

**Technical University of Cluj-Napoca**

**Faculty of Automation and Computer Science**

Programming Techniques

Assignment 3

Order management

Teacher: Prof. Ioan Salomie Student: Alina Mihuț

Teaching assistant: Ing. Ciprian Adrian Stan Group 30422

Table of contents

1. Assignment objective
2. Problem analysis, modeling, scenarios, use cases
3. Design (design decisions, UML diagrams, class design, relationships, packages, algorithms, user interfaces)
4. Implementation
5. Results
6. Conclusions
7. Bibliography
8. Assignment objective

Consider an application **Order Management** for processing client orders for a warehouse. Relational databases are used to store the products, the clients and the orders. Furthermore, the application should be structured in packages using a layered architecture presented in the support presentation and should use (minimally) the following classes:

• **Model classes** - represent the data models of the application

• **Business Logic classes -** contain the application logic

• **Presentation classes –** GUI related classes

• **Data access classes -** classes that contain the access to the database

Other requirements include:

* Use javadoc for documenting classes and generate the corresponding JavaDoc files
* Use relational databases for storing the data for the application, minimum three tables: Client, Product and Order
* Create a graphical user interface including:

- A window for client operations: add new client, edit client, delete client, view all clients in a table

- A window for product operations: add new product, edit product, delete product, view all product in a table

- A window for creating product orders - the user will be able to select an existing product, select an existing client, and insert a desired quantity for the product to create a valid order. In case there are not enough products, an under stock message will be displayed. After the order is finalized, the product stock is decremented.

1. Problem analysis, modeling, scenarios, use cases
2. Analysis

The application should implement the basic operations on a database ( displaying, updating, inserting, finding and deleting) and keep track of the clients, products and orders. The three tables are stored in a MySQL relational database, which is connected to the application and updated in real time. All this operations and the data which has to be updated in the tables are set by the user from a graphical interface.

B) Modeling, scenarios and use cases

The main scenario is the following one:

The user starts the application and selects from the main scene the button corresponding to the table he wants to work on. He will then be taken to another scene, where he has several options: to display all the entries in the table, to add a new entry to the table, to delete an entry or to update one. This is the scenario for all the three main use cases (orders, products and clients). The data in the tables is updated after each operation.

There are several use cases, for each of the tables (for the client table – Insert Client, Update Client, Delete Client, Display all the clients). The data introduced by the user is validated when necessarily. For the product tables, the use cases are similar: – Insert Product, Update Product, Delete Product, Display all the products. Data is, again, validated. For the order table, there are just two use cases: Insert Order (the user selects a client, a product and a quantity and makes an order, which will appear in the orders’ table and the stock from the products’ table will be decremented and Display all the orders.

1. Design

A Layered Architecture is the organization of the project structure into four main categories: presentation, application, domain, and infrastructure. Each of the layers contains objects related to the particular concern it represents.

The presentation layer contains all of the classes responsible for presenting the UI to the end-user or sending the response back to the application.

The application layer contains all the logic that is required by the application to meet its functional requirements and, at the same time, is not a part of the domain rules.

The domain layer ( business layer ) represents the underlying domain, mostly consisting of domain entities and, in some cases, services. It contains the classes that encapsulate the application logic

The Data Access Layer contains all the classes containing the queries and the database connection.

I chose to design my application using a Layered Architecture. My application has six packages corresponding to the 4 design layers: dataAccessLayer, businessLayer, model and presentation.

The **model** package contains classes mapped to the database table (Client, Product and Order). This package is related to all of the four layers, being accessed in all layers of the application. The **connection** package contains the Connection Factory class which contains the name of the driver, the database, location (DBURL), and the user and the password for accessing the MySQL Server.

The **dao** package contains the Data Access Layer of the application. Each of its classes contain the queries and the methods that define the common operations for accessing one of the three tables (client, product or order).

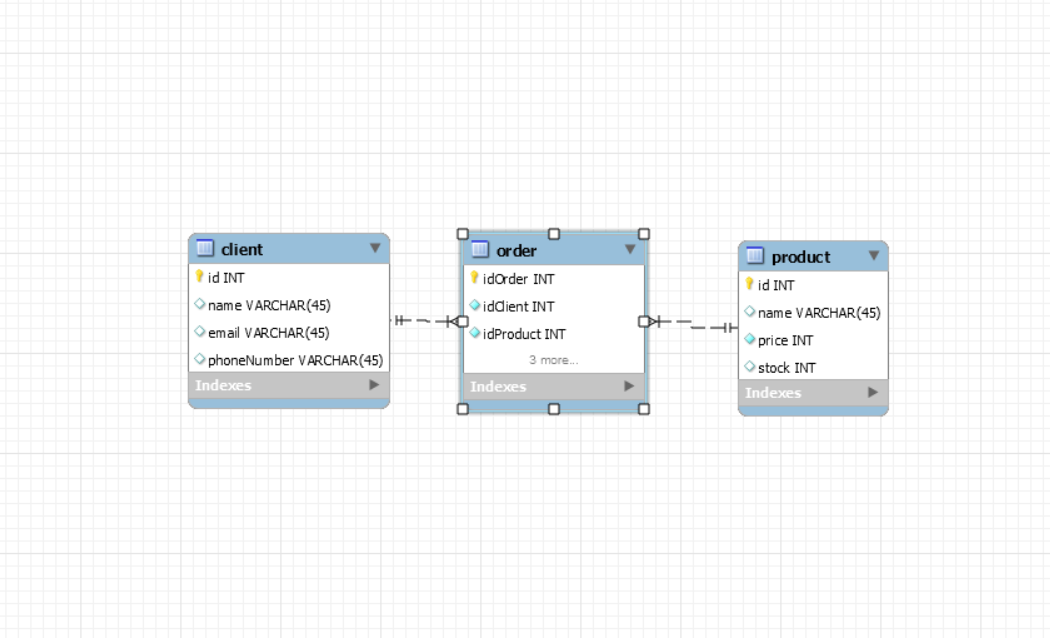
The **bll** package contains the business layer of the application: the classes which encapsulate the application logic for the operations on one of the three tables and the classes which contain methods for validating the inputs given by the user, in order to insert in the tables only relevant and correct data.

The **presentation** package the package with the controller classes ( 4 in total – one controlling each scene), as its name suggests, which controls both the GUI classes and the model classes and the data flow between the two. It receives keyboard inputs from the GUI as well as button presses and translates the events into requests, which are sent to the model and the user interface class, with the methods concerning the running of the Graphical User Interface and its’ proper working.

Last but not least, the **application** package contains the main class, initializes the primary stage and displays it on the running of the application.

1. Implementation

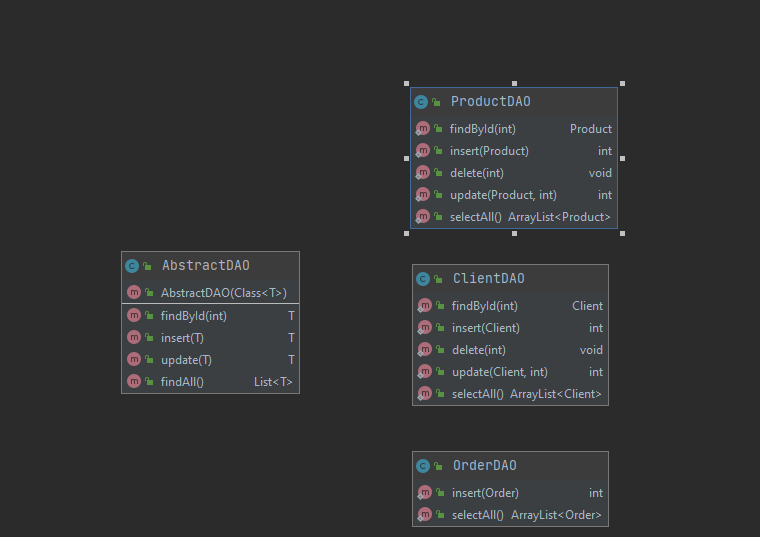
The first thing to do is to design a MySQL relational database. The database schema is presented below.



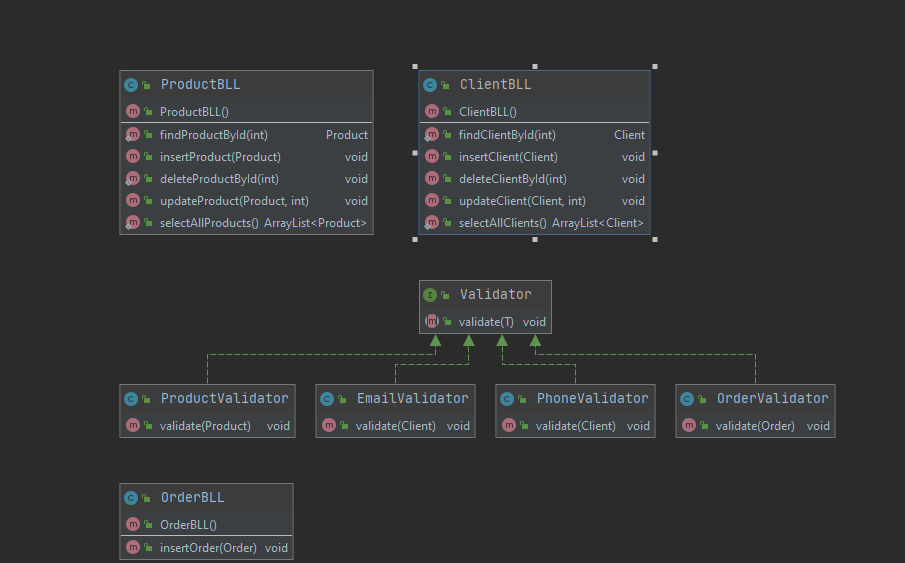
I connected the database to my java application in the Connection Factory class. It contains the name of the driver, the database, location (DBURL), and the user and the password for accessing the MySQL Server. Also, it contains methods for creating a connection, getting an active connection and closing a connection, a Statement or a Result Set.

Each of the six packages in the Layered Arhitecture pattern is divided into several classes, containing specific methods.

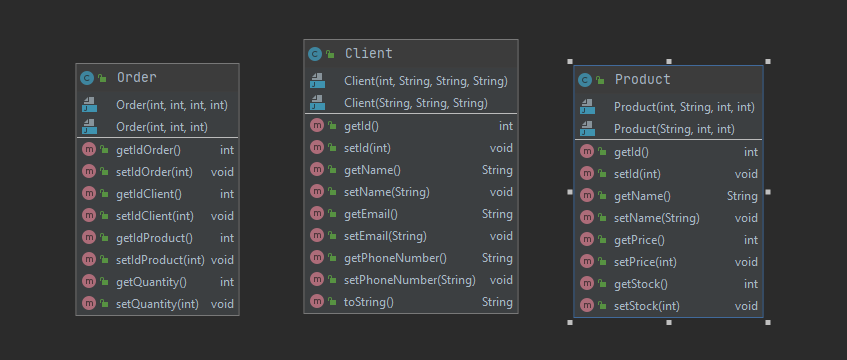
The dao package contains the classes ClientDAO, ProductDAO and OrderDAO. ClientDAO class contains the queries and the methods that define the common operations for accessing the clients table: **findById(int clientId), insert(Client client), delete( int clientId ), update (Client client, int clientID), selectAll().** These are the lowest level functions, working directly with the database and performing queries. The other two classes perform similar operations. The diagram of the dao package is presented below.



The business layer package in multiple classes, performing two important tasks. First, we have the ClientBLL, ProductBLL and OrderBLL classes, with methods which encapsulate the application logic for the operations on the three tables (finding an entry by id, inserting an entry in the table, deleting an entry, updating and displaying all entries in the table). The diagram of the bll package is presented below. Then, there are the classes which perform validations on the entries of the three tables. Each validator implements the Validator interface and contain methods which override the ***validate()*** method. Each method verifies if the entry (email, price, phone, stock or quantity) to be inserted or updated in the database is valid and if not, it throws an exception.

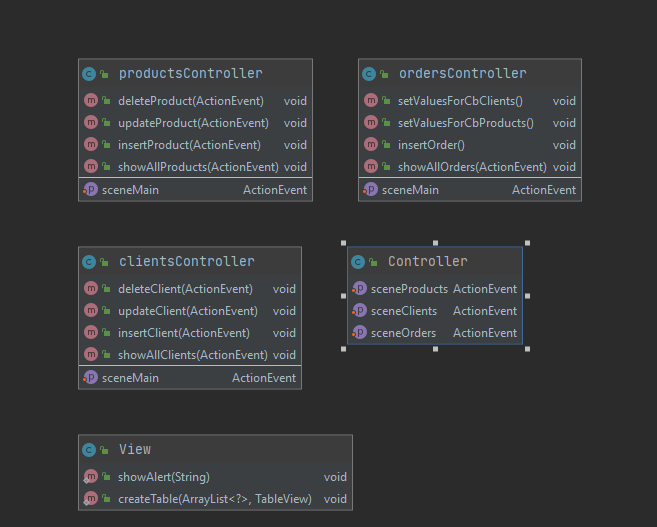


The model package is divided into 3 classes, Client, Product and Order. The Client class resembles the client table in the database and has the fields exactly the same type as the columns from the corresponding table. These classes contain constructors, getters and setters for creating and accessing the objects.



The presentation package, has the user interface classes and the controller classes, with methods which perform actions on buttons or display alerts in the graphical interface for data validation. Each of the four controller classes corresponds to one of the scenes in the graphical user interface. They have methods which get the inputs from the user and send them in order to be validated and inserted in the tables. They also have methods actions on buttons, allowing the user to select the operation he wishes to perform on the tables. The buttons also allow the user to easily navigate between the scenes. Displaying the output on screen is handled by the View class, specifically the method ***createTable( ArrayList<T> listOfObjects, TableView table )***, which takes as parameters 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. This method is called for all three types of objects and the contents f the tables are displayed in a table view in each scene.

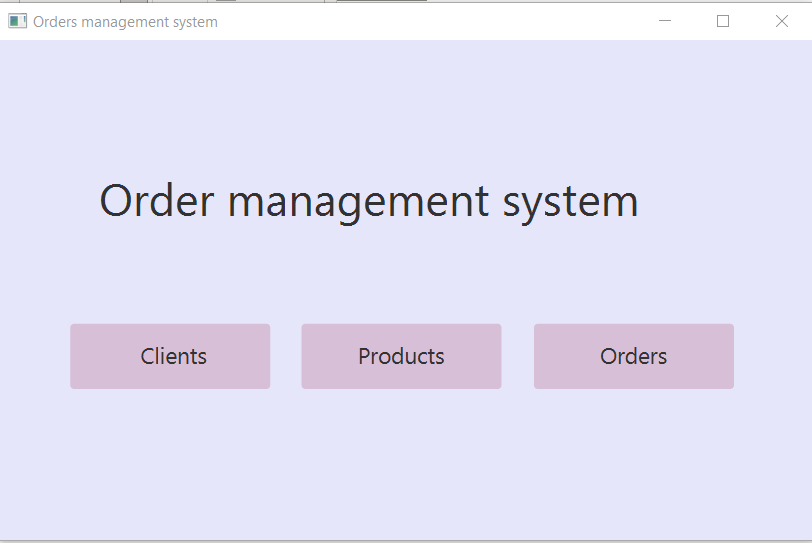


The Main class in the application package initializes the primary stage and displays it on the execution of the application.

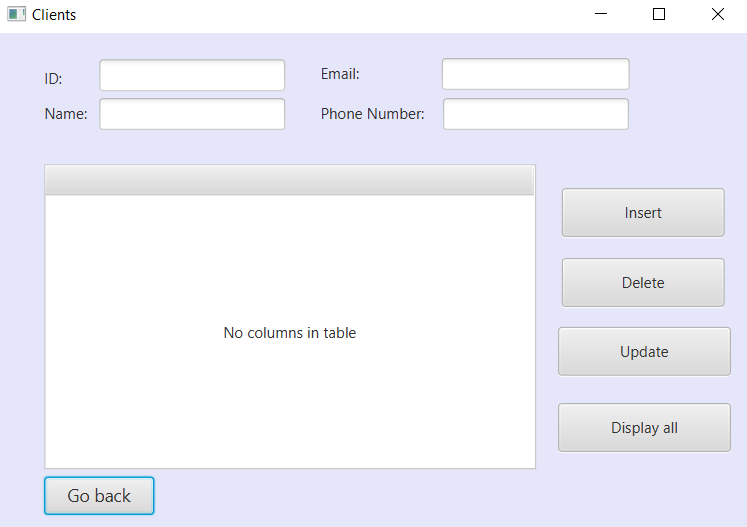
In the implementation of the graphical user interface, I used Scene Builder. The GUI contains four windows, a main one and three for accessing each table. The user has to enter in each of the text fields or choice boxes the parameters of the entry to be inserted/updated or the id of the entry he wants to delete and press a button for choosing the operations he wants to perform on the table. The data he enters is verified and in case it’s not correct, he receives an alert box on the screen. The user can see the data in each table, updated after each operation. He can also navigate through the windows with the actions set on buttons.

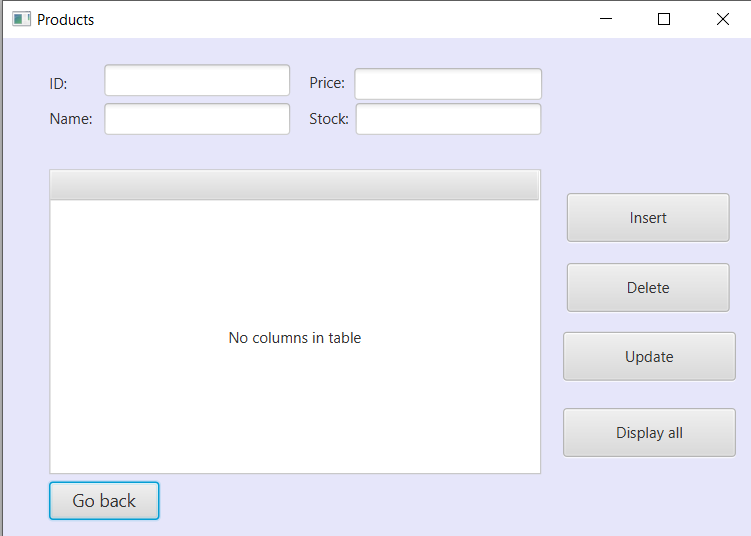
I designed the user interface in order to be easy and inductive for the user. The design of the user interface window can be seen in the following pictures.

The main window:

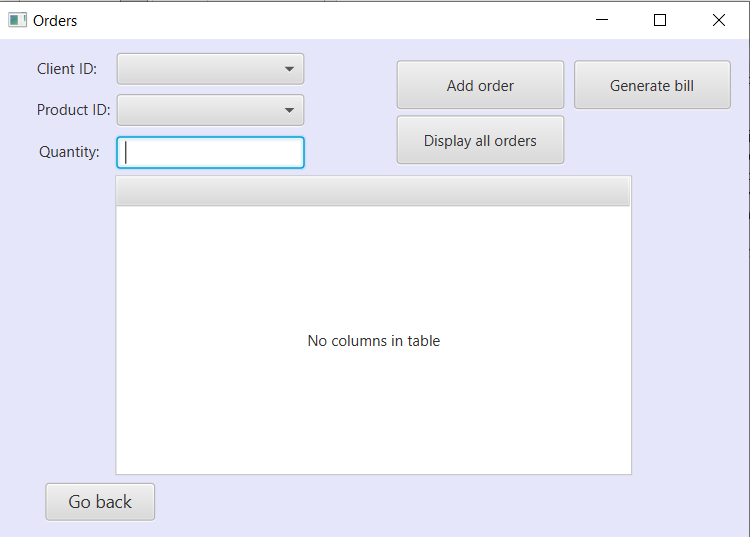


The clients window:



The products window:

And the orders window:



1. Results

The results of the performed operations can be seen by accessing the database from MYSQL workbench or by displaying the contents of the tables in the graphical user interface. All operations are performed in real time. Data is only inserted or updated in the tables if the parameters are valid. Orders can only be placed between an existing customer, chosen from a combo box and an existing product. If the quantity inserted by the user is not valid, the order can’t be placed and it won’t appear in the table. Once a valid order is made, it appears in the table and moreover, the stock from the product table is decremented with the ordered quantity.

1. Conclusions

Working on this project was a good opportunity for me to gain a deeper understanding of the OOP concepts learned in the first semester as well as working on an application with a graphical user interface. It was a bit challenging at first, until I understood how to work with a database and make the connection between it and the java application, perform queries, etc. and how to structure and organize the project in order to respect the layered architecture pattern. I realized that a good modelling of the problem from the beginning really helps a lot in the future development.

I’m really glad that I now know how to work with a relational database, because I feel it will really help me a lot in other future projects.

All in all, I liked working on this project and I feel like I gained a lot of knowledge from it.

As further improvements, a lot could be made. For example, to make a more complex database and add new features, like searching for an exact keyword or entry. As for the user interface, I feel like it could also be improved to be more user-friendly.

1. Bibliography

- ***Assignment\_3\_Support Presentation*** PowerPoint presentation