Order management application for processing

customer orders

Homework 3

Technical University of Cluj-Napoca

Student: Natalia Carina Sîrbu

Computer Science, 2nd year, group 30425

April 20, 2017

Table of Contents

[1. Introduction to Database 3](#_Toc481085290)

[2. Objectives 3](#_Toc481085291)

[2.1 Main objective 3](#_Toc481085292)

[2.2 Problem analysis and approach 4](#_Toc481085293)

[3. Scenarios 4](#_Toc481085294)

[4. Design 5](#_Toc481085295)

[4.1. Class Diagram 5](#_Toc481085296)

[4.3 Class Details 9](#_Toc481085297)

[4.3.1. Customer 9](#_Toc481085298)

[4.3.2. Order 9](#_Toc481085299)

[4.3.3. Product 10](#_Toc481085300)

[4.3.4. OrderedProduct 10](#_Toc481085301)

[4.3.5. Connection Factory 11](#_Toc481085302)

[4.3.6 PrepStatement 11](#_Toc481085303)

[4.3.7 Table Creator 11](#_Toc481085304)

[4.3.8 CustomerBLL 12](#_Toc481085305)

[4.3.9 ProductBLL 12](#_Toc481085306)

[4.3.10 OrderBLL 12](#_Toc481085307)

[4.3.11 Controller 12](#_Toc481085308)

[4.3.12 CustomerController 12](#_Toc481085309)

[4.3.13 CustomerFrame 12](#_Toc481085310)

[4.3.14 OrderController 12](#_Toc481085311)

[4.3.15 OrderFrame 12](#_Toc481085312)

[4.3.15 ProductController 13](#_Toc481085313)

[4.3.16 ProductFrame 13](#_Toc481085314)

[4.3.17 TableFiller 13](#_Toc481085315)

[4.3.18 FrameStack 13](#_Toc481085316)

[4.3.19 View 13](#_Toc481085317)

[5. Bibliography 13](#_Toc481085318)

# **Introduction to Database**

In order to get a better understanding of this project, the term of database should be explained. A database is a collection with organized data so that it can be easily accessed, managed and updated. This collection may include schemas, tables, queries, reports, views and other objects.

Data is organized into rows, columns and tables, and it is indexed to make it easier to find relevant information. Data gets updated, expanded and deleted as new information is added. Databases process workloads to create and update themselves, querying the data they contain and running applications against it. . . . . . . . . . . . . . . . . . . . . . . . . . .. . . . . . . . . . . . . .. . . . . .. . . . . . . . . . . . . . . . . . . . . . .. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .. . . . . . . . . . . . . . . . .. . . . . .. . . . . . . . . . .. . . . . . . . . . . . . .. . . . . . . . . . . . . . . . . . . . . . . . . . .. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .. . . . . . . . . .

# Objectives

## Main objective

The purpose of this project is to develop an application that will manage and process orders for the customers of a warehouse. Relational databases are used to store the products, the clients and the orders.

The application should follow the pattern: the attributes of a customer are added by the user and then the information can be inserted, deleted or updated from/in the database. Moreover, the attributes of a product can be added and then the information can be inserted, deleted or updated from/in the database. Also, one can place an order that will be added into the database.

The graphical user interface will give the user the possibility to insert the data and then decide what will do with that. All the information needed for application will be provided from the user: the id of the product, name of the product, price of the product, quantity of the product in case the user clicks the *Product* button from the interface; the id of the customer, the name of the customer, the age of the customer in case the user clicks the *Customer* button from the interface; the id of the order, the id of the client, the id of the product, the quantity in case the user clicks the *Order* button from the interface.

It is known that not every user will be necessarily a programmer or a person with basic knowledge in programming. Therefore, the interface should be clear and user friendly, displaying different messages that can guide the user.

## Problem analysis and approach

The way the project was implemented using the bottom-up approach. This means that the first thing to be created is the base of the problem that is also presented using a great amount of details. By doing so, we will end up connecting these items together in order to obtain the whole project.

Before everything else was built, a connection to the database from MySQL Workbench was created so that the data introduced by the user in the graphical user interface will be permanently synchronized with the database.

The problem was split in more packages because a layered architecture was used in this project. The idea of this technique is that one upper layer has access only to its inferior layer and not below. This said, the application is divided into four packages: Presentation Layer, Business Logic Layer, Data Access Layer and Model. The Presentation Layer package has access only to the Business Logic Layer package, the Business Logic Layer package has access only to the Data Access Layer package and the Data Access Layer package has access only to the Model package.

One good starting point is to pick different nouns that are related to the subject and they will be the name of the classes. Moreover, in each class, the methods will bear the name of different verbs that are also related to the subject.

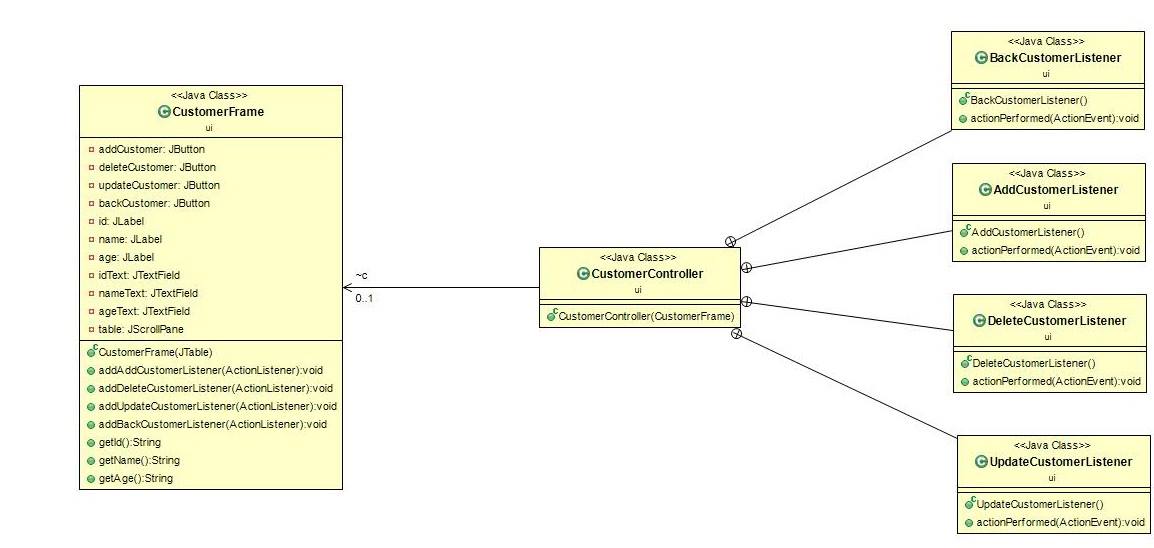
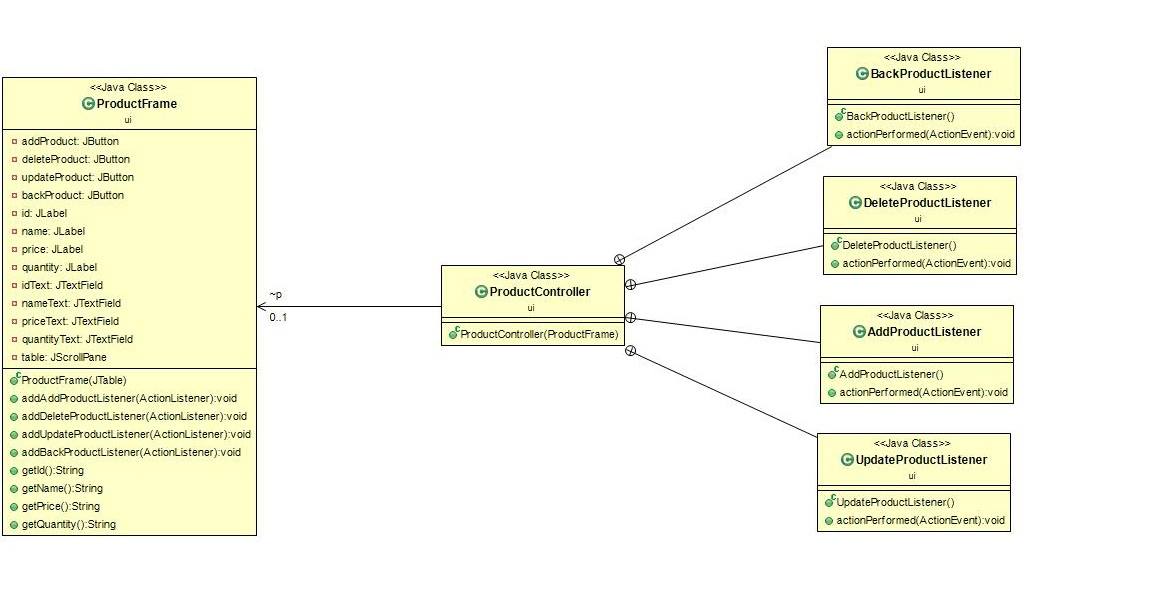
# Scenarios

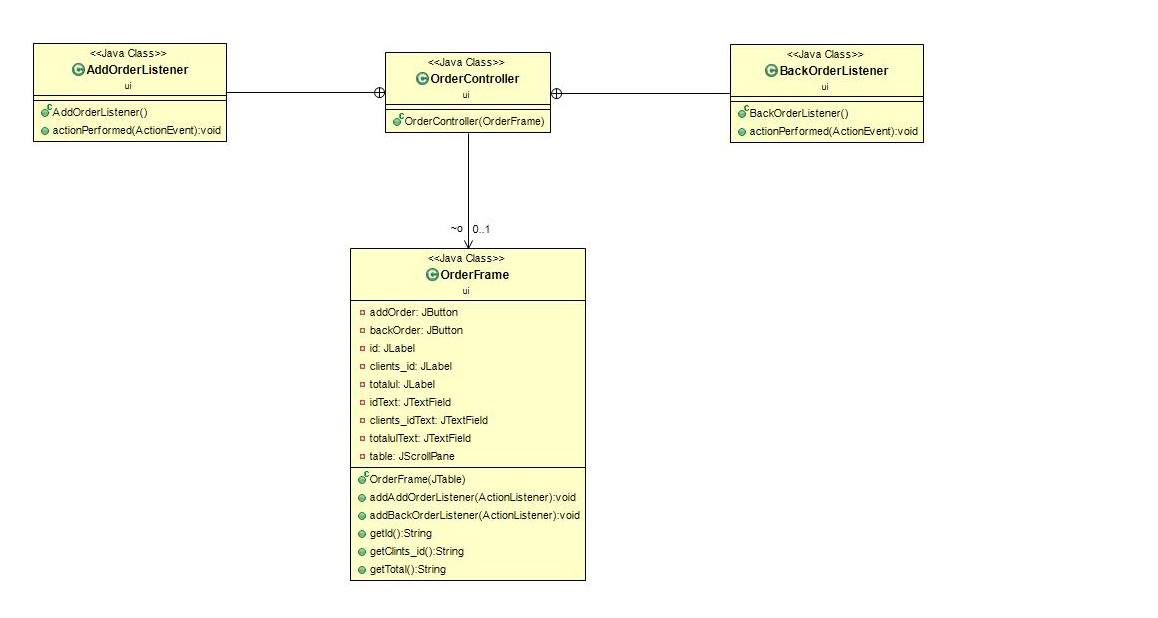
The scenario for this project is plain simple: the user selects from the graphical user interface on what field will perform the action and a new customized window will open for that action. In the new window, the user will insert the required information and then will press the right button according to his or her desire. The new information will be added in a table in the graphical user interface and in the database from MySQL Workbench.

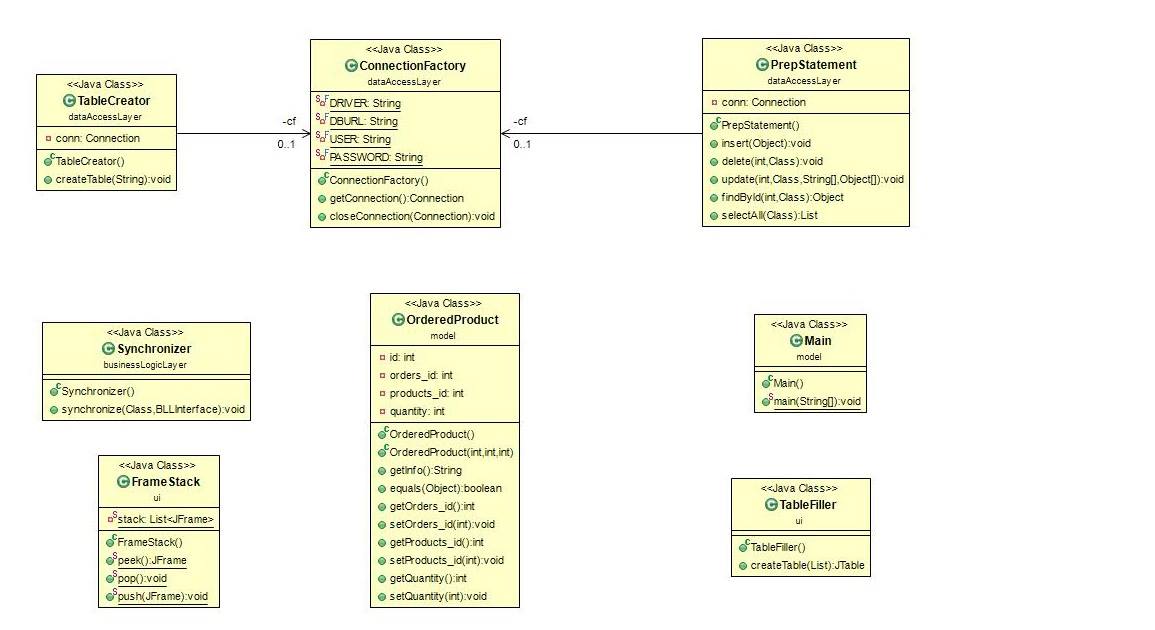
# Design

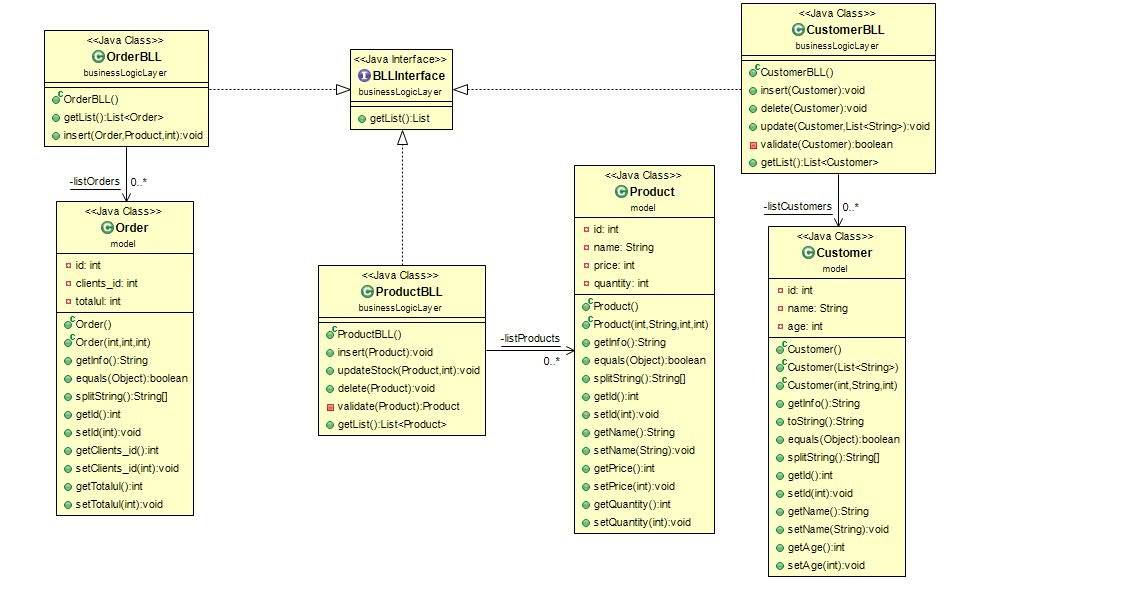
## 4.1. Class Diagram

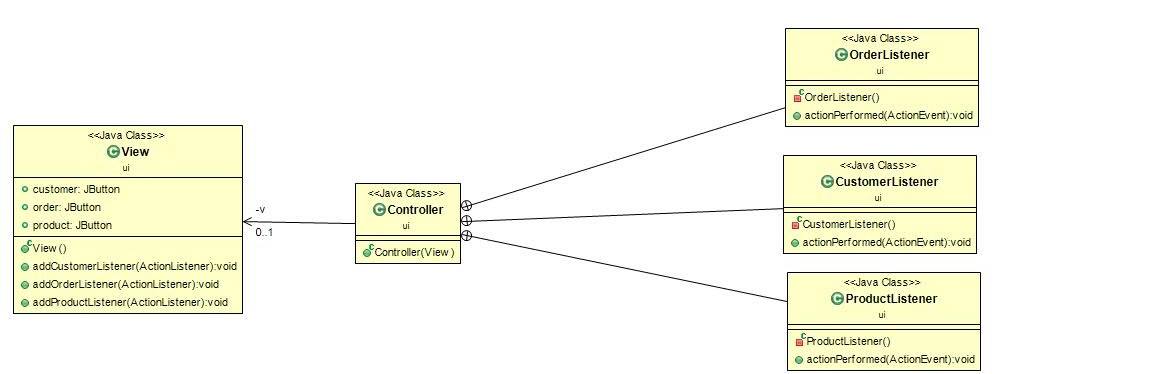
This class diagram was designed using UML, i.e. Unified Modeling Language.

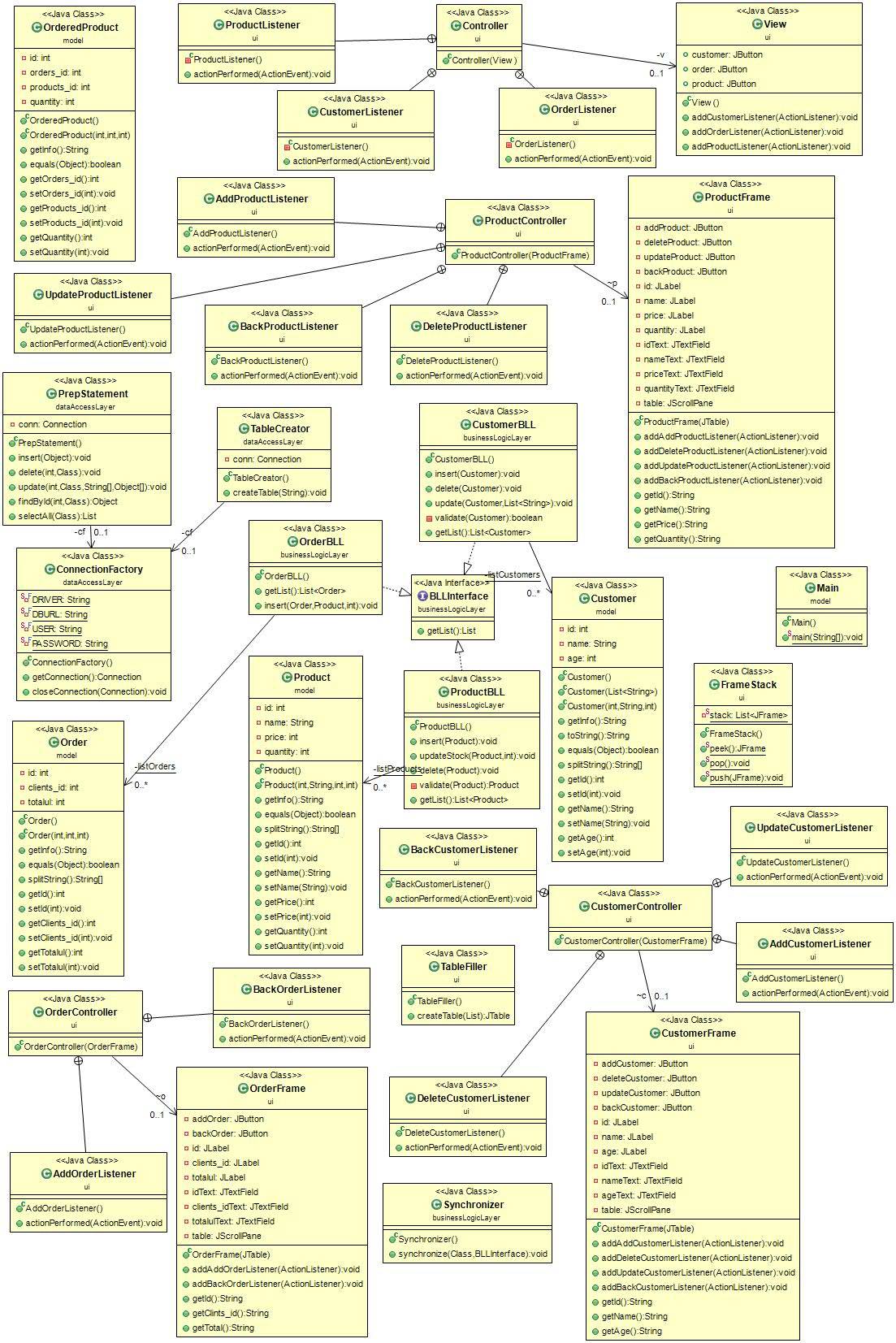












## 4.3 Class Details

MODEL

### 4.3.1. Customer

### 

This class has three attributes declared as private. One attribute is the *id* which is an unique identifier of each customer. The other one id the *name* which represents the name of each customer, *age* which represents the age of each customer. The getters and setters for each attribute are declared here.

* *public Customer () ­*– this is the empty constructor of the class;
* *public Customer (List<String> list) ­*– this is another constructor that has a list of strings as parameters;
* *public Customer (int id, String name, int name)­* – this is the constructor that has as parameters the id of the customer, the name of the customer, the age of the customer.
* *public String getInfo ()* – this method will return the attributes of the customer in a customized string in order to be added in the right way into the database;
* *public String toString ()* - this method will return the attributes of the customer in a string;
* *public Boolean equals (Object o)* – this method will overwrite the basic *equals* method.
* *public String [] splitString () ­–* this method will return an array of strings that will contain the id, name and age of the customer in exact this order.

### 4.3.2. Order

This class has four private attributes that are declared private. The first one is *id* which is the unique identifier of each order. The second one is *clients\_id* which is a foreign key from the Customer. Another one is *product­­­*\_*id* which is a foreign key from the Product. The last one is *totalul*. For each attribute from this class, the getters and setters are declared here.

* *public Order () ­*– this is the empty constructor of the class;
* *public Order (List<String> list) ­*– this is another constructor that has a list of strings as parameters;
* *public Order (int id, String name, int name)­* – this is the constructor that has as parameters the id of the order, the name of the order, the age of the order.
* *public String getInfo ()* – this method will return the attributes of the order in a customized string in order to be added in the right way into the database;
* *public String toString ()* - this method will return the attributes of the order in a string;
* *public Boolean equals (Object o)* – this method will overwrite the basic *equals* method.
* *public String [] splitString () ­–* this method will return an array of strings that will contain the id, name and age of the order in exact this order.

### 4.3.3. Product

This class has four private attributes that are declared private. The first one is *id* which is the unique identifier of each product. The second one is a string *name*. There is also a price for the product and a quantity for the products. Foe each attribute from this class, the getters and setters are declared here.

* *public Product () ­*– this is the empty constructor of the class;
* *public Product (List<String> list) ­*– this is another constructor that has a list of strings as parameters;
* *public Product (int id, String name, int name)­* – this is the constructor that has as parameters the id of the product, the name of the product, the age of the product.
* *public String getInfo ()* – this method will return the attributes of the product in a customized string in product to be added in the right way into the database;
* *public String toString ()* - this method will return the attributes of the product in a string;
* *public Boolean equals (Object o)* – this method will overwrite the basic *equals* method.
* *public String [] splitString () ­–* this method will return an array of strings that will contain the id, name and age of the product in exact this order.

### 4.3.4. OrderedProduct

This class has three attributes declared as private. One attribute is the *orders\_id* which is an unique identifier of each order. The other one id the *product\_id*  which represents the unique identifier of each product, *quantity* which represents the quantity of each product. The getters and setters for each attribute are declared here.

* *public OrderedProduct () ­*– this is the empty constructor of the class;
* *public OrderedProduct (List<String> list) ­*– this is another constructor that has a list of strings as parameters;
* *public OrderedProduct (int id, String name, int name)­* – this is the constructor that has as parameters the id of the ordered product, the name of the ordered product, the age of the ordered product.
* *public String getInfo ()* – this method will return the attributes of the ordered product in a customized string in ordered product to be added in the right way into the database;
* *public String toString ()* - this method will return the attributes of the ordered product in a string;
* *public Boolean equals (Object o)* – this method will overwrite the basic *equals* method.
* *public String [] splitString () ­–* this method will return an array of strings that will contain the id, name and age of the ordered product in exact this order.

DATA ACCESS LAYAER

### 4.3.5. Connection Factory

The **Connection Factory** class has only one purpose: to connect the application to the database from the MySQL Workbench. It has four private static final attributes of type string that are: DRIVER, DBURL, USER, PASSWORD.

* *public ConnectionFactory ()* – this is the constructor of this class that had only a “try-catch” block in it;
* *public Connection getConnection ()* – this method will establish the connection to the database;
* *public void closeConnection (Connection conn)* – this method will close the connection to the database.

### 4.3.6 PrepStatement

The **PrepStatement** class has two private attributes. The first one is an instance of the ConnectionFactory and the second one is an instance of the class Connection from the library included.

* *public PrepStatement ()* – this is the constructor of the class the will create a new connection to the database every time it gets instantiated;
* *public void insert (Object o)* – inserts a new object into the database using reflection;
* *public void delete (int id, Class<*?> c) *–* deletes a certain object from the database;
* *public void update (int id, Class<?> c, String [] column, Object [] newValues) ­*– updates a certain object using reflection
* *public Object findById (int id, Class<?> c)* – finds a certain object using only its id;
* *public List selectAll (Class c)* – this method will return a list of all the objects that are in the class c, i.e. it will return all the rows from the table given by the class c.

### 4.3.7 Table Creator

The **TableCreator** class will only create the tables in the MySQL Workbench database using sql language.

**BUSINESS LOGIC LAYER**

### 4.3.8 CustomerBLL

The **CustomerBLL** class has the customized insert, delete, update methods for the customer.

### 4.3.9 ProductBLL

The **ProductBLL** class has the customized insert, update and delete for the product inserted from graphical user interface.

### 4.3.10 OrderBLL

The **OrderBLL** class has the customized insert method for each order entered in tha database.

**UI**

### 4.3.11 Controller

In the **Controller** class there are explained the action listeners for every button from the first frame showed to the user. For each button (Client, Order, Product) a different action is implemented when the button is pressed.

### 4.3.12 CustomerController

In the **CustomerController** class the actions performed by the ADD, UPDATE and DELETE buttons are implemented. Each action has a different method.

### 4.3.13 CustomerFrame

In the **CustomerFrame,** the elements that compose the graphical user interface for the special frame designed for the customer are declared here. Their other properties, such as bounds and layouts are specified here.

### 4.3.14 OrderController

In the **OrderController** class, the actions performed by each button available to the user are described in a different method.

### 4.3.15 OrderFrame

In the **OrderFrame,** the elements that compose the graphical user interface for the special frame designed for the order are declared here. Their other properties, such as bounds and layouts are specified here.

### 4.3.15 ProductController

In this class, **ProductController**, the action listeners of each button from the graphical user interface is described in a new method.

### 4.3.16 ProductFrame

In the **ProductFrame,** the elements that compose the graphical user interface for the special frame designed for the product are declared here. Their other properties, such as bounds and layouts are specified here.

### 4.3.17 TableFiller

The **TableFiller** class will return a JTable. This class has only one method that creates the table, i.e. it fills its cells with the required data. This class was build using reflection. Thus, with only one list of objects, the table is created. Because reflection was use, the type of object does not influence its functionality.

### 4.3.18 FrameStack

This class was created using the stack principle. That’s why the class has the name **FrameStack**.

### 4.3.19 View

In the **View** class, there is a constructor with the elements that compose the graphical user interface. In this constructor, to each element is set a certain bound, background, border and size.

# **Bibliography**

<https://en.wikipedia.org/wiki/Database>

<http://searchsqlserver.techtarget.com/definition/database>

<https://msdn.microsoft.com/en-us/library/ee658109.aspx>

<https://docs.oracle.com/javase/tutorial/reflect/member/methodInvocation.html>

<https://docs.oracle.com/javase/tutorial/uiswing/components/table.html>