Programming techniques

*Homework 2*

Moldovan Horia Andrei

Group 30424

Problem Specification :

*TP Lab* – Homework 3: Consider an application OrderManagement for processing customer orders for a warehouse. Relational databases are used to store the products, the clients and the orders. Furthermore, the application uses (minimally) the following classes:

• Model classes - represent the data models of the application

• Business Logic classes - contain the application logic

• Presentation classes – classes that contain the graphical user interface

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

Other classes and packages can be added to implement the full functionality of the application.

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. Use javadoc for documenting classes.

c. Use reflection techniques to create a method createTable 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.

Requirements :

• Graphical interface: o Window for client operations: add new client, edit client, delete client, view all clients in a table (JTable) o Window for product operations: add new product, edit product, delete product, view all product in a table (JTable)

• Create a product order for a client: the application user will be able to select an existing product, select an existing client, and insert a desired quantity for the product to create a valid order. In case that there are not enough products, an under stock message will be displayed. After the order is finalized, the product stock is decremented.

• Use relational databases for storing the data for the application, minimum three tables: Client, Product and Order.

• 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

• Documentation

Problem Analysis :

In order to design an application which fulfills the above mentioned description we have to firstly understand what a database is. A database is a structured set of data held in a computer, especially one that is accessible in various ways. So basically a database is a collection of connected table, with various data, on which a set of particularly operations can be done. A table is made of his header with the column specification and of the data it contains. For this application we will made use of the following on a single table operations.

* Insert – this operation will create a new row in the table containing the information we give as its parameters.
* Delete – this operation will delete a row from the table depending on the row number we specify.
* Update – this operation will update a specified row with some changed values we specify.
* Find – this operation will return a specific row depending on the row number we specify
* Select all – this operation will return the entire table with its header and data.

A valid database has to implement correctly all this operations we will try to do so with our tables.

The database we will be working on is the well known MySql Workbench. So one of the goals of this application is to make the MySql database resemble our graphical user interface part of the application. In order to be able to write MySql queries a minimum amount of database knowledge about the MySql statements will be necessary. The other goal which does not depend on the MySql database is the part where we will have to create a bill for every approved order.

Problem Approach :

Normally if we want to do some stuff in our application and see the result in the database then the logical thing to do would be to make a connection between the java project and the database. We do this by downloading some Java Database Connector provided by the MySql environment and inserting it in our workspace. Next we have to make a connection in MySql with a database server. After all this work is done we will try to connect to the database from the java project by using the java library provided by MySql java.sql and the information we received when making the connection to the server(host, root, password).

When making the database operations described above we will make use of the MySql statements which are some predefined queries which we will have to use in our program.

The java classes will be grouped into packages in order to maintain an order and to make the code more easy to understand and more importantly to allow the reusability of the code with perhaps different tables or different parts of the code.

Every table of the relational database will correspond to a class in our workspace, having as attributes the same name as the header description of the columns.

We will have three database tables which we will work with therefore three table classes.

The description of the problem requires us to make use of reflection techniques. The reflection approach was indeed very useful to the design of the application because without it we would have been required to create a lot of methods and to write a lot of cod but what reflection does is that it allow us to create a method which can take as parameter a class or an object and when used in a class we simply just set the parameter to the class we are in. So other said, instead of creating 3 similar methods, for the Client, Product and Order tables, which do almost the same thing, for example the insert into table operation, we design just once applicable for all of the three tables.

User case :

The user has a lot of options from which he can choose freely. On both Client and Product table panels he can introduce the values he wants to introduce in the table, the id of the row he wants to delete, or to find and the values of the row he wants to update. For each action a button was made. On the last panel the user is required to check from a box the name of the client that wants to make the order, the product he wants to buy and its size. After this the order button should be pressed.

Design :

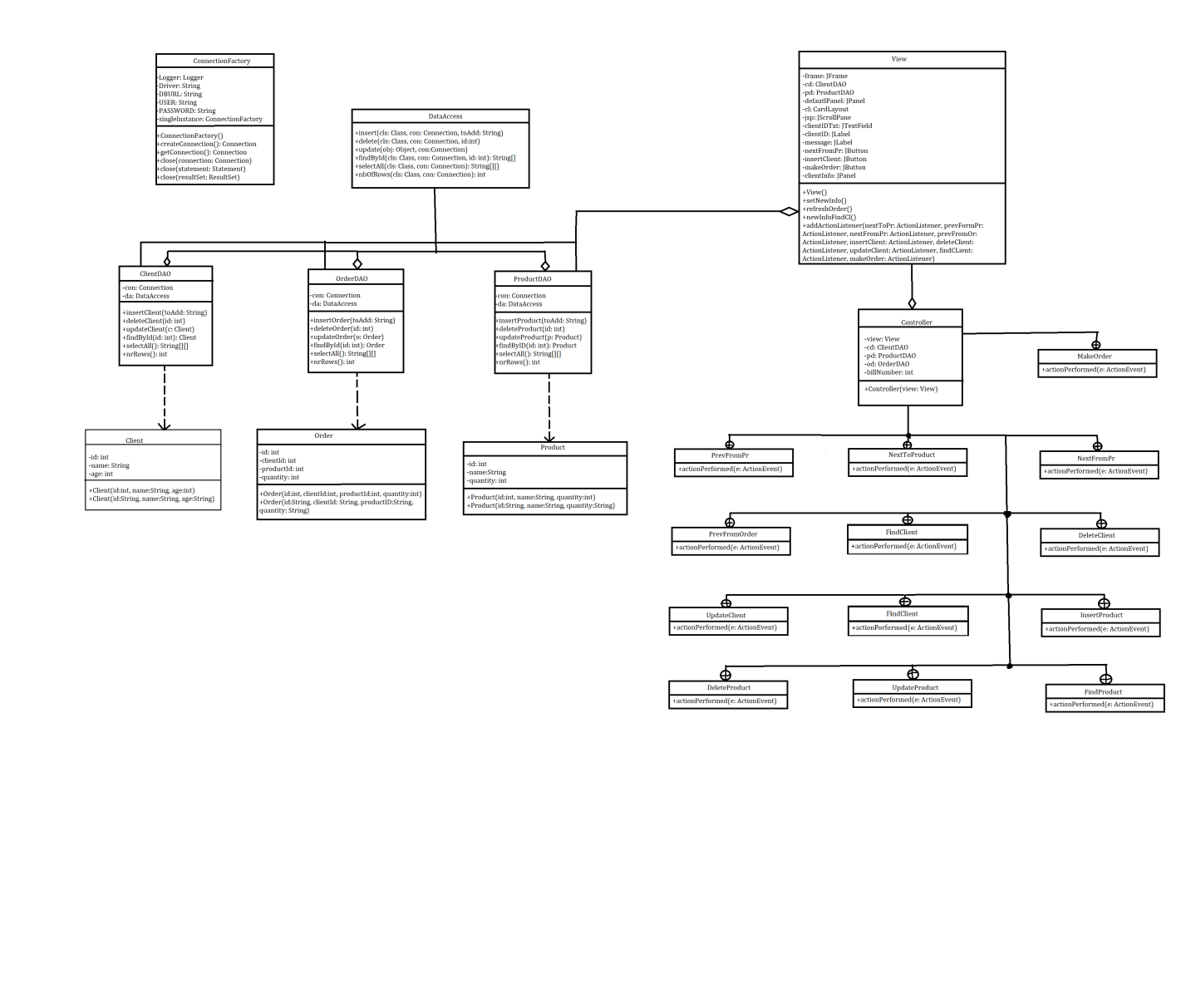
We can divide the project in two major parts: the logic of the application and the user interface. They both must run flawless in order for the application to work correctly. The design follows a Model-View-Controller pattern.

* The logic consists of the approach used(described above), and of the implementation(the classes with their attributes and methods), which glues all the parts of the MVC model together.

* -The GUI part was implemented according to the MVC pattern by splitting the written code in three big parts. -The Model contains the algorithms which run “in the back”, to which the user has no access.

-The user has access only to the provided text fields which can be found in the View part.

UML :



Implementation :

We will now take a closer look at the role and the implementation of each class with its attributes, constructors and methods.

Firstly lets discuss about the classes which are resembling the tables from the database. This are Client.java, Product.java and Order.java corresponding to the Client, Product and Order tables from the javadbconnection database project. All this classes can be found in the model package.

* ***Client.java***

The attributes of this class are the same as the header of the Client table: id, name and age. When calling the constructor of this class, which creates a new Client object with the given parameters, so basically a new row in our table, we have to keep in mind that in the database the id field is the primary key of the table which means that its value of the id on each row must be unique so there won’t be two rows with the same id. Furthermore a restriction was made on the age of the clients in the validator package and as a consequence the age of each client must be greater than 8 and less than 80 years. The attributes of this class were declared private for security purposes so in order to access their data getters and setters methods have been made.

* ***Product.java***

The attributes of this class are the same as the header of the Product table: id, name and quantity. When calling the constructor of this class which creates a new Product object with the given parameters, so basically a new row in our table, we have to keep in mind that in the database the id field is the primary key of the table which means that its value of the id on each row must be unique so there won’t be two rows with the same id. The quantity represents the number of products existing or available in stock. The attributes of this class were declared private for security purposes so in order to access their data getters and setters methods have been made.

* ***Order.java***

The attributes of this class are the same as the header of the Order table: id, ClientID, ProductID and size. The size defines how many types of the product the client has bought. When calling the constructor of this class which creates a new Order object with the given parameters, so basically a new row in our table, we have to keep in mind that in the database the id field is the primary key of the table which means that its value of the id on each row must be unique so there won’t be two rows with the same id. The table representing this class was made in order to map the relation between the Client and Product table. The attributes of this class were declared private for security purposes so in order to access their data getter and setters methods have been made.

Next we will describe the classes from the dataAccess package.

* ***ConnectionFactory.java***

This class is the one that makes possible the connection between the java workspace and the MySql database. The declared string type attribute represent the connection information received when the connection to the database server was created together with the username and password of the localhost. The constructor is the one who makes use of the downloaded jdbc connector in order to facilitate the connection. The function createConnection is very important because with this function we can actually create a connection towards the database server. The other functions are used in order to close the created connection, the executed statement and the result set obtained after the execution of the querry.

* ***DataAccess.java***

This class is the most complex class of all because here are made the general methods using the above presented reflection technique, which will be used by each table class in order to do all of the presented operations.

The first method is the insertion method, having as parameters the connection we are working with, the Class representing the table in which we want to do the insertion and a string containing the values we want to insert. We use some predefined reflection methods in order to find the header of the table we are working with, and its name and together with this information and with the string given as parameter we create a SQL statement which covers the insertion of our values and execute it. The delete method takes a parameter the connection and the table we are working with and an integer which indicates the id of the row we want to delete. We again create an SQL query with fields obtained using reflection and the integer passed as parameter to the function and execute it in order to delete the desired row. The update function takes as parameter the connection we are working with and an object. The given object will be compared with an existing object in the table by having the id as common field and the values of the old row object will be updated to the values of the new row object, given as parameter. The findByID function takes as parameter the class and the connection we are working with and the id of the row we want to find and its data is returned into a string from the function. The selectAll function will return a matrix of strings equivalent to the table given as parameter, matrix which will later be used to create the actual table. The nbOfRows function will retuen and integer representing the number of rows of the class table we need.

The business package is where all the reflection methods from the dataAccess class are implemented on each existing table.

* ***ClientDAO.java | ProductDAO.java | OrderDAO.java***

All this classes have a connection and an instance of the dataAccess class as attributes and the table operation methods which make use of this. All methods, beginning from the insert one to the one that returns the number of rows have the same purpose as the ones describes in the dataAccess class and their implementation does nothing more than calling those methods.

The next package is the presentation package which contains the classes used for the graphical user interface.

* ***View.java***

This class has as attributes all the components used to make the interaction with an user possible : panels, layouts, labels, text fields, buttons, scroll panels, and a frame which contains all the components. A card layout has been used which allows us to switch between panels in order to see the two tables(Client and Product) and the order panel. The constructor of this class initiates the declared components with the already existing information in the tables which will be later changed using the database operations. The most important function is the one which adds to each button an action listener, which will be implemented in the Controller class.

* ***Controller.java***

This class is the one that says what will be done when pressed on each one of the buttons. For example, when one of the NextToProduct, PrevFromProduct, NextFromProduct or PrevFromOrder is pressed the default panel for the cards layout will be switched. The insert, delete, update and find buttons have the same functionality as described above. On the other hand when pressed the MakeOrder buttons the following will happen: if the ordered quantity of the product is bigger than the existing quantity an out of stock message will appear. Otherwise a text folder with the bill order will be created, the size of the product will be decremented and the specific order will be inserted in the Order table.