(after pressing the view button in order to also see the table)



IMPLEMENTATION

This assignment is based on the reflection technique provided by JAVA. Its mainly objective is writing generic code which can be used by other people when developing different applications. This is possible because Java reflection makes it possible to inspect classes, interfaces, fields and methods at runtime without having known the names of the classes.

In the implementation, this technique was used in the AbstractDAO class in the following methods:





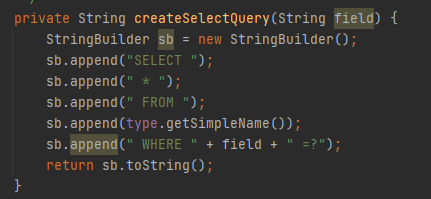
The method above constructs a DefaultTableModel having as parameter a list of objects of type T.

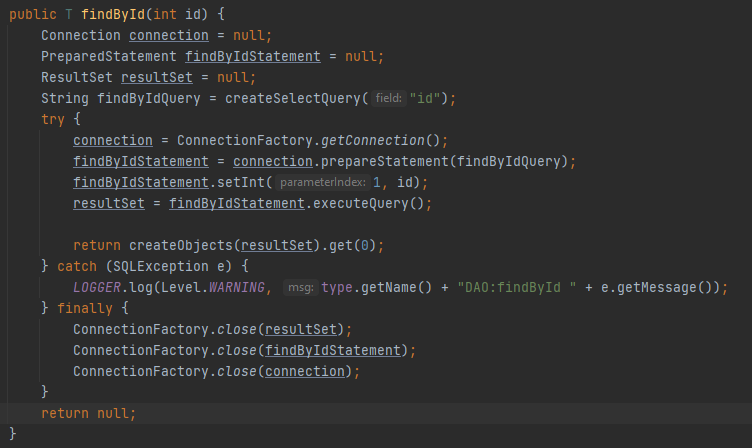
In order to construct the table, we need to know how many collums the table we will build has. By using the getClass() and getDeclaredFieds() we are able to identify on which table we are working on (if it is the Client/Product/Order one) and get all the declared fields in that table (for e.g on the Client we would have id, name, address, email and age => a total of five fields).

Moving on, on the first for loop we go through all the fields declared in the table and access their names. By having their names now, we can add the columns with their respective field in the tableModel.

On the next for loop, we go through the list of objects which represents our elements from the table (each elemet of the list has its own data and is equivalent to a row in the table). Going through each element, we access the information stored on each field and place it in an array. In the end, the build array is inserted as a row in the tableModel, the latter being then „converted” into a JTable.

Another example is the following piece of code which finds an element from a table based on its id value. In the StringBuilder sb that stores the query statement we use *type*.getSimpleName to get the name of the table we will work on and insert it in the statement.





CONCLUSIONS

This assignment represents a good example for using reflection tehcniques. You can see its utility when executing queries (insert, select, update, delete) on tables from a database, especially because not all the tables contain the same fields or have the same number of elements.

BIBLIOGRAPHY

<https://gitlab.com/utcn_dsrl/pt-reflection-example/-/tree/master/src/main/java>

<https://gitlab.com/utcn_dsrl/pt-layered-architecture/-/tree/master/src/main/java>

<https://alvinalexander.com/java/java-mysql-update-query-example/>

<https://www.tutorialspoint.com/how-to-create-defaulttablemodel-which-is-an-implementation-of-tablemodel>

<https://student.cs.uwaterloo.ca/~cs446/1171/Arch_Design_Activity/Layered.pdf>