Documentation
Customer Order Management

Student: Rusu Daniel
Group: 30422

Table of Contents

1.Objective

Main objective of the assignment:

Consider an application for processing customers orders for a warehouse.
Relational databases are used to store the products, the client and the orders.

Secondary objectives:

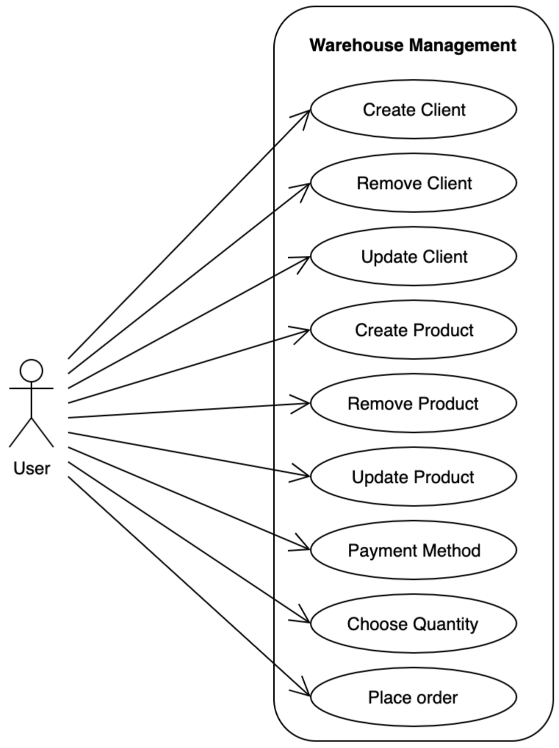
- Creating a good looking, fully-functional user interface
- Creating a database that matches the requirements
- Implementing connection to the database
- Implementing algorithms to run on the data in the database tables
- Linking the user interface to the back-end of the application, in order to display the output correctly to the user and receive the right input from the user.

All the secondary objectives presented above will be detailed in the fourth chapter of this documentation.

2.Problem analysis, modelling, use-cases

The application is built to perform operations on databases: create, read, update, delete. It shows the user the contents of 'client' and 'product' tables, and allows him/her to operate on these tables: add a new client/product, remove or update the data of an existing one.
The main goal of the application is to provide a way to place and manage orders easily, without having to interact with the database. It will also block any attempts of introducing wrong data, using the implemented input validators.

To better illustrate how the application can be used, I have built a use-case diagram:



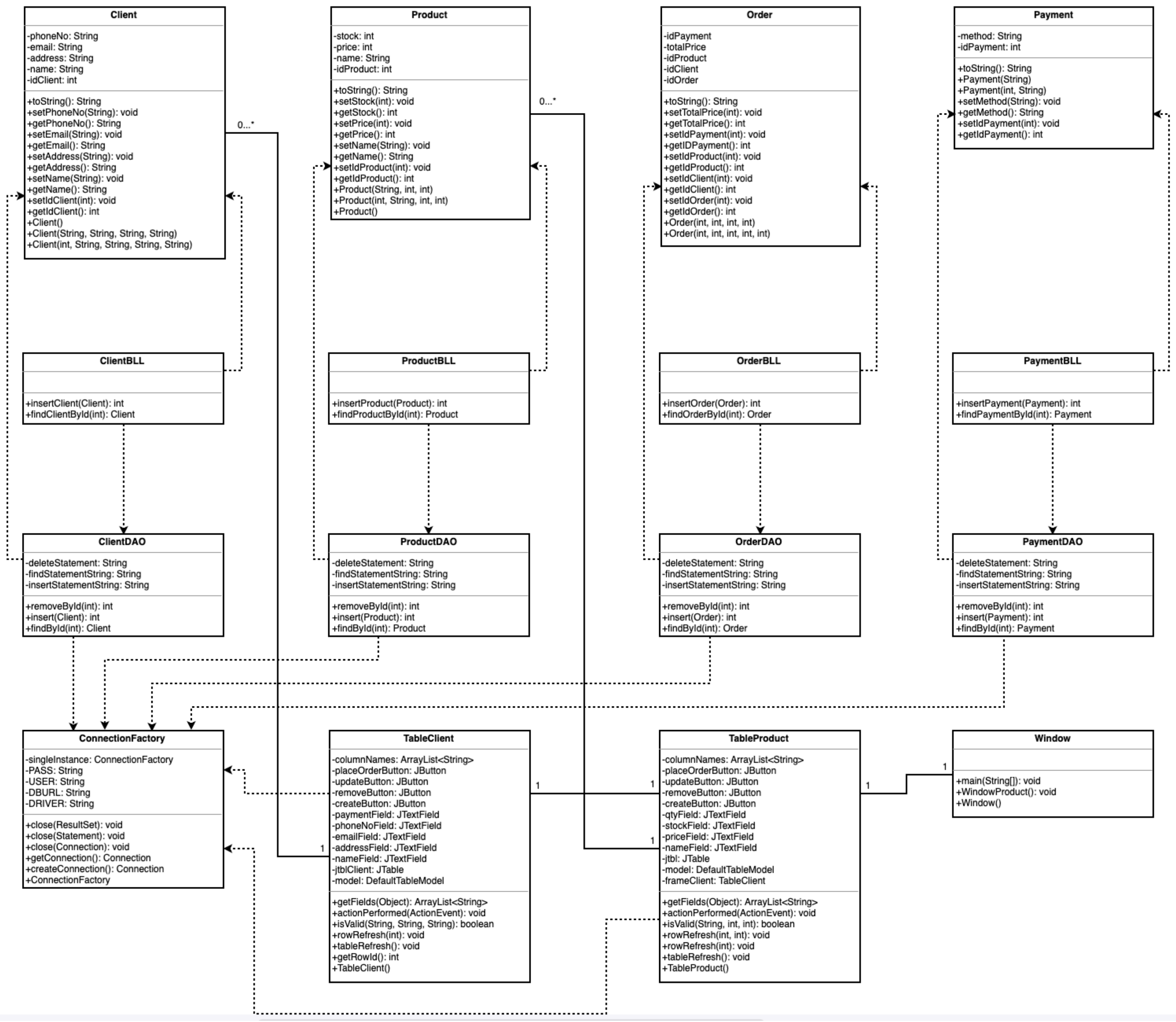
Use-cases description:

- **Create Client:** The input used to provide data about the client to be introduced is checked to see if everything is valid. Then, the provided data is loaded into the database ‘client’ table, on a new row, and then the whole table is displayed to the user in the graphical user interface. The display process is done by erasing all the rows in the JTable and reading the contents of the database table again. This process makes sure that the JTable contents in the user interface match the contents of the database table all the time.
- **Remove Client:** This button is used to delete a row in the table. The row to be deleted needs to be selected by the user first. Then, the table display process is similar to the one used for the ‘Create Client’ use-case: all the rows are removed from the JTable in the user interface, then read from the database table and displayed again.
- **Update Client:** Input used to modify the fields of a client in the database table based on the selection made in the UI JTable and the input given on the UI text fields. The input is checked to be valid, and if it is, the database table is modified and the user interface table is refreshed, as in the cases above.
- **Create Product:** Input similar to the ‘Create Client’ one. It is used to read data provided in the UI text fields and load it into the database ‘product’ table. When this button is pressed, the input on the text fields is verified and if it is valid, the database table will be appended this product. Then the UI JTable is refreshed, so that it displays the latest state of the database table.
- **Remove Product:** Input similar to the ‘Remove Client’ one. It is used to remove a row (entry) of the ‘product’ database table. The row to be deleted needs to be selected in the UI JTable before pressing the ‘Remove’ button. After the deletion process is over, the UI table is refreshed, as in the cases above.
- **Update Product:** Input similar to the ‘Update Client’ one. It is used to modify the fields (other than ID) for a certain row (entry) in the database table. The row to be updated needs to be selected in the UI JTable before the ‘Update’ button is pressed. After the button is pressed and the update process is over, the table is refreshed.
- **Payment Method:** Input used to select which type of method the user will use to pay for the chosen products. The user needs to introduce ‘1’ for cash and ‘2’ for card in the text field. These values correspond to the id’s in the ‘payment’ table (of the database). When processing a new order, the payment method id will be inserted in the ‘idPayment’ column of the ‘order’ table in the database, after pressing the ‘Place Order’ button in the graphical user interface.
- **Choose Quantity:** This input is used to let the user choose the amount he/she wants to buy, once they have selected the desired product. The value is used to calculate the totalPrice in the ‘order’ table of the database.
- **Place Order:** Button used to signal that the client was selected in the client table, the product was selected in the product table, the payment method was chosen and the desired quantity was specified, so the application can retrieve that information and process the order. All the data specified above is read and processed by the algorithms, and then (assuming that the data is valid) a new row is created in the ‘order’ table of the database, containing the client id, product id, total price and payment id. The product stock is then decremented.

3. Design (Decisions, UML Diagram, Class Design, User Interface)

The software design approach is OOP style. The application uses a system of interacting objects for the purpose of solving the proposed problem.

The UML class diagram of the application is presented below:



The classes are illustrated as rectangles divided into 3 sections: the first one (up) contains the name of the class, the second (middle) contains the class fields and the third (down) contains the implemented class methods.

The relationships between different classes are represented using different types of links. There are multiple 'dependency' relationships drawn on the diagram. For example, there is such a relationship between 'TableProduct' (UI class) and 'ConnectionFactory' classes, because in order to display in the JTable of the user interface the data in the database tables, there needs to be a connection between them, so that the application can read from the database and add rows to the JTable. The same type of relationship exists between the 'ClientDAO' and 'Client' classes, because some of the ClientDAO methods need information about the client they need to perform operations on, i.e. they are dependent on Client fields.
There are also association relationships drawn on the UML diagram above. The most obvious ones are between Client and TableClient, Product and TableProduct. These are one-to-many relationships, because each table of clients can contain multiple clients, and each table of products can contain multiple products, but they can only be added to one table of clients/products.

4. Implementation

The application consists of 16 classes, grouped in 5 packages. Each of the packages represents a part of the 3-tier architecture approach suggested for solving the proposed problem: Model, Presentation Layer, Business Logic, Data Access and MySQL Connection.
All the classes will be presented and detailed below.

- **The Client class (Model package)**

This is the class used to represent the client. It contains all the specific fields: idClient, name, address, email, phoneNo, that match the database 'client' table template.
Important methods implemented in this class:

- 'Client(String, String, String, String)' is one of the class constructors. It is used whenever the user wants to add a new client to the database 'client' table. The Strings are read from the input text fields in the graphical user interface and, if valid, used as parameters to this constructor.
- getters and setters for the class fields, so the fields can be read and updated from other classes.

- **The Product class (Model package)**

This is the class used to represent the product. It contains all the specific fields for such an object: idProduct, name, price, stock, that match the database 'product' table template.
Important methods implemented in this class:

- 'Product(String, int, int)' is one of the class constructors. It is used whenever a user wants to add a new product to the database 'product' table. It is similar to the constructor of class Client described above.
- getters and setters for the class fields, so that other classes can read and update the data of client objects, without the need to make them public in the Client class.

- **The Order class (Model package)**

This class is used to represent the order. It is linked to the 'order' table in the database that contains all the foreign keys to other tables. It needs to know fields such as 'idClient', 'idProduct' and 'idPayment' of other tables. These represent the class fields:idOrder, idClient, idProduct, totalPrice, idPayment.
Important methods implemented in this class:
  - 'Order(int, int, int, int)' is one of the class constructors. It is used when the 'Place Order' button is pressed in the graphical user interface, and the necessary data is read from the UI. This constructor is called to create an object of type order that will be then inserted into the database 'order' table.
  - getters and setters for the class fields, similar to the ones in classes described above.

- **The Payment class (Model package)**

This class is used to describe the payment. It is linked to the 'order' table in the database, to provide the payment id, when it is needed. The purpose of this class is to provide various methods of payment (cash or card) to the user, once he/she decides what to order. The class fields are: idPayment and method.
Important methods implemented in this class:
  - 'Payment' is the only class constructor. Used to create a new payment object and initialise its idPayment and method fields with the values provided as parameters.
  - getters and setters for the class fields.

- **The ClientBLL class (bll package)**

This class is used to encapsulate the business logic level of the application, with respect to the Client class. It should contain both methods that need to communicate with the database (that call the respective method in the 'dao' package classes) and methods that have nothing to do with the database (algorithms that work with objects of type client).
Important methods implemented in this class:
  - 'findClientById' is a method that receives as parameter an integer (the id of the client to be found' and calls the ClientDAO class method 'findById', in order to search the database 'client' table for the client with the given id. Can throw a NoSuchElementException, if the id provided does not exist in the client table.
  - 'insertClient' method receives as parameter an object of type client and calls the ClientDAO method 'insert', in order to access the database and add the client to the database 'client' table.

- **The ProductBLL, OrderBLL and PaymentBLL classes (bll package)**

These classes are used to encapsulate the business level of the application, with respect to the Product, Order and Payment classes. They are very similar to the ClientBLL class described above, each working with the specific object type (product, order and payment, respectively).

• The ClientDAO class (dao package)

Class used to provide methods to interact with the database, with respect to the objects of type client. Methods that implement basic CRUD operations on the database 'client' table should be declared here.
Important methods implemented in this class:

- 'findById' is the method used to search in the database 'client' table for a client with the idClient specified as parameter. It returns an object of type client (the one with the requested id in the database 'client' table). In order to search in the database, the method uses an sql 'SELECT' statement.
- 'insert' is the method used to add a new row (entry) in the database 'client' table. It receives as parameter an object of type Client and returns an integer that indicates the idClient the inserted client was assigned (it is auto-incremented by the MySQL database), or the value '-1' if it failed.
- 'removeById' is the method used to delete a row (entry) from the database 'client' table. It receives as parameter an integer indicating the idClient of the row to be deleted. The method uses a delete sql statement to remove the specified row and returns the id of the deleted row (just as a confirmation that it worked), or '-1' if it failed.

• The ProductDAO, OrderDAO and PaymentDAO classes (dao package)

These classes are very similar to the ClientDAO class described above. Their role is to interact with the database tables (ProductDAO with 'product' table, OrderDAO with 'order' and PaymentDAO with 'payment'). The CRUD operations on the database tables are implemented in these classes.
Each of these three classes has 'findById', 'insert' and 'removeById' methods, similar to the ones in the ClientDAO class, that manipulate the corresponding database tables.

• The ConnectionFactory class

This class is used to establish, create and close connections with the MySQL database. It has the specific fields: DRIVE, DBURL, USER, PASS.
Important methods implemented in this class:

- 'createConnection' is the method that creates the connection with the database and returns it (object of type Connection)
- 'close' - there are three 'close' methods, one for connection, one for statement and one for result set.

All the methods and fields present in this class are specific to such an application and are vital to it, since it has no use without the link to the database.

• The TableClient class

This class is used to build the graphical user interface window that contains the Clients JTable, the create, delete, update buttons and fields associated with them, and the payment method input. It also contains the action listener part of the user interface, designed for the buttons mentioned above.
Important methods implemented in this class:

- 'TableClient' is the class constructor. Since this is a user interface class, this is where all the buttons, fields and labels are implemented. Also, this constructor is used to create the JTable that displays the contents of the 'client' database table.

- ‘tableRefresh’ method is used to delete all the JTable rows and re-load them from the database ‘client’ table. This method is useful after the contents of the database were modified, to make sure that what the user sees in the user interface always matches what is in the database table.
- ‘rowRefresh’ method is used when the ‘Update’ button is pressed in the UI. It makes changes in the database, in order to edit a certain row. After the ‘Update’ button is pressed, the selected row in the user interface JTable is edited and then the whole table is refreshed, using the ‘tableRefresh’ method described above.
- ‘isValid’ is the method used to validate the input on the text fields whenever the ‘Create’ button is pressed.
- ‘actionPerformed’ method is used to check which button was pressed and perform the corresponding action: create, remove or update a row in the database ‘client’ table.
- ‘getFields’ is the method that uses reflection in order to generate the JTable header. It receives as input an object of type Object and returns an ArrayList of Strings containing the names of all the declared fields in parameter’s class.

• The TableProduct class

This class is very similar to the TableClient class, except that this is the class where TableClient is initialised (so that they are linked).
This is the class used to build the graphical user interface window that contains the ‘Products’ JTable, the create, update, delete buttons and the fields and labels associated with them, and the product quantity field and ‘Place Order’ button.
Important methods implemented in this class:

- ‘TableProduct’ is the constructor of this class. It is very similar to the one in the TableClient class. This one misses the payment method field and label, but adds the desired quantity field and label, and the ‘Place Order’ button.
- ‘tableRefresh’, ‘rowRefresh’, ‘isValid’ and ‘getFields’ methods are similar to the ones described above, in the ClientProduct class.
- ‘actionPerformed’ is the method that checks which button was pressed and performs the corresponding order. The only difference between this and the actionPerformed method of the TableClient class is that this one check for the ‘Place Order’ button. When this is the case, the application reads all the necessary inputs in the user interface and creates a new order, by inserting a new row (entry) in the database ‘order’ table.

• The Window class

This class is used to start the application.
Important methods implemented in this class:

- ‘Window’ is the class constructor. All it does is to call the constructor of the TableProduct class (which calls the constructor of the TableClient class).
- ‘main’ is the main method of the application. It calls the Window constructor to start the whole process.

5. Results

When running the application, the user can observe some of the results in the user interface. For example, the client table and product table in the UI can be modified directly by the user (using the create, remove and update buttons) and the response is immediate: the tables displayed change.
However, when placing an order, the only hint that it worked (other that the pop-ups for invalid input) is the fact that, in the product table, the selected product stock was decremented with the amount ordered. The main change, in this case, is in the database 'order' table. The result of the 'Place Order' button press is the insertion of a new row (entry) in this table. This table is not visible to the user and remains visible only to the application because of privacy policies between users.

6. Conclusions

Designing this application was very interesting and useful, because it helped me understand and learn how a Java application can communicate with a MySQL database. This is very important because nowadays, most of the applications have a database to work with (whether it is only for storing user account data or a lot more user information).

Future developments:

- Implement an AbstractDAO class that can work with any type of Object, in order to automate the whole process of interaction with the database.
- Implement a feature that allows users to log in using an username and password, instead of selecting clients from the clients table.

7. Bibliography

• Pdf presentation document received on the link below:
http://coned.utcluj.ro/~salomie/PT_Lic/4_Lab/HW3_Tema3/

• Java reflection tutorials:
https://www.geeksforgeeks.org/reflection-in-java/

• Java Swing tutorials:
https://docs.oracle.com/javase/tutorial/uiswing/layout/gridbag.html
https://www.javatpoint.com/java-swing

• Other links indicated in the assignment paper:
http://coned.utcluj.ro/~salomie/PT_Lic/4_Lab/HW3_Tema3/Tema3_HW3.pdf