Warehouse Order Processor

Vlad-Cristian Buda 30423

Table of Contents (no numbering)

1. Objective
2. Analysis of the problem
   1. Assumptions
   2. Modeling
   3. Use cases
   4. Errors
3. Design
4. Implementation
   1. Business Layer
   2. Model Layer
   3. Data Access Layer
   4. Presentation Layer
5. Results
6. Conclusions
7. Biobliography

1. Objective

The main objective of this project is to design and develop a Java program specialized in processing customer orders for a warehouse.

In order to obtain this functionality, the main problem can be decomposed into several steps needed to reach the goal: obtain a proper representation for each type of object being manipulated (clients, products and orderds in our case); correctly design the methods used for performing different actions involving the objects we represent (add a new client, delete an existing one, place an order etc.); design a friendly and intuitive graphical user interface for the user to easily take advantage of the functions provided by the program.

2. Analysis of the problem

The solution of the problem should be able to correctly process customer orders and maintain an relevant SQL database for a warehouse. In order to obtain this functionality, several operation must be implemented: insert a new client/product/order into a table; delete an existing client/product; update the information related to an existing client/product. First of all, in order to do that, a correct representation of the real life objects must be proposed. The fact that the user is responsible with populating the database requires some assumptions to be made.

a) Assumptions

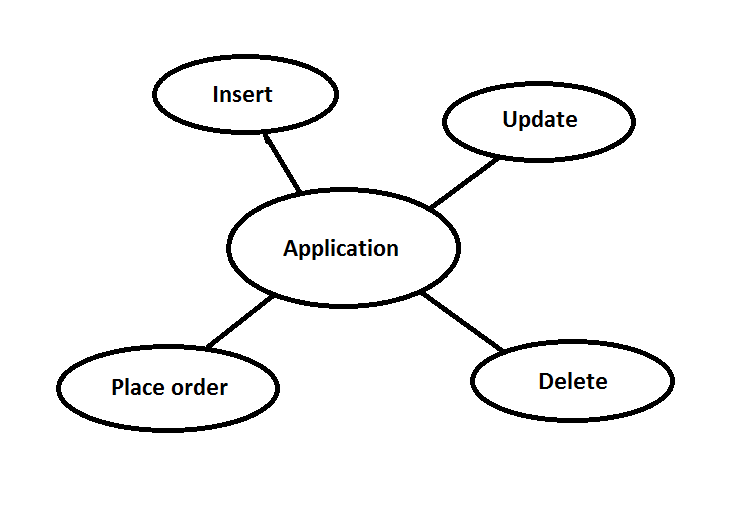
The assumptions that should be made regarding this problem are stricly related to the corectness of the information the user enters to populate the database. Altough we suppose that the user correctly enters the information corresponding to each field, the entry in a table is checked before inserting it in the desired table and the user is notified when incorrect information is being entered. We also assume that the user does not want to place more than a product in an order.

b) Modeling

As mentioned above, an important aspect in finding a solution is to correctly represent the data. In the case of our application, 3 real life object should be represented. The first one is the client, the person which places the order. Each client has a unique id which identifies him in the database. The other relevant information for representing a client is its name, address and email address. The next on is the product, the item being ordered and shipped to a client. Similar to a client, each product is uniquely identified by an id. Besides that, the product has name, price and the quantity being available in stock. Everytime an order is placed for a product, the quantity of that product is updated. The last one, the order, is identified by an unique id and contains other relevant information such that: the id of the client who placed the order, the id of the product being ordered, the quantity of the product being ordered and the total cost of the order.

c) Use cases

The use-cases are represented by the functions the program offers: the capability of managing a database with clients, products and orders.. All the available operations, i.e. use cases possible are shown in the following diagram:



For each of the above mentioned use-cases, a button on the graphical user interface will be present. In order for the user to perform an operation, he will first enter the data having the correct syntax and then will push the button corresponding to the desired use case. Depending on the chosen operation, the results will be immediately displayed on the graphical user interface or the user will be notified in case something went wrong. The steps described above are the same for each use-case.

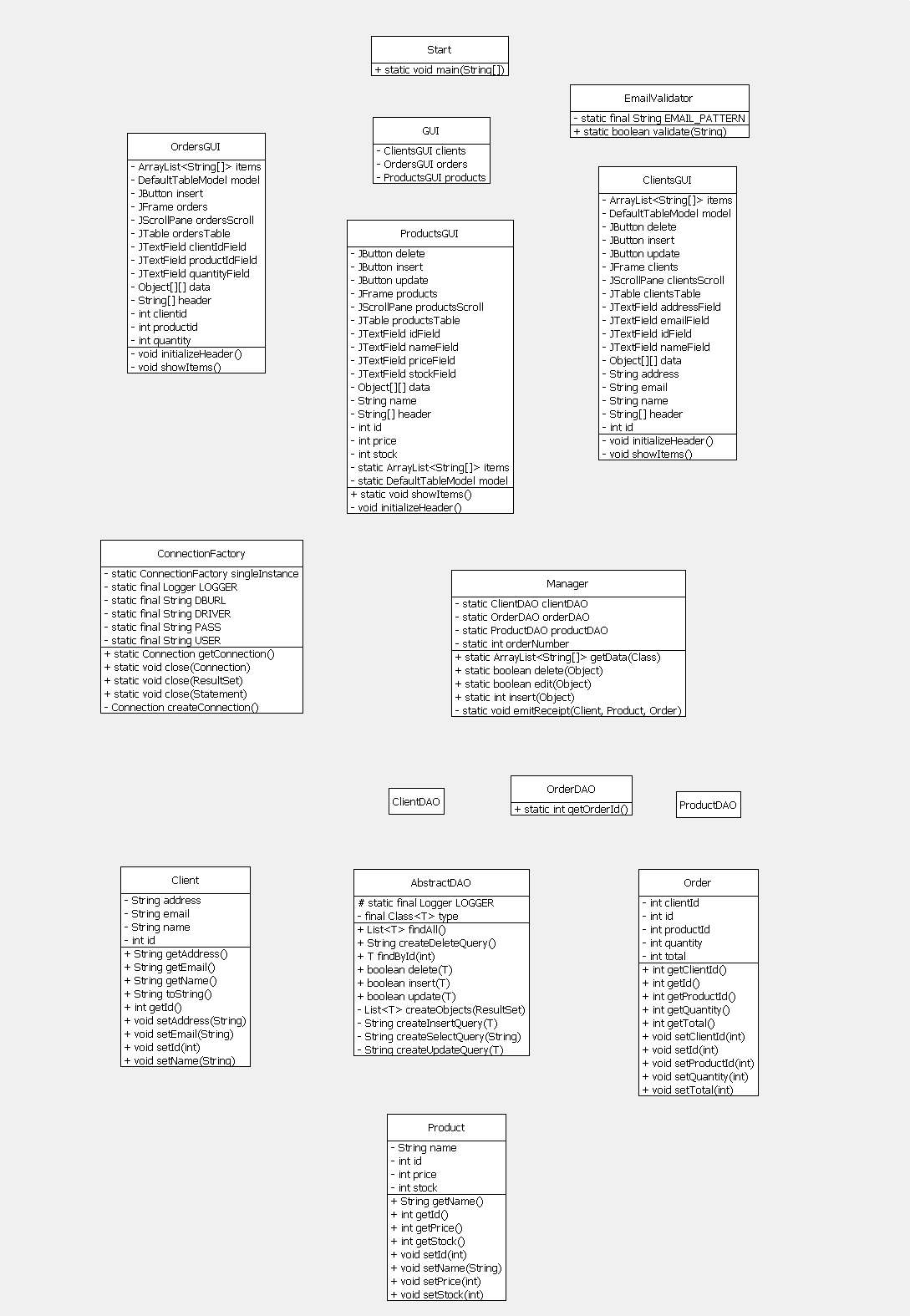
d) Errors

The corectness of the program guarantees no errors in terms of computing the result if the input is correctly entered. Even though the input entered may have a wrong format, the user will be immediately prompted and the program should not encounter any errors. At this point, it is safe to say that all the exceptions that might appear due to invalid input are addressed.

3. Design

The design of the application is made in an object oriented manner. As mentioned earlier, the classes corresponding to the modelled object contains the relevant information for that object.

The following UML diagram shows the entire structure of the program along with all the classes, interfaces and relationships between them.



There are 6 packages containing the main classes: Business package, Connection package, Data Access package, Model package and Validator package. Business package contains the main logic of the program and is represented by the class Manager. Connection package has the class ConnectionFactory which is used to establish the connection to a database. Data Access package contains the generic class AbstractDAO which implements the main methods used for performing database operations. Classes ClientDAO, OrderDAO and ProductDAO inherits AbstractDAO and instantiate it with the corresponding type. Model package contains the classes representing the models being used : Client, Product and Order. Validator package has a static method which checks whether a given email has the correct syntax or not.

When it comes to data representation, an ArrayList of strings was used to obtain the representation of the content of each table for displaying in the JTable. The ArrayList was chosen due to the fact that it offers useful methods such as add, delete and its dimension changes dynamically.

All the main classes will be explained in detail in the following section along with their implementation, main methods and also the graphical user interface.

4. Implementation

We already mentioned above the main packages and their corresponding methods. Each class implementation will be thoroughly explained next.

The program is structured in 4 layers: Model Layer, Business Layer, Data Access Layer and Presentation Layer.

Model Layer is represented by the Model package which contains the classes which represents the modeled data. Class Client represents the client in the database. It contains relevant information such as its unique id, name, address and email. The class mainly consists of getters and setters along with a function which returns a string representation of an object of type Client. Class Products represents the product along with its corresponding characteristicts: unique id, name, price and stock. Similar to the client class, it consists of setter and getter functions. Same can be said about Order class which has the information relevant for an order: order unique id, client id, product id, quantity and total.

Business Layer is represented by the Business package. This layer is responsible with application logic. It has a single class, Manager, which contains methods used to perform operations on objects. Its methods are all static and receive an object of type Object. The methods use reflection in order to find the corresponding class to each received object and then perform the specified operation using an instance of class DAO corresponding to each class. The insert methods inserts and object in its corresponding table in the database. The method returns 1 if the insertion was successful, 0 in case of an error when trying to insert a client or a product (id already exists) and negative values for errors specific to placing an order (-1 for client not found, -2 for product not found and -3 for insufficient stock). The other methods for editing or deleting an entry are similar, the only difference is that they return a boolean, true in case of success and false otherwise. Method getData() returns an ArrayList of Strings which represents all the entries in the table corresponding to a class (Client, Product or Order). This method is used for displaying each table’s content in the JTable. Method emitReceipt() is a private static method called inside the class when an order was successfully placed. It generates a text file in the form of a receipt which contains the details for that order.

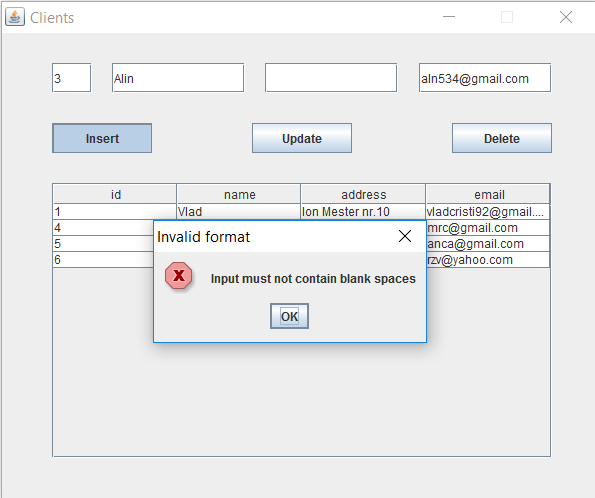
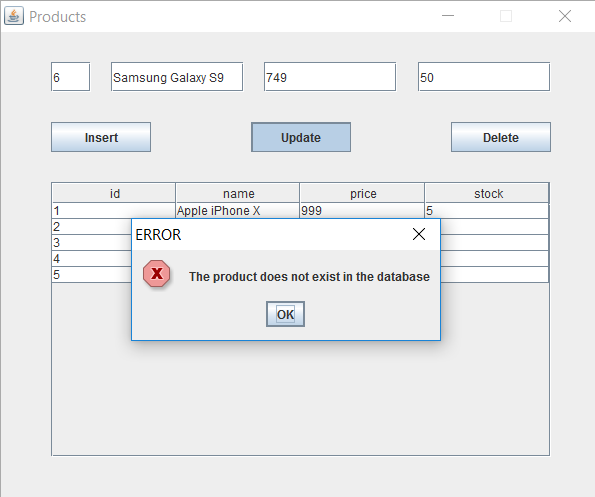
Data Access Layer is represented by the DAO package. It contains the classes used for performing SQL statements corresponding to each operation. Class AbstractDAO is a generic class which implements the operations through reflection. Each method receives a generic object T. Method insert() is used for inserting an entry in a table. It calls the private method createInsertQuery() which returns a string representation of an SQL statement for inserting an element in the corresponding table. This method uses reflection to access the fields of the object’s class in order to create the statement. Moreover, through reflection it gets the value corresponding to each field and inserts it in the SQL statement which is then returned to the Insert() functon. Having the corresponding SQL statement, the Insert() function then connects to the database using ConnectionFactory class, executes the statement and then closes the connection. If the insertion was successful, the function will return true or false otherwise. ConnectionFactory class contains the methods necessary to establish a connection to the database. The private method getConection() retuns an object of type connection used by the DAO classes to perform SQL operaitons. The other methods used for performing database operations, update() and delete(), works the same as insert() functions. Each functions has a corresponding function which creates an SQL statement specific to that operation. After the function receives the statement, it connects to the database and executes the statement. Each function returns true if the operation was successful or false otherwise.

There is also a find() method which searches for an entry in a table based on its id. If the item exists in the database, an object representation will be returned or null otherwise. Similar to find(), the findAll() function returns all the entries of a table. It returns the result in a fost of a list of objects of the corresponding type. Both functions, find and findAll, call the method createObject which through reflection instantiates the corresponding class and creates objects based on the ResultSet object which contains the results of the SQL statement execution.

Classes ClientDAO, ProductDAO and OfferDAO all extends the AbstractDAO class and specifies a concrete type, corresponding to each model in the database.

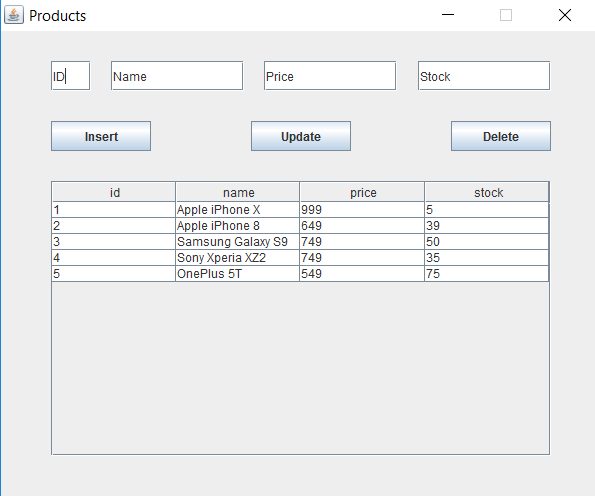
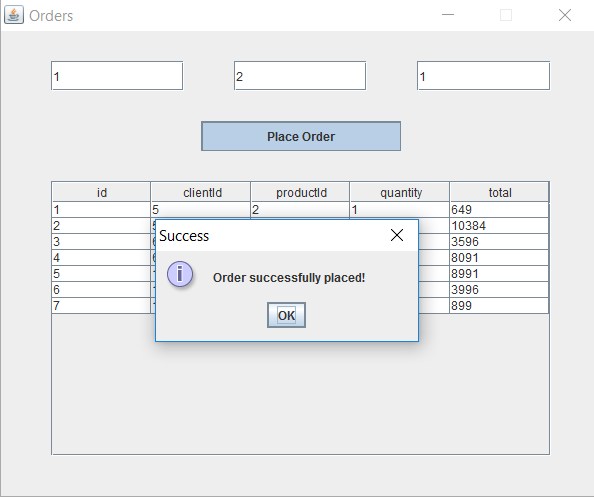
Presentation Layer is responsible with the graphical user interface of the program. It consists of the GUI package where the graphical user interface for each table is implemented. Each class represents a different frame and each one contains a JTable for displaying the information in the database. Each frame contains the corresponding JTextField for getting information from the user. Each operation has a corresponding button which user must press after he entered the input. Type errors are addressed inside this class and the user will be prompted by a message when an invalid data type was entered of when a blank space in left. Class ClientGUI uses an email validator which validates the syntax of the email. After the input was validated, the class will call the static methods in Manager class coresponding to the desired operation. An object of the current class type is also sent to that function.

The JTable header is generated through reflection for each table in the database by accessing the filed corresponding to that class and getting their name. Each time the information in the JTable needs to be refreshed, the method showItems() is called which gets the string representation of the content of its corresponding table.

5. Results

After all the necessary steps (design, implementation, testing) we managed to obtain a fully functional program for processing orders for a warehouse. Also, besides the main functionality, a simple and intuitive graphical user interface was created to ease the access to the program’s functions. The following screenshots present the application in various use cases:

6. Conclusions

During the design and implementation of this project I was able to learn how to connect a Java program to an SQL database. Also, I this was a good oportunity to reacap the SQL staments. I also managed to learn some new things and improve my programming skills. For example, Java reflection was a new thing for me at this point and through this project I was able to use this powerful tool.

For further improvements of this application, I can suggest the addition of new feature and more complex SQL functions. Also, the database can be easily expanded with more tables and more information can be stored.

8. Biobliography

<https://stackoverflow.com>

<http://coned.utcluj.ro/~salomie/PT_Lic/4_Lab/HW3_Tema3/HW3_Mysql_Installation_Guidelines.pdf>

<http://coned.utcluj.ro/~salomie/PT_Lic/4_Lab/HW3_Tema3/Tema3_HW3_Indications.pdf>

<http://tutorials.jenkov.com/java-reflection/index.html>

<https://www.draw.io/> (for drawing the UML diagram)

Java: How to Program, 9th Edition (Deitel)