MINISTERUL EDUCAȚIEI NAȚIONALE

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

Assignment 3
Order Processing
Warehouse Management
- Documentation –

24.04.2018

Linca Paul Tudor

Group: 30422

1. Task

Consider an application OrderManagement for processing customer orders for a warehouse. Relational databases are used to store the products, the clients and the orders.

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.

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.

d. Implement a system of utility programs for reporting such as: under-stock, totals, filters, etc.

2. Brief description:

This assignment consists of developing an application which simulates a kind of online shop that works based on a database. Thus, the application will be linked to a database and the processing of the data will be done in the dao classes.

The application will use the following cases:

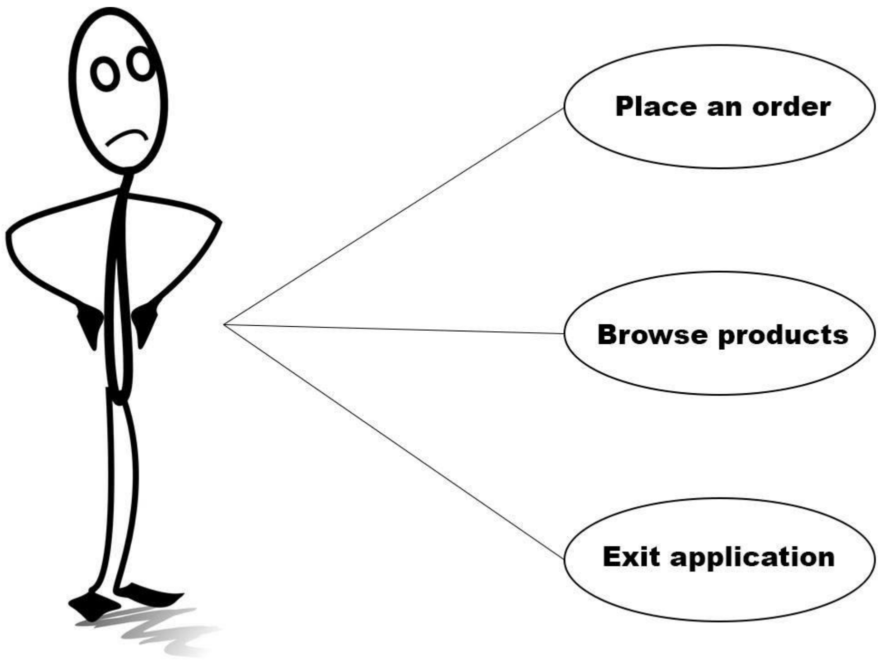
- Model classes - represent the data models of the application (for example Order, Customer, Product);
- Business Logic classes - contain the application logic (for example OrderProcessing, WarehouseAdmin, ClientAdmin);
- Presentation classes – classes that contain the graphical user interface

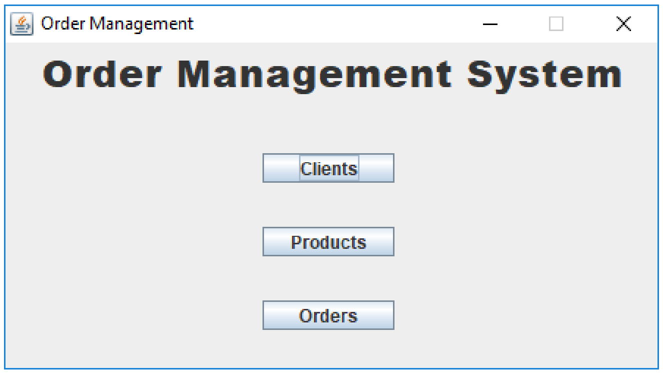
- Data access classes - classes that contain the access to the database

With all this in mind, the assignment simply comes down to:

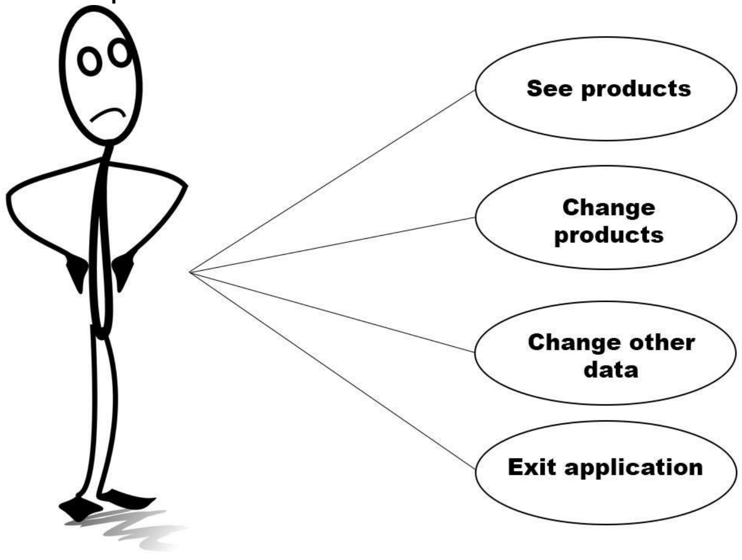
- Building a nice looking and intuitive graphical interface;
- Building a database
- Linking the application to the database
- Implement the database access layer
- Use reflection

3. User Diagram





The user is presented with a simple and smooth graphical interface. The main menu simply consists of 3 buttons labled with the section pressing said buttons takes the user to. Thus, we have “clients” for client processes, “products” for product processes and “orders” for order placements. So the user can place orders, make a new account and browse through the available products.



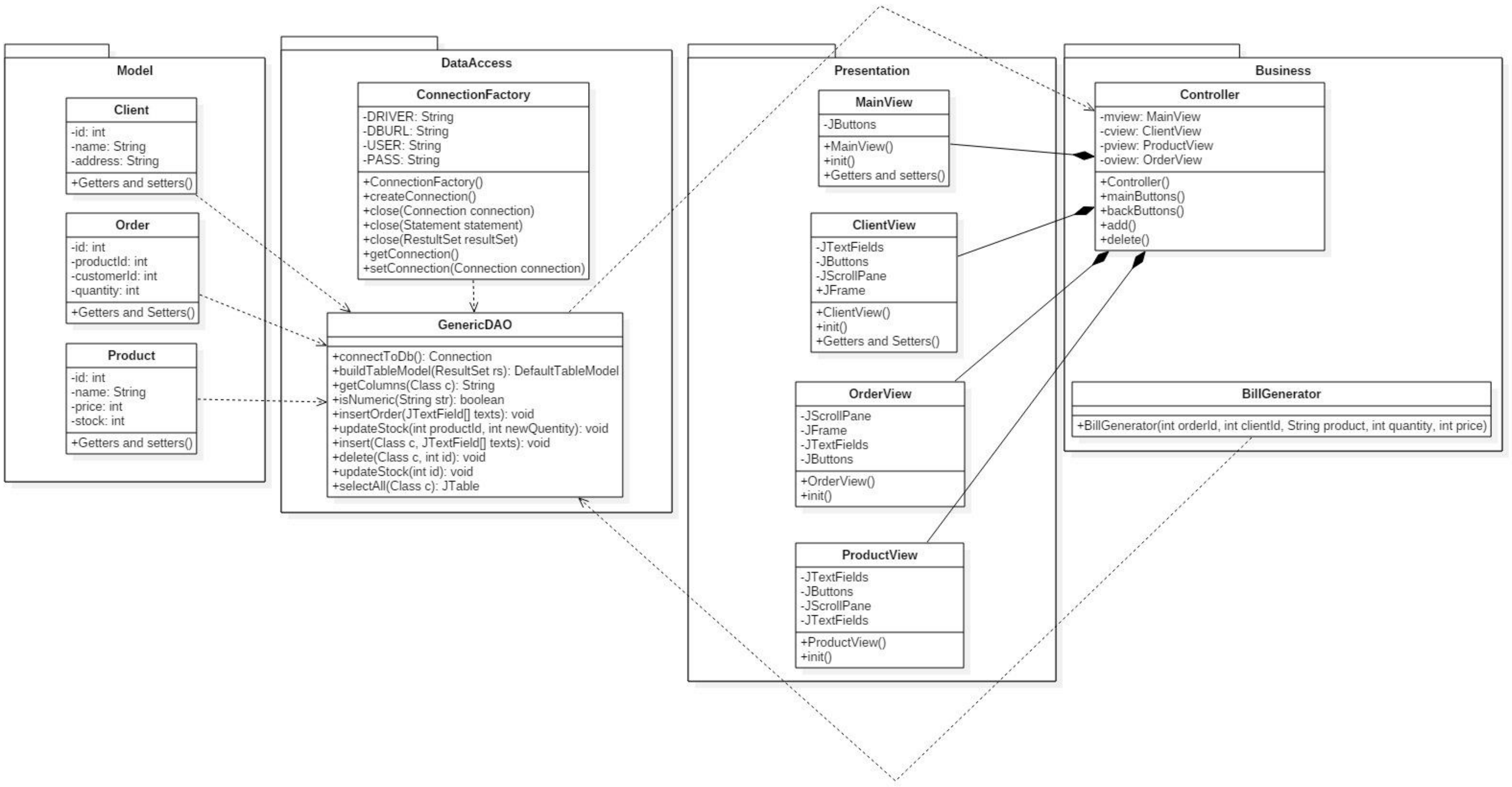
The admin has the same GUI as the user but has more permissions as seen in the figure above.

In my opinion, the graphical user interface is pretty straight forward in how the user is supposed to use the system.

4. Class diagram

A class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects.

Even though until now I used the Model – Controller – View (MCV) architectural pattern but for this assignment another architecture was imposed. Here’s the class diagram:


Model
Client
-id: int
-name: String
-address: String
+Getters and setters()
Order
-id: int
-productId: int
-customerId: int
-quantity: int
+Getters and Setters()
Product
-id: int
-name: String
-price: int
-stock: int
+Getters and setters()
DataAccess
ConnectionFactory
-DRIVER: String
-DBURL: String
-USER: String
-PASS: String
+ConnectionFactory()
+createConnection()
+close(Connection connection)
+close(Statement statement)
+close(RestultSet resultSet)
+getConnection()
+setConnection(Connection connection)
GenericDAO
+connectToDb(): Connection
+buildTableModel(ResultSet rs): DefaultTableModel
+getColumns(Class c): String
+isNumeric(String str): boolean
+insertOrder(JTextField[] texts): void
+updateStock(int productId, int newQuentity): void
+insert(Class c, JTextField[] texts): void
+delete(Class c, int id): void
+updateStock(int id): void
+selectAll(Class c): JTable
Presentation
MainView
-JButtons
+MainView()
+init()
+Getters and setters()
ClientView
-JTextFields
-JButtons
-JScrollPane
+JFrame
+ClientView()
+init()
+Getters and Setters()
OrderView
-JScrollPane
-JFrame
-JTextFields
-JButtons
+OrderView()
+init()
ProductView
-JTextFields
-JButtons
-JScrollPane
-JTextFields
+ProductView()
+init()
Business
Controller
-mview: MainView
-cview: ClientView
-pview: ProductView
-oview: OrderView
+Controller()
+mainButtons()
+backButtons()
+add()
+delete()
BillGenerator
+BillGenerator(int orderId, int clientId, String product, int quantity, int price)

The imposed architecture was one that is commonly used for applications that work with databases. So for this assignment I used the layered architecture which consists of organizing the project structure into four main categories / packages: presentation, business, data access and model. Each of the layers contains objects related to the particular concern it represents:

- Model classes - represent the data models of the application (for example Order, Customer, Product);
- Business Logic classes - contain the application logic (for example OrderProcessing, WarehouseAdmin, ClientAdmin);
- Presentation classes – classes that contain the graphical user interface;
- Data access classes - classes that contain the access to the database;

One of the most important rules for this architectural pattern is that all the dependencies should go in one direction, from presentation to infrastructure (data access). As it can be seen from the UML diagram this rule is followed in my project: all the dependencies start at the presentation package, go to the business and then to the data access package in which the models go as well.

5. Reflection techniques

For this assignment, it was required that I u se 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 and to create a generic class that contains the methods for accessing the DB: create object, edit object, delete object and find object.

The queries for accessing the DB for a specific object that corresponds to a table will be generated dynamically through reflection

Reflection is a feature in the Java programming language. It allows an executing Java program to examine or "introspect" upon itself and manipulate internal properties of the program. For example, it's possible for a Java class to obtain the names of all its members and display them. Java is one of the only languages in which this feature exists. This makes Java a very powerful programming language and makes reflection a very useful, though tricky and hard to learn, technique.

So, Java Reflection makes it possible to inspect classes, interfaces, fields and methods at runtime, without knowing the names of the classes, methods etc. at compile time. This is the only way the requirements of the assignment could be met. In the next section, in which the classes will be described, it will also be described how I used reflection.

6. Class description

- ### MainView (GUI)

This class represents the main GUI (Graphical User Interface). This is the window which the user will be presented when they first run the program. The private attributes are the GUI’s 3 JButtons which are needed in other classes for the correct execution of the program. They also have getters and setters needed for said classes to access them.

The init() method is a pretty straight forward window building method. It uses the three JButons aswell as a JFrame and a JLabel used to communicate the title of the application. The three JButton will be used for the user to access the other parts/windows of the application which will be described next.

- ClientView (GUI)

Just like in the MainView class, this class is a GUI class which has as attributes the components which are needed in other classes. So the attributes are: 3 JTextFields, 2 JButtons, a JFrame and JScrollPane. While the first 3 types of components are easy to understand I will explain how I used the JScrollPane. The use of the JScrollPane is pretty simple. On this window all the clients will be shown and if there are more clients than the window can show at one time it would be a problem. This is where the scroll pane comes into action. With it, the user will be able to scroll through the clients, thus able to view all of them even if they don’t fit in the window.

Other than that, the methods for this class are the same as in the main window: a Constructor that will call the init() method used for building the window. Since this window is not meant to be closed from the “x” button I added a “back” JButton for the user to be able to return to the main window.

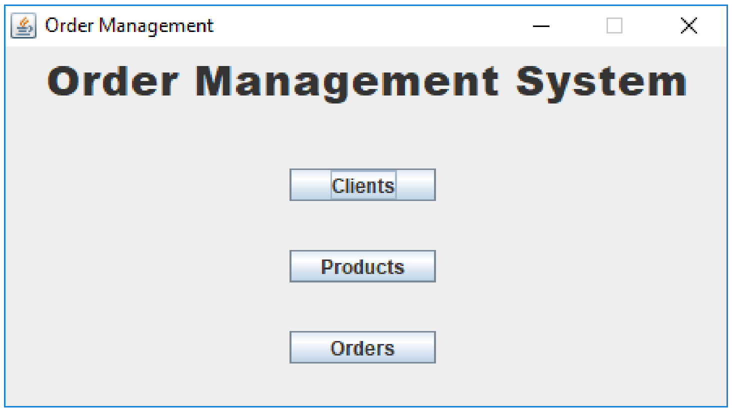
- ProductView (GUI)

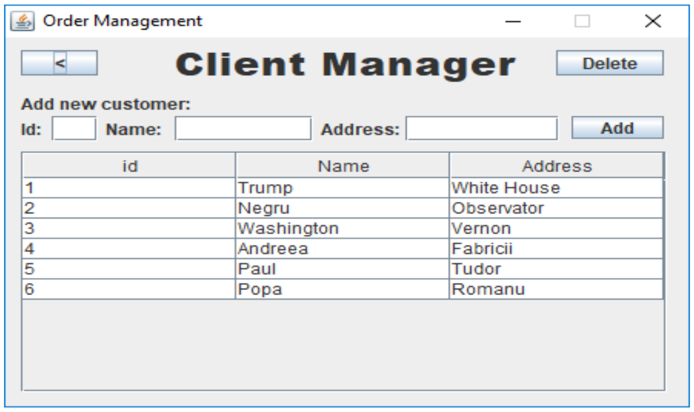
This view is almost an exact carbon copy of the previous one but instead of displaying all the clients, it displays all the products.

- OrderView (GUI)

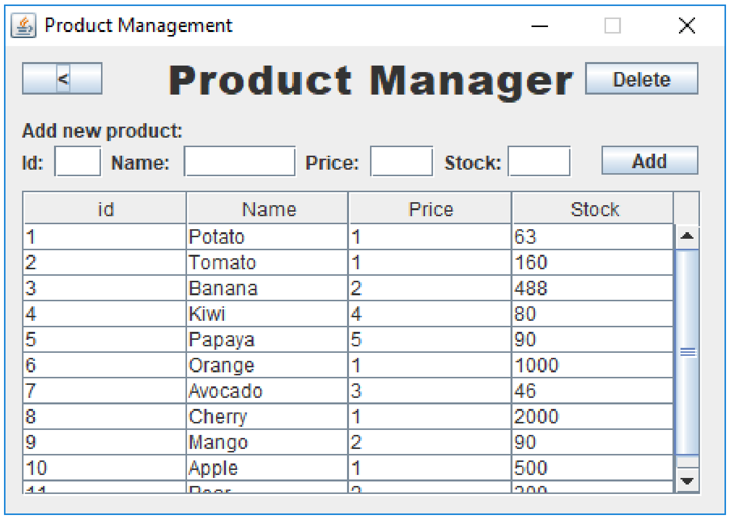
This view is almost an exact carbon copy of the previous two but instead of displaying all the orders, it displays all the products.

These last 4 classes make up the Presentation package. All the JTextFields in the Windows are used to add a new client, product or order depending on the current window.

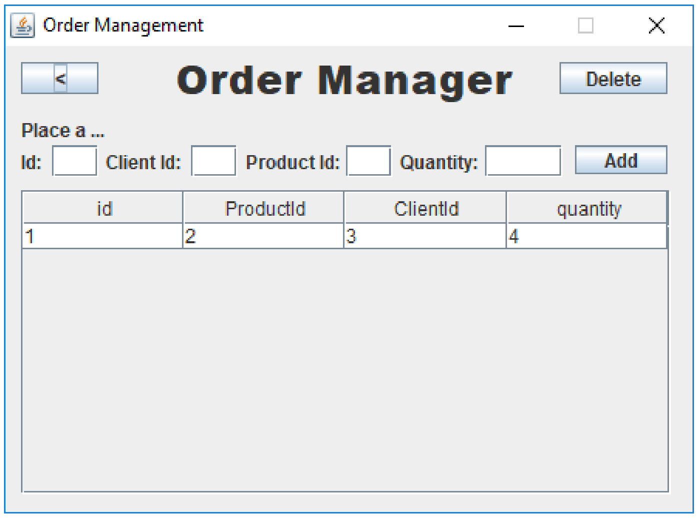




id	Name	Address
1	Trump	White House
2	Negru	Observator
3	Washington	Vernon
4	Andreea	Fabricii
5	Paul	Tudor
6	Popa	Romanu



id	Name	Price	Stock
1	Potato	1	63
2	Tomato	1	160
3	Banana	2	488
4	Kiwi	4	80
5	Papaya	5	90
6	Orange	1	1000
7	Avocado	3	46
8	Cherry	1	2000
9	Mango	2	90
10	Apple	1	500



id	ProductId	ClientId	quantity
1	2	3	4

- Client

The client class basically, as the name would suggest, represents each customer. It has 3 attributes: id, name and address, all the important information that is needed for an online shop to deliver the desired products to the customer.

All the methods for this class are getters and setters for the attributes.

- Product

This class has the same functionality and composition as the Client class. Out of the 4 attributes only 2 of them are not in the Client class: price and stock which are pretty straight forward and an explanation is not needed.

All the methods for this class are getters and setters for the attributes.

- Order

This class represents each of the orders. As attributes, beside the id, it has a productId and a customerId. These two are commonly used in databases. Thus, the productId corresponds to a product and the customerId corresponds to the customer that ordered that product. Other than those there's also a quantity attribute which tells how much of that product is ordered.

All the methods for this class are getters and setters for the attributes.

The previous 3 classes are all part of the model package.

- ConnectionFactory

This class contains the name of the driver (initialized through reflection), the database location (DBURL), and the user and the password for accessing the MySQL Server. The connection to the DB will be placed in a Singleton object.
The class contains methods for creating a connection, getting an active connection and closing a connection, a Statement or a ResultSet.

Basically, this class connects the project to a mySql database.

It has methods for creating the connection as well as for closing it.

- GenericDAO

This class is the most important one out of the whole project. It lets us access and manipulate the data of a database. To build this class I used reflection techniques. While it may not have any attributes, this class uses most of the other classes for it to work properly.

I'll go through each of the methods explaining what they do and how they work:

- Connection createConnection()
  - This class simply connects the project to the desired database using the ConnectionFactory class.
- DefaultDataModel buildTableModel(ResultSet rs)
  - This class builds a table model from the given result set and then returns it. The headers of the models are built from the metadata of the result set which gives us the names of the colums. These names are stored in a Vector. Then another Vector of Vectors is built out of the data received from the database, each element of the main vector being all the attributes of an entity that has been read.
  - After these 2 Vectors are built, a DefaultTableModel with the 2 vectors as parameters is returned for a table to be constructed.

- JTable selectAll(Class c)
    - This method connects to the database and selects all the elements from the table corresponding to the class c from the parameter list. This table will be inserted In the corresponding window's JScrollPane for the user to view.
- String[] getColumns(Class c)
    - This method simply returns a string array that contains all the column names of a table from the database given by the class c
- Boolean isNumeric(String str)
    - This method returns true if the given string corresponds to a number and false if it doesn't.
    - It will be used for constructing the query for inserting elements into a table.
- Void Insert(class c, JTextField[] texts)
    - This method connects to the database and inserts a new row based on the information on the JTextFields given in the parameters.
    - The method uses reflection techniques so that it can be used to insert any row on any table. Of course the table will correspond to whatever the Class c from the parameters is.
    - The isNumeric method is used here to know where to put " ' "s in the query. Thus, if a JTextField holds a String , it will be surrounded with " ' " for the query to work correctly.

- Void delete(Class c, int id)
  - This method connects to the database and deletes a row given by the id in the parameter list
  - This method uses reflection techniques so that it can be used to delete any row on any table given by the class c.
- Void insertOrder(JTextFields[] texts)
  - This method is used to connect to the database and insert a new order. I used a different method for orders because it is needed to decrement the stock of a product each time a new order is placed. Thus, after adding a row to the order table, the corresponding product is selected and its stock is updated using the next method.
- Void updateStock(int productId, int newQuantity)
  - This method connects to the database and updates the stock of a product given by the productId such that it gets the value stored in newQuantity.

The last two classes are part of the DataAccess package.

Controller

This class connects the presentation classes with the dataAccess classes, thus handling all the data manipulation of the database correctly through the application. It's methods implement action listeners for the JButtons of the presentation classes and assigns them the correct functionality:

- ~ mainButtons(): implements action listeners to the buttons on the main view so that each one takes the user to the corresponding window.
- ~ backButtons(): implements action listeners to the "back" buttons on each of the secondary windows such that when pressed they take the user back to the main window
- ~ add(): implements action listeners to the "add" buttons on each of the secondary windows such that when pressed they add a new row on the corresponding table with the information written in the JTextFields
- ~ delete(): implements action listeners to the "delete" buttons on each of the secondary windows such that when pressed they delete the row corresponding to the JTextField that holds the id

• BillGenerator

This class which has only one method, its constructor, generates a bill for each order based on the indormation given on the parameters of the constructor. The bill will be of type .txt.

7. Results

By running the application and using it, it can be seen that after adding or deleting a row on a table, the JTable that shows all the rows will not be updated, the operation is completed successfully. This is a problem that I struggled with a lot and could not resolve. Other than that the application works correctly.

8. Conclusion

In my opinion, just like the other 2 assignments, this project was chosen very wisely. It is a very good way of polishing our OOP skills and of learning 2 new Java techniques: working with a database and reflection.

Further improvements to the program would be:

~ Updating the table upon every operation
~ Have the classes and methods in them take less rows of code
~ Building a better bill, with more data and even displaying it on the application or mailing it to the customer

9. Bibliography

https://en.wikipedia.org

http://www.coned.utcluj.ro/~salomie/PT_Lic/

https://softwareengineering.stackexchange.com

https://www.google.ro

MINISTERUL EDUCAȚIEI NAȚIONALE

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA