Technical University of Cluj-Napoca

Faculty of Automation and Computer Science

2nd Semester 2016-2017

**Programming Techniques**

**Assignment no. 3**

Student: Crișan Dan

Group: 30424

Supervising teacher: Viorica Chifu

**Table of contents**

1. **Objective** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3
2. **Problem analysis, modeling, scenarios and use cases** . . . . . . . . . . . . . 3

2.1. Problem analysis . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .3

2.2. Modeling . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .3

2.3. Scenarios and use cases . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4

1. **Design** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .5

3.1. Design decisions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .5

3.2. UML diagram . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5

3.3. Data Structures . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .8

3.4. Class design . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .8

3.5. Algorithms . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .9

3.6. GUI . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .9

1. **Implementation and testing** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10
2. **Results** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 11
3. **Conclusions** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .11
   1. Things I have learned . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .11
   2. Future improvements . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 11
4. **Bibliography** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .12
5. **Objective**

The specified objective of this assignment was the following: Consider an order management application for processing customer orders for a warehouse. Relational databases are used to store the products, the clients and the orders.

The requirements are as follows:

a. Analyze the proposed application, determine the structure and behavior of its classes and draw an extended UML class diagram.

b. Design, implement and test the application classes. Use javadoc for documenting classes.

c. Define, design and implement a system of utility programs (examples: reports for under-stock, totals, etc.).

d. Design and implement a comprehensive demo driver for the order management application.

1. **Problem analysis, modeling, scenarios and use cases**
   1. Problem analysis

The program should provide a user-friendly interface so that a user could do the following operations:

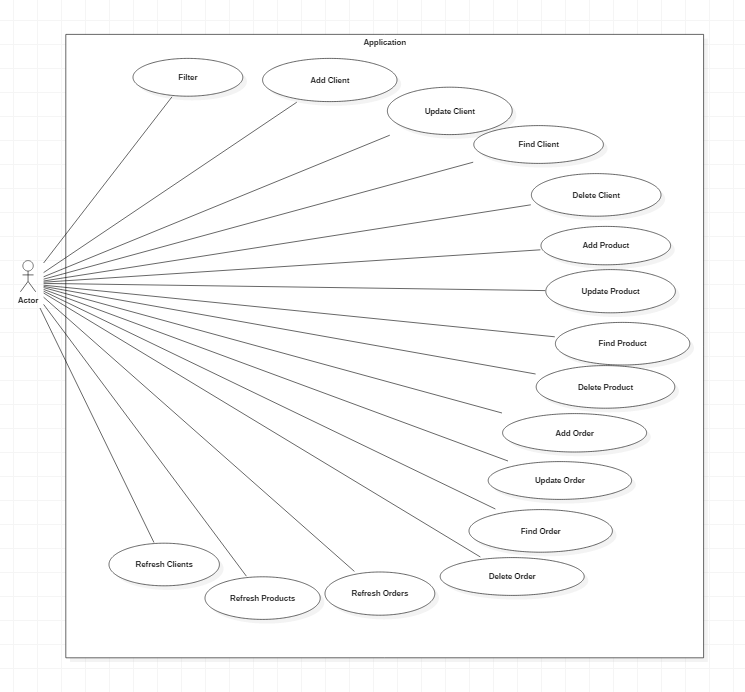
* Add a new client
* Edit / Update a client
* Delete a client
* Find a specific client
* Show all clients (in a JTable)
* Add a new product
* Edit / Update a product
* Delete a product
* Find a specific product
* Show all products (in a JTable)
* Add a new order
* Edit / Update an order
* Delete an order
* Find a specific order
* Show all orders (in a JTable)

These said operations could be easily changed to fit other tables in order to simulate a different database.

* 1. Modeling

The approach I have chosen involves creating 3 main additional classes: ClientDAO, ProductDAO, OrderDAO (DAO – Data Access Object). These classes are used to make connections and implement SQL statements between our database and our Client, Product and Order classes.

* 1. Scenarios and use cases



The main (and only) actor in this program is the user. The user can perform any of the actions mentioned above, considering the input is valid. Interaction between the user and the system is done via the GUI, by typing in data for a client, product or order. Then the user proceeds by pressing the button specific for the desired operation.

There are several use cases we should mention:

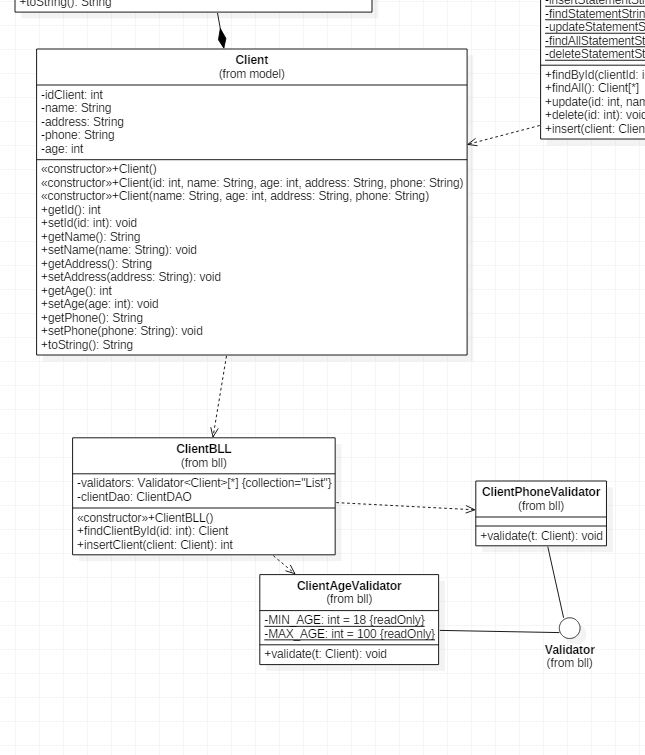
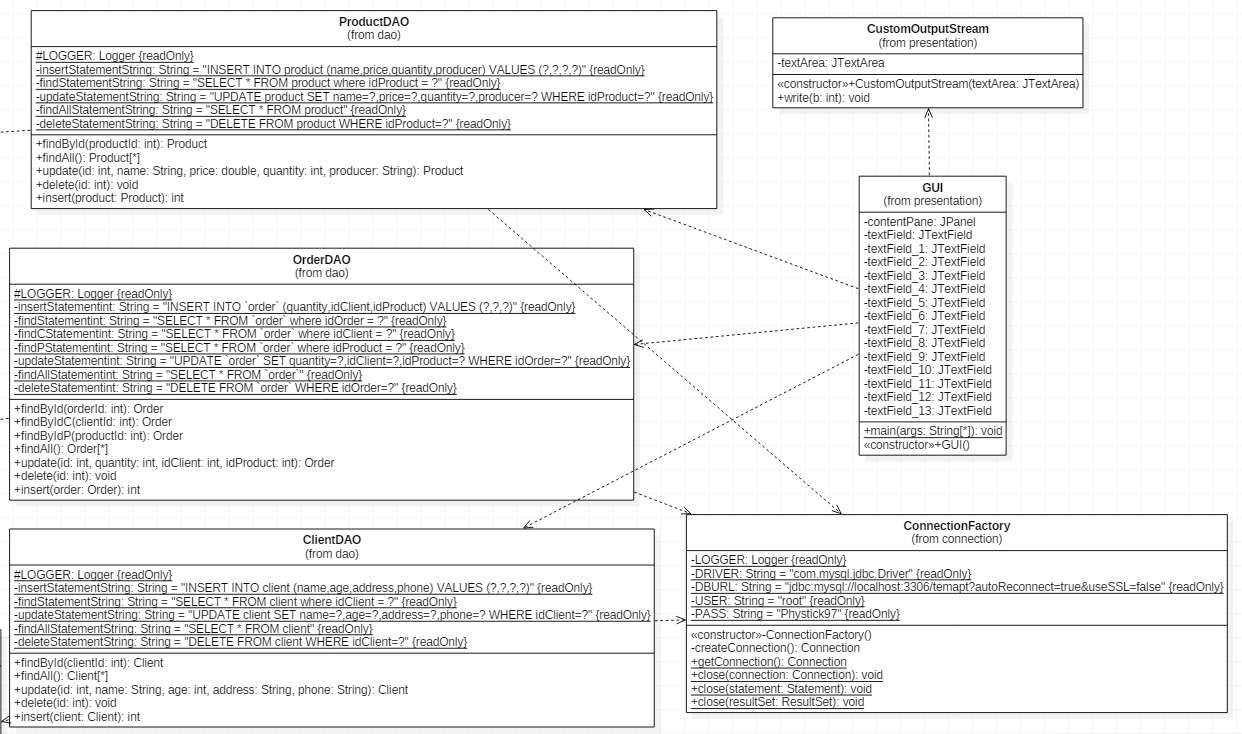
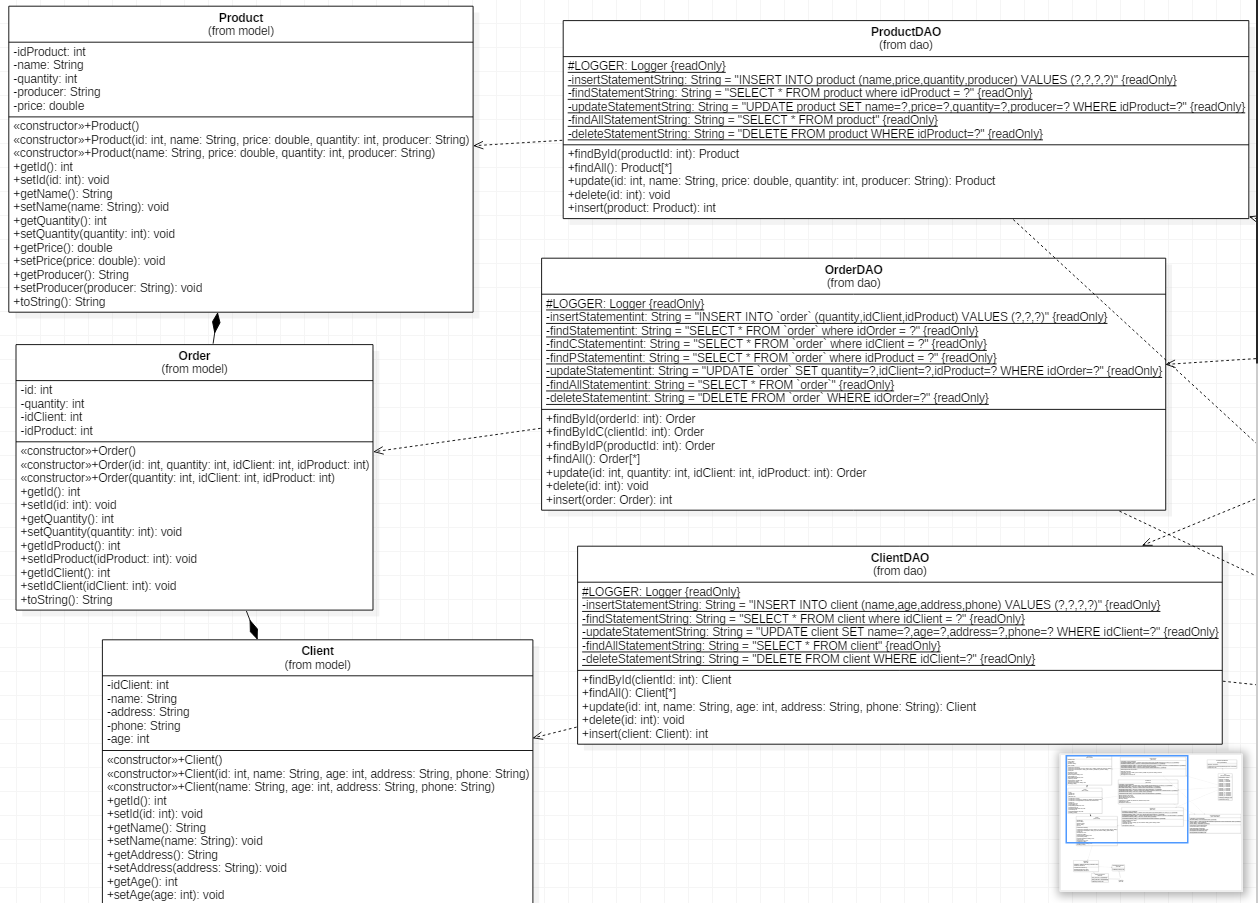
* Adding a client / product / order: The user should insert data in all the fields except the ID which is incremented automatically.
* Updating a client / product / order: Same as for the adding operation, only in this case the user needs to input the ID of the entry that needs to be updated.
* Finding a client / product / order: The operation is done by simply typing in an ID and pressing the find button. All data with respect to that entry will be printed in a JTable.
* Deleting a client / product / order: Just like finding an entry, deleting one is done by entering the ID of the entry that we want to delete.
* After every operation (except for the finding one) the user has time to analyze data and then press a “refresh” button to see the changes made in the table. If the operation done was the “Add order” one, a file will be generated, called “Receipt” which holds a “Total price” calculated for the person who made the order. E.g. Total price for ‘Name’: ‘order cost’.

1. **Design**
   1. Design decisions

As requested in the assignment, I decided to have the classes Client, Product and Order hold the simple methods (setters and getters) and create 3 other classes, ClientDAO, productDAO and orderDAO which implement the SQL statements (find, delete, update, insert). I also used a class provided to us, ConnectionFactory, in order to connect my database to these classes. As it was suggested, I created several packages which separate the interface from the classes which implement the connection and from the Data Access Object classes (DAO).

* 1. UML class diagram (Unified Modeling Language)

In the following 3 pictures, we can see the diagram for the classes found in my program. The diagrams were made using StarUML. Due to the size of the diagram, I’ve grouped the classes according to the connections they have with other classes so that each connection can be seen.

­

* 1. Data structures

As data structures I use an ArrayList of clients, products or orders for the findAll function. Basically, I store all entries (clients, products and orders respectively) in these ArrayLists so the content of the tables can be printed at any time. However my most useful data structure is the one named rs which is a Result Set. After I am connected to my database, I use this Result Set in order to run SQL statements and store the data retrieved from them.

* 1. Class design
* Class Client:

Here I have some basic attributes of a client. Each client has an ID (int idClient), a name (String name), an address (String address), a phone number (String Phone) and an age (int age).

* Class Product:

Just like class Client, here are a few attributes that may describe best a product, these being: name (String name), quantity (int quantity), it’s producer (String producer), it’s price (double price) and of course, an ID (int idProduct).

* Class Order:

This is a class which makes the connection between the client and the product he/she wants. Therefor aside from it’s own ID (int id) this class holds the ID of a client (int idClient), the ID of a product (int idProduct) as well as the quantity the client wants (int quantity).

* Class ClientDAO:

This class has a couple of Strings implemented. Since this class makes the connection between our Client and the database these Strings (which are static and final) hold SQL statements to be executed, namely: insertStatementString, findStatementString, updateStatementString, findAllStatementString and deleteStatementString, which are used for inserting, finding by ID, updating, selecting all data and deleting respectively.

* Class ProductDAO:

Just like ClientDAO this class has implemented a few String that are final and static: insertStatementString, findStatementString, updateStatementString, findAllStatementString and deleteStatementString, which are used for inserting, finding by ID, updating, selecting all data and deleting respectively.

* Class OrderDAO:

This class has a small twist to it compared to the previous 2. Aside from the Strings presented before, here we have 2 more. So the Strings are: insertStatementint, findStatementint, updateStatementint, findAllStatementint, deleteStatementint and the 2 new Strings, findCStatementint and findPStatementint which are used for inserting, finding by ID, updating, selecting all data, deleting and finding a client and a product by ID respectively.

* Class ConnectionFactory:

This class is basically the connection itself to our database. It has 4 Strings that are static and final, which hold the Driver, the database URL, the server user and it’s password.

* Class CustomOutputStream:

All this class has is a JTextArea in which it will print the redirected output.

* Class GUI

Aside from the textFields used to retrieve data, the GUI has 3 JTables which are updated using an Object named row ( Object[] row = new Object[5] ). This object holds my data for each row and has maximum 5 columns. Everytime I change something in the table I call the method model.addRow(row) or model.removeRow(row).

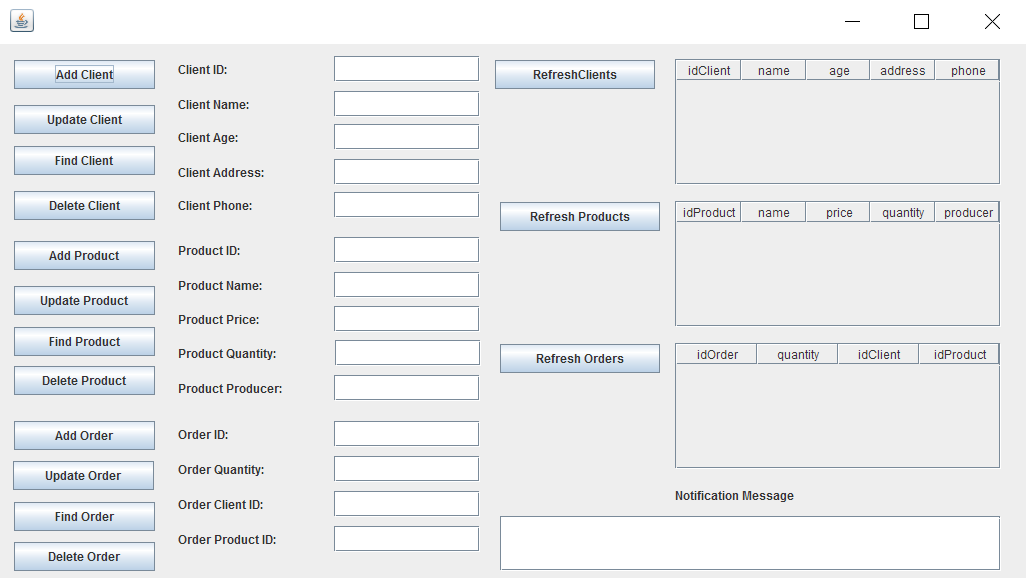
* 1. Algorithms
* Classes Client, Product and Order have only setters and getters and not any complicated algorithms.
* Classes ClientDAO, ProductDAO and OrderDAO are pretty much the same.

They all have a method to insert, update, delete or find by id and to findAll (select all data from a table). The methods findById and update return a client/product/order, so that when called all data related to the specific client/product/order is acquired. Method findAll returns a list of clients/products/orders. Since method delete purely deletes a client by executing the SQL statement, it returns void. Finally the method insert has a parameter of type client, which after the method is executed receives an id which is automatically updated by the database. Moreover, all those methods have if or catch statements which print in the notification area in the GUI if an exception appears (operation can’t be completed). Examples of such exceptions would be: client does not exist/can’t be found, client cannot be deleted (due to having an order placed), out of stock, etc.

* 1. GUI (Graphical User Interface)

The graphical user interface I have proposed is clearly labeled and easy to use.

The GUI has 14 textFields in which the user may enter data related to client, product and order’s attributes mentioned in a previous section of this documentation. Having specified in the Scenarios and use cases section what each operation requires, we know what fields we have to complete in order to press and execute the button(operation) desired. After each operation the user has to press the refresh button corresponding to the table he/she wants to see updated. In the user interface there is also a textArea with a label called Notification Message above, which prints any error message if there is one (E.g. “Out of stock, only ‘value’ pieces left.”).



1. **Implementation and testing**

In my implementation, the classes which do the hard work are in the dao package.

At their core, the thing they all have in common and helps them do what they do are the SQL statements written as this:

**private** **static** **final** **String** ***insertStatementString*** = "INSERT INTO client (name,age,address,phone) VALUES (?,?,?,?)";

**private** **final** **static** **String** ***findStatementString*** = "SELECT \* FROM client where idClient = ?";

**private** **static** **final** **String** ***updateStatementString*** = "UPDATE client SET name=?,age=?,address=?,phone=? WHERE idClient=?";

**private** **final** **static** **String** ***findAllStatementString*** = "SELECT \* FROM client";

**private** **static** **final** **String** ***deleteStatementString*** = "DELETE FROM client WHERE idClient=?";

Then there is the connection made to the database and the result set, which as previously explained, keeps the results gathered from running the SQL statements.

Connection **dbConnection** = **ConnectionFactory**.*getConnection*();

PreparedStatement **findStatement** = **null**;

ResultSet **rs** = **null**;

The result set receives the results in a try catch like this:

**try** {

findStatement = dbConnection.prepareStatement(***findStatementString***);

findStatement.setInt(1, clientId);

rs = findStatement.executeQuery();

// …

} catch {

// …

}

In the same try catch for each SQL statement we give to our variables (e.g. name, address, etc.) the result we have stored in the Result set.

**if** (rs.next()) {

name = rs.getString("name");

age = rs.getInt("age");

address = rs.getString("address");

phone = rs.getString("phone");

}

This is in my opinion the most complicated part as far as implementation goes (Leaving making the connections and putting all classes together aside).

1. **Results**

The program resulted is a user friendly application which would be a good starting point for creating a fully functional database. Sure, there is still a long way to go before this may be useful for a real company since for now you can only have 1 product per order and there are many functionalities still to be implemented.

1. **Conclusions**
   1. Things I have learned

The most useful thing I think I have learnt during this assignment, was how to connect a program written in java to a database, which to my surprise was pretty simple. Further more it was an improvement for me to learn how layered architecture works and why it may be useful.

* 1. Future improvements

There is a whole bunch of new features to be implemented. First of all the database could be more accurate having more than those 3 tables. Having made a more accurate database we can also start thinking how to implement a way for clients to be able to have more than 1 product in the same order. The verification and restrictions could be improved (e.g. check to see if the phone number has only numbers, check the quantity to be over 0, check the price to be over 0, etc.).

1. **Bibliography**

It was very helpful for me in order to finish this assignment to use parts of the code provided by the university.

<http://coned.utcluj.ro/~salomie/PT_Lic/3_Lab/HW3_Tema3/HW3_Indications.pdf>

<https://bitbucket.org/utcn_dsrl/pt-layered-architecture/src/fa3f32f80f9b36680ed07186808b5110ec6f305f/src/main/java/?at=master>

In order to learn how to print to a text file, I have found a very useful tutorial on the following website:

<http://stackoverflow.com/questions/2885173/how-do-i-create-a-file-and-write-to-it-in-java>

In order to learn how to redirect the output from the console to my text area in the Graphical User Interface, I used the class CustomOutputStream and a few other commands found on the following website:

<http://www.codejava.net/java-se/swing/redirect-standard-output-streams-to-jtextarea>