Order Management

Faculty of automation and computer science | TEHNICAL UNIVERSITY – CLUJ NAPOCA

GROUP 30423

Adrian - Radu Macocian

2018

# Assignment objective

Consider an application Order Management for processing customer orders for a warehouse. Relational databases are used to store the products, the clients and the orders.

Secondary Objectives:

* Use a model package for the model classes that represent the data of the application
* Use a business logic package containing the logic classes such as order processing and database management
* Use a presentation package that has all the swing classes and the graphical user interface implementation
* Data access classes which will keep the connection between the application and the database
* 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
* Implement a system of utility programs for reporting such as: under-stock, totals, filters, etc
* Create a product for a client
* Create the graphical user interface (Projection)
* Have a start frame which from where all the input data can be accessed
* Display all the occurring actions on the graphical user interface
* Create a bill for each order as a text file

# Assignment analysis, assumptions, use-cases, errors

The order management system should be able to connect to a database and then store or retrieve info from it. It is also able to change data already stored in the database.

The use-cases are a little more complicated: once we run the application, we have to choose whether we want to run client operations, or management operations. If we chose to run client operations we then have a choice of operating as an already existing client or to create a new client which we will be operating under afterwards. Choosing to run management options shows a list of all the existing orders and products, and gives the possibility to add new products or to cancel existing orders. The exceptional use cases are whenever the user tries to ignore an input field, or tries to pass invalid data to the application, in which case the program will signal the user that the input is invalid and not procced until valid input is given.

# Projection

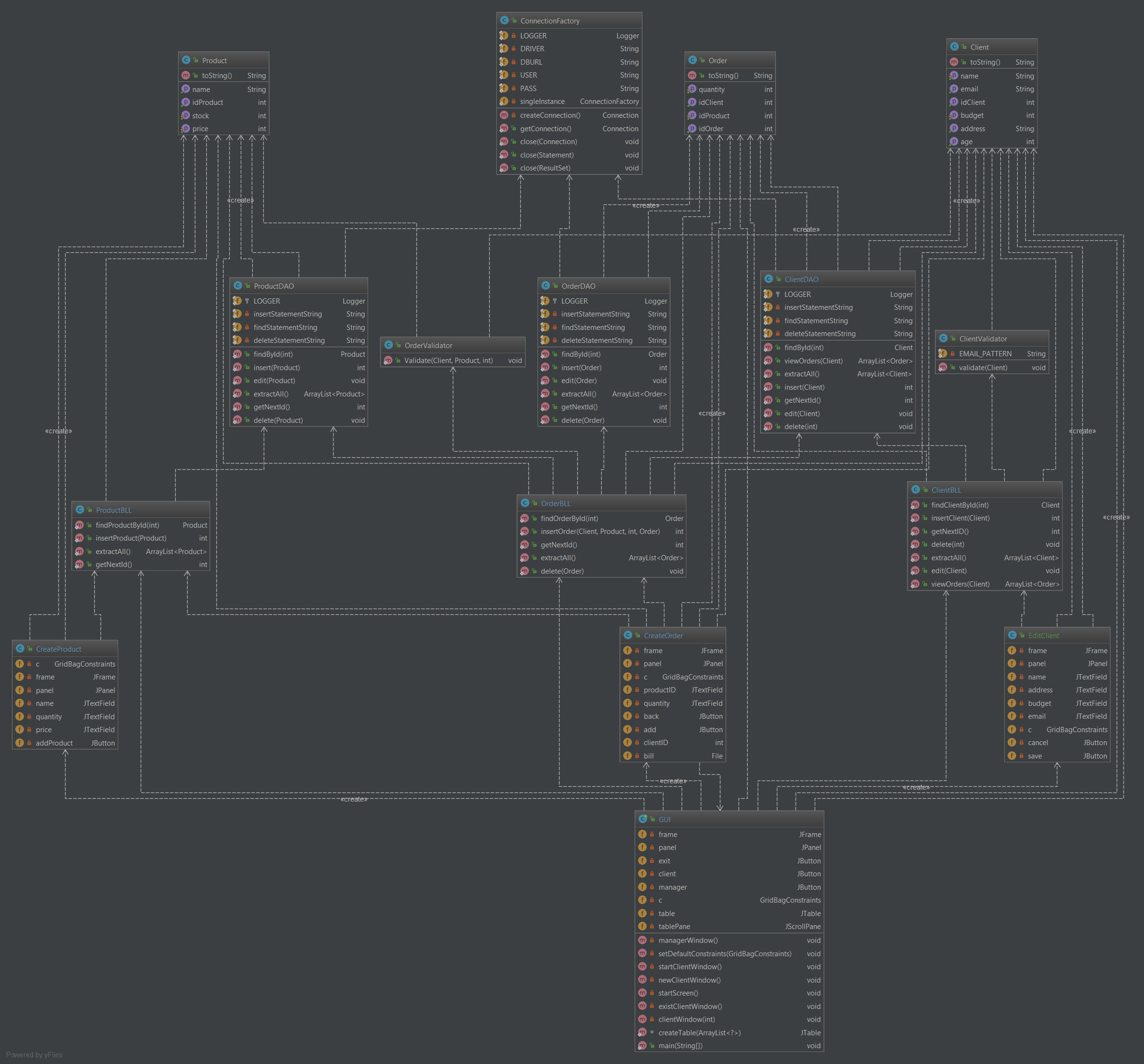
The whole project is split into 4 large areas: the model package, the data access package, the presentation package and the business logic package. There is a 5th smaller package used to validate the data entering the database called the validation package.

The model package stores the classes which would be equivalent to the data in the database so that they can be easily manipulated inside the application. Except for being passed as parameters for some of the methods for the database those classes have no other interaction with the database.

The data access package establishes the connection to the database and hold all the methods used to gather, edit, store or delete data from the database. It has some extra unused methods for deleting or getting some data which was not necessary for this application but were let there since they could be useful in some future upgrades of the application.

The business logic layer is the main connection between the application and the data access layer. Since the data access layer has direct access to the data base, some of the data that get there should be filtered so that we don’t end up storing unwanted or faulty data. The business logic package does just that with the help of the validation package. No other class or package in the application has any access to the data access layer except for the business layer

The presentation layer is the what its name says. Except it actually does all the heavy lifting all the application. While the data access and the business logic layers have the complicated logic behind them, the presentation layer is the wall of text – brute force that runs the whole application. Anything the can be done inside the application is done in the GUI therefore it is handled by the presentation layer.



# Implementation

The whole graphical user interface uses grid bag layout for all of the only panels it uses. The resolution for the main frame is set with an absolute value on 640 x 480 pixels and the position is set for the center of the screen. All the other frames are set on 320 x 240 pixels. The main GUI class has 6 methods, each of them restarting the panel and creating a new “frame”. It also stores the recursive method which creates a table based on an array of objects.

There are some buttons which reappear throughout the application which are stored as attributes of the class such as the exit button and the back button. The tables are created recursively and the panels reset whenever there is an update to one of the active tables

The other classes of the presentation layer are createObject, editClient and createProduct. Each of them does exactly what it says: create an object and store it on the database, edit one of the clients in the database or create a product and store it in the database respectively. All of them create a new frame which is half the size of the main one, on which the attributes of the chosen class can be entered and then passed on to the business layer.

# Testing

The application is used to create a connection to a database and manipulate its data. The Junit tests are not efficient in this case since there is no complicated logic happening in the backend and everything can be tested and verified through the graphical user interface. The only possible tests would be checking whether the connection to the database and the data manipulation is accurate, which can be easily checked through the user interface or in other cases, it can be checked through a database management application such as mysql workbench.

# Conclusions

The order management system works as a warehouse or store simulator that is capable of storing a number of clients, products and orders in a database and then communicate with it constantly. The application is also capable of exporting a receipt for each of the orders as it is created. It’s structured in an object oriented programming style having the 5 packages, each communicating and doing its own thing in order to create a fully working database management application. Each class has its own functionality and is essentially vital to the program. The data is passed through them continuously so each of them has to do its job correctly. The whole graphical interface is done manually, and although I considered (and tried) creating the user interface without using a layout I came to the conclusion that the software would look much cleaner and would be more responsive if I did use a layout. Since I already learned some of the basics of the GridBagLayout I used it for this application as well. No helping software was used, so now I’m even more capable of finding any bugs and fixing them a lot easier since I know exactly why I did something or why something is in a specific place. I’ve learned a lot about how to update a panel and frame constantly during runtime so that the data shown on screen is always accurate.

The only problem I found with the program is that some of the time the tables would not update when changed. Because of that I had to completely rerun an entire method to make sure that the table would update which is not entirely ideal.

I feel like I’ve learned a lot about database while working on this assignment. I knew vaguely how java applications connected to databases should work but I had no idea on how to use them and could barely design a software without even considering using connection factories and sql statements. This assignment thought me a lot about recursive programming since I didn’t understand it before and I was avoiding it until I had to really use it just now. In this assignment I’ve also learned how to update a GUI, and make sure its data is always up to date, while executing it through and outside class. Recursive programming is still confusing for me, but I can now handle the basics and I’m pretty sure that, if handled right, recursive programming can have a huge positive impact on a given software. I’m still unsure on when an error or warning will be triggered, that’s why I avoided creating the whole data access layer based on recursive programming.

One of the other big improvements in my coding abilities following this assignment is my ability to generic, although I couldn’t implement them in this application, I studied a lot about them while learning about recursive programming. I’ve never had to use them before and tried mostly using the same entities as for a normal application. Being forced to write a program that uses recursive programming I spent a lot of time digging through all of its possibilities. And to my surprise this was not even that hard to learn once you understand the basic principles of recursive programming.

Possible improvements:

One of the biggest improvements I could bring to this program would be to find a way to use all of the methods created in the data access layer since they are already there. Due to a lack of time I was unable to do this but giving full control to the user over the database is not impossible.

Another big improvement would be in the accessibility of the graphical user interface. It’s not very clear what each option would do so that’s one of the first things to be modified. Also there are too many panels the user has to go through before reaching one desired action.

A big clarity improvement would be changing the sql statements so that the tables do not show numbers and ids but they actually show names, and real values. This is again not a logic problem but more of a lack of time problem. Seeing numbers is easier for management but if the table showed the name of the client or the product as well it would give the whole application a much familiar filling.

Also one improvement would be that the bill that’s created after each order, could be created in a whole separate text file instead of just writing over the same one, and risking losing data in the case of an overwrite.

The business logic layer could also use some improvements since it could have some extra validations. Right now except for some validations to 2 of the methods in the business logic layer it’s methods just return what their respective methods in the data access layer return, giving them a sense of uselessness.

The flow through the graphical user interface could also be improved. While most of the time I tried to keep the possibility of going back or canceling an action, in some cases it would just ruin the whole look of the user interface so I chose to abandon it. Together with this improvement to the user interface I could also add more color to it. Right now it looks exactly like a java low-level beginner interface, and although it has a lot of panels it could still be changed to look a lot better with the right kind of imagination and time.

# Bibliography:

Most of the project was done by myself with the help of the laboratory professor, who gave us some ideas on how to use threads and on how to make our application as thread safe as possible. The only two external information sources were used for the creation of the graphical user interface and for creating the division of polynomials:

<https://docs.oracle.com/javase/tutorial/uiswing/layout/visual.html>

For a debugging and problem solving there were a lot of searches on stack overflow but I can’t pin point exactly to which url’s I’ve used:

<https://stackoverflow.com/>