Technical University Of Cluj-Napoca

Programming Techniques

Assignment 3

ORDER MANAGEMENT



Student: Bogdan Tudor-Alexandru

Group: 30424

Teacher: Ioan Salomie

Teacher Assistant: Pop Cristina

**Main Objective**

Design and implement an application aimed to process customers, products and orders for a warehouse. To store the data, relational databases will be used.

**Sub-Objectives:**

* Analyze the problem and identify the requirements
* Design the application
* Implement the application
* Provide the user with an easy way to manage the clients
* Provide the user with an easy way to manage the products
* Provide the user with an easy way to manage the orders
* Provide the user with bills for the fulfilled orders

**Project Specifications**

1. **Problem Analysis**

The warehouse holds products and receives orders from certain clients. The application provides an easy way for the user to manage the clients, the products in the warehouse and the orders made. Through the application, the user can INSERT, UPDATE or DELETE certain clients or products from the warehouse’s databases, and can also PLACE ORDERS when the requirements are fulfilled (the stock quantity >= the wanted quantity) which will also provide the user with a bill for the processed order as a text file.

For storing the information regarding the warehouse, the application is using relational databases. The database name is “schooldb” and has 3 tables:

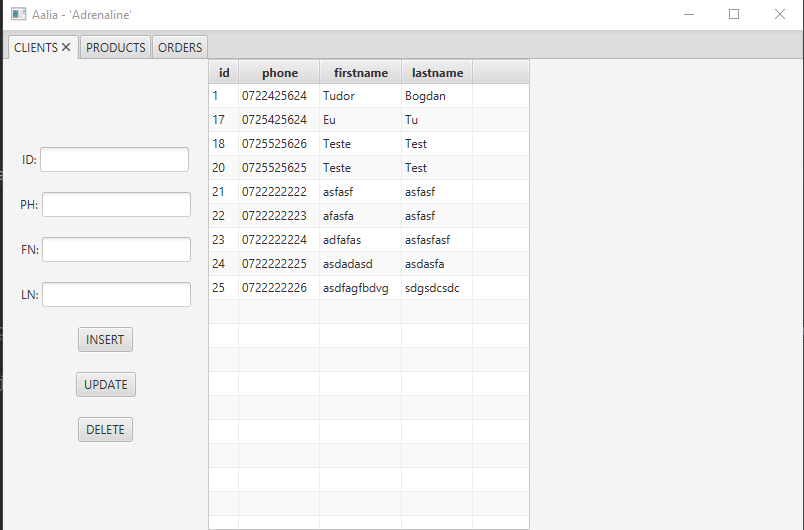
* Clients: for storing the data of the clients (id, phone number, first name and last name), with the phone number being unique
* Products: for storing data about the products in warehouse (id, name, quantity and price per unit), with the name of the producy being unique
* Order: for storing the information about the finalized order (id, id of the clients that made the order, id of the product wanted, quantity of the products wanted)

1. **Functional Requirements**

* The application should allow the user to INSERT, UPDATE and DELETE clients from the database of the warehouse.
* The application should allow the user to INSERT, UPDATE and DELETE products from the database of the warehouse.
* The application should allow the user to PLACE(insert) orders into the database of the warehouse.
* The application should allow the user to visualize the modification of the state of the warehouse (clients, products, orders) as soon as a modification has been done (an insertion, deletion, update).

1. **Uses Cases**

In order to make the application easier to use and reduce the errors made by the user the application should have and easy to use and intuitive user interface.

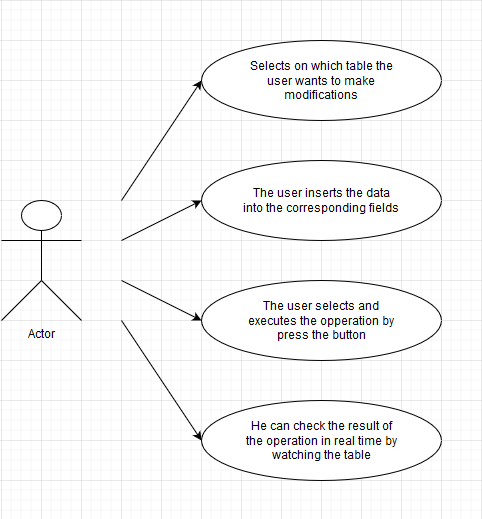


For a proper functioning of the application, the user should fill in all the fields in the left side of the selected tab, each text field corresponding to one of the fields in the corresponding class and column in the table. The Labels are an intuitive abbreviation to the name of the fields in classes and column in the tables in the relational database. Also, the ID, QUANTITY and UNITPRICE should be nonnegative integers, and nonnegative double, while the PHONE should be a 10 characters long String. The rest of the fields are just Strings with a maximum length set.

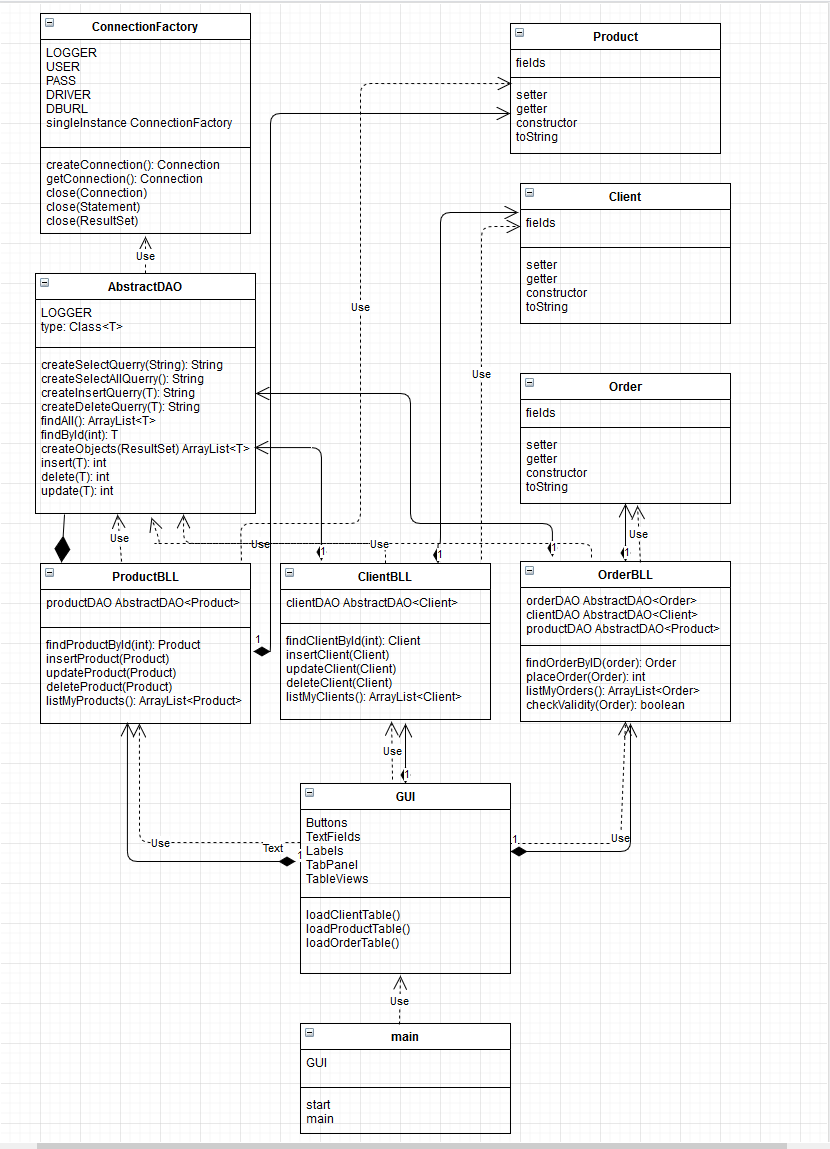
After filling in the corresponding fields the user just has to click on one of the buttons mentioned under the fields to execute the wanted operation. To do a simple INSERT the user can set the ID to 0 and the next id in order will be automatically selected. Also PHONE (client) and NAME(product) should be unique for all the data in the table.

1. **Design**

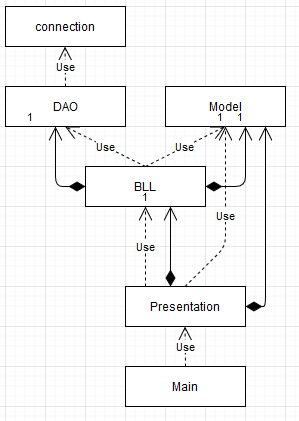
**Use Case Diagram:**



**Class Diagram:**

****

**Package Diagrams:**

****

**Data Structures:**

The data structures used to implement the application are both primitive and complex. The primitives ones are INT and DOUBLE, mostly used for the fields of the objects describing the tables in the relational database, while the complex ones are STRING and ARRAYLIST, used for also describing the content of the database but also to retrieve the content of the database. The class AbstractDAO is using generics and reflection to be able to implement the same methods for all of : Client, Product and Order. Array List is very useful because it is easy generalized to return the content of the Table.

**Packages**

For the structuring of the application I used a Layered Architecture pattern, which is very common and used in these kinds of projects. The main package (ordermanagement) contains the following packages:

* BLL – the business Layer
* Connection – the data access Layer
* Dao – also part of the access Layer
* Model – the Model Layer
* Presentation – the Presentation Layer

The business layer (BLL) contains the classes that encapsulates the classes containing the application logic. It contains the classes:

* ClientBLL
* ProductBLL
* OrderBLL

The data access layer is composed of the 2 packages : Connection and Dao, which contains classes that have the querries and the connection between the data base and the application. It contains the classes:

* ConnectionFactory
* AbstractDAO

The Presentation Layer contains the classes defining the user interface. It contains the following classes:

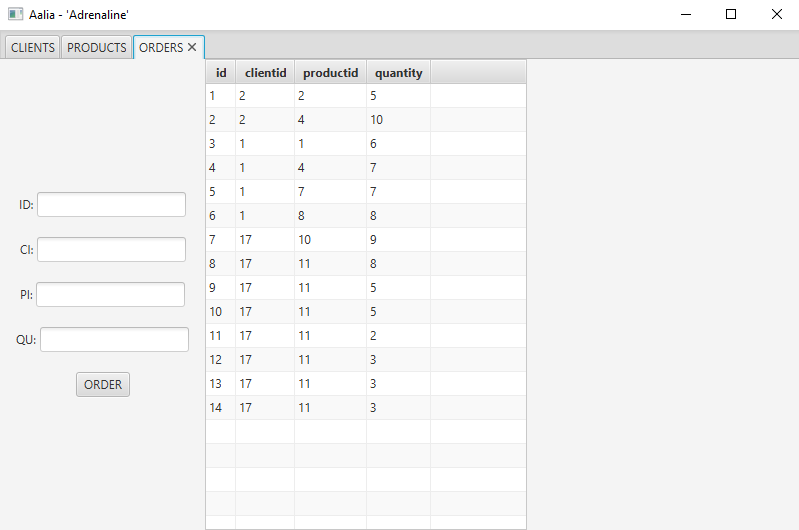
* GUI

The Model contains the classes mapped to the tables in the database. It contains the following classes:

* Client
* Product
* Order

**User Interface:**

The GUI (graphical user interface) is that the user of the application sees when running the application. To make the application easier to use and reduce the mistakes made by the user, the user interface must be user-friendly.The user interface have been done using the JavaFX library, and by seeting on the main scene a TabPane with 3 tabs, each pane for a corresponding table: one for clients, one for products and one for orders. Each tabe is further composed by one controllers (for the inputs and the buttons) and the tableView to see the current elements in the database. The user should fill in all the fields in the left side of the selected tab, each text field corresponding to one of the fields in the corresponding class and column in the table. The Labels are an intuitive abbreviation to the name of the fields in classes and column in the tables in the relational database. Also, the ID, QUANTITY and UNITPRICE should be nonnegative integers, and nonnegative double, while the PHONE should be a 10 characters long String. The rest of the fields are just Strings with a maximum length set.



**Implementation**

The implementation is done using and object oriented approach, with object oriented concepts, the most remarkable ones being generics and reflection technique. Each class is enrolled in a coressponding package to fulfill the Layered Architecture.

1. Model classes: Client, Product, Order

They are composed from setters, getters and constructors and helps quering data from the database through the reflection technique.

1. GUI
   * Public void loadClientTable() – refreshes the table showing the clients in the database
   * Public void loadProductTable() – refreshes the table showing the products in the database
   * Public void loadOrderTable() – refreshes the table showing the orders in the database
   * Public GUI() – the constructor class. It lays all the components in place and sets the action of the buttons when pressed.
2. AbstractDAO
   * Contains the generalized querries that will be particularized in the Business Logic Layer classes and will be further.
   * public AbstractDAO( Class<T> tipClasa) : constructor assigning the class type to the generic
   * private String createSelectQuery(String field) : Create Select Querry for a given field
   * private String createSelectAllQuery(): Create Select Querry for a all field
   * private String createInsertQuery(T myT) : Create Insert Querry for a given type
   * private String createDeleteQuery(T myT) : Create Delete Querry for a given type
   * public ArrayList<T> findAll() : returns an Array List containg all of the entires in the database
   * public T findById(int id): returns the first entry of the given type t if the id was matched, or null if id was not found in the table
   * private ArrayList<T> createObjects(ResultSet resultSet) : it creates an Array List of our objects for a given ResultSet
   * public int insert(T t) : it inserts the given item in the corresponding table and returns the id of the new entry if inserted succefull or 0 otherwhise
   * public T findById(int id): returns the first entry of the given type t if the id was matched, or null if id was not found in the table
   * public int delete(T t) : it removes the entry with the ID passed throught the object as a parameter if found in the table.
   * public int update(T t): if possible to update, it first removes the item from the table and makes a new insertion with the modified fields.
3. ConnectionFactory
   * The class implements the connection to the database and the methods related to its closure.
   * private ConnectionFactory() : instantiates a singleton obj
   * public static Connection getConnection() : retrives the connection with the database
   * public static void close(Connection connection) : closes the connection to the database
   * public static void close(Statement statement) : closes the connection to the database
   * public static void close(ResultSet resultSet) : closes the connection to the database
   * private Connection createConnection(): makes the connection to the database
4. ClientBLL class
   * containing the application logic regarding the client
   * public ClientBLL(): Instantiates a new Client bll
   * public boolean checkValidity(Client myClient): Check validity of the given Client
   * public int checkAvailability(String phone) : Check availability of the phone number among clients
   * public Client findClientById(int id): Find client by id client.
   * public void insertClient(Client myClient): Insert client in the Client table
   * public void updateClient(Client myClient): Update client.
   * public void deleteClient(Client myClient): Delete client.
   * public ArrayList<Client> listMyClients(): List my clients array list
5. OrderBLL class
   * containing the application logic regarding the orders
   * public OrderBLL (): Instantiates a new Order bll.
   * public boolean checkValidity(Order myOrder): Check validity of the given Order with regards to the product id and the client id.
   * public Order findOrderById(int id): Find order by id order.
   * public int placeOrder(Order myOrder): inserts order into the table  
     returns (-3) invalid data, (-2) no such product or client, (-1) not enough items, (0) orderPlaced **AND CREATES THE BILL IF ORDER IS SUCCESFULLY PLACED.**
   * public ArrayList<Order> listMyOrders(): List my orders array list.
6. ProductBLL class
   * containing the application logic
   * public ProductBLL(): Instantiates a new Product bll.
   * public boolean checkValidity(Product myProduct): Check validity of the given Product
   * public int checkAvailability(String name): Check availability of the name among products
   * public Product findProductById(int id): Find product by id product in product table
   * public void insertProduct(Product myProduct): Insert product into product table
   * public void updateProduct(Product myProduct): Update product into product table
   * public void deleteProduct(Product myProduct): Delete product.
   * public ArrayList<Product> listMyProducts(): List my products array list.

**Conclusion:**

I enjoyed working on this project because I got the opportunity of creating a java application connected to a database which helped me deepen my knowledge in both fields, while also getting hand on experience with working with the JavaFX and writing user interfaces using this library. I am also happy to learn how to work with the Javadoc, because it helped me see a new side of project documentation, which until now I found it pretty boring, even those it is one of the most important(if not the most important) part of the entire project, since it tell the user how, where, when, what and what not to do, but also everything in between. I enjoyed deepening my knowledge with regards to generics, but also with reflection, because, even though I find the user interface and the javaFX not to be so pleasant to look at(at least for me), I think that the backend part and the functionality of the application to be easily applied even now, with a much more developed technology.I think that this kind of application if the foundation to many apps that are very popular nowadays and I think that it has many ways to develop, either by adding an authentication feature and allow clients to place the orders themselves, and register themselves into the database, but also by making a much more pleasant interface, which usually boosts the popularity of the application. I hope that for a future project I will be able to try a much more please graphical user interface with the FXML features that JavaFX can work really nice with.

**Useful links:**

<https://www.oracle.com/ro/technical-resources/articles/java/javadoc-tool.html>

<http://tutorials.jenkov.com/javafx/tabpane.html>

<https://www.oracle.com/technical-resources/articles/java/javareflection.html>

<https://www.youtube.com/watch?v=FIACGmzibAM>

<https://www.javatpoint.com/java-jdbc>

<https://openjfx.io/>

<https://app.diagrams.net/?src=about#>