

FACULTY OF AUTOMATION AND COMPUTER SCIENCE

FUNDAMENTAL PROGRAMMING TECHNIQUES

ASSIGNMENT 3: ORDER MANAGEMENT

Coordinator: Dr. Eng. Cristina Bianca Pop

Student: Bianca-Veronica Avram

2nd year, 2nd semester

2020

Contents

1. *Objectives*
2. *Problem analysis, modelling, scenarios, use case*
3. *Design*
4. *Implementation*
5. *Results*
6. *Conclusions*
7. *Bibliography*

*1.Objectives*

This assignment has as its main objective working with a database by means of the java programming language. The requirement of order management is mainly related to operations executed to insert, find, delete and update in a database, practically using SQL and database manipulation and using all the CRUD operations by means of MySQL program.

As secondary objectives, the following can be identified:

1. Creating the database in MySQL
   * This point is further developed in the 3rd section and it is all contained in the SQL dump file that is uploaded alongside the source code and the present documentation
   * This objective includes the steps of building the database in MySQL, which involves the tables (client, product, orderitem and order) with their columns, the primary keys and the foreign keys
2. Creating the connection to the database in IntelliJ
   * This will be further developed in the implementation section
   * It is mainly related to the ConnectionFactory class, which is a singleton class that contains methods for creating a connection to the database, getting that connection and closing it when necessary
3. Creating the layered architecture
   * Model layer, Data Access layer, Business Logic layer, Presentation layer, Start layer, each of them having a certain purpose and functionality, which ultimately give the whole application
   * This point is presented in sections 2, 3 and 4 with further details about the classes
4. Creating the classes in each layer
   * This point is presented in section 4
   * The first three layers are related to the database, so the classes that are implemented strongly correspond to the tables from the database
5. Creating a file parser for the command given in a text file
   * These details are included in section 4
   * It is based on reading from file and providing the information needed for decrypting and interpreting the given commands
6. Taking the proper decision for each command
   * Also in section 4
   * Here it’s worth to mention introducing the Command class for the ease of processing and organizing data needed for the operations to be executed, even if work has been done for effectively creating those command objects
7. Working with iText for generating the pdfs that are required as output
   * From adding Maven dependencies, to the functions that were implemented to create the pdf and store it, these can all be found in section 4

*2.Problem analysis, modelling, scenarios, use case*

This database management application has as its input a text file that contains a number of lines with commands that are accepted. These are: insert client, insert product, report client, report product, delete client, delete product, order and report order. The non-report commands have parameters which are the attributes that serve at executing the command properly. Each command has a strict structure, so a template for writing a command must be followed so that the application works.

“Insert client: name, address”

“Insert product: name, quantity, price”

“Report client”

“Report product”

“Delete client: name”

“Delete product: name”

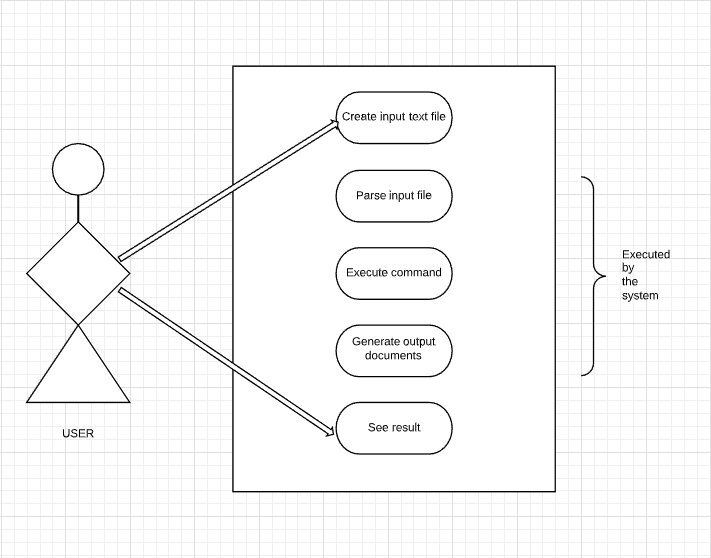
“Order: client\_name, product\_name, product\_quantity”

“Report order”

The uppercase of the command name is not relevant, because when converting the command string to a Command object, the name is made all lowercase.

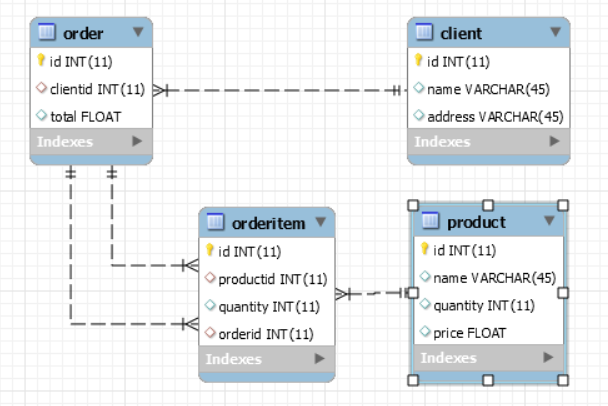
The commands are all read from file, each line is send to a method that converts it into a Command object that contains the information needed to execute the specified command. Then, in the main method, there is a switch that analyses each command separately. Depending on the type of command, the corresponding method from business layer is called and the job is done. What the user can see after executing the program for a certain text file is the updates folder that contains the pdfs that were generated. They can be either reports (they are all counted), bills (also counted) or under stock messages.

So, the possible scenarios are determined by the input commands, as the user gets from the application results under the form of pdf documents. The command is considered to be valid, so there is no validator implemented for it. In case of a wrong input command, either an exception will occur within the Command class or when attempting to insert wrong values in the database, or an unsuccessful operation will take place.



*3.Design*

First of all, developing this application started from creating the database in MySQL. The dump file with the statements that were executed in this purpose will be delivered alongside the project, but the tables will be described right below.



The figure above represents the relational schema of the database. As noticed, there are 4 tables, each having an integer as primary key.

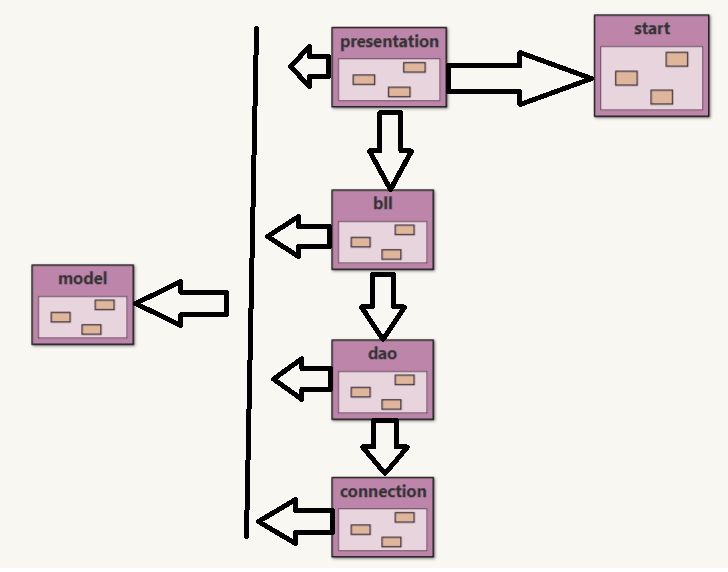
Table client contains the information abut the clients, which include name and address, both represented as strings.

Table product is just similar to product, but the information about a product include name, represented as string, available quantity, which is an integer, and price per unit, which is a float.

Now, each client has an order associated to them, so the id in order table is the same as clientid in the same table. This table contains only the total price of the expenses the client cumulated from orders.

The orderitem table is the suggested extra table that actually records what product and in what quantity a client orders at one time. This table only contains ids, which are actually foreign keys from the other tables. All of them are constructed on delete cascade, so when deleting a client, for example, the order corresponding to it and all the order items are deleted as well.

Next on, the package diagram will be presented and, in the next section, the packages will be “open” so that the classes will be analysed



The very base of the project is the model package, which contains as many classes as the database, namely four, each being the equivalent of a table. Those are used to practically simulate a record of the database and objects instances of these classes are used in all the other classes in order to maintain an order and a level of organization in the code. Going deeper, with the scheme presented in the following section, the classes contained inside model are Client, Product, Order and Orderitem. They will be detailed in the implementation section.

Then, the next package is strictly linked to realizing the connection to the database. Package connection only contains one class, ConnectionFactory that implements the methods needed to create the connection, get the connection and close the connection.

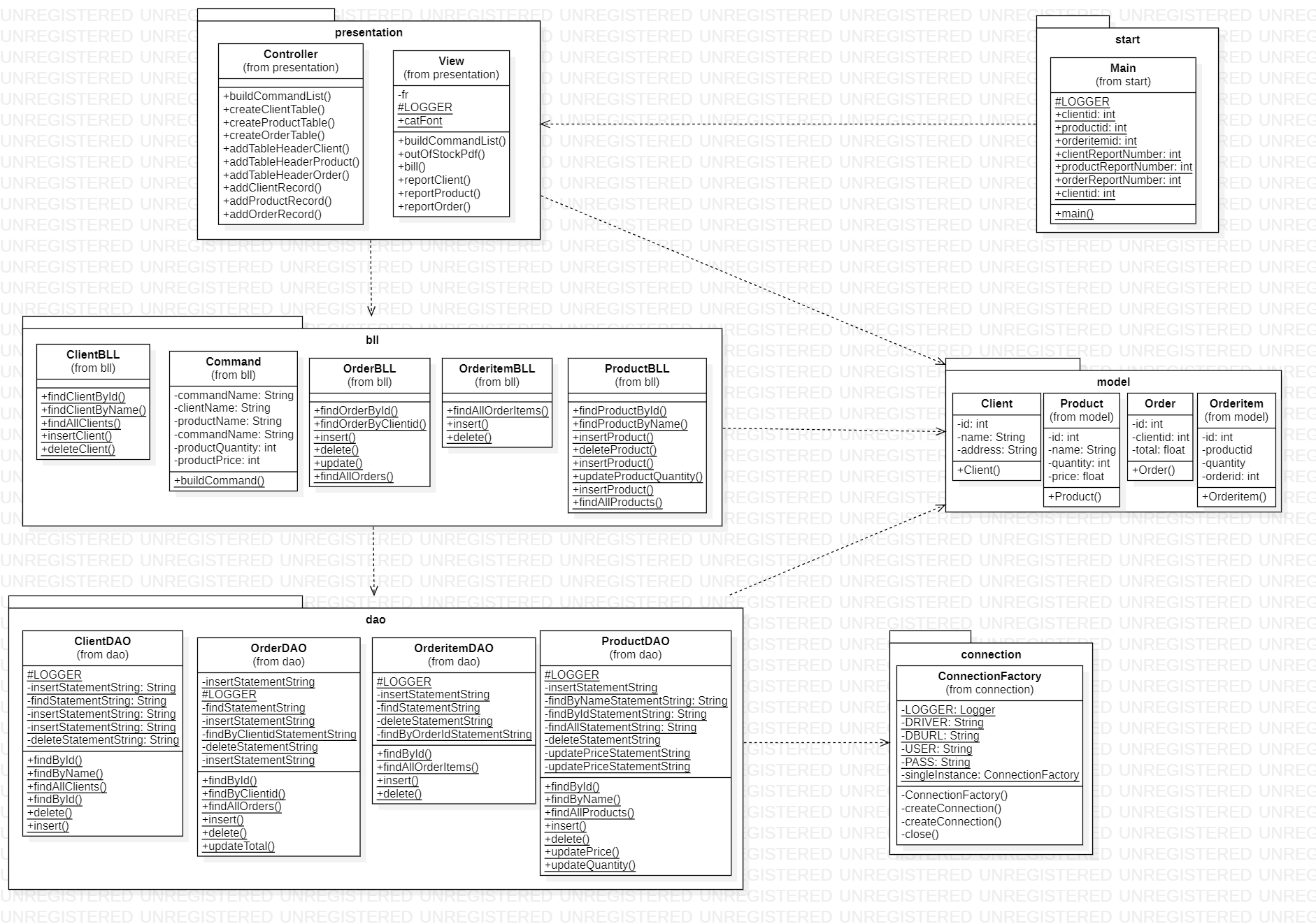
The dao package also has 4 tables, again corresponding to the database tables, but here, the interaction with the database takes place. Data access is done using the methods from connection package and a set of SQL statements that are executed.

The next package, bll, calls the methods from dao package. Also here, there are the same classes corresponding to the tables and an extra one, called Command, which is a class introduced by the author in order to make parsing commands easier.

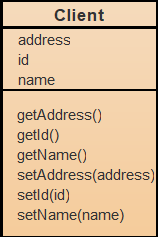
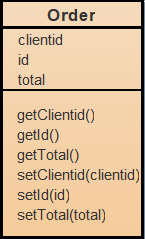
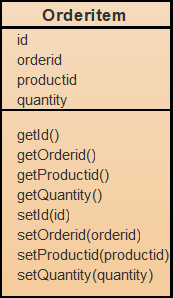
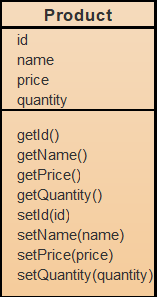
The presentation package is the one that ensures user interface, even if it is not a graphical interface. It still respects the model-view-controller prototype, of course, excepting the model part, as there is no java Component or java Container structure. This application only operates with one text file that contains a list of commands discussed in the previous section. This text file is only present in the View class, which is responsible with reading from file and generating a list of Strings that actually represent a list of commands These are passed to the Controller to create the list of actual Command objects. Also, the View class is in charge of generating the pdf documents which are of three types: report, bill and out-of-stock notification. The task of literally creating the pdf tables for the reports comes to the Controller, which has methods for creating table, adding header to table and adding records to table.

Last but not least, there is the start package that contains only once class, namely Main class. Here the main method of the application is implemented, based on all the structures mentioned before and detailed in the next part of this document.

Before going to class design, here can be found a more detailed diagram that includes all the packages, all the classes and all the methods implemented by them.

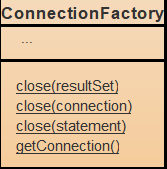
*4.Implementation*

This section is dedicated to describing all the classes, their fields and their methods. First of all, the model package classes:

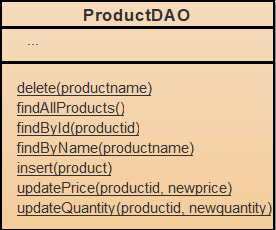
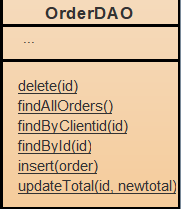
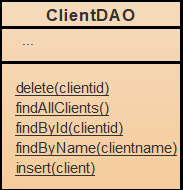
It is not relevant to take each of them separately, as they are exact copies of the entities of the database, so they contain the same fields and the same names. The only methods here are the getters and the setters, which are needed to ensure encapsulation, as all the fields are private.

Package connection has only class ConnectionFactory.



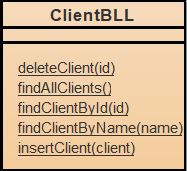
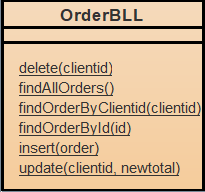
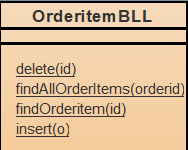
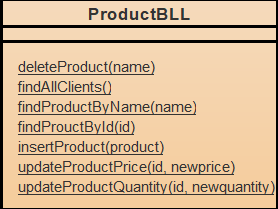
This class is the actual connection to the database. It is a singleton class, which means that it can have only one instance, which, once created is referred over and over again. The close methods are used to close the result set, the statement and the connection after executing a SQL query.

Package dao is the direct access to the data from database and use connection package to access it. These classes will also be presented all at once, as they are very similar and all the methods do, in essence the same task.

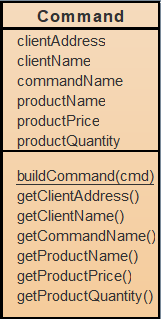


These classes have as fields public static final String variables, each defining a SQL query that has to be executed, depending on the needs of the problem. There are SELECT statements, INSERT INTO statements, DELETE FROM statements and UPDATE statements. The latter are present only in ProductDAO class, as the price needs to be updated if a product with the same name is inserted. In this case, the price is updated to the price of the product that attempts to be introduced. Then, the updateQuantity() method is used each time an order is placed, as the available quantity of that product needs to be decremented with the number of ordered products. Also, there is an update statement in the OrderDAO class, because every time an order is placed, the total cost for the client with id equal to clientid increases with the cost of the current order (which is actually orderitem). This is calculated by multiplying the ordered quantity and the price of the ordered product.

Package bll is very similar to dao package, as it mainly calls the methods from dao. Once again, the classes correspond to the previous ones.

In addition, in this package, an extra class is added. That is Command class which represents a command read from the text file. It actually receives the command string and splits it into smaller parts so that the command name is extracted, alongside the parameters that are necessary.

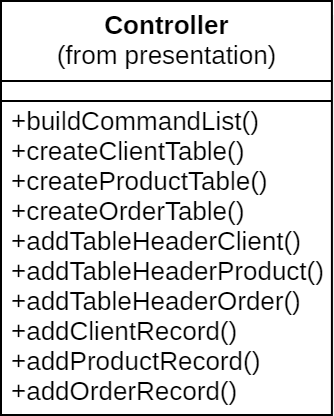


The fields of this class are the command name, a String that is actually the name of the command read from file (insert client, insert product, order, report client, report order, report product, delete product, delete client). This field tells next what is the purpose of the command. Then as this Command object needs to satisfy all the cases, the minimum number of parameters was chosen such that all the parameters given in the command can be stored into a Command object. So, there is clientName, prodeuctName, clientAddress, productQuantity and productPrice. Now the method that actually builds a command will be presented.

public static Command buildCommand(String cmd)

A string is passed as parameter and that represents the line of text fount in the text file. As known, one command starts with its name, then the parameters. This cmd string is split apart using String.split(“[:,]”) method. Like this, in the first member of the result there is the command name, which is made lower case for ease, then, depending on this name, there is a switch that detects what parameters will be allocated for the new created Command object. They are also obtained, then a new Command object is created and returned. In case there is no need for a particular field, it is simply initialized with null or 0, because, further in the implementation, when a Command is used, it is known exactly which fields are needed and which are null, so that there is no risk of NullPointerException.

Moving on to the next package, presentation, it contains classes Controller and View that rely on user interface.



Controller class provides the View everything it needs in order to generate the output and also takes from the View the input and processes it. As consequence, it does not have any fields, only some methods implementations which will be discussed below.

1. public ArrayList<Command> buildCommandList(ArrayList<String> list)

This method takes as input the list of commands as strings obtained by the View after reading the file. That list is iterated and each string is converted into a Command object as it was presented before. A new list is created and returned, but this time of type Command.

1. public PdfPTable createClientTable()

This method is used to generate the PdfPTable object that is going to be used by the View to be inserted in the pdf document. It is based on iterating through the ArrayList obtained from ClientBLL.findAllClients(). For each client, a record is inserted into the table.

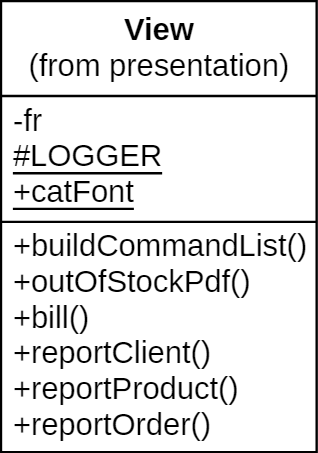
1. public void addTableHeaderClient(PdfPTable table)

This method creates the header of the table using methods that are specific to the com.itextpdf package. It is actually a lambda function that is used here.

1. public void addClientRecord(PdfPTable table, Client c)

This method is used to add a record into the table.

All the other methods from Controller are similar to the last three presented, with the exception that other reports are built, with different information. There is actually a fact that is worth to be mentioned, as for the order report, a more complex method is implemented. As shown in this documentation, the order and orderitem tables do not contain actual relevant information for the user, so it would make no sense to show as output a table full of numbers having a meaning that needs to be searched in order to be understood. Because o this, reportOrder method unites all the entities of the database, as SELECT statement are used within all of them in order to build a record that actually has a meaning for the person checking the report. So, first of all, the order table is iterated. Then, a SELECT statement based on orederid is executed to obtain all the individual orders of a certain client. Then, the client is searched in its table by id, so is the product, and the names (relevant information) is obtained.



View class uses a FileReader object in order to read all the command in the text file . The list of String objects is generated for further processing using the buildCommandList() method. For the output, there are methods to generate the pdf documents. All of them are void functions and use methods specific to the com.itext package.

*5.Results*

Junit was not used for testing as it was not required. The input text file served as the testing means, so, during developing, the program was successively run on that input file and the results were noted until no other errors occurred.

*6.Conclusions*

As conclusion this assignment was useful, because it was the first interaction of the author with a relationship between java language correlated with database connection. There was though a background in object oriented programming and database connection, but c# was used instead of java. By contrast, the approach from this assignment and the one from the past were quite similar, but more work was done here, as the rules were followed more precisely and the whole project was way more organized and logical, using new concepts from java.

The singleton class was used here for the first time, so now the author has a pretty clear idea of what this design pattern refers to and how it can be used.

The layered architecture was also a new concept, because usually there was only one package that contained all the classes, but, as a matter of fact, it made the project look and actually be smartly organized and categorized. Each layer has its own specific role and functionality, but, it is unfair to admit that everything was clear from the beginning or that there are no more doubts on this subject. The business layer still has an uncertain position, as, as noticed from the example given in the assignment documentation, it only calls the methods implemented in the data access layer, so, it might not be completely understood by the author.

The project was realized using IntelliJ IDEA, which provides a very friendly way of managing a maven project, so all the dependencies were successfully added and the .jar file was easily generated.

The author also had the intention to work a little bit more on the main class, in order to shorten the main method by splitting the code for each case into methods of the main class, but as many of the cases only required a method call plus an if clause to check the result, it was not of such importance to have a short main method. This was mentioned only to motivate the fact that this method overtakes the limit of 30 lines of code.

As further development, the database could be expanded, with tables like producer or store in order to create a more complex and widely useful application.

Also, the input as a text file could be changed with keyboard input so that the application could be used in a real situation, in real time. Obviously, there is place for more commands like update price(which actually happens in this implementation, as, if one tries to insert a product with the same name, the price that is considered valid is the second one and it is updated in the database).

*7.Bibliography*

1. <https://www.baeldung.com/java-pdf-creation>
2. <https://dzone.com/articles/spring-auto-wiring-beans-with-autowired-annotation>
3. <https://bitbucket.org/utcn_dsrl/pt-layered-architecture/src/master/>
4. <https://alvinalexander.com/java/java-mysql-select-query-example/>
5. <https://stackoverflow.com/questions/37413538/how-to-check-if-deletion-was-successful-in-the-database/37413628>
6. <https://stackoverflow.com/questions/33030422/can-be-used-singleton-pattern-with-mysql-jdbc-connection>
7. <http://coned.utcluj.ro/~salomie/PT_Lic/4_Lab/Assignment_3/Assignment_3_Indications.pdf>
8. <https://www.baeldung.com/executable-jar-with-maven>
9. <https://www.jetbrains.com/help/idea/working-with-code-documentation.html>
10. <https://www.tutorialspoint.com/java/java_documentation.htm>