



Documentation

Assignment no. 3

Student name: Baciú Maria-Simina

Group: 30424

Contents

1. Objectives
2. Problem analysis, structure, scenarios, uses
3. Implementation
4. Results
5. Conclusions
6. Bibliography

1. Objective

Consider an application OrderManagement 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:

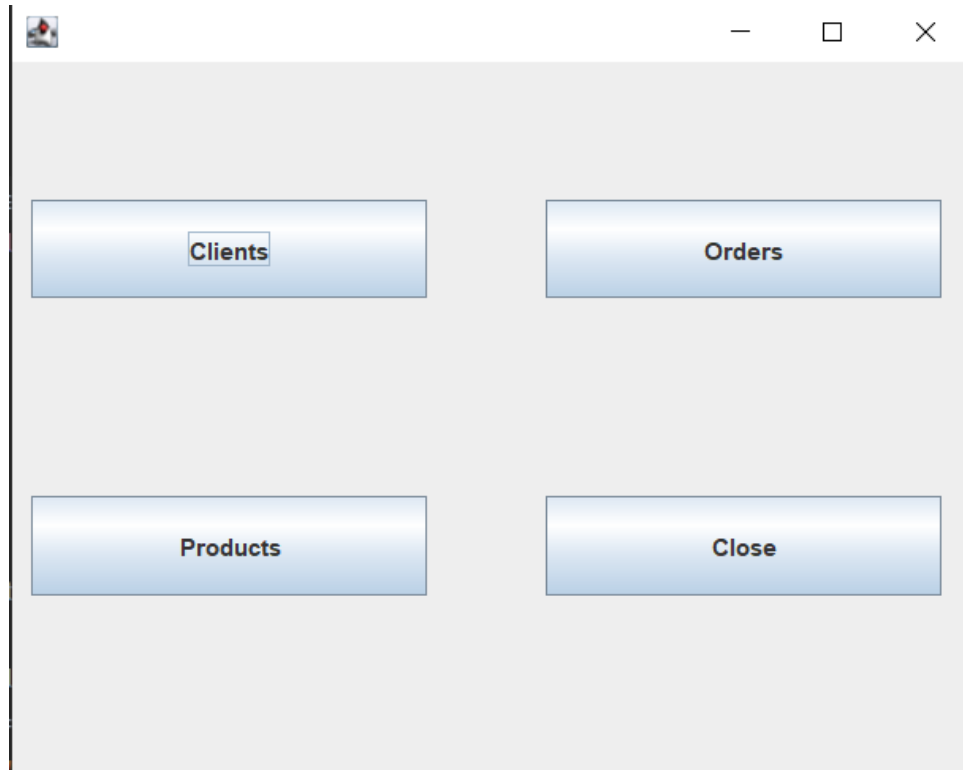
- Model classes - represent the data models of the application (for example Order, Client, Product)
- Business Logic classes - contain the application logic
- Presentation classes – classes that contain the graphical user interface
- Data access classes - classes that contain the access to the database

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

- a. Analyze the application domain, determine the structure and behavior of its classes and draw an extended UML class diagram.
- b. Implement the application classes. Use javadoc for documenting classes.
- c. 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.

2. Problem analysis

This application should be able to fulfil all the requirements in order to display, modify and to keep track of orders, clients and the products. This are stored in a relational MySQL database, along with the information of the users which have access to the system. This way, all the data is easier to retrieve and access from different computers.



The application has multiple so said pages or graphical user interfaces, the main one, in which the user has to choose which object from the project that it wants to manage. So, it has to choose between the clients, orders and products. A close button was also included in the main interface, in order to make the design look better. Depending on the table chosen, different pages will pop-up.

Clients

ID:

First Name:

Last Name:

Email:

ID	firstName	lastName	email
2	ferads	cdsf	sada@fds.csd
4	Voic	Andrei	voicandrei22@gmail.com
25	dfd	fdf	fd@d.v
34	ewf	rewbfb	fsdf@fds.co
43	ewf	rewbfb	fsdf@fds.co
434	dsff	dsfstgfg	fdst@er.c
543	fd	dsf	fds@gfg.c

Insert

Delete

Update

Back

For the clients, the user will have to choose the ID, the first name, last name and the email address. All CRUD operations, create, read, update and delete have been implemented. Moreover, all the data that will be inserted will go through a validation first, so the ID can not coincide with any other ID, first and last name need to contain only letters and the email needs to contain “@” and “.”.

Products

ID:

Name:

Price:

Stock:

id	name	price	stock
1	rgf	43	39
2	df	23	32
3	mere	34	221
4	Banane	10	0
12	portocale	43	10
13	dsdac	32	392
34	fds	235	26

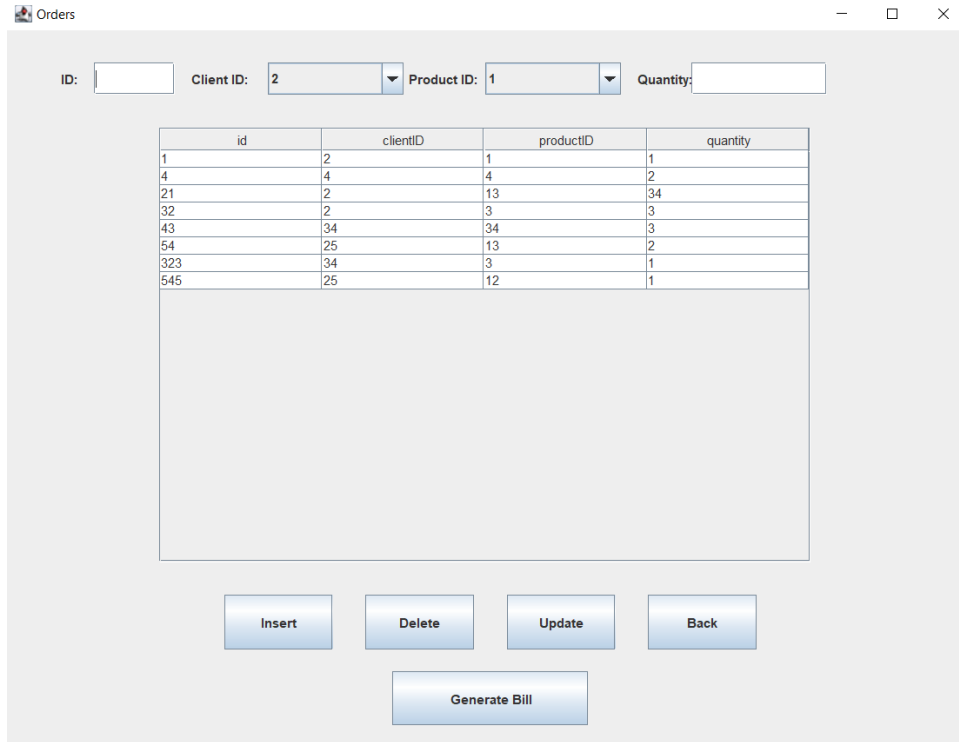
Insert

Delete

Update

Back

As for the products, the user will have to choose an ID, a name, a price and a stock. All CRUD operations, create, read, update and delete have been implemented. Moreover, all the data that will be inserted will go through a validation first, so the ID can not coincide with any other ID, the name contains only letters, the price and the stock need to be over 0.



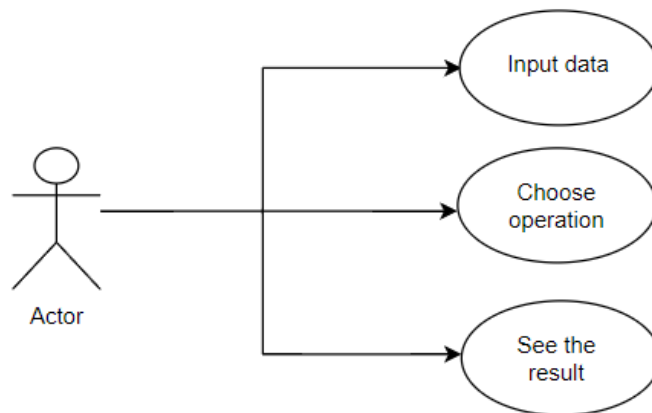
The screenshot shows a window titled "Orders" with a standard Windows-style title bar (minimize, maximize, close buttons). Inside the window, there is a form at the top with four fields: "ID:" (a text input), "Client ID:" (a dropdown menu showing "2"), "Product ID:" (a dropdown menu showing "1"), and "Quantity:" (a text input). Below the form is a table with four columns: "id", "clientID", "productID", and "quantity". The table contains the following data:

id	clientID	productID	quantity
1	2	1	1
4	4	4	2
21	2	13	34
32	2	3	3
43	34	34	3
54	25	13	2
323	34	3	1
545	25	12	1

Below the table is a large empty rectangular area. At the bottom of the window, there are five buttons: "Insert", "Delete", "Update", "Back", and "Generate Bill" (which is centered below the other four).

As for the orders, the user will have to choose an ID, pick a client and product ID from the list and choose the quantity needed. All CRUD operations, create, read, update and delete have been implemented. Moreover, all the data that will be inserted will go through a validation first, so the ID can not coincide with any other ID and the quantity must be over 0. Furthermore, the user can choose to generate a bill, congaing details about the order made.

3. Implementation:

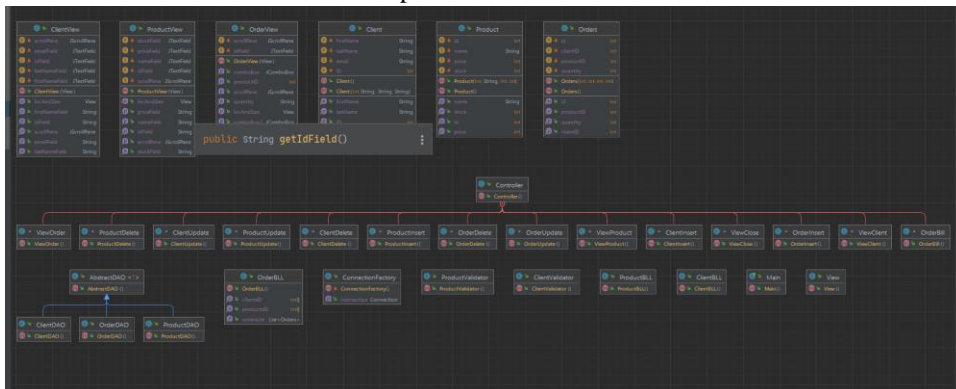


Use case diagram

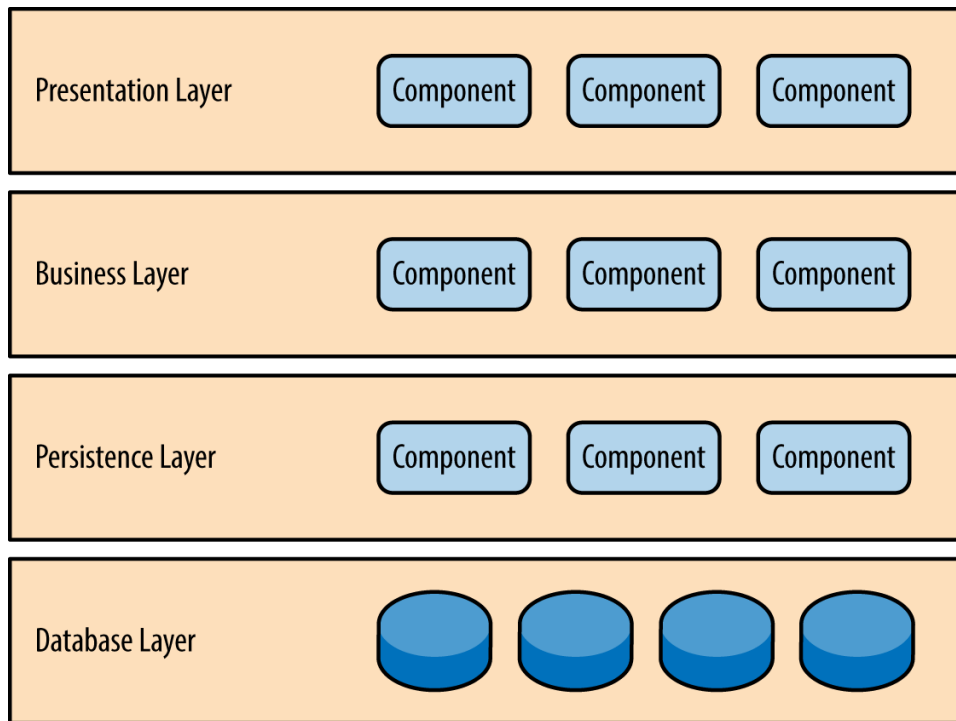
Shows the actor, the user, interacting with the application. They can perform a series of actions in the application, like insert, delete, update.

Class diagram

Shows the classes and the relationship between them.



As for data structures, I have used simple data types such as integer and String, but also more complex ones such as Array Lists in the implementation. Furthermore, to simplify the understanding of the program, an layered architecture was used.



- Presentation layer: UI layer, view layer, presentation tier in multitier architecture UI layer, view layer, presentation tier in multitier architecture
- Application layer: the service layer.
- Business Layer: business logic layer, domain logic layer
- Data Access Layer: persistence layer, logging, networking, and other services which are required to support a particular business layer

As the project is heavily based on the layered pattern, it is parted into 4 distinct parts:

- Model: containing the classes Clients, Products, Orders
- View: containing the class GUI
- DAO: containing the data access layer, the abstract DAO, the Client DAO and the product DAO.
- BLL: containing the client, product and order BLLs, calling the methods from the daos.

The program is split this way based on the divide and conquer method. Divide the problem into smaller ones and solve the simple already known problems.

First the model package:

➤ Client Class

It contains information about the clients of this application, their ID, an integer, first and last name, which are Strings and the email, which is a String as well.

Constructor:

- `public Client(int ID, String firstName, String lastName, String email):` it initializes the clients with the right ID, first name, last name and email

Methods:

`public int getID() ():` it returns the id of the client.

`public double setID():` it sets the ID of the client.

`public void getFirstName(String firstName):` it returns the first name of the client

`public void setFirstName():` it sets the first name of the client.

`public int getLastName(String lastName):` it returns the last name of the client

`public String setLastName():` it sets the last name of the client.

`public int getEmail():` it returns the email of the client

`public String setEmail():` it sets the email of the client.

`public String toString():` it returns the clients as Strings.

➤ Product Class

This class contains information about the products of this application, their ID, an integer, name, which is String and the price and stock, which are integers as well.

Constructor:

`public Product(int productId, String name, int price, int stock):` it initializes the products with the right ID, name price and stock.

Methods:

➤ `public int getID() ():` it returns the id of the products.

`public double setID():` it sets the ID of the products.

`public void getName(int power):` it returns the name of the Products

`public void setName(double coefficient):` it sets the name of the products.

`public int getPrice():` it returns the price of the products

`public String setPrice():` it sets the price of the products

`public int getStock():` it returns the stock of the products

`public String setStock():` it sets the stock of the products.

`public String toString():` it returns the clients as Strings.

➤ Orders Class

This class contains information about the orders of this application, their ID, an integer, quantity, which is an integer, as well as the ID of the product wanted and the ID of the client that orders said product.

`public Orders(int orderID, int clientID, int productID, int quantity):` it initializes the orders with the right ID, quantity, product ID and client ID.

`public int getId():` it returns the id of the order

`public void setId(int id):` it sets the id of the order

And all the other setters and getters for the productID, clientID and the quantity and also a method to String that returns the orders as Strings.

Secondly, the view package:

➤ View

This class is the main frame, all the other views being started after the buttons on the main view were selected. It has four buttons, one for the client view, one for the product view, one for the order view and one that closes the application window. It extends JFrame, and has a series of arguments for setting the layout manager, setting the location and sizes of the buttons, text fields, etc., adding components and adding action events to the buttons

➤ ClientView

Extending JFrame, the GUI class is used to implement the client interface, written in Java Swing.

It has a series of arguments for setting the layout manager, setting the location and sizes of the buttons, text fields, etc., adding components and adding action events to the buttons.

Methods:

```
public void setLayoutManag()
```

```
public void setLocAndSize()
```

```
public void addComponent()
```

```
public void addActionEvent()
```

```
public void actionPerformed(ActionEvent e):
```

 this method sets the action listeners. So, for example, when the user wants to perform an insertion, they push the insert button, and the GUI performs the operation.

➤ ProductView

Extending JFrame, the GUI class is used to implement the product interface, written in Java Swing.

It has a series of arguments for setting the layout manager, setting the location and sizes of the buttons, text fields, etc., adding components and adding action events to the buttons.

Methods:

```
public void setLayoutManag()
```

```
public void setLocAndSize()
```

```
public void addComponent()
```

```
public void addActionEvent()
```

`public void actionPerformed(ActionEvent e):` this method sets the action listeners. So, for example, when the user wants to perform an update to a field, they push the insert button, and the GUI performs the operation.

➤ OrdersView

Extending JFrame, the GUI class is used to implement the order interface, written in Java Swing.

It has a series of arguments for setting the layout manager, setting the location and sizes of the buttons, text fields, etc., adding components and adding action events to the buttons.

Methods:

`public void setLayoutManag()`

`public void setLocAndSize()`

`public void addComponent()`

`public void addActionEvent()`

`public void actionPerformed(ActionEvent e):` this method sets the action listeners. So, for example, when the user wants to generate a bill, they push the generate bill button, and the GUI performs the operation.

Thirdly, for the DAO package:

➤ AbstractDAO

This class was created to define the common operations for accessing a table, so for methods we have:

`private String createSelectQuery(String field):` it creates a query in order to select everything from a table that fulfills the condition.

`public String findAllQuery():` it creates a query in order to select everything from a table

`public List<T> findAll():` it verifies the connection and executes the findAllQuery;

`public T findById(int id):` it verifies the connection and executes the findById;

`public JTable createTable():` it creates a JTable

There are methods for the update, delete, insert queries, as well as functions to verify the connection and execute the query.

➤ ClientDAO, ProductDAO, OrdersDAO

All have the same functions, the create table, update, delete and insert functions. All this functions come from the abstract dao, but they are specifically implemented for said classes.

Fourthly, for the BLL package:

It verifies if the information given by the user is correct and then executes the functions in the daos.

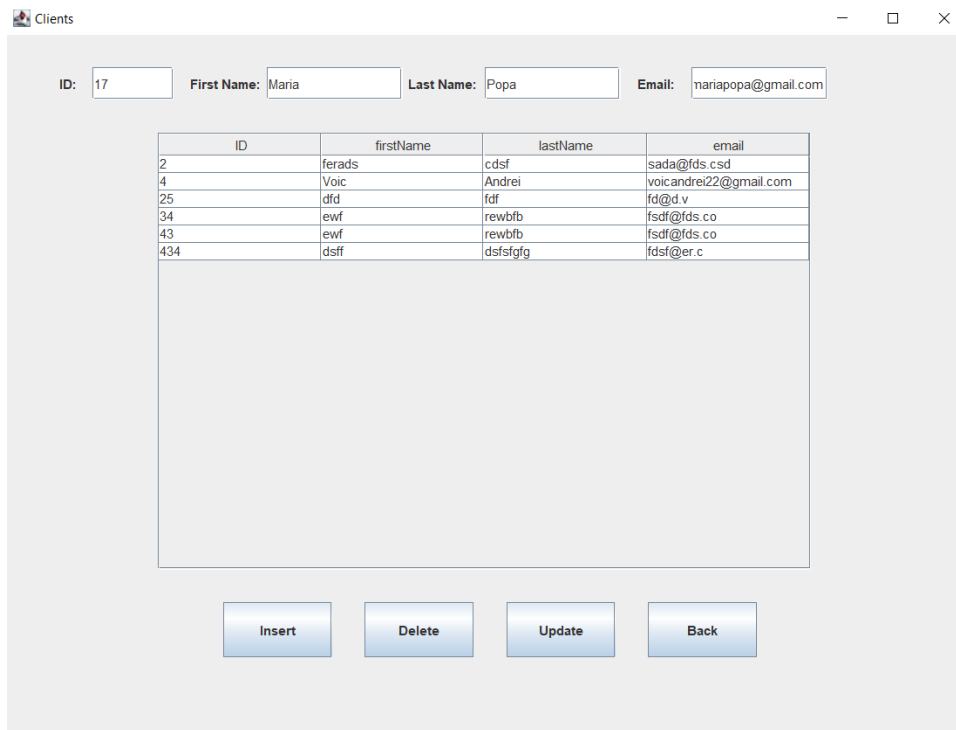
➤ ClientBLL, ProductBLL, OrdersBLL

All have the same functions, calling the methods in the daos for creating the tables, updating the object chosen, deleting the object chosen and inserting the object chosen after their validation.

There is also the Controller part, in which the action listeners for the views were implemented.

4. Results

As for the results, one can see in the photos bellow that the client with ID 17 first name Maria last name Popa and email mariapopa@gmail.com was inserted in the table.



ClientBLL, ProductBLL, OrdersBLL

All have the same functions, calling the methods in the daos for creating the tables, updating the object chosen, deleting the object chosen and inserting the object chosen after their validation.

There is also the Controller part, in which the action listeners for the views were implemented.

4. Results

As for the results, one can see in the photos bellow that the client with ID 17 first name Maria last name Popa and email mariapopa@gmail.com was inserted in the table.

ID	firstName	lastName	email
2	ferads	cdsf	sada@fds.csd
4	Voic	Andrei	voicandrei22@gmail.com
25	dfd	fdf	fd@d.v
34	ewf	rewbfb	fsdf@fds.co
43	ewf	rewbfb	fsdf@fds.co
434	dsff	dsfstfg	fsdf@er.c

Insert Delete Update Back

Clients

ID: 17 First Name: Maria Last Name: Popa Email: nariapopa@gmail.com

ID	firstName	lastName	email
2	ferads	cdsf	sada@fds.csd
4	Voic	Andrei	voicandrei22@gmail.com
17	Maria	Popa	mariaPopa@gmail.com
25	dld	fdf	fd@d.v
34	ewf	rewbfb	fsdf@fds.co
43	ewf	rewbfb	fsdf@fds.co
434	dsff	dsfsgfg	fsdf@er.c

Insert Delete Update Back

5. Conclusion

This project took a lot of time and patience to write and finish. I started writing the code for it the moment I received it, but I could not manage to do it in time. For starters, the database connection would not work. I kept trying to implement the ConnectionFactory code that was given to us, but it would not work for me. In the end, through some miracles (I did not do anything, it just connected after 100 tries) it worked. After that, I spent a ton of time trying to understand the AbstractDAO code that was uploaded to git. It took so long to manage to understand how it works that the deadline passed. I realized after some time, that modelling the problem in the right way from the beginning can make it a lot easier to implement all the requirements. It was a lot harder to finish this project as in the beginning I did not do enough research on what a layered architecture means, what daos and blls are, so I was implementing things that I did not understand. That meant that I was doing everything badly, but after doing my research and asking all the questions I had, I managed to understand the problem and to implement it. From this project I learned multiple things, from managing my time better to how to do your research before the deadline.

5. Bibliography

- <https://stackoverflow.com/>
- https://gitlab.com/utcn_dsrl/pt-layered-architecture/-/tree/master/src/main/java
- https://gitlab.com/utcn_dsrl/pt-reflection-example/-/tree/master/src/main/java

