DOCUMENTATION

ASSIGNMENT 3

WAREHOUSE MANAGEMENT APPLICATION

STUDENT'S NAME: SANDOR SERBAN VLAD
GROUP: 30422

CONTENTS

1. The assignment's objective

The assignment's objective is given by the following problem: managing the products, the clients and the orders for a warehouse using hand-written registries is difficult and time-consuming. So the objective is actually the solution to the previously mentioned problem: to design and implement an application for managing the client orders for a warehouse.

The sub-objectives that need to be completed in order to reach the main goal are:

- Analyze the problem and identify requirements
- Design the orders management application
- Implement the orders management application
- Test the orders management application

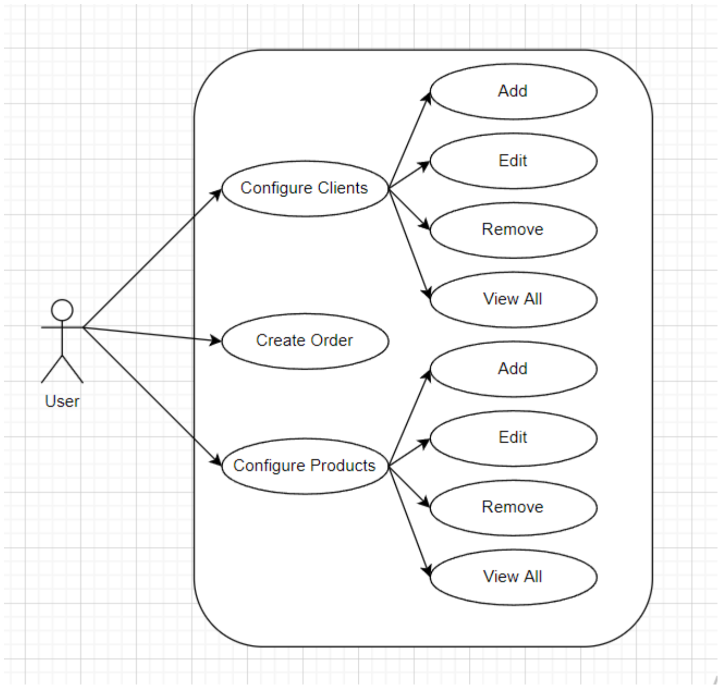
2. The problem's analysis, modelling, scenarios and use-cases

In order to complete the implementation of the simulation, there have to be defined some functional requirements, such as:

- the application has to be able to respond to the user's interactions and accept the user's input
- the application should be able to access the database corresponding to the warehouse
- the application should execute various queries based on the desired action of the user
- the application should execute actions such as add/remove/update/view all clients/products
- the application should be able to create orders based on existing client IDs, product IDs and the desired quantity
- the application has to manipulate the user-entered data in order to create client, product or order object types.

There can also be defined some non-functional requirements, as the following:

- the graphical user interface should be easy to understand.
- the graphical user interface should not ask the user to perform any unnecessary actions.
- the application should navigate through the GUI windows easily and efficiently.
- the bill that is created through the creation of an order should be easy to understand and read.



Use Cases:

- Use Case: Add Client
  Primary Actor: the user
  Main Success Scenario:
  1. The user clicks on the button "Configure Clients"
  2. The user selects the "Add Client" option
  3. The user enters the required data for the client entry: id, name, address and age.
  4. The user selects the "Add Client" option again.

- Use Case: Update Client

Primary Actor: the user
Main Success Scenario:
1. The user clicks on the button “Configure Clients”
2. The user selects the “Update Client” option
3. The user enters both the updated data and the one that is not changed.
4. The user selects the “Update Client” option again.

• Use Case: Remove Client
Primary Actor: the user
Main Success Scenario:
1. The user clicks on the button “Configure Clients”
2. The user selects the “Remove Client” option
3. The user enters the id of the client that is to be removed (other data can be inserted, but it is not mandatory)
4. The user selects the “Remove Client” option again.

• Use Case: View All Clients
Primary Actor: the user
Main Success Scenario:
1. The user clicks on the button “Configure Clients”
2. The user selects the “View All Clients” option
3. The table with all the clients from the database shows up.

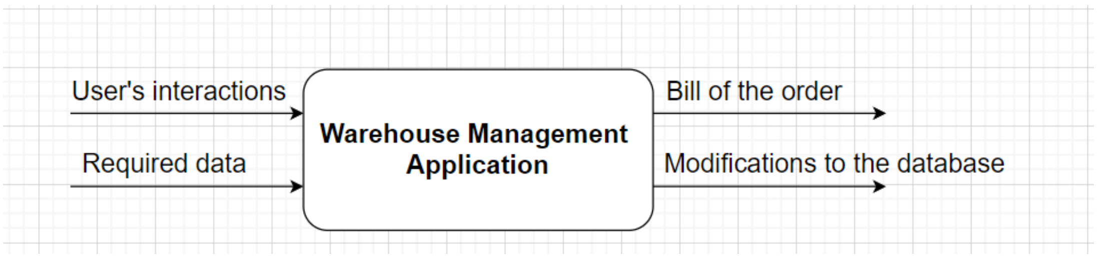
• The same 4 use cases can be taken for the products as well, having the same steps but instead of the required fields for the client, there will be product id, name of the product, the quantity and the price per unit.

• Use Case: Create an Order
Primary Actor: the user
Main Success Scenario:
1. The user clicks on the button labeled “Create an order”
2. The user selects the id of the product that is to be bought
3. The user selects the id of the client that is making that purchase
4. The user enters the desired quantity of that product
5. The user selects the “Place Order” option.

Alternative Sequence: the user enters a quantity value greater than the quantity in stock
1. The application creates a pop-up message that the stock is insufficient and that they should try again
2. The scenario returns to step 2 of the main success scenario.

3. The Design

a. The overall system’s design


User's interactions
Required data
Warehouse Management Application
Bill of the order
Modifications to the database

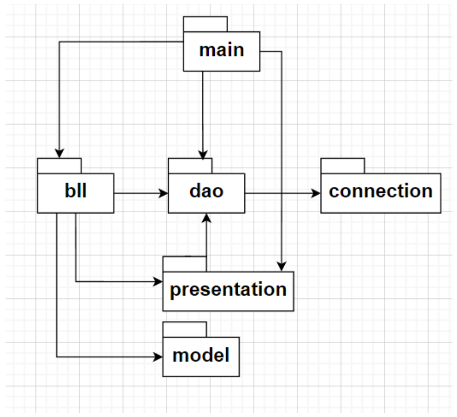
The application takes as input the interactions from the user with the various buttons to navigate throughout the GUI windows, and when needed the data that represents the fields of the selected items to be configured. The application manipulates the inputs and creates as output the bill made by the creation of an order, a table with all the products/clients of the database or just manipulates the entries in the database.

b. The division of the project

The solution is divided into 6 packages, and those are:

-main: Main Class, starting the whole program
-presentation: AllView, ClientBoxView, ClientView, Controller, OrderView, ProductBoxView, StartView, ProductView used for the Graphical User Interfaces and also the connections between them
-model: Client, Order, Product stating the base models of the tables in the database
-connection: ConnectionFactory used for establishing the connection to the database
-dao (Data Access Object): AbstractDAO, ClientDAO, ProductDAO, OrderDAO
-bll (Business Logic): ClientBLL, ProductBLL, OrderBLL used for handling the logic in the program

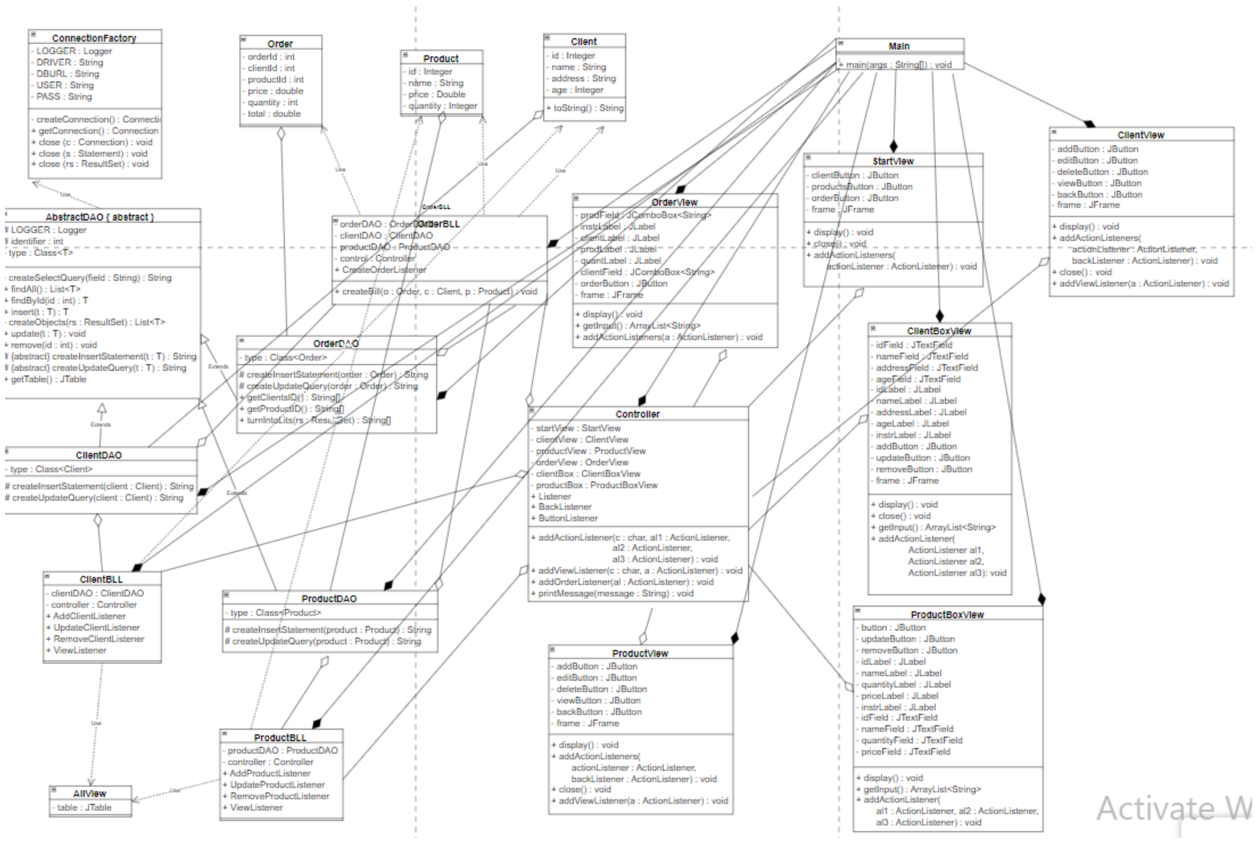
The relationships between these packages are shown in the diagram below:



Moving on to the classes, they consist of:

- Main, that instantiates every class of the presentation package (except for the AllView), all the bll classes and every dao class besides AbstractDAO.
- AbstractDAO, which uses the static methods of ConnectionFactory.
- ClientDAO, which extends AbstractDAO and has as an instance variable a Client object.
- ProductDAO, which extends AbstractDAO and contains an instance variable of type Product.
- OrderDAO, which extends AbstractDAO and has an attribute of type Order.
- ClientBLL, that has as instance variables an object of type ClientDAO and one of type Controller; it also instantiates a new AllView object.
- ProductBLL, that is very similar to ClientBLL but instead contains ProductDAO.
- OrderBLL, that has as attributes a ClientDAO, a ProductDAO, an OrderDAO and a Controller.
- ConnectionFactory
- Client
- Product
- Order
- StartView
- ClientView
- ProductView
- OrderView, that uses the static methods from OrderDAO
- ClientBoxView
- ProductBoxView
- AllView
- Controller, that has as instance variables all the classes from the presentation package, except for the AllView

The relations between the classes are represented in the diagram below:


ConnectionFactory
- LOGGER : Logger
- DRIVER : String
- DBURL : String
- USER : String
- PASS : String
- createConnection() : Connection
+ getConnection() : Connection
+ close (c : Connection) : void
+ close (s : Statement) : void
+ close (rs : ResultSet) : void
Order
- orderId : int
- clientId : int
- productId : int
- price : double
- quantity : int
- total : double
Product
- id : Integer
- name : String
- price : Double
- quantity : Integer
Client
- id : Integer
- name : String
- address : String
- age : Integer
+ toString() : String
Main
+ main(args : String[]) : void
ClientView
- addButton : JButton
- editButton : JButton
- deleteButton : JButton
- viewButton : JButton
- backButton : JButton
- frame : JFrame
+ display() : void
+ addActionListeners(actionListener : ActionListener, backListener : ActionListener) : void
+ close() : void
+ addViewListener(a : ActionListener) : void
StartView
- clientButton : JButton
- productsButton : JButton
- orderButton : JButton
- frame : JFrame
+ display() : void
+ close() : void
+ addActionListeners(actionListener : ActionListener) : void
AbstractDAO { abstract }
LOGGER : Logger
identifier : int
- type : Class<T>
- createSelectQuery(field : String) : String
+ findAll() : List<T>
+ findById(id : int) : T
+ insert(t : T) : T
- createObjects(rs : ResultSet) : List<T>
+ update(t : T) : void
+ remove(id : int) : void
(abstract) createInsertStatement(t : T) : String
(abstract) createUpdateQuery(t : T) : String
+ getTable() : JTable
OrderBLL
- orderDAO : OrderDAO
- clientDAO : ClientDAO
- productDAO : ProductDAO
- control : Controller
+ CreateOrderListener
+ createBill(o : Order, c : Client, p : Product) : void
OrderView
- prodField : JComboBox<String>
- instrLabel : JLabel
- clientLabel : JLabel
- prodLabel : JLabel
- quantLabel : JLabel
- clientField : JComboBox<String>
- orderButton : JButton
- frame : JFrame
+ display() : void
+ getInput() : ArrayList<String>
+ addActionListeners(a : ActionListener) : void
OrderDAO
- type : Class<Order>
# createInsertStatement(order : Order) : String
# createUpdateQuery(order : Order) : String
+ getClientsID() : String[]
+ getProductID() : String[]
+ turnIntoLits(rs : ResultSet) : String[]
ClientBoxView
- idField : JTextField
- nameField : JTextField
- addressField : JTextField
- ageField : JTextField
- idLabel : JLabel
- nameLabel : JLabel
- addressLabel : JLabel
- ageLabel : JLabel
- instrLabel : JLabel
- addButton : JButton
- updateButton : JButton
- removeButton : JButton
- frame : JFrame
+ display() : void
+ close() : void
+ getInput() : ArrayList<String>
+ addActionListener(ActionListener al1, ActionListener al2, ActionListener al3) : void
Controller
- startView : StartView
- clientView : ClientView
- productView : ProductView
- orderView : OrderView
- clientBox : ClientBoxView
- productBox : ProductBoxView
+ Listener
+ BackListener
+ ButtonListener
+ addActionListener(c : char, al1 : ActionListener, al2 : ActionListener, al3 : ActionListener) : void
+ addViewListener(c : char, a : ActionListener) : void
+ addOrderListener(al : ActionListener) : void
+ printMessage(message : String) : void
ClientDAO
- type : Class<Client>
# createInsertStatement(client : Client) : String
# createUpdateQuery(client : Client) : String
ClientBLL
- clientDAO : ClientDAO
- controller : Controller
+ AddClientListener
+ UpdateClientListener
+ RemoveClientListener
+ ViewListener
ProductDAO
- type : Class<Product>
# createInsertStatement(product : Product) : String
# createUpdateQuery(product : Product) : String
ProductView
- addButton : JButton
- editButton : JButton
- deleteButton : JButton
- viewButton : JButton
- backButton : JButton
- frame : JFrame
+ display() : void
+ addActionListeners(actionListener : ActionListener, backListener : ActionListener) : void
+ close() : void
+ addViewListener(a : ActionListener) : void
ProductBoxView
- button : JButton
- updateButton : JButton
- removeButton : JButton
- idLabel : JLabel
- nameLabel : JLabel
- quantityLabel : JLabel
- priceLabel : JLabel
- instrLabel : JLabel
- idField : JTextField
- nameField : JTextField
- quantityField : JTextField
- priceField : JTextField
+ display() : void
+ getInput() : ArrayList<String>
+ addActionListener(al1 : ActionListener, al2 : ActionListener, al3 : ActionListener) : void
ProductBLL
- productDAO : ProductDAO
- controller : Controller
+ AddProductListener
+ UpdateProductListener
+ RemoveProductListener
+ ViewListener
AllView
- table : JTable
Use
Extends
Activate W

c. Data Structures Used

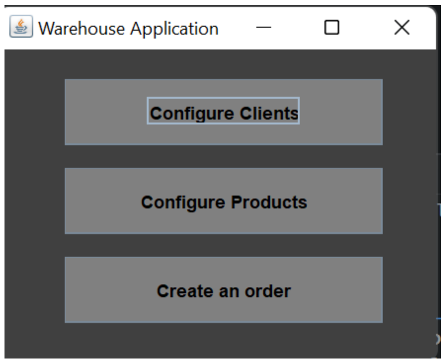
For the implementation of this solution, I used ArrayLists and Lists to manage data easier and to have a mean to send it all in one step from class to class.

d. The Graphical User Interface

The Graphical User Interface consists of multiple windows, basically one for each activity of the user:

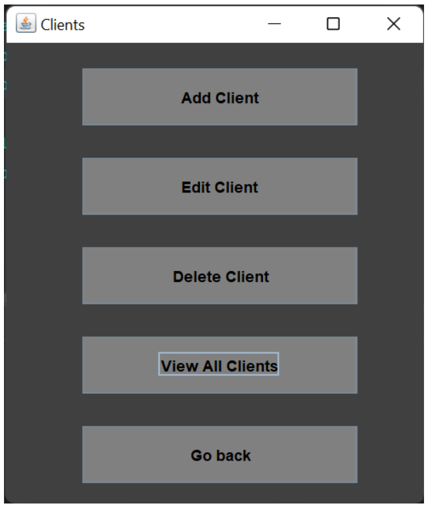
- Selecting the items to be configured, having the options:

  - Configure Clients
  - Configure Products
  - Create an order



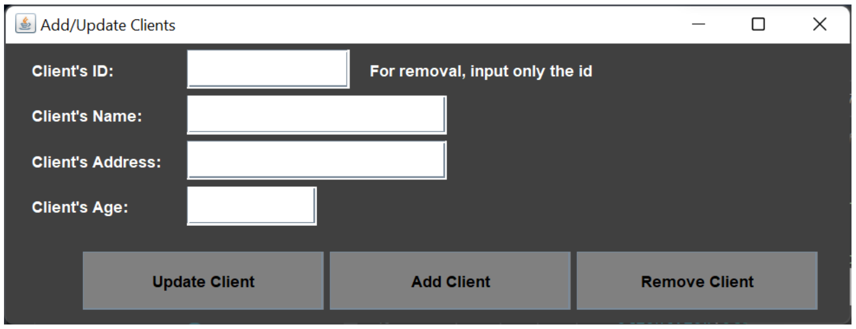
- Selecting what operation should be executed on clients, with the following options:

- Add Client
- Edit Client
- Delete Client
- View All Clients
- Go back (to the previous window)

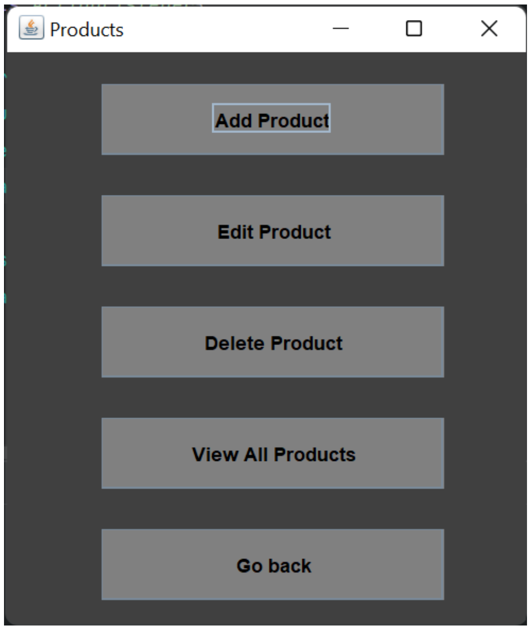


- Entering the data required for a client entry and selecting the final operation, having as fields:
  - client's id
  - client's name
  - client's address
  - client's age,

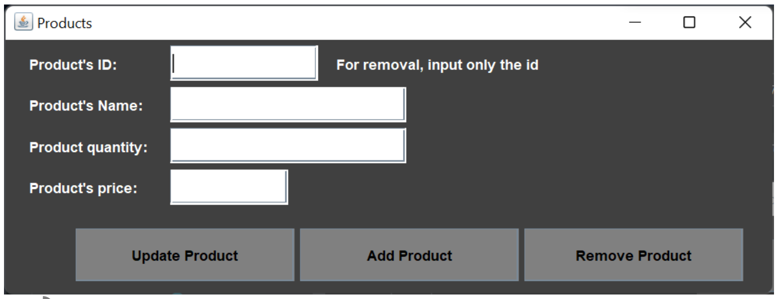
  and having the options:
  - Update Client
  - Add Client
  - Remove Client



- Selecting the options for configuring products, such as:
  - Add Product
  - Edit Product
  - Delete Product
  - Go back (to the first window)

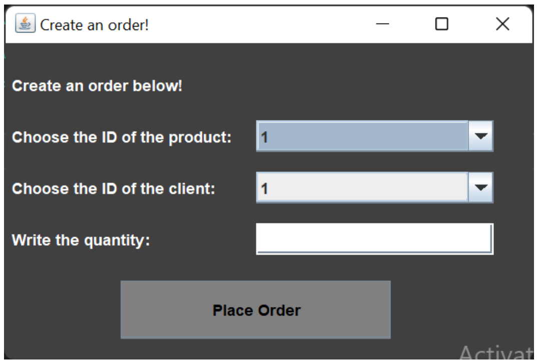


- Entering the data required for a product entry and selecting the final operation, having as fields:
  - product's id
  - product's name
  - product's quantity
  - product's price



and having the options:
- Update Product
- Add Product
- Remove Product

- Creating an order, by selecting the ID of the product and the one of the client from the corresponding drop-down lists and entering the desired quantity, then finalizing the order by pressing the button.



4. The implementation

1. Main

- contains the main method that instantiates the necessary objects and starts the application.

2. Client

- represents the Client table from the database, having the required fields (id, name, address and age).

3. Product

- represents the Product table from the database, having the required fields (id, name, price, quantity).

4. Order

- represents the Order table from the database, having the required fields (orderId, clientId, productId, quantity, price, total).

5. AbstractDAO

- it has as a field an object of type Class<T>, which will be parametrized for other classes
- contains for relevant methods:
  - findById, findAll, insert, remove, update, that gets the connection to the database, creates the corresponding query for said operations and execute them; they return (if that is the case) the object(s) that were to be selected from the tables.
  - createObjects, that takes as a parameter a ResultSet object and using reflection, generates the corresponding objects and return them.
  - createInsertStatement and createUpdateQuery, that are abstract methods that are to be used in the classes that extend this one
  - getTable, that uses the findAll method to retrieve all the entries from a table, and then generates the header of the table using reflection and filling the data of the table into a new JTable object that is returned.

6. ClientDAO

- extends AbstractDAO, having then the same methods but its field is now an object of type Class<Client>.
- the abstract methods from AbstractDAO are overridden to create the specific queries for the client table.

7. ProductDAO

- the same as ClientDAO, but the field is of type Class<Product>.

8. OrderDAO

- the field is now of type Class<Order> and the createInsertStatament is overridden too.

- it also contains a method turnIntoList to turn a ResultSet object into an array of String objects.
- the latter method is used in getClientsID and getProductID to get all the ids of clients/products placed in an array.

9. ConnectionFactory

- it contains as fields a Logger objects and several String ones to realize the connection to the database.
- it has methods to fetch the connection and to close it as well.

10. Controller

- it contains a couple of inner classes that implement the ActionListener interface, that result in the navigation from one GUI window to the other

11. ClientBLL

- it has as instance variables a ClientDAO object and a Controller one.
- it contains several inner classes that implement the ActionListener interface and realize the operations selected by the user:
  - AddClientListener : fetches the inputs from the ClientBoxView window, instantiates a Client object and uses it as a parameter to the insert method of the ClientDAO field.
  - UpdateClientListener : the same as the previous one but calls the update method.
  - RemoveClientListener : gets the input from the GUI and uses the id to call upon the delete method.
  - ViewListener : instantiates a new object of type AllView, and in its constructor the getTable method is called.

12. ProductBLL

- has the same behavior as the ClientBLL class but instead of Client objects, there are used instead Product ones.

13. OrderBLL

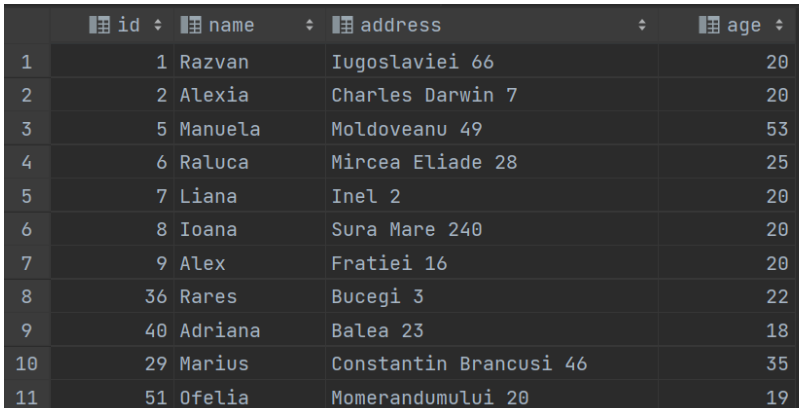
- it contains an inner class that implements the ActionListener interface, CreateOrderListener, that instantiates a Product and a Client object by calling the findById method, and by using the fields of the previously mentioned objects a new Order object is instantiated; it is also checked if the quantity desired is in stock, otherwise it would create a message dialog to warn the user; finally, it calls upon the createBill method.
- it also has a createBill method, that is used for creating a file and writing in it (with the help of a BufferedWriter) the actual date, the name of the client, the name of the product, the quantity bought, the price per unit and the total that was paid, with the help of the fields of the Order object that is set as a parameter.

14. The Graphical User Interface Classes

- the GUI is completed by using classes that extend either JFrame or JPanel (such as StartView, ClientView, OrderView, ProductView, ClientBoxView, ProductBoxView, and Allview) and by using various classes that implement the ActionListener interface the navigation between them is done.

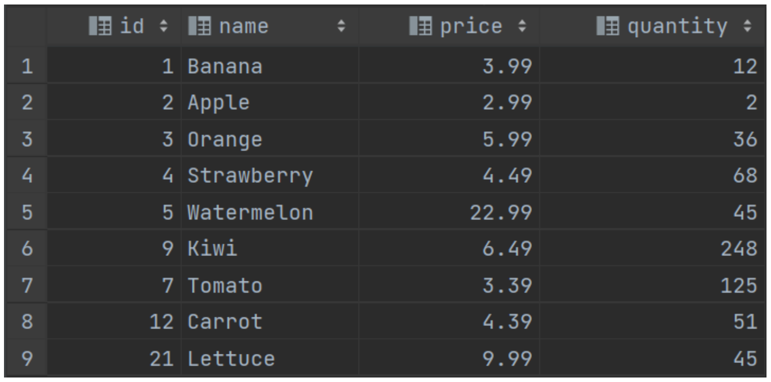
15. The database

- the database was concepted in MySQL and consists of three tables: client, product and order.
- the client table has as columns the id, name, address and age of the client :



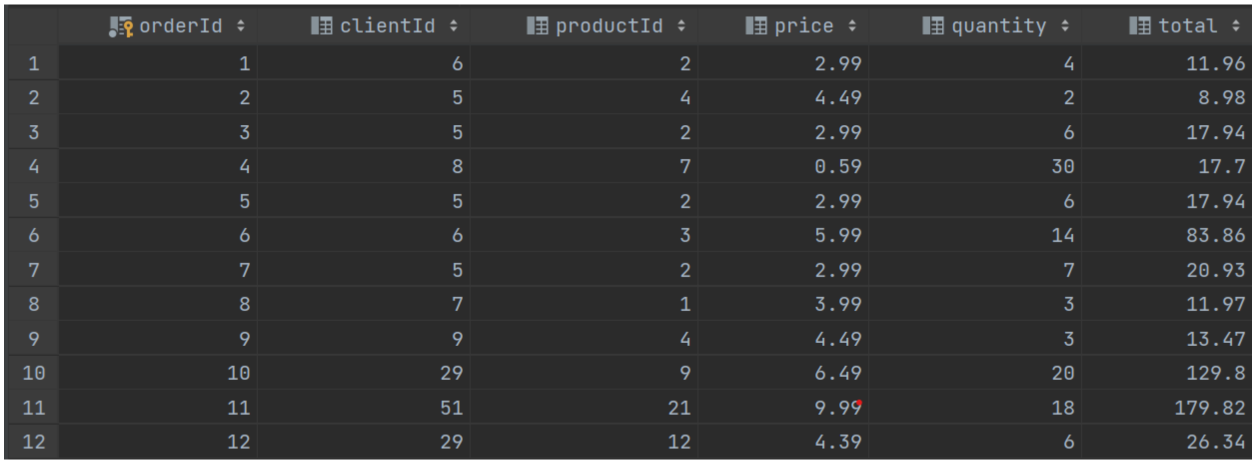
	id	name	address	age
1	1	Razvan	Iugoslaviei 66	20
2	2	Alexia	Charles Darwin 7	20
3	5	Manuela	Moldoveanu 49	53
4	6	Raluca	Mircea Eliade 28	25
5	7	Liana	Inel 2	20
6	8	Ioana	Sura Mare 240	20
7	9	Alex	Fratiei 16	20
8	36	Rares	Bucegi 3	22
9	40	Adriana	Balea 23	18
10	29	Marius	Constantin Brancusi 46	35
11	51	Ofelia	Momerandumului 20	19

- the product table has as columns the id, name, price per unit and quantity of the product :



	id	name	price	quantity
1	1	Banana	3.99	12
2	2	Apple	2.99	2
3	3	Orange	5.99	36
4	4	Strawberry	4.49	68
5	5	Watermelon	22.99	45
6	9	Kiwi	6.49	248
7	7	Tomato	3.39	125
8	12	Carrot	4.39	51
9	21	Lettuce	9.99	45

- the order table has as columns the id of the order, id of the client, the total to be paid and the id, price per unit and quantity of the product :



	orderId	clientId	productId	price	quantity	total
1	1	6	2	2.99	4	11.96
2	2	5	4	4.49	2	8.98
3	3	5	2	2.99	6	17.94
4	4	8	7	0.59	30	17.7
5	5	5	2	2.99	6	17.94
6	6	6	3	5.99	14	83.86
7	7	5	2	2.99	7	20.93
8	8	7	1	3.99	3	11.97
9	9	9	4	4.49	3	13.47
10	10	29	9	6.49	20	129.8
11	11	51	21	9.99	18	179.82
12	12	29	12	4.39	6	26.34

5. Results

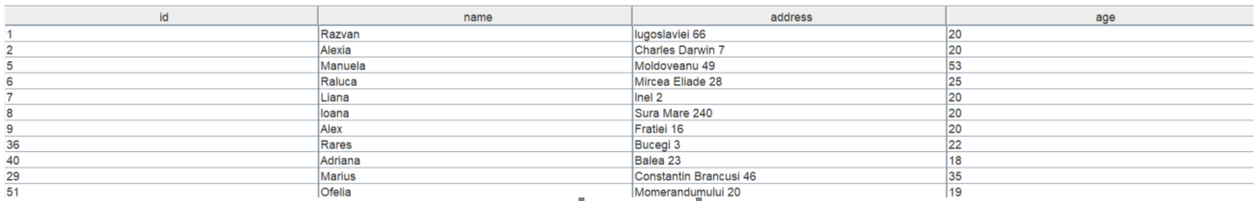
The results of the application can be divided into 4 categories:

- the database modification
- the table viewing
- the error finding
- the bill creating.

For the database modification, it can be checked that after performing any operations of insert, update, delete in the database that the desired item was configured as previously desired.

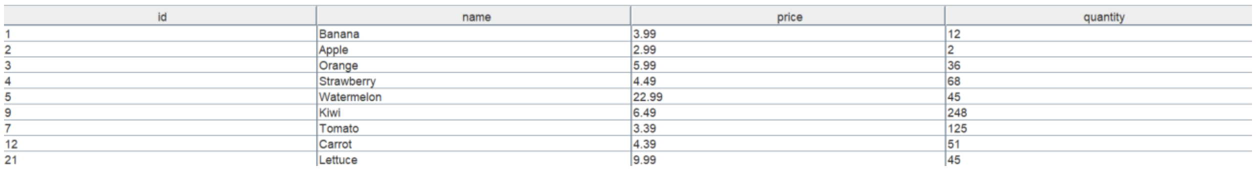
For the table viewing, 2 images will be provided below presenting the case of view all clients and view all products.

1. Client's Table



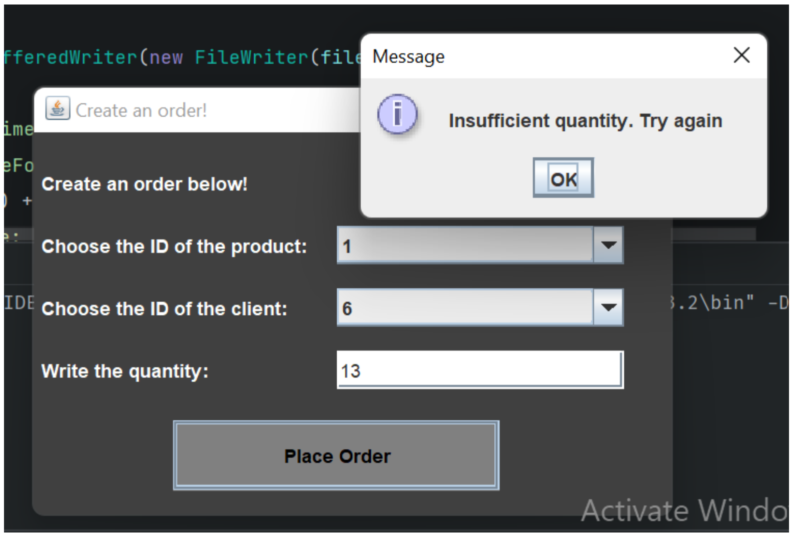
id	name	address	age
1	Razvan	Iugoslaviei 66	20
2	Alexia	Charles Darwin 7	20
5	Manuela	Moldoveanu 49	53
6	Raluca	Mircea Eliade 28	25
7	Liana	Inel 2	20
8	Ioana	Sura Mare 240	20
9	Alex	Fratiei 16	20
36	Rares	Bucegi 3	22
40	Adriana	Balea 23	18
29	Marius	Constantin Brancusi 46	35
51	Ofelia	Momerandumului 20	19

2. Product’s Table



id	name	price	quantity
1	Banana	3.99	12
2	Apple	2.99	2
3	Orange	5.99	36
4	Strawberry	4.49	68
5	Watermelon	22.99	45
9	Kiwi	6.49	248
7	Tomato	3.39	125
12	Carrot	4.39	51
21	Lettuce	9.99	45

As for the error finding, this can be checked if while creating a new order, the user inputs a quantity greater than the one in stock (in the table). For the following example we can see that in the table there are 12 bananas and if the user requests 13 the error will show up.



Message

Insufficient quantity. Try again

Create an order!

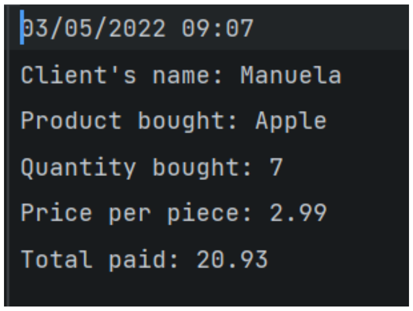
Create an order below!

Choose the ID of the product: 1

Choose the ID of the client: 6

Write the quantity: 13

As for the bill creating, when the “Place Order” button is pressed and the inputs are correct, a text file representing the bill is created, as in the following image:



```
03/05/2022 09:07
Client's name: Manuela
Product bought: Apple
Quantity bought: 7
Price per piece: 2.99
Total paid: 20.93
```

It can be seen that in the bill there is printed the date when the order was completed, the name of the client, the name of the product that was bought, the quantity bought, price per unit and the total paid.

6. Conclusions

Working on this assignment I have got acquainted to reflection and got to use it and get the basics of it, I also revised working with databases, establishing connections, creating queries and executing them and working with result sets. Some modifications I would bring would be an update to the GUIs, getting them a better look and also improve the efficiency of the program overall.

7. Bibliography

1. https://www.baeldung.com/java-jdbc
2. http://www.mkyong.com/jdbc/how-to-connect-to-mysql-with-jdbc-driver-java/ https://dzone.com/articles/layers-standard-enterprise
3. http://tutorials.jenkov.com/java-reflection/index.
4. https://www.baeldung.com/java-pdf-creation
5. https://www.baeldung.com/
6. https://dev.mysql.com/doc/workbench/en/wb-admin-export-import-management.html