DOCUMENTATION

ASSIGNMENT3

NAME: Muntean Cristian

GROUP: 30424

Contents

[1. Objective 3](#_Toc100861146)

[2. Analysis, modelling, scenarios and use cases 3](#_Toc100861147)

[3. Design 6](#_Toc100861148)

[4. Implementation 8](#_Toc100861149)

[5. Results 10](#_Toc100861150)

[6. Conclusions 10](#_Toc100861151)

[7. Bibliography 10](#_Toc100861152)

# Objective

Create an application responsible with managing client orders for a warehouse. The user should be able to perform operations on clients: add a new client, remove a client, edit a client, find a client; perform operations on products: add a new product, remove a product, edit a product, fin a product; and perform operations on orders

# Analysis, modelling, scenarios and use cases

1. Analysis

This application simulates the managing of orders in a warehouse and it interacts with a database, allowing the user to perform different actions on clients, products and orders. For the client operations, a user is able to add, edit, find and remove a client. In order to perform these operations to a client, the user has to click on the button from the main menu that takes him/her to the client operations window, then click on the button of their choice. After that, if the “add client” button is clicked, a second window will pop up and the user has to input the fields of the new client (id, name, surname, address). In order for the edit operation to be performed, the user has to select an existing client from the table, then click on the “edit” button. In this way, the selected client will be edited with the fields that the user completes in the pop-up window. For the remove operation, the user has to select a client from the table, then click said button and the client will be removed. The find operation requires an id as the input from the user and will show the user with the given id.

The product operations work in the same way as the client ones. In order to perform them, the user has to click on the button from the main menu that takes him/her to the product operations window, then select the operation that he/her wants to be performed.

The add operation for the orders is different from the ones that correspond to client and products. In order for a user to add an order, he has to input the id and the quantity of the product, then select from two tables the client and the product that will be linked to the order. If the selected product’s stock is less than the quantity that the new order demands, an “out of stock” message will be displayed. For orders, there is a special operation called “Print bills” which, when selected by the user, a bill will be printed in the form of a .txt file for each of the orders.

1. Modelling

The for the client and product insertion and edit operations, the user will have to input the following values with the mentioned restrictions in order for an operation on the database to be successful:

* Id, different from the ones that are already in the table (not null in the case of insertion)
* Name, surname and address (for client) and name (for product) – not null in the case of insertion
* Price and stock (for product) – not null in the case of insertion and must be a number

For the order insertion and edit, the user must input the following values with the mentioned restrictions in order for an operation on the database to be successful:

* Id, different from the ones that are already in the table (not null in the case of insertion)
* There must be a row selected in both tables (in the case of insertion)

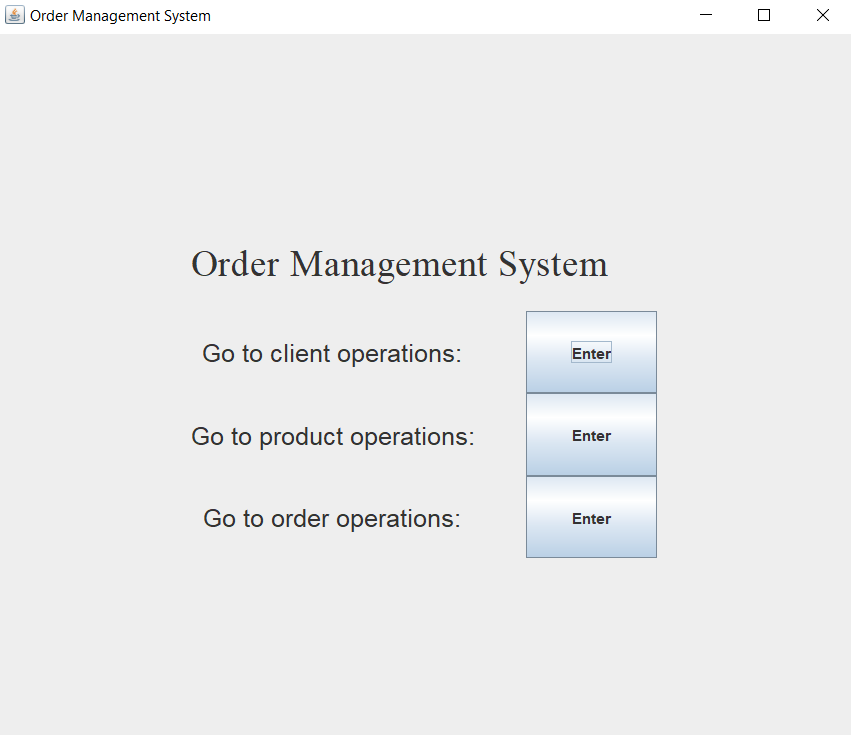
If the inputs follow these restrictions, then the operation will be stored in the database and the tables containing clients/products/orders will be updated.

1. Scenarios and use cases

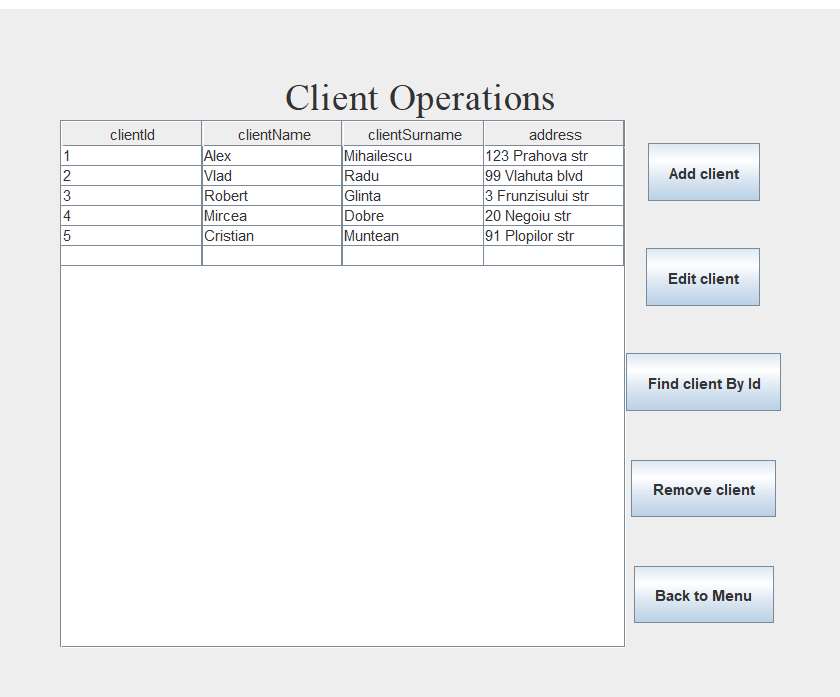
A use case is a methodology used in system analysis to identify, clarify, and organize system requirements. The use case is made up of a set of possible sequences of interactions between systems and users in a particular environment and related to a particular goal.

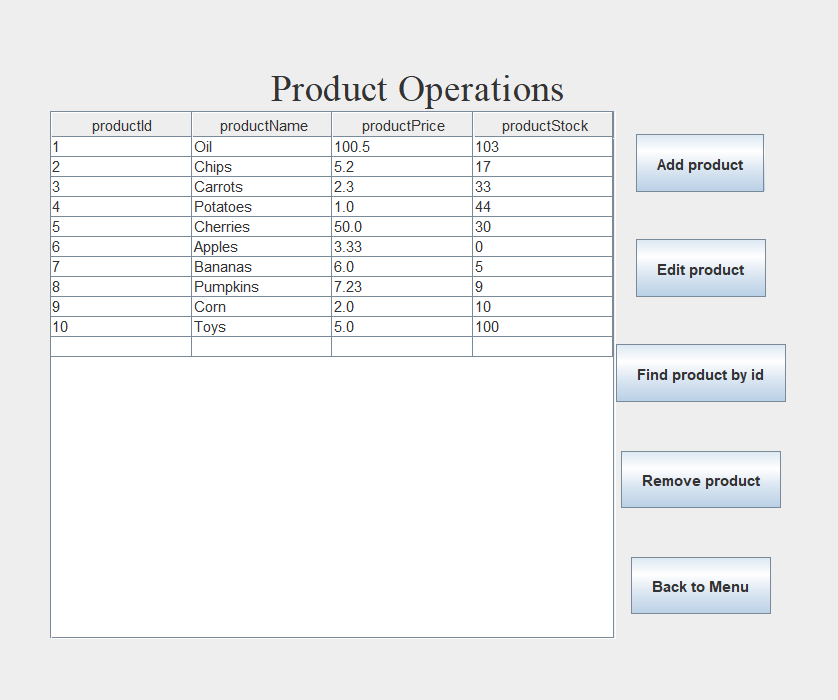
The use cases are strongly connected with the steps the user has to make, so I created the user interface with this in mind, resulting in a straight forward design.

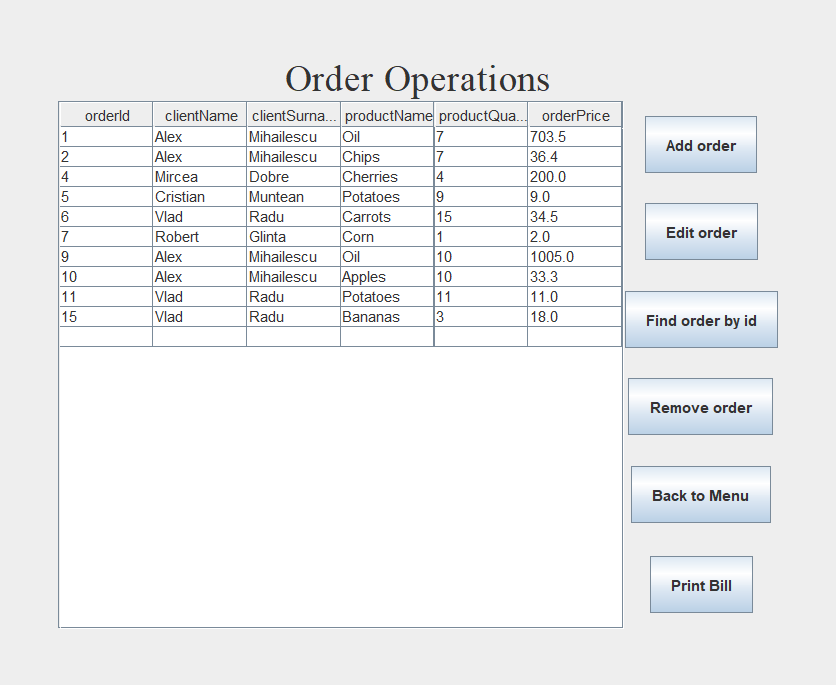
The first window that the user will see is the menu, where he/her has to select one of the three kinds of operations that the application can perform.



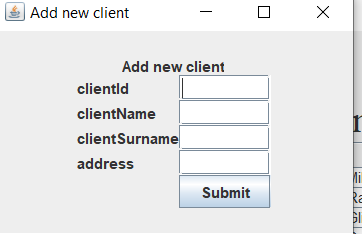
The next three windows are the ones where the operations will be performed and showed to the user.

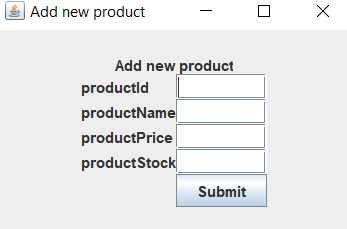


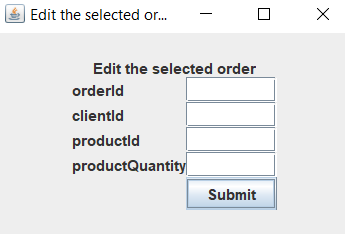




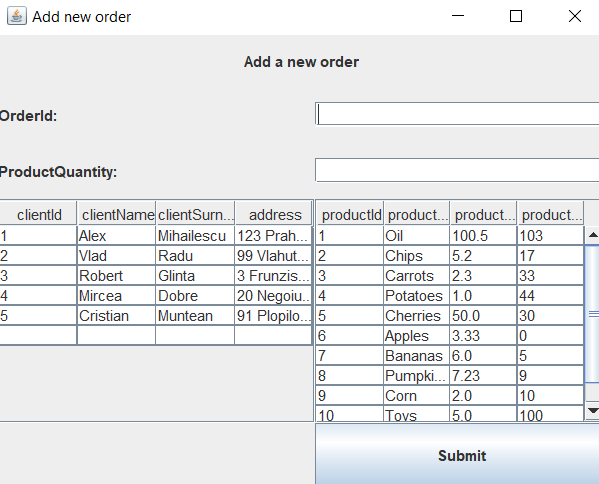
If the user selects the “Add” or “Edit” operation, the pop-up window will show the inputs necessary for an operation to be performed.





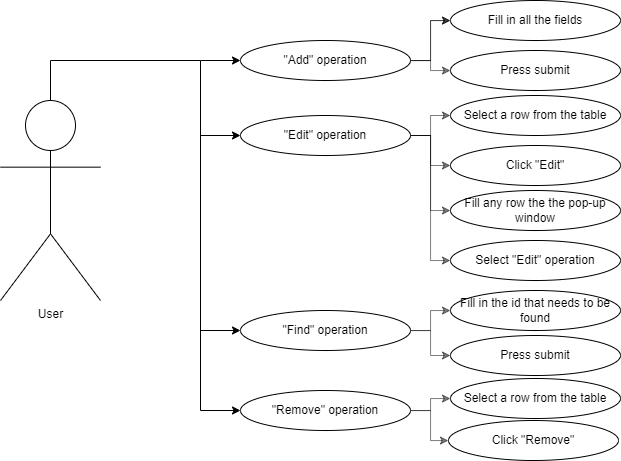


For the “Add order” operation, the pop-up window looks a bit different.



The for any of the “Add” operations, the user will have to fill all the fields and then click the “Submit” button. For the “Edit” operations, the user has to select a row from the table, then press the edit button where he can fill or not any field that is available to him in the pop-up window. For the “Remove” operations, the user has to select a row from the table, then press the “Remove” button. For the “Find by id” operations, the user has to click on the button, fill in the id he wants to find and then click the “Submit” button. For the “Print bills” operation that is present for the orders, the user has to click said button.

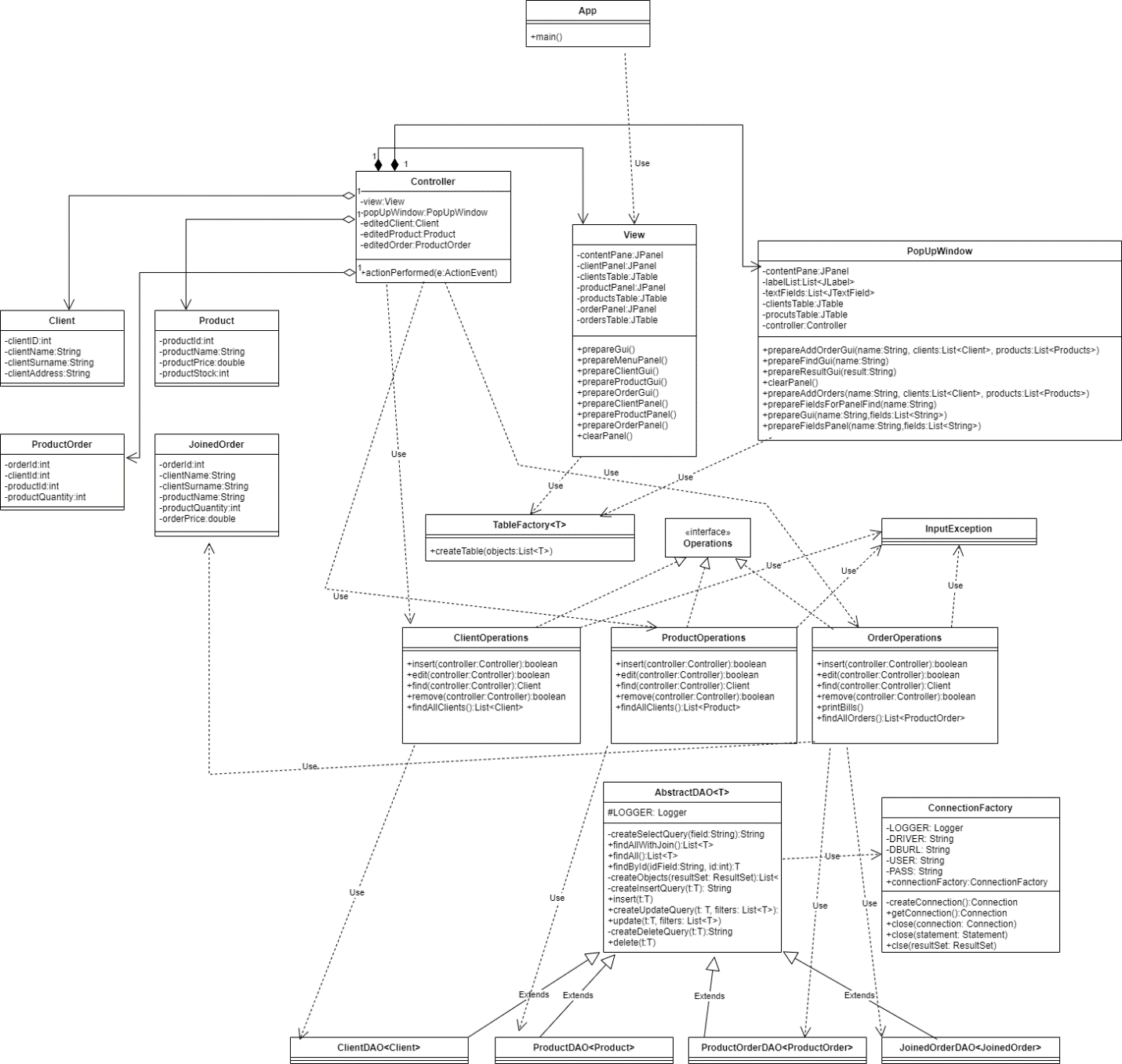
The use case diagram for these processes is presented below:



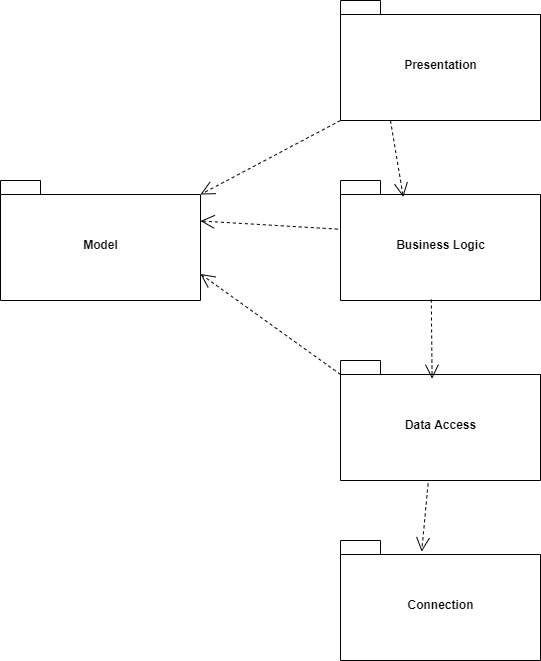
# Design

In the design of the application, I used the layered architecture that consists of the following layers:

* Model – contains the classes mapped to the database table. In my application, the classes “Client”, “Product”, “ProductOrder” and “JoinedOrder” belong to this package
* Presentation – contains the classes responsible with the user interface. In my application, the classes “View”, “Controller” and “PopUpWindow” belong to this package
* Business Layer – contains the classes that encapsulate the application logic. In my application, the interface “Operations” and the classes “ClientOperations”, “ProductOperations”, “OrderOperations” (which implement the “Operations” interface), “TableFactory” and “InputException” belong to this package
* Data Access Layer – Contains the classes containing the queries and connect to the database. In my application, the classes “AbstractDAO” (generic base class for the following classes), “ClientDAO”, “ProductDAO”, “ProductOrderDAO” and “JoinedOrderDAO” (which extend “AbstractDAO”) belong to this package.
* Class diagram



* Package diagram



* Data Structures

The data structures I have used in this project are either primitive and nonprimitive types (such as int, double, String), or more complex data structures such as ArrayLists. I used ArrayLists instead of simple Arrays because the maximum size of the Lists are unknown (the user can add as many cliends, products and orders as he pleases), so the dynamic size allocation of ArrayLists comes in handy in this case.

# Implementation

Because I used the layered architecture, my application has 5 main parts:

1. The *Business Logic* part

This part contains the classes that use the Model classes in order to create the functionality of the application. It contains 5 classes and 1 interface:

1. The “Operations” interface

This interface is the blueprint for the classes that will implement the concrete operations. It contains 4 methods: add, edit, find and remove

1. The “ClientOperations” class

This class is responsible with implementing the operations that will affect the “Client” table from the database. It implements the “Operations” interface and its methods. Every method receives as a parameter an object of type “Controller” because the operations receive the inputs from the graphical user interface. When extracting the inputs from the user interface, the reflection technique has been used in order to create a new object of type “Client” and to set the values of said object to the values that have been inputted in the text fields from the user interface.

1. The “ProductOperations” class

This class is responsible with implementing the operations that will affect the “Product” table from the database. It implements the “Operations” interface and its methods. It works in the same way as the “ClientOperations” class, but on a different table and with different fields. Because the reflection technique has been used, there isn’t much difference between this class and the previous mentioned one.

1. The “OrderOperations” class

This class is responsible with implementing the operations that will affect the “ProductOrder” table from the database. It implements the “Operations” interface and its methods, and has a unique method named “printBills”. In comparison with the previous two classes, the methods that are present in this one are different. When adding a new order (by using the “add” method), there needs to be a check present to see if there is enough stock of the ordered product for the order to saved and inserted into the database. If there isn’t enough stock, the operation will not succeed. The same check is performed in the “edit” method. When changing the product of an order, the product quantity of the order will be added back to the previous product and then subtracted from the new product of the order. When removing an order (by using the “remove” method), the product quantity of the order will be added back to the product’s stock.

1. The “TableFactory” class

This class is responsible with creating the tables that will be displayed on the user interface. It is a generic method that can receive any type of objects. It has a single method: “createTable”. This method receives as a parameter a list of object of that generic type and, by using the reflection technique, it generates the header of the table and populates it with the objects from the list. It returns a new JTable that has its cells restricted from being edited.

1. The “InputException” class

This class is a simple exception that is thrown when the user didn’t fill all the necessary inputs in the graphical user inteface.

1. The *Data Access* part

This part contains the classes that are responsible with connecting to the database and performing operations on said database. It contains 5 classes:

1. The “AbstractDAO” class

This class is a generic class that is responsible with communicating with the database and performing actions on said database. Its most important methods are the ones that perform insertion, deletion, updates and searches in the database’s tables. Each of these methods connect to the database by using the “ConnectionFactory” class and use the reflection technique to perform the actions that they need to perform.

1. The “ClientDAO” class

This class is a subclass of the “AbstractDAO” class that performs operations on the “Client” table of the database.

1. The “ProductDAO” class

This class is the subclass of the “AbstractDAO” class and performs operations on the “Product” table of the database.

1. The “ProductOrderDAO” class

This class is a subclass of the “AbstractDAO” class and performs operations on the “ProductOrder” table of the database.

1. The “JoineOrderDAO” class

This class is a subclass of the “AbstractDAO” class and performs operations on a “virtual table” that results from the joining of the “ProductOrder” table with the “Client” and the “Product” table.

1. The *Model* part

This part contains the classes which model the application and are mapped to the database table. The model has 4 classes:

1. The “Client” class
2. The “Product” class
3. The “ProductOrder” class
4. The “JoinedOrder” class

All of these classes contain the same fields as each respective table’s columns from the database, except the “JoinedOrder” class, which maps a “virtual” table that is a result of the join previously metioned.

1. The *Presentation part*

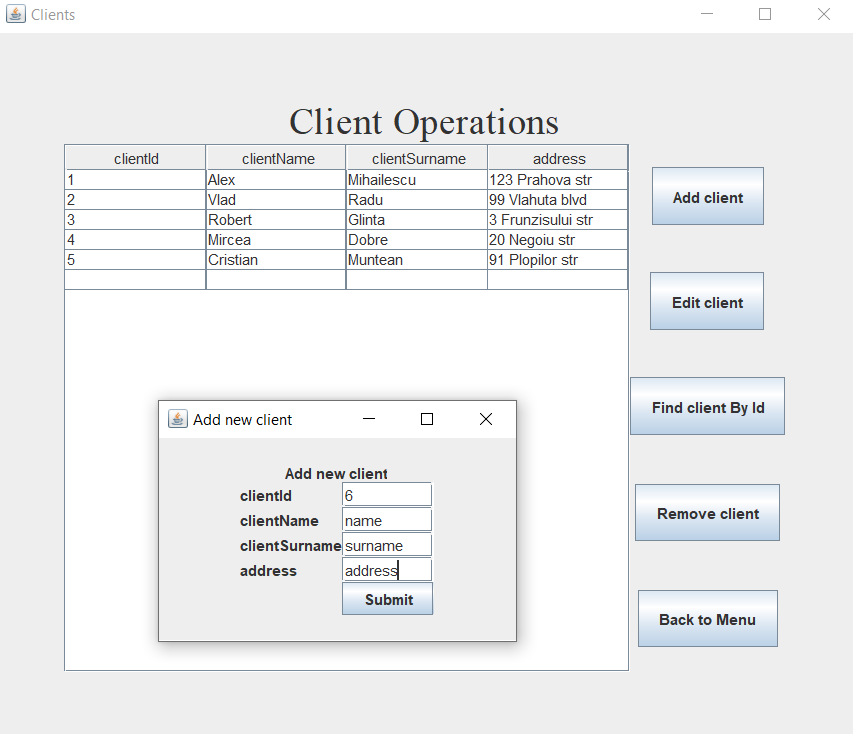
This part contains the classes that are responsible with the creation of the user interface. There are 3 classes that belong to this package:

1. The “View” class
2. The “PopUpWindow” class
3. The “Controller” class

The “View” and “PopUpWindow” classes are responsible with creating the user interface, whereas the “Controller” class is responsible of linking the user interface with the logic and model parts of the application.

# Results

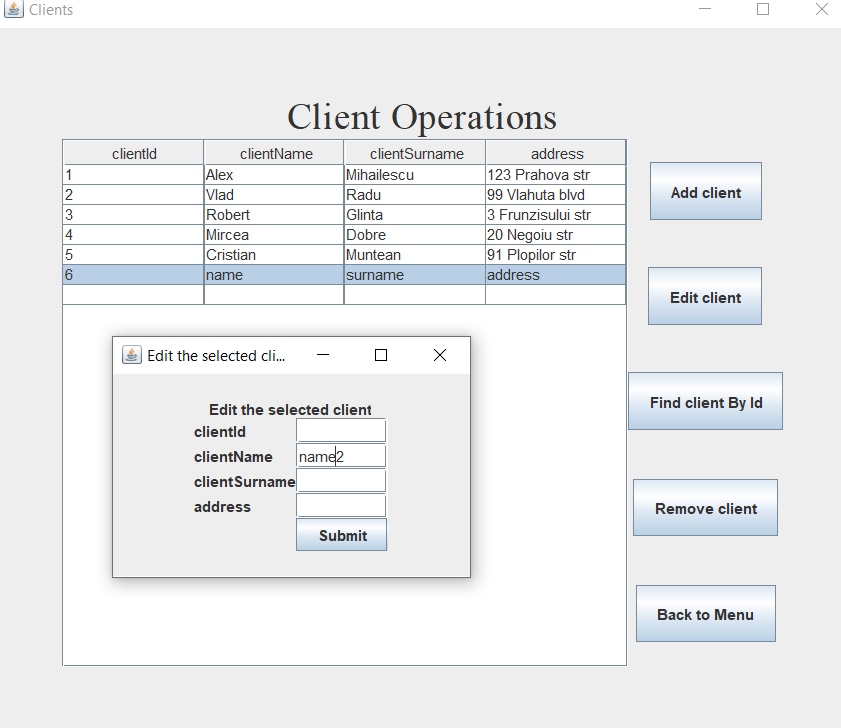
For testing the application, I introduced some inputs manually. Firstly, for the client operations I added a new client that had all of the fields valid and the result will be seen in the updated table once the pop-up window closes.

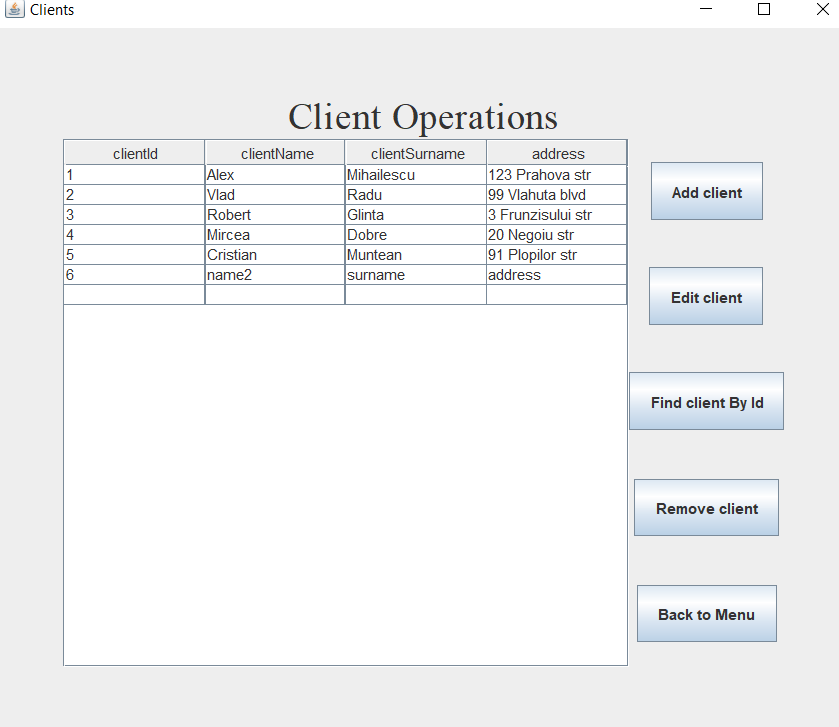




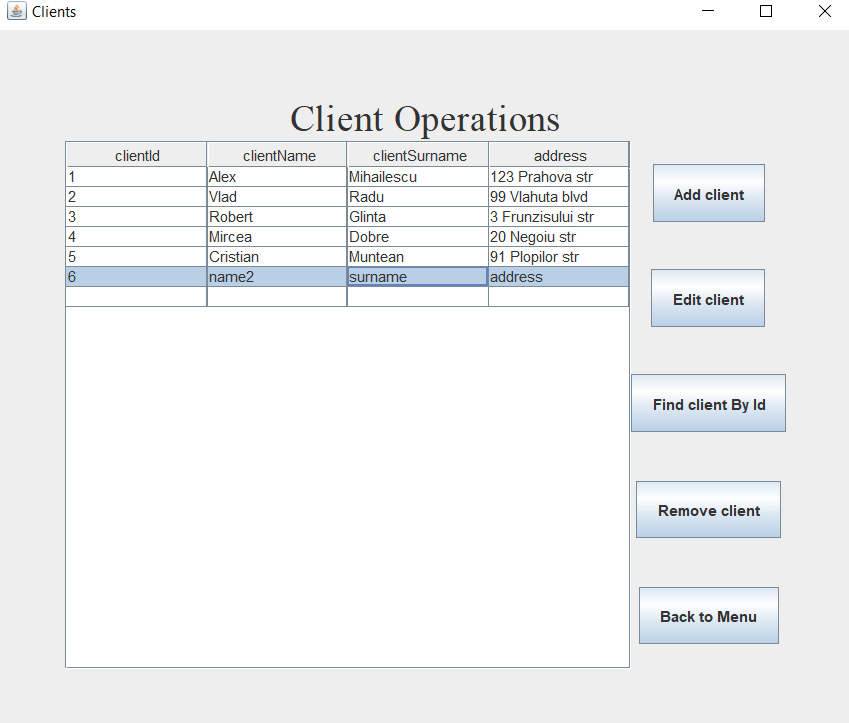
If one of the inputs are not valid, the pop-up window will not close automatically and no action will be performed on the database.

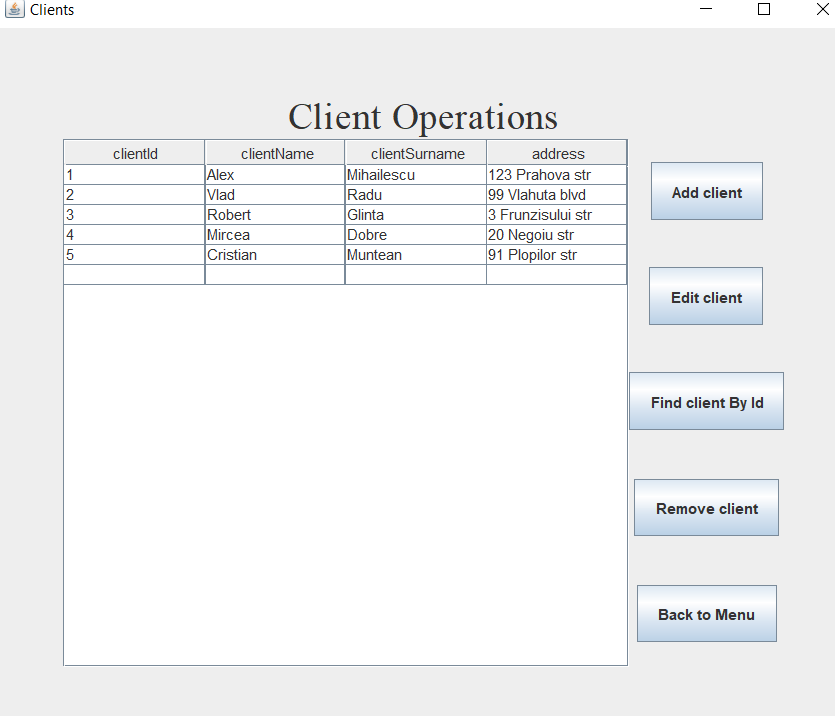
In the “Edit” option, I selected a user from the table, then clicked the “Edit” button and filled some of the fields with new values, then clicked “Submit”. The updated results can be seen in the table once the pop-up window closes automatically. If one of the inputs is not valid, the pop-up window will not close and no action will be performed on the database.



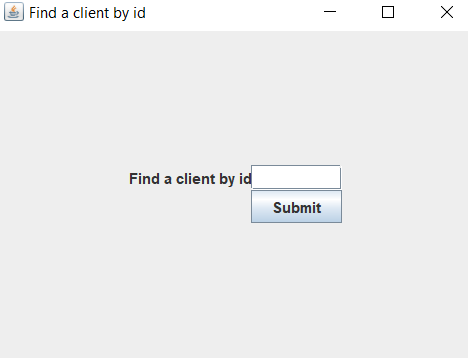


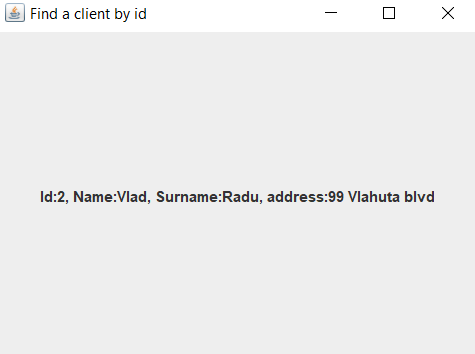
For the “Remove” option, I selected a client from the table, then clicked the “Remove client” button. If there is no row selected, nothing will happen.

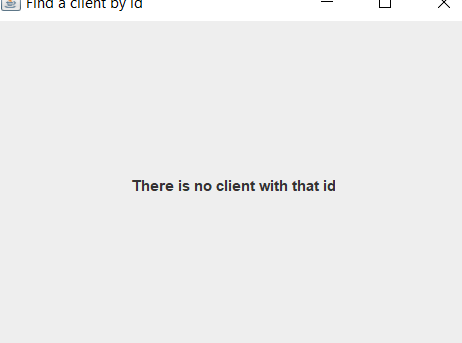




For the “Find client by id” I clicked on the “Find client by Id”, then filled the text field with a value and pressed submit. If there is no client with the mentioned id, a message will be displayed.



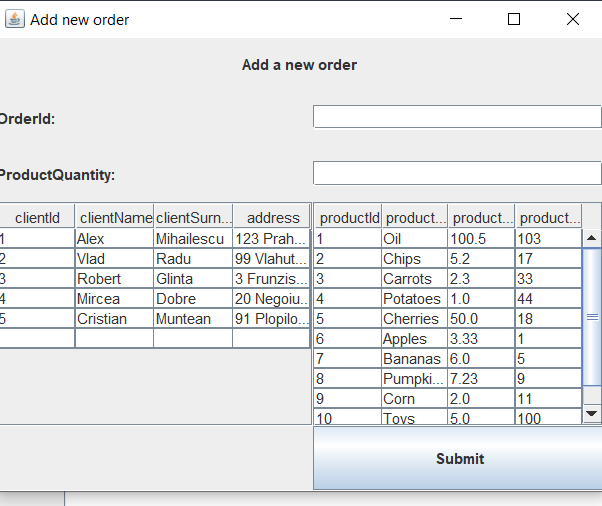




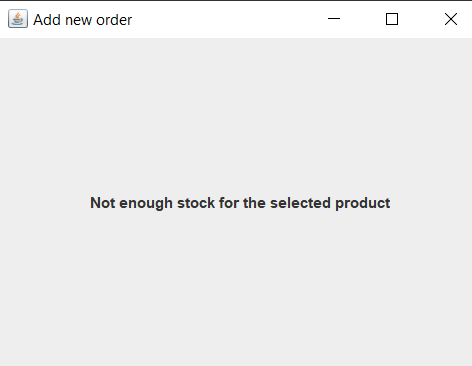
The operations on the “Product” table are the same as the ones on the “Client table”.

The operations on the “ProductOrder” table that are different from the “Client” table are the “Add order” and the “Print bills”.

When selecting the “Add order” operation, the pop-up window will be different.



If the user writes invalid inputs, a “not enough stock” window will pop-up.



# Conclusions

By doing this project, I managed to learn how to work with databases using Java and how to use the reflection technique. Some good examples of the use of reflection technique in this project are the “createTable” method and the “AbstractDAO” and the operations classes.

# Bibliography

* <https://docs.oracle.com/> - for learning how to work with various layouts and graphical user interface components
* <https://stackoverflow.com/> - for various bugs that I encountered during the development
* Programming Techniques in Java – Lectures of prof Cristina Bianca Pop – for familiarizing myself with the reflection technique