Technical University of

Cluj-Napoca

Faculty of Automation

and

Computer Science

Warehouse(Database Connection)

Discipline: Programming Techniques

Gavril Luca

Date: 05.05.2018

Group:30422

1.Main objective:

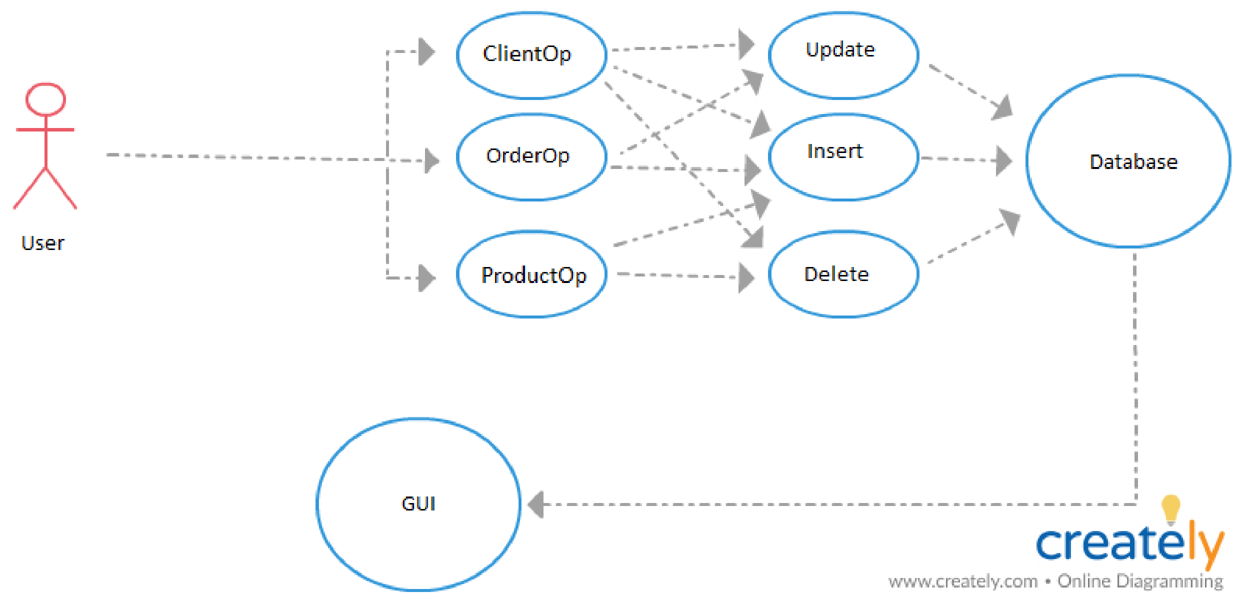
The objective of this homework is to design and implement an Order Management system, by processing the orders of different customers in regard to a warehouse which holds certain products. In order to store data about products, information about customers and orders, we use relational databases, working in MySQL Workbench. It is desired that the project be organized in packages, holding different classes that accomplish multiple roles. In order to implement the system, it will be decomposed into a multiple phase process:

- Data Access operations : communication with the database, in regards to customers, orders and products. The system will need to retrieve, implement data and information in the database. The following operations will be in direct contact with the database:
  Customer operations: Insertion of customers in the database, updating the customer's data in the database, deletion of a customer from the database.
  Product operations: Insertion of products in the database, updating the product's data in the database, deletion of a product from the database.
  Order operations: Insertion of an order in the database, deletion of an order from the database.
- Business Logic operations: in order to have proper data in our database schema, we need to validate, approve of certain operations before they can be ran.
- Graphical interface: The system is designed for an user who is in charge of a warehouse, meaning that he must be able to insert, update or delete customers and/or products, as well as adding or erasing the orders. For this, the system must have an user friendly graphical interface, evident in the system's capabilities.

2.Analysis:

From the get-go, it is evidently important that there must be a connection between the system and a database holding the information. The connection with the database must be implemented in the right way, meaning that there cannot be any sudden drop in connectivity. The graphical user interface also contains the ability to show the customer and product tables, and this has to be automatically updated for the user.

2.1.Use case diagram:



Again, the system is designed for a user that is in control of the warehouse. As it can be seen in the use case diagram, the user has multiple ways to communicate with the customer, order and product tables. This is done by the different operations implemented in the system. In general, the user can insert, update or delete customers and products. He can also add or erase orders.

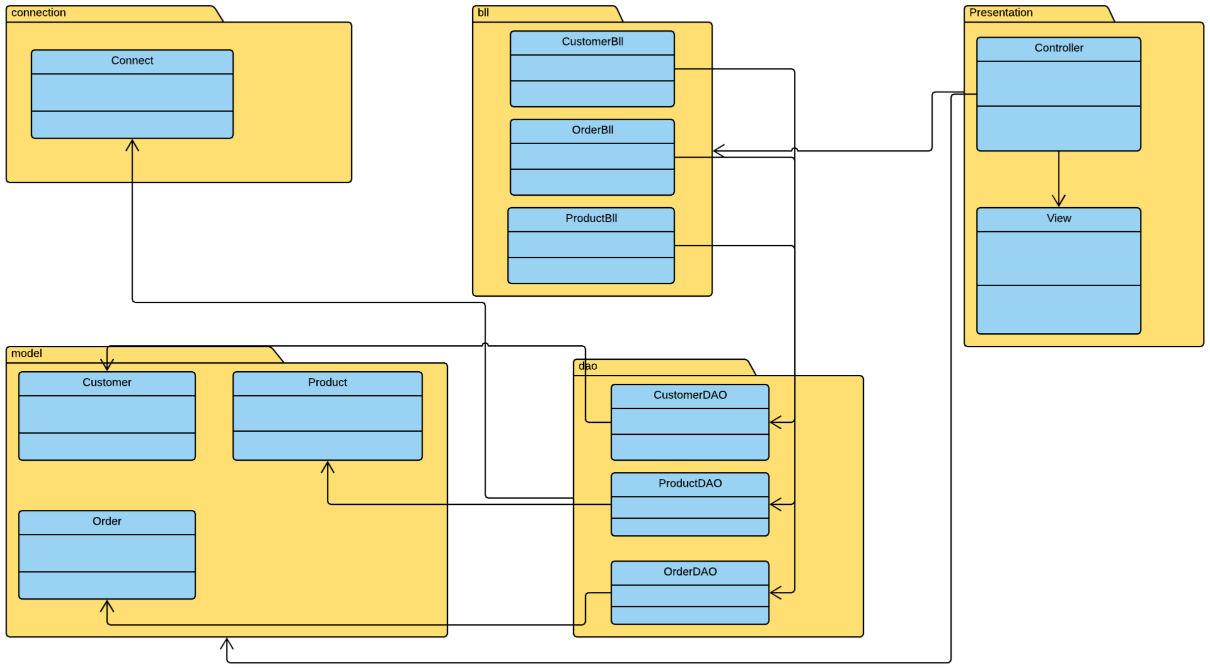
2.2.Assumptions:

Because of the business logic area of our system, and also because of some

attributes of the SQL programming language, there are some realistic restrictions to the data the user can input. However, it is very important that the user is capable of understanding the type of data he must input. Otherwise, the program can stop and the user might experience unpleasant / false results.

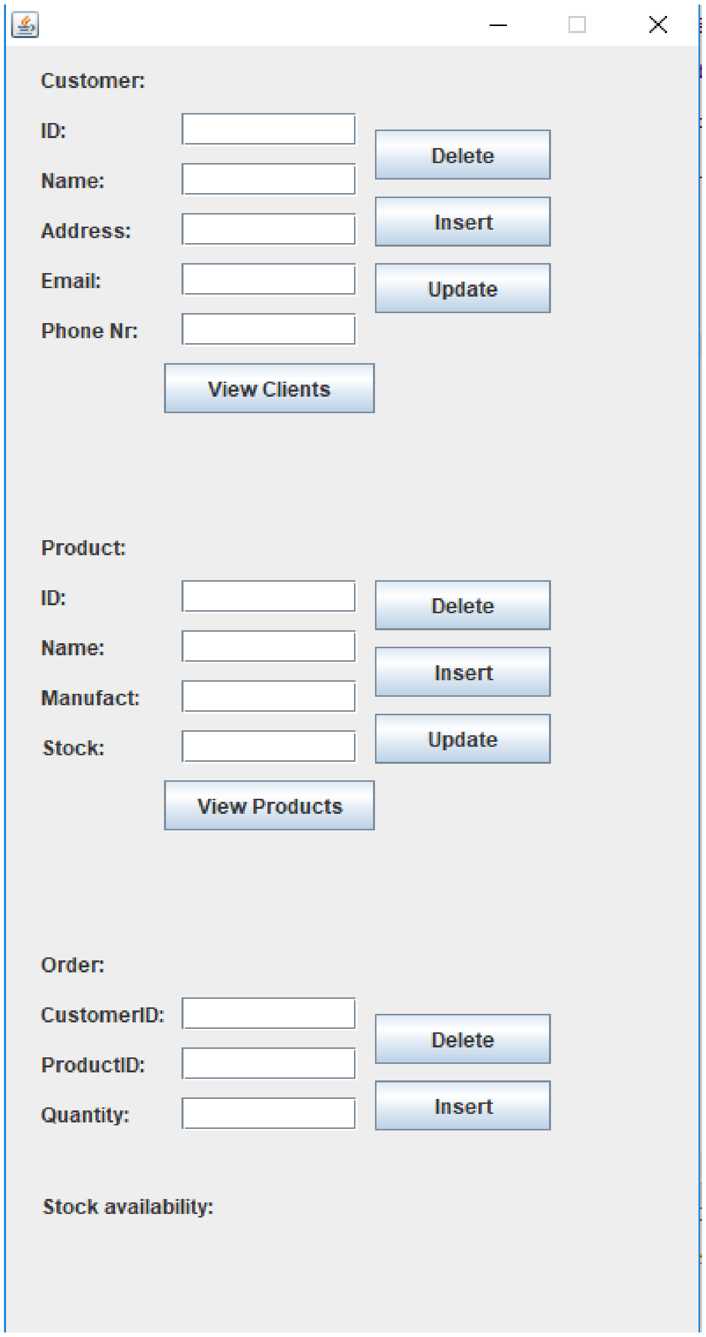
2.3.Design:

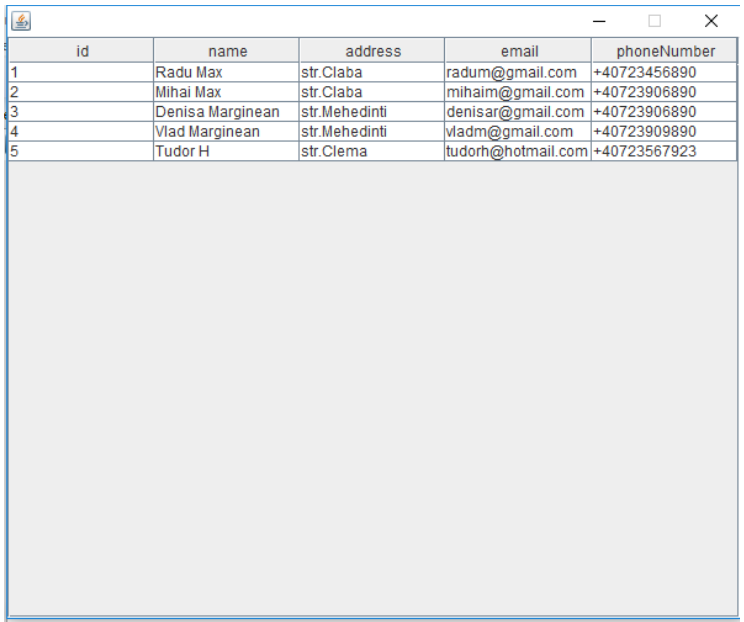
2.3.1:UML Diagram:



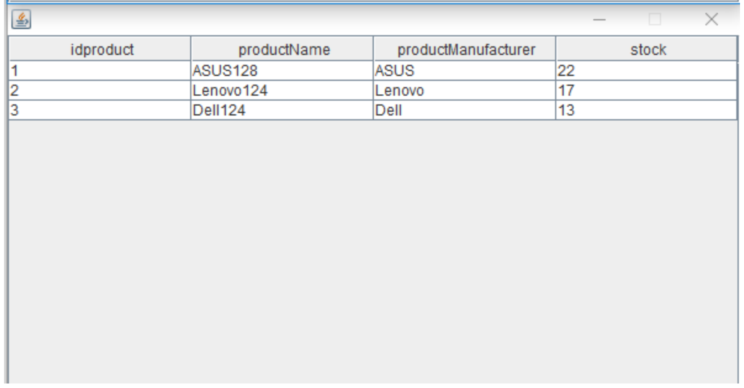
As it can be seen from the UML Diagram , the system is split in multiple packages, each holding a specific purpose in the implementation of the program. The presentation package controls the GUI, the model package gives us the basic entities later put in connection with the database through the dao and bll packages. The connection package holds the system's connection to the MySQL relational database.

2.3.2:Graphical User Interface:





id	name	address	email	phoneNumber
1	Radu Max	str.Claba	radum@gmail.com	+40723456890
2	Mihai Max	str.Claba	mihaim@gmail.com	+40723906890
3	Denisa Marginean	str.Mehedinti	denisar@gmail.com	+40723906890
4	Vlad Marginean	str.Mehedinti	vladm@gmail.com	+40723909890
5	Tudor H	str.Clema	tudorh@hotmail.com	+40723567923



idproduct	productName	productManufacturer	stock
1	ASUS128	ASUS	22
2	Lenovo124	Lenovo	17
3	Dell124	Dell	13

The Graphical User Interface presents the user with an easily understandable preface. It holds 4 different main labels: Customer, Product, Order and Stock availability. Beneath the Customer label, the user can see 5 more helping labels, each put in front of 5 text fields, where the user can introduce the data he desires. Next to the text fields, there are 4 buttons, 3 of which are used for the customer operations : Delete, Insert and Update. The fourth button, if pressed, allows the user to see the table of customers, automatically updated after the MySQL database.

Similar to the Customer side of the GUI, the Product label is accompanied by 4 different helping labels, again in front of 4 text fields. The Product area also has 4 buttons, 3 for the product operations : Delete, Insert and Update. The fourth button, if pressed, allows the user to see the table of products, automatically updated after the MySQL database.

Inside the Order area we find 3 more helping labels accompanied by 3 text fields. In this area, we only have 2 different buttons : Delete and Insert for the operations with orders.

On the bottom of the frame, the user will find the Stock availability label. If the user inserts an order and the quantity in said order cannot be met by the number of products in stock, he will receive an error message telling him that there is no way the order can go through, because there are not enough products on stock. If the order can be placed, the user will receive a validation message.

2.3.3.MySQL:

Regarding design, we must also look at the way our warehouse database schema is designed. There are 3 tables in the database: customer, order and product. The customer and product tables are straight forward, holding data and information about each inserted customer/product. However, the order table holds three different attributes: customerid, productid and quantity. The order table is built in such a way that it holds two different One-to-Many relationships: a customer can be involved in more than one order, as well as a product can be requested by more than one customers.

3.Implementation:

As previously mentioned, the system is split in packages, each holding one or more classes.

- connection package:
  It only contains one class, Connect. In the class Connect, the following methods are implemented:
  -Connect()-constructor that uses the DRIVER static attribute to return the class associated with the DRIVER string;
  -createConnection()-this is the class in which we create and return a viable connection to the database, using the DBURL,USER and PASS attributes;
  -getConnection()-a simple getter for the created connection, using a local root;
  -close(Connection connection)-the method closes our connection;
  -close(Statement statement)-the method closes our statement;
  -close(ResultSet resultSet)-the method closes our resultSet;

- bll package:

  The bll package contains 3 different classes: CustomerBll,OrderBll and ProductBll.
  The business logic layer classes have the role of validating the Data Access Operations, described in an ulterior package. For example, the class CustomerBll has the following methods:
  -findById(int id)-a Customer c is initialized as c1.findById(id), where c1 is CustomerDAO type. If there is no customer but such an id, the method returns an exception.
  -insert(Customer customer)-if the customerid is smaller than 0, or if the text fields for either one of : address, email, phoneNumber, name is empty, the method will throw an exception. Otherwise, the customer can be inserted.
  -update(Customer customer)- if the customerid is smaller than 0, or if the text fields for either one of : address, email, phoneNumber, name is empty, the method will throw an exception. Otherwise, the customer can be updated.
  -delete(int id)-if the id is smaller than 1, the method throws an exception. Otherwise, the customer with said id can be deleted.
  -selectAllC()-a selection of all the customers in the database. The validation is basically done because of the restrictions of an ArrayList. The method selectAllC() will be executed without any validation.

  The other two classes, OrderBll and ProductBll are implemented in similar ways, with specific requirements regarding their data access operations.

- dao package:

  The dao package contains three classes : CustomerDAO, OrderDAO and ProductDAO. Each class makes use of SQL queries to create the desired operations for our order processing system.
  As attributes, each class holds a logger, and a number of Statement strings that are final, each containing different SQL queries. Both

CustomerDAO and ProductDAO contain similar methods: findById(), insert(), delete(), update() and selectAll(). In each method , the LOGGER attribute creates warnings in case a query of an operation is inadequate.
The ProductDAO class contains the updateStock(int quantity, int prodid) method which has to be used in order to properly update the availability of a product, after each order has been made.

- model package:

  The model package contains three classes: Customer, Order and Product. Each of these classes has attributes that represent columns in the MySQL database schema tables, followed by a constructor, getters and setters for each attribute and in the case of Customer and Product, a toString method.
  The Customer class has the following attributes: id of type int, name, address, email and phoneNumber, all of type String. Each one of these attributes can be found as a column in the MySQL customer database table.
  The Product class has the following attributes: idproduct of type int, productName and productManufacturer of type String, and stock of type int. Each one of these attributes can be found as a column in the MySQL product database table.
  The Order class has the following attributes: customerID , productID, and quantity, all of type int. Each one of these attributes can be found as a column in the MySQL order database table.

- presentation package:

  The presentation package contains 2 classes: View and Controller.

  -View Class:
  This class holds numerous attributes, all being components of the javax.swing package : buttons for the operations of the customers, products and orders, text fields for the information that the user inputs, three frames, one JTable for showing the elements of the customers and products tables.

The main method of the View Class is the constructor : View(). Inside the constructor we build the graphical user interface, with all of its necessary components. There are many other methods, but most of them are getters and setters for the view's attributes. Apart from those, there are the button methods, with ActionListener parameters, 3 init methods for the 3 frames, 2 of which also implement ScrollPanes and finally, the viewAllTable method.

The viewAllTable(ArrayList<Object> objects) method, of type JTable, uses reflexive techniques to create a JTable that will hold the elements of a particular MySQL table. By using reflexive techniques, we work on the default Java class Object, meaning that we can replace the class with any other that fits our needs. The parameter of this function is an ArrayList of Object type objects. Inside the method, we use the variable tablemodel, of type DefaultTableModel, a javax.swing component that uses a vector of vectors to store cell value objects. The reflexive method returns a newly formed JTable, which we will use to display the values of MySQL tables customer and product.

-Controller Class:
The main purpose of this class is to properly control the functionality of the View component. As such, it takes as attributes a View instance, an instance of each of the model package classes, as well as instances of each of the bll package. The controller only has one method: the Controller() constructor. In this method we firstly initialize the View attribute, setting it to visible. The constructor then takes each actionListener method of the View instance, and assures that the proper operations will be performed when the user inserts sets of data and presses different buttons. Regarding the reflexively built JTable, the constructor method uses two different actionListener methods: one for showing the customer MySQL table and another for showing the product MySQL table. Once one of the two buttons is pressed, the table automatically generates in a new frame, being completely updated and identical to the table the user would see in MySQL.

- main package:
  The main package only holds one class: Main, inside of which we initialize a new Controller c1. When the program is ran, the main

system frame appears, having all its text fields and buttons coordinated.

4.Results:

The application is fully functional, depending on the user's capability of properly managing the insertion of data. The stock label will always point to whether or not an order can be placed and the graphical user interface data is fully connected to the MySQL database schema.

5.Conclusions:

The way in which the homework was presented, a student can learn multiple things. For one, the connection between a MySQL Database and a Java App is on full display. Regarding MySQL, the student has to remember from the previous semester how to create a relational database with multiple tables and one to many relations between those tables. Also, the MVC type of object-oriented programming style is furthermore developed, by gaining better knowledge on the layered architecture of a project. The Java Reflection Framework is also something the student is imposed to learn because of this homework.

The project can be upgraded by creating a few more operations for perhaps a more complex warehouse database, with many-to-many relations between the tables. Also, the project could fully use the newly studied reflection, by creating an AbstractDAO class, that implements the database operations reflexively.

6.Bibliography:

https://www.geeksforgeeks.org/reflection-in-java/

https://www.coned.utcluj.ro/~salomie/PT_Lic/4_Lab/HW3_Tema3/