Stoica Razvan Cosmin

Group 30424

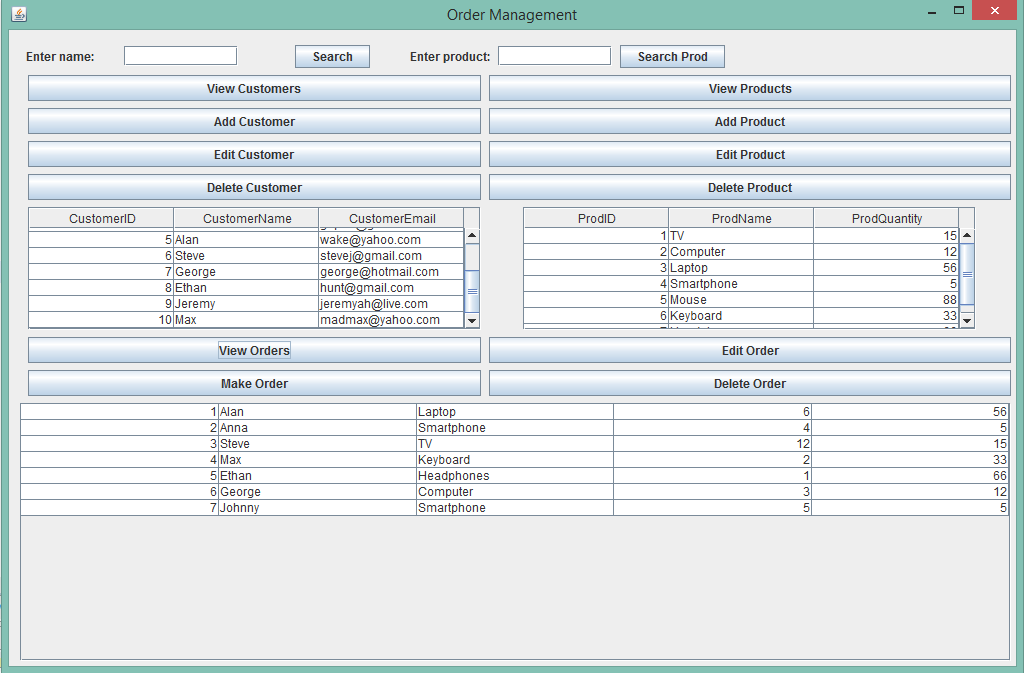
**Order Management**

1. **Homework objective**

Make an “Order Management “application for processing customer orders which connects to a database and modifies the data inside of it.

1. **Problem analysis**

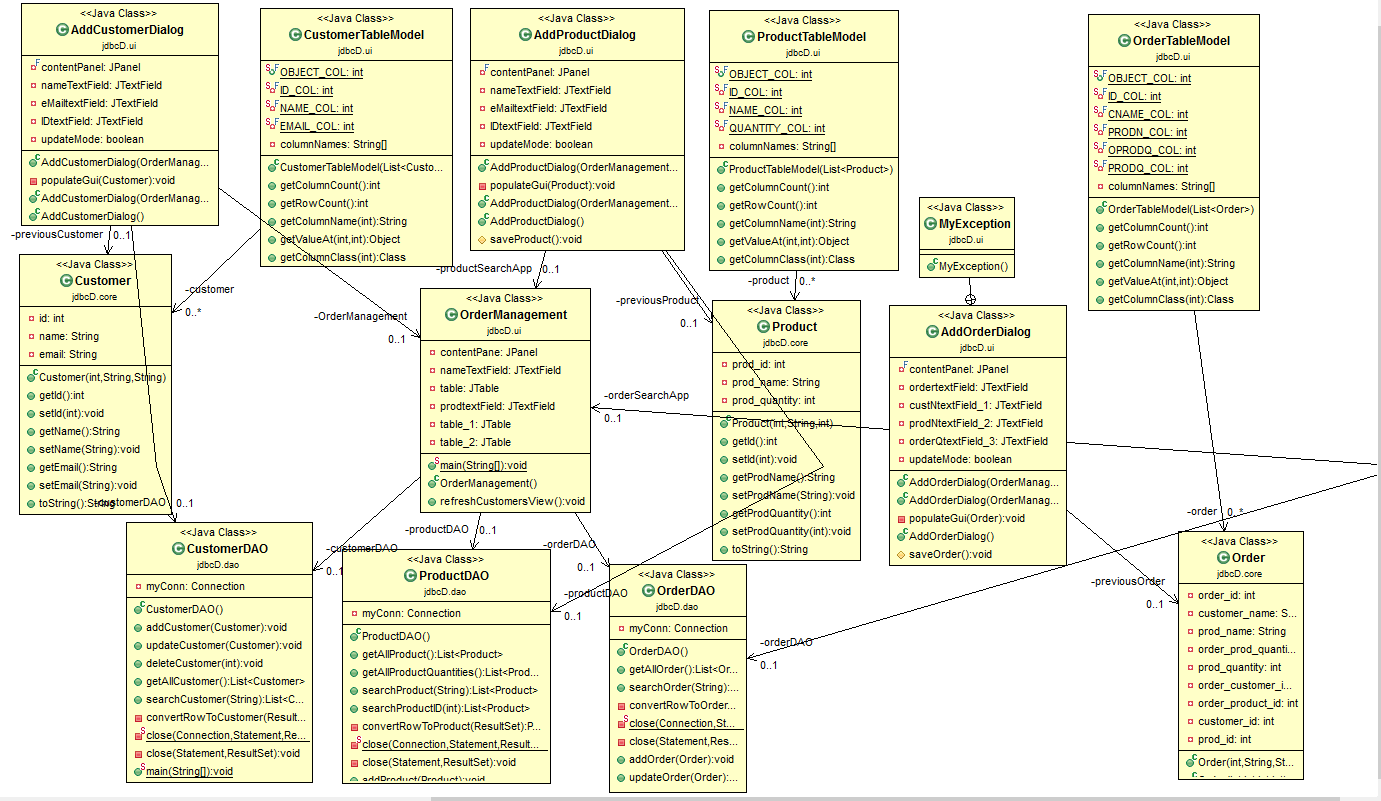
The program should be as user friendly as possibly. So there should be a button for every operation. For the Customers we have View Customers, Edit Customers, Delete Customers and Add Customers, as well as a search bar, to search them by the name. For the Products we have the same buttons as Customers. And for the Order we have a different view of the table, because we are interested in the name of the product, customers and order quantity not in the id (which is found in the order table in the database). Also for the orders we don’t need a search button.



1. **Design**

For the design of the project I’ve tried to use the DOA (Data access object) pattern in order to connect to the database. The database that I’ve created is called “ordermanagement” and has 3 tables that will be used by the applications: customer, product, order. I’ve created the database in MySQL Workbench, and I’ve stored the address to it in a file called om\_properties. The order table contains the rows order\_product\_id and order\_customer\_id which are used to join the product and the customer tables.

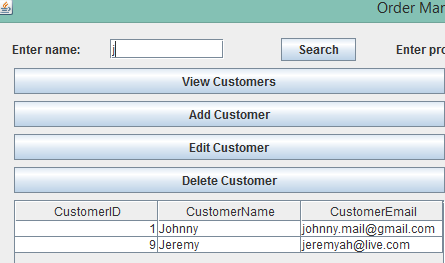
The project contains the following customer classes: Customer, CustomerDAO, CustomerTableModel, AddCustomerDialog. It also contains the following products classes: Product, ProductDAO, ProductTableModel, AddProductDialog and the following order classes: Order, OrderDAO, OrderTableModel and AddOrderDialog. All of the classes are used in the interface class called OrderManagement. There is also a MyException class used to display a pop-up in case the product quantity is lower than the product ordered quantity.



UML DIAGRAM

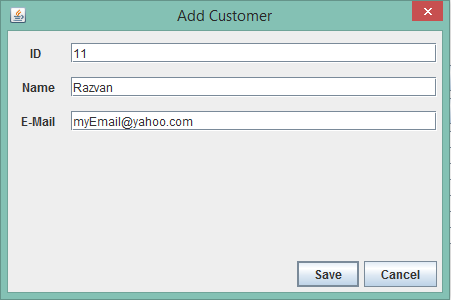
The Customer class is used in order to get the information about the customer, like the id, the name and the e-mail. It contains a constructor as well as getters and setters for the object attributes. The CustomerDAO class is used to connect to the database (via constructor) and it also contains methods that operate on the database like: addCustomer which inserts into the database a new customer, updateCustomer which updates an already existent customer, deleteCustomer which deletes a customer. Another methods are the getAllCustomer() which returns a list of all the customers from the database and searchCustomer which searches for a customer name and returns a list containing the customer with the searched name. The method converRowToCustomer() gets the values from a row and calls the constructor with them and the returns it. The close method is used to close the statement used for connecting to the database which are called in the methods that I just mentioned. The CustomerTableModel creates the table rows of the customer with a string and a list of customers. Which are set by the overridden getValueAt() method.

The Product classes do pretty much the same thing as the Customer classes except for some additions. In the ProductDAO class we have a searchProductID() class which will be later used in order to verify if the quantity of a product is the same as the quantity of the ordered product. The method uses an SQL statement and returns a list with the product found at the specified id.



The Order class is a bit different then the Customer and Product class because here we have 2 constructors one used when displaying the table, because we have to join the two table in order to display the name of the customer and the name of the product and one used when adding/deleting or editing an order. It also contains the getters and setters for each column in the table. For the OrderDAO we have a getAllOrder() method which now uses a more complicated SQL statement in which we inner join by the id, order table with customer and product table and then we return a list with it. The insertOrder() and updateOrder() methods use the columns found in the order table (order\_id, customer\_id, order\_product\_id and order\_prod\_quantity). The order table model contains only the rows to be displayed which are done with the getValueAt() method.

The OrderManagement class represents the graphic interface of the application, here we buttons for View/Add/Eddit/Delete Customer/Product which call the methods previously mentioned, as well as View/Make/Edit/Delete Order. We also have 3 scroll panels each containing the tables in which the contents of the database tables are shown. By adding a customer we call a dialog which is set in the AddCustomerDialog class which has 3 text fields in which the user writes the id, name and e-mail of the customer these are saved and create a new customer whith the saveCustomer() method which is used for adding and updating the table. The AddProductDialog class is basically the same but with corresponding data modified. The AddOrderDialog however has a new exception for the case in which the product quantity is smaller then the ordered product quantity which will throw an exception message “OVERSTOCK” using J Option Pane. The Search buttons use the searchCustomer() or searchProduct() methods in the case in which we type something in the search text field, if not it displays all the customer/products.

****

As for the used packages we have java.util.\*, java.sql.\* and java.io.\* for the DAOs, java.util.List and javax.swing.table.AbastractTable for the TableModels, and for the interface classes we have java.awt.EventQueue, javax.wing.JFrame, javax.swing.JPanel, javax.swing.border.EmptyBorder, javax.swing.JLabel, java.awt.FlowLayout, javax.swing.JOptionPane, javax.swing.JTable, javax.swing.JTextField, javax.swing.JButton, javax.swing.JScrollPane, java.util.List as well as action listeners and com.jgoodies.froms.\*. I also created 3 packages for the ui, core and the dao where I put the created classes.

1. **Implementation and testing**

The operations work correctly if they are written in the expected format, if not the program will throw an exception (the user will see a pop-up message that will warn him/her if he did something wrong). I tested each class before using the user interface, and I tried to print every possible outcome to make sure that the application doesn’t break or throws unexpected exceptions.

1. **Results**

All the given operations work correctly and the application displays error messages in case the user inputs something wrong, I tried to cover most of the cases in which this would happen.

1. **Conclusion and further developments**

As a conclusion the project needs a lot of improvements in order to be optimal and bug free, and as for further developments I think that having an invoice when making an order and having more tables like the price of the product and the address of the customer would improve the quality of the project. Also a better graphic interface with better placed buttons and tables would be a plus.