Dan Mitrea

Group 30421

Year of study : 2016-2017

Technical University of Cluj Napoca

Programming Techniques

* Assignment 3 -

Order Management System

Contents

Assignment Objective . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3 Problem Analysis . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3 Modelling . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4

Scenarios. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5

Use Cases . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5 Design . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7 Class Diagrams . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7 Packages . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .7 “ Model classes ” . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .. . . . . . . . . . 8

“ bll ” classes . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8 “ connection ” class . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8

“ dao ” classes . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9 GraphicDesign Class . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .10 “ start ” classes . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 11

User Interface . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .11 Testing and Implementation . . . . .. . . . . . . . . . . . . . . . . . . . . . . . . . .14 Results . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 14 Conclusions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 14 What has been learned . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 14 Further improvement possibilities . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 15 Bibliography . . . . . .. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .15

Assignment Objective

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. Furthermore, the application uses (minimally) the following classes:

1. Domain specific classes: Order, Customer and Product
2. Business Logic (warehouse-specific processing) classes: OrderProcessing, WarehouseAdministration, ClientAdministration

1. Presentation classes: GUI related classes
2. Data access classes: Database access related classes

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

Requirements

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, filters, etc.).

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

Problem Analysis

Nowadays, almost every domain has to deal with big data, endless information about everything. For example, Facebook has to work with billions of users, more billions of videos and photos they post, and so on. This being said, in order for bussinesses that have the purpose of selling items to people, to expand more quickly, creating an internet site for people to buy their products is the best way to do so. This is where an order management application can greatly help any bussiness that sells products online. Clients can be seen in tables and their information can be processed more easily, products’ stock and order quantities can also be processed more easily.

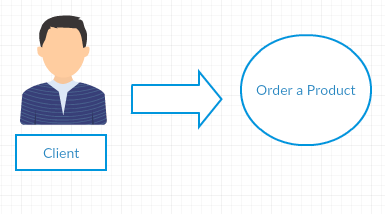
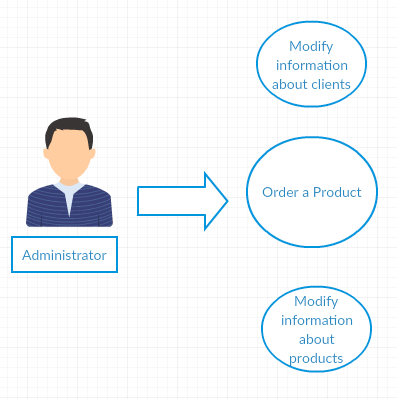
This applications always have two types of users, the administrators ( the ones that run the site ) and regular customers. The main difference between them is that the administrator has more rights than an ordinary client, that has only the option to place an order. These additional rights include updating the information of a client or a product, inserting a new client or product into the database, deleting a client or a particular product from the database, or he can just check everything that is in the database in the form of a table. More easily said, they can modify any data that is inside the database, therefore the tools that they use need to be able to do more complicated calculations, while the interface does not have to be so fancy, mainly because they are not interested in the aspect of the application, but in its utility. Users on the other hand, should have at their disposal a user friendly interface, making it easy for them to order what they want, and to see what is available in the store.

Modelling

Scientific modelling is an activity with the aim of making a particular part or feature of the world easier to understand , define, quantify, visualize , or simulate by referencing it to existing and usually commonly accepted knowledge . It requires selecting and identifying relevant aspects of a situation in the real world and then using different types of models for different aims, such as conceptual models to better understand, operational models to operationalize , mathematical models to quantify, and graphical models to visualize the subject.

In our case, the best way to model our application is by creating specific sets of classes that deal with their corresponding areas of interest. The data acces classes are the ones that directly interfere with the database and modify information inside it. The bussiness logic classes are like an intermediary, taking the information from the data access ones, and create corresponding classes and objects. This classes also have the role to check if the data is valid, or that the user does not order a product that has no stock left. Simpler said, they place constraints on what the user can do, so that the enterprise has the last word in what the client can do.

Use cases



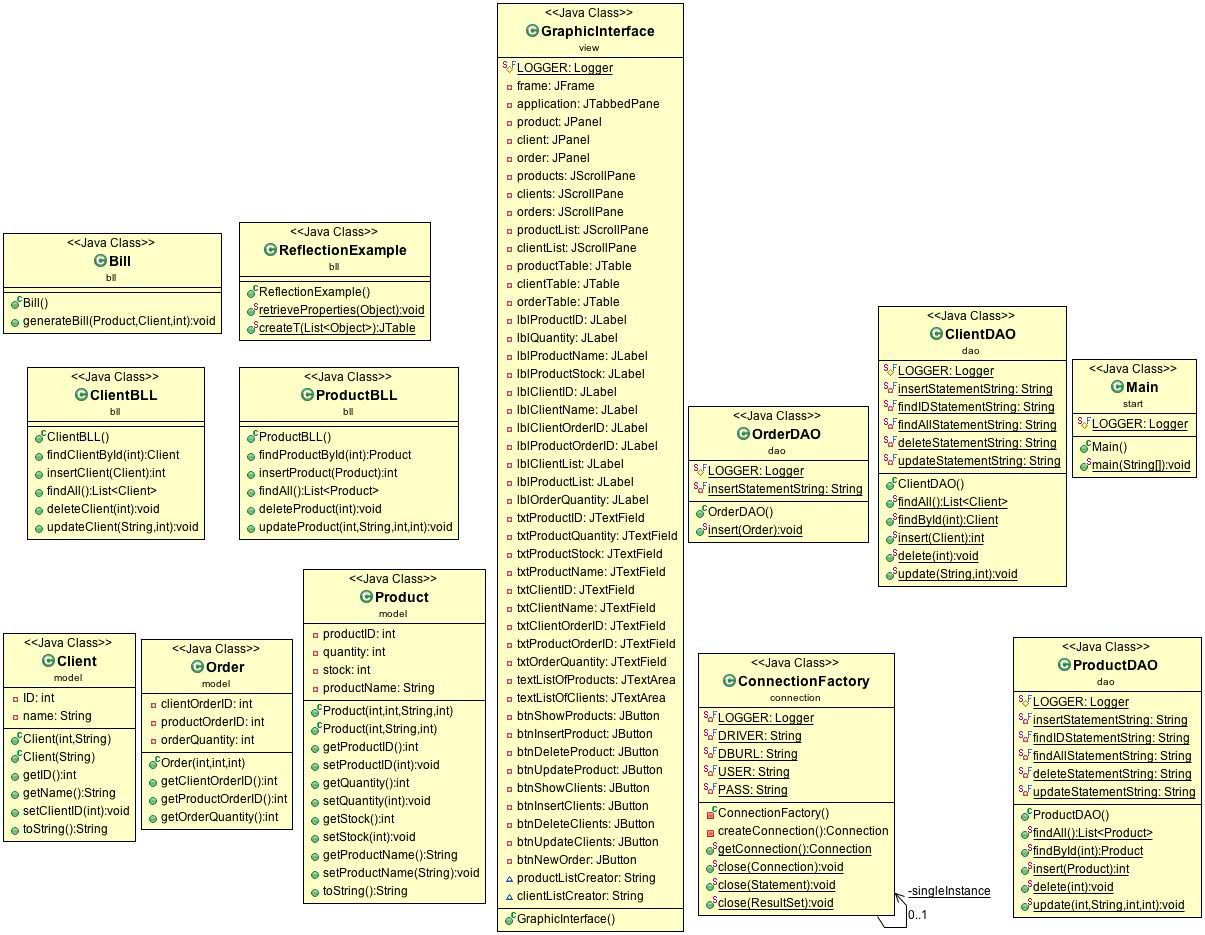
Scenarios : user places an order

1. The user starts the application
2. The user goes to the Order panel, which is the only panel that he / she should work on.
3. The application displays a list of all the available products and all the clients in the database.
4. The user chooses the id of him in the database, and the id of the products that he / she wants to order.
5. The user also enters a quantity, meaning how many products of that kind does he / she want to order.
6. Beneath, in a table, all the orders and products ordered are shown.
7. The application terminates.

Error case:

If the user enters a quantity that is greater than the available stock and tries to place an order, the application will display that there are not enough products in the warehouse, and what the available stock for that particular product is.

Also, if the user tries to enter a product id that is not in the list, an error message will be displayed.



Design

Packages:

For this particular project, I have used many packages in order to better divide the classes by their functionality. These packages are based on the concept of 3-layer architecture.The “ bll ” package contains the classes that deal with the bussiness logic of the application, where the enterprise running the app can choose restrictions on what the user can do, and that also make the connection between the database and the abstract objects of the application, which all have to be linked together. Said more simply, the “ bll ” package contains the classes that encapsulate the application logic. The “ connection ” package contains the class that makes the connection between the application and the local server, in order for it to be synchronized with the database. The “ dao ” package contains the classes that do the required CRUD operations directly on the database. The “ view ” package defines the presentation layer, containing the classes that define the user interface. The “ model ” package contains the classes that model the actual tables from the database, these being mapped between them. The “ start ” package contains the class that starts the application, and also the class that creates the displayer JTables using generics.

Classes:

Model classes :

As I have previously stated, these classes are mapped to the database. They must contain the exact same attributes as the entities in the database, and should replicate their behaviour. This package contains three classes : “ Order ”, “ Product ”, “ Client ”. Other than creating a second constructor for the class “ Product “, so that when inserting a new one, we do not need the ID, as it is generated automatically, these classes contain just what you would expect from them : attributes, the exact same ones as the entities they model, constructors and getters and setters for each attribute, so that we can obtain and set them anywhere in our application. Also, each of these classes contains the method toString(), overriden, so that we can display information about each model, where it is needed. These are useful because not only do we want to make changes in the database with our operations, but also to have lists of objects instantiated from these classes. This way, our database and application are strongly connected, each entry in the database having its corresponding object in its designated list.

“ bll “ classes

These are the bussiness logic classes, the ones that act as an intermediary between the data access layer and the presentation layer. The application has a separate class for every entity, namely “ ProductBLL ” and “ ClientBLL ”. They work with the methods from the “ dao ” classes, and create new objects, or lists of objects, where it is the case. Also, where it is needed, they check if the information provided is valid, and if not, they throw exceptions that signal the user of the mistake that has been made. This package also contains the class responsible for creating a bill for each order. The bill, in fact, is a text file containing information about the orders, the clients that placed orders and the products they have bought.

“ connection ” class

This package contains the class with the same name ( but with upper case letter at the beginning! ) that manages the connection to the database. It

contains the name of the driver (initialized through reflection), the database location (DBURL), and the user and the password for accessing the MySQL Server. The connection to the DB will be placed in a singleton object. The singleton pattern is a [software design pattern](https://en.wikipedia.org/wiki/Software_design_pattern) that restricts the [instantiation](https://en.wikipedia.org/wiki/Instantiation_(computer_science)) of a [class](https://en.wikipedia.org/wiki/Class_(computer_programming)) to one [object](https://en.wikipedia.org/wiki/Object_(computer_science)), because only one object is required to coordinate actions across the system. The class contains methods for creating a connection, getting an active connection and closing a connection, a Statement or a ResultSet.



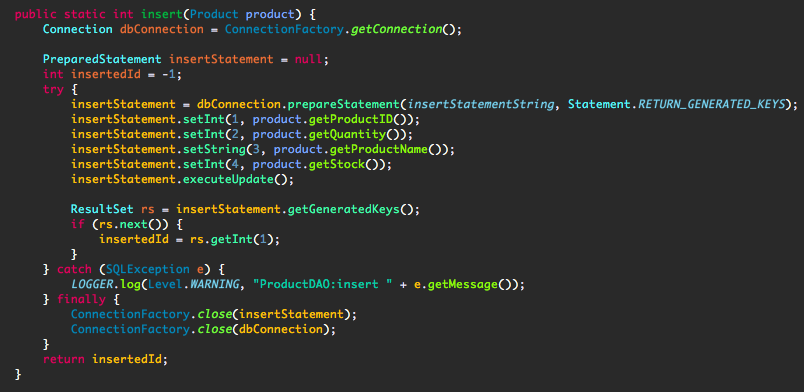


“ dao ” classes

These are the classes containing the queries. They operate directly on the database and they execute queries on it, using statements and result sets. A statement is used to create and execute a query. A string containing the query is created. Then, a connection to the database is created, and the query is initialized. Each parameter of the query (“?”) will be replaced with the corresponding values received as parameters by the method. The results of the query are placed in a result set. Each element of the result set corresponds to a row from the table. It can be iterated, so that we can easily reach every row from the table that we need.

Each query is executed inside a specific method. The basic principle is the same : we take the string containing the query, we fill out the unspecified parameters with the ones that we receive from the header of the method and we execute the now complete query. Depending on the case, if the method returns something or not, we can iterate through the result set containing the results of the applied query, and create a new object with these attributes or not. For example, the method that inserts a client / product into the database, return the id of the newly introduced object, because the ID is generated by the database automatically, and so we do not know it by default, only if we set the ID to be the one returned by the “ insert ” function.

At the end of each operation, the connection must be closed (the result set, the the statement and also the connection).



Graphic Design class

This is the class that implements the graphical user interface. It has the aim of helping the user of the application ( client or administrator ) to do his / hers available operations, in a friendly and intuitive way. It is based on a tabbed pane, each pane having specific operations on one of the three tabels : “ Order ”, “ Product ”, “ Client ”. It may not be the most fancy, nor the most complex user interface, but the text fields and labels make it easy for the user to know what kind of data he / she should introduce, and where. Also, the buttons that perform the desired operations are very visible and have suggestive names, so that the user knows exactly how to do what he / she wants. The client and product panels are pretty much the same. On the top of the panel, the user can introduce the data he wants in text fields, each text field being preceded by a label telling the user what data is needed. Following this, there is a button for each operation, that, after being pressed, performs the specific operation. On the second half of the page, there is a scroll pane that contains the representation of the database table in the form of a JTable.

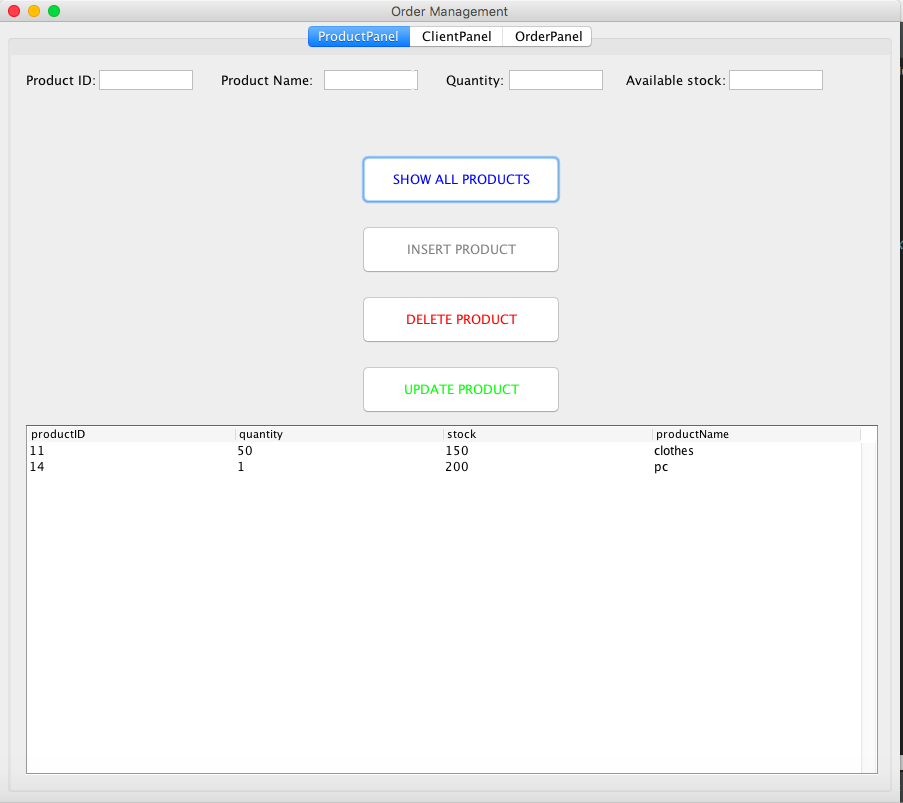
The order section is created for the user. The client can introduce an id of the product he / she desires, the id of him / her, and a quantity of the product to be ordered. Below, there is a list for all the available products and also for every client, so that the user knows what data to introduce. Again, on the second half of the page, each order is represented in a JTable.

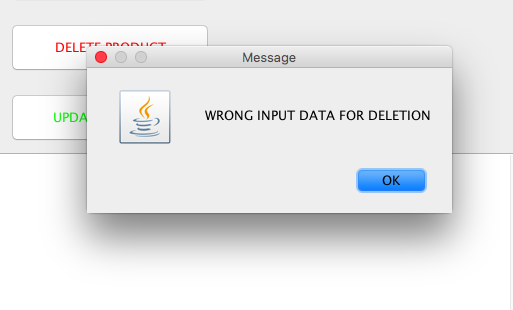
“ start ” classes

This package contains the class that starts the graphical user interface. Also, it contains a class that uses reflection techniques in order to retrieve properties of an object, and to create a JTable. Both methods are created using generics, and they can receive and object as a parameter, not only a specific set of them. It creates an array of strings representing the columns of the table, filling them with the name of the corresponding fields, obtained through reflection. Then, a default table model is created using these already created columns. for each column field from each row, the table is filled with the information from the actual database and displayed in a pleasing manner. Then, the method returns the created table so that it can be used anywhere it is necessary.

User interface

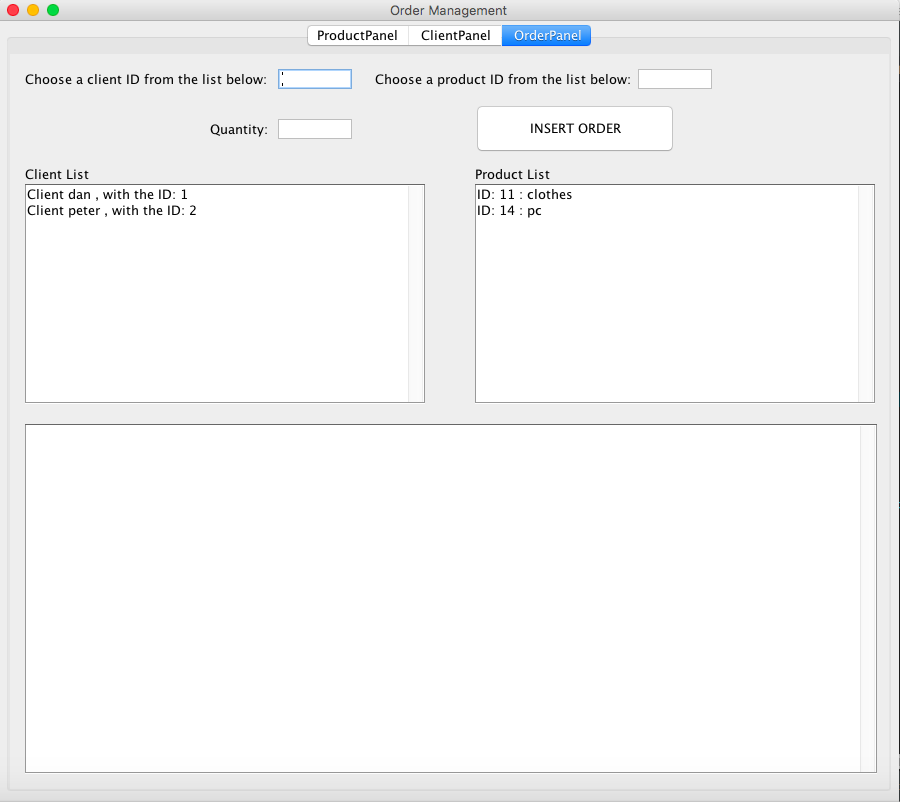
The first window that appears when the user opens the application is the Client pane. This one, and also the Product pane, are the ones that should be accessed only by the administrator, as they have the possibility to alter the database, and this kind of actions should not be permitted to any other user of the application.



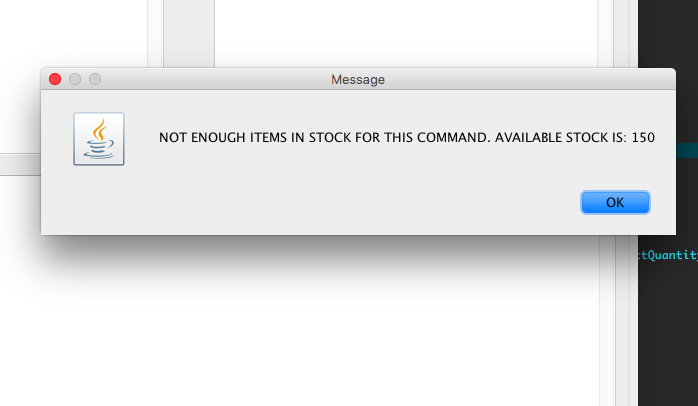
Each button, when pressed, checks if the input data of the administrator is correct, and if not, it displays a message telling him / her that the information entered is invalid. 

Each alteration of the objects from the database can be directly seen in the JTable from the bottom of the page, just by clicking on the “ Show all products ” button. The client panel has a very similar structure and the same functionality, the only difference being in the input text fields, that correspong to the actual table from the data base. The “ OrderPanel ” is the panel corresponding to the

client that wants to order something.



Each order will be displayed in the JTable from below. If the quantity entered exceeds the remaining stock of a product, a message telling this to the user will be displayed.



Implementation and testing

I began implementing the application by using the code we had at our disposal, to connect to the database. After I made sure I could connect to the server, I started implementing the model classes and the ones from the data access layer. Each method from each class has been tested using the already overriden toString() methods to see if they do what they should. I tested the queries by checking if the desired operations could be seen in the database.

After creating the GUI and displaying the information on each panel, I could easily test each method again, directly on the interface.

Results

The result is, after many hours of coding, an application that does the basic CRUD operations on a database containing multiple tables. Also, the resulting application is a perfect example of a database management system. With some improvements, I believe it can actually be used in a real situation.

Conclusions

What I learned

Besides refining my skills in writing java code, creating meaningful classes that replicate real objects, the main thing that I have learned is working with a database through a java program. I learned how to connect to a database, how to execute queries on it, and, all in all, how to create an application that is close to what most of the companies work with nowadays. I believe that this particular project will help me a lot when I will have to work for a company and deliver such applications.

Further possibilities for improvement

Although the application does all the basic operations it should, there are some areas that could be improved. First of all, when starting the application, there could be a panel that makes you choose the type of user that you are(client or administrator), so that not all the panels are available for every user. Another improvement that could be made, is creating more constraints. As an example, the administrator could not introduce more than 10000 items of one kind at a time. What is more, the client could have the option to see his past orders if he wants to.

Bibliography

1. <http://stackoverflow.com>
2. <https://docs.oracle.com/javase/tutorial/uiswing/layout/visual.html>
3. <https://wordcounter.net>
4. <https://en.wikipedia.org/wiki/Singleton_pattern>
5. <https://creately.com>