Documentation

Assignment 3
ORDERS MANAGEMENT

Student name: Bojan Darian-Adelin

Group: 30423/1

CONTENTS

Contents 1
Assignment Objective 2
Problem Analysis, Modeling, Scenarios, Use Cases 3
Design of the application 5
Implementation 8
..... 12
Results 14
Conclusion 14
Bibliography 15

``

ASSIGNMENT OBJECTIVE

The objective of the assignment is to design and implement a management application which uses a database to create basic CRUD operations for the tables: Order, Client, Product. The application should be designed according to the layered architecture pattern and use reflection techniques.

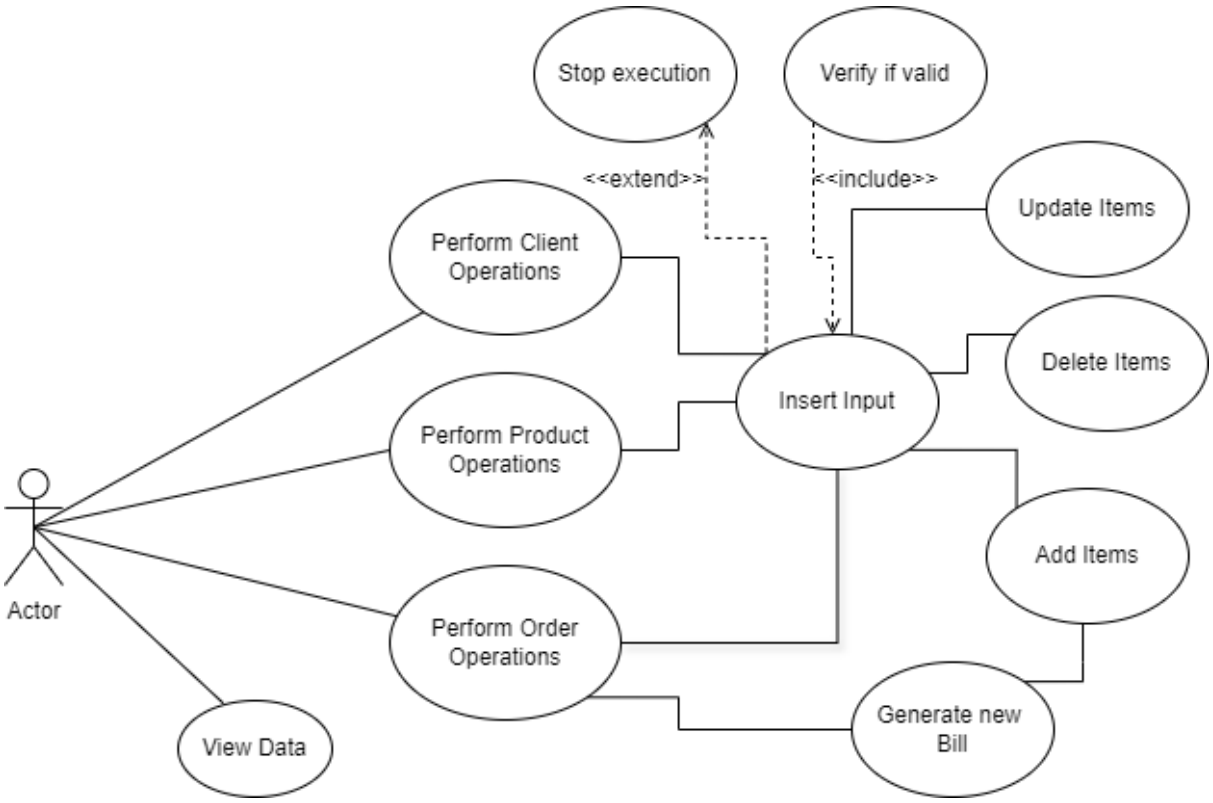
To achieve this task, it is necessary to satisfy the subsequent sub-objectives:

- Use object-oriented programming style (encapsulation, appropriate classes etc.).
- Implement a graphical user interface in Java Swing.
- Implement two JTable windows for Client and Product operations with the following functionalities: add new client, edit client, delete client, view all clients in a table.
- Implement a window for orders, each order will add a Bill object to a Log table.
- Usage of reflection techniques for generating the header of the tables and creating a generic class that contains the methods for accessing the DB (all tables except Log).
- Good organization of the code.
- Use JavaDoc.
- Use relational databases for storing the data.
- Java naming conventions.

PROBLEM ANALYSIS, MODELING, SCENARIOS, USE CASES

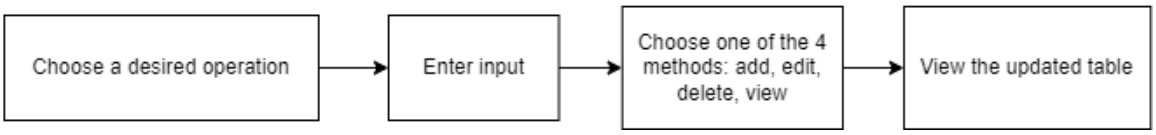
In order to solve this task, it is important to take into consideration the class model and the use cases such that we can have a clear view of the future development.

By starting with the use case diagram, we can gain a clearer understanding of the application we are developing. The user has the following interactions with the graphical user interface: choosing what type of operation the application will perform (Client, Product or Order operations). Then, the user may insert the inputs. Then, the result will be stored in a database and the user can see the results inside the application.



Firstly, the user chooses one of the three types of operations to perform. They will select add, edit, delete, or view all elements from table. In the case of adding or editing, they will also have to enter a new entity.

The following **list** outlines the execution steps of the application:



DESIGN OF THE APPLICATION

Classes, interfaces, and their responsibility:

- MainView - initializes the application's graphical interface for a simulation and sets up an exit mechanism when the interface is closed.
- ProductOrders – creates a GUI where the user can insert desired inputs to perform order operations.
- Controller – controls and links the actions from Presentation package with the functionalities from BusinessLogic package.
- Client – a simple class used to create a client object, similar in logic to a record.
- Product – a simple class used to create a product object, similar in logic to a record.
- Bill – a record which creates a bill object that will be stored in a Log table.
- Orderr – a class like client and product, used to create an object of type order which will also be stored in a database table.
- ObjectModel – an interface used more as a helper for parsing arguments in methods of type Bill, Client, Product, or Orderr
- AbstractDAO - A DAO class providing common functionality for accessing and manipulating entities in the database.
- ClientDAO – a DAO class which focuses on CRUD operations for Client table from the database.
- ProductDAO – a DAO class which focuses on CRUD operations for Product table from the database.
- LogDAO – a DAO class which focuses on CRUD operations for Log table from the database.
- OrderDAO - a DAO class which focuses on CRUD operations for Orderr table from the database.
- ConnectionFactory – a class responsible for making the connection with the database.

- BaseBLL - BaseBLL is a class which is parent to all the classes from the BusinessLogic package: ClientBLL, LogBLL, OrderBLL,ProductBLL
- ClientBLL – a class which uses the functions from the ClientDAO
- ProductBLL – a class which uses the functions from the ProductDAO
- OrderBLL – a class which uses the functions from the OrderDAO
- LogBLL – a class which uses the functions from the LogDAO

There are five packages in the application: BusinessLogic, Presentation, DataAccess, BusinessLogic and Connection. Each one of them includes classes which have a common objective.

BusinessLogic contains 5 classes: ClientBLL, OrderBLL, ProductBLL, LogBLL, and BaseBLL. These classes are responsible for implementing the logic of adding, removing, updating the database tables.

The connection package includes ConnectionFactory class and a file called “db.properties”. The latter is the file where the user can insert his server details. Once all configurations are in place, DAO classes invoke a ConnectionFactory object before proceeding with the SQL implementation.

DataAccess package contains the classes: AbstractDAO, ClientDAO, ProductDAO, LogDAO, OrderDAO. Here the classes provide frameworks for performing CRUD (Create, Read, Update, Delete) operations on entities in a database.

The model package includes simple classes which encapsulate the data and behavior related to a client, product, bill, or order entities, providing methods to access its attributes.

Finally, the presentation package includes the classes meant for visual interaction with the user. The user can insert inputs and access information from the database according to their desire.

Interactions

- Controller class interacts with Business Logic classes and other Presentation classes by starting processing, updating UI elements, and calling the methods for updating the database.
- BusinessLogic classes interact with DataAcess methods and classes from Model package. For each method, a Model type object is created and then parsed as argument to a method from the DataAccess package.
- ConenctionFactory is mainly used in the class AbstractDAO which uses reflection methods to update the database tables.
- MainView interacts with Controller and ProductOrders. MainView represents the starting frame.

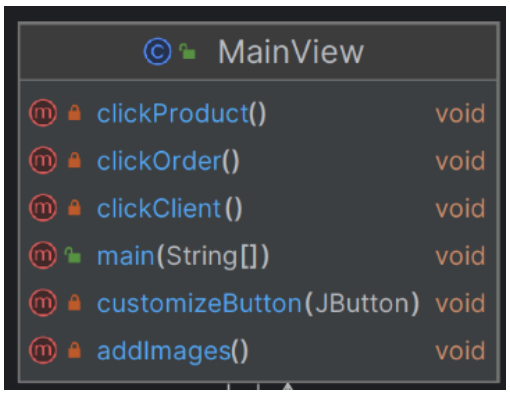
Important data structures used

ArrayList<Object[]> objectsInTable - The data structure used to store the objects retrieved from the database.

DefaultTableModel tableModel - This data structure is used to represent the data in a tabular format, suitable for display in a JTable.

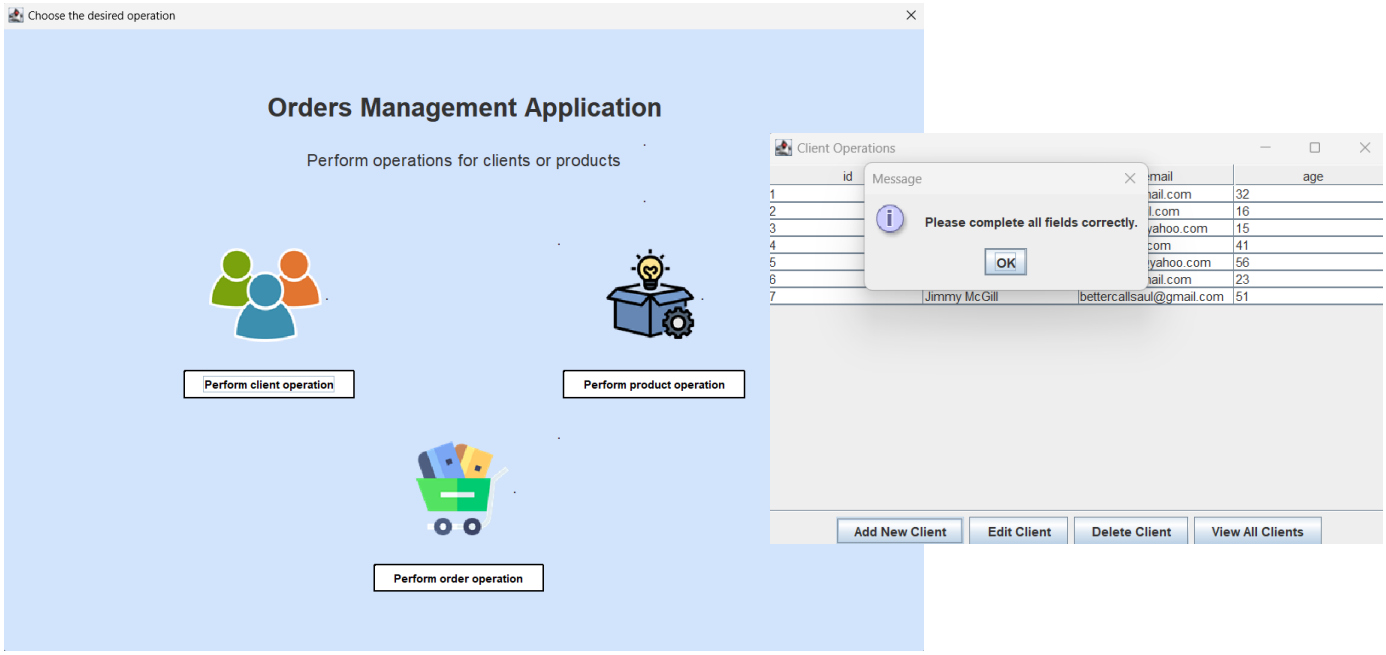
JTable elementsInTable – Used to add entities retrieved from the database into a JTable.

IMPLEMENTATION

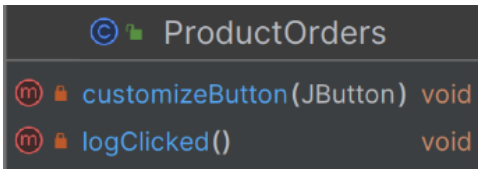


MainView represents the main graphical user interface of the application. Its objective is to construct and display a dialog window for the user interface. MainView serves as the intermediary for user interactions with the application's interface, preparing the inputs for the next phase. The interface includes 3 buttons: performOrderButton, performClientButton, performProductButton.

The UI of the project is simple and effectively allows users to enter desired inputs. If the values are not of the desired type, an error message will be displayed:



The class ProductOrders represents the class for creating a new order. After the inputs are tested, a new bill object is created, and information is updated in database tables. This class is only a GUI and does not provide any special algorithms.



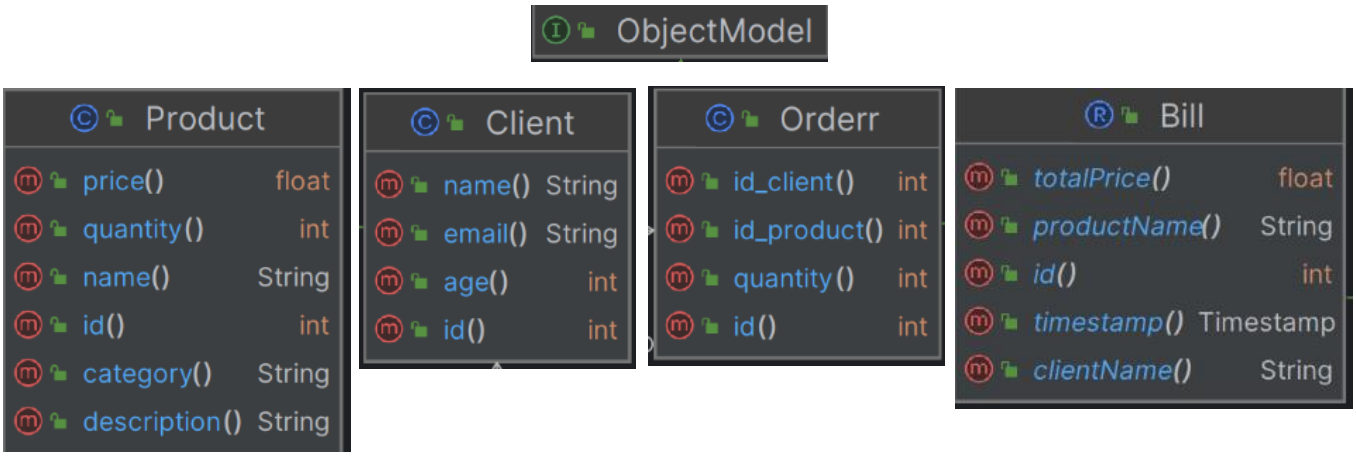
The most important class, Controller, acts as a bridge between the presentation layer and the business logic layer, orchestrating user interactions and managing data flow within the application. Its main objective is communication with the other classes, ensuring error handling and creating JTables after information is retrieved from the database. One vital method is the showClientOperations which creates a JTable when a button from MainView is pressed.

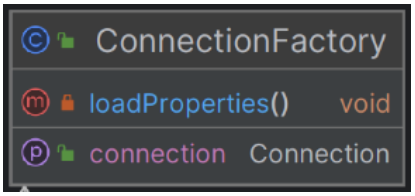
```
private void showClientOperations() throws SQLException {
    ClientBLL clientBLL = new ClientBLL(this);

    JFrame clientFrame = new JFrame("Client Operations");
    clientFrame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
    clientFrame.setSize(600, 400);
    clientFrame.setLayout(new BorderLayout());

    JScrollPane clientScrollPane = getScrollPane();
    clientFrame.add(clientScrollPane, BorderLayout.CENTER);
```

```
JPanel clientButtonPanel = getButtons("Client");

addButton.addActionListener(e -> addClient(clientBLL));
editButton.addActionListener(e->{
    int idClicked = elementsTable.getSelectedRow();
    editClient(idClicked, clientBLL);
});
deleteButton.addActionListener(e->{
    int idClicked = elementsTable.getSelectedRow();
    deleteClient(idClicked, clientBLL);
});
viewButton.addActionListener(e -> clientBLL.viewClients());
clientFrame.add(clientButtonPanel, BorderLayout.SOUTH);
clientFrame.setVisible(true);

clientFrame.addWindowListener(new WindowAdapter() {
    @Override
    public void windowClosed(WindowEvent e) {
        super.windowClosed(e);
        new MainView();}  });}
```

It creates a JTable and adds 4 buttons for performing certain CRUD operations.

The Model classes are used to create simple objects in which certain information is stored temporarily. They all implement the ObjectModel interface.

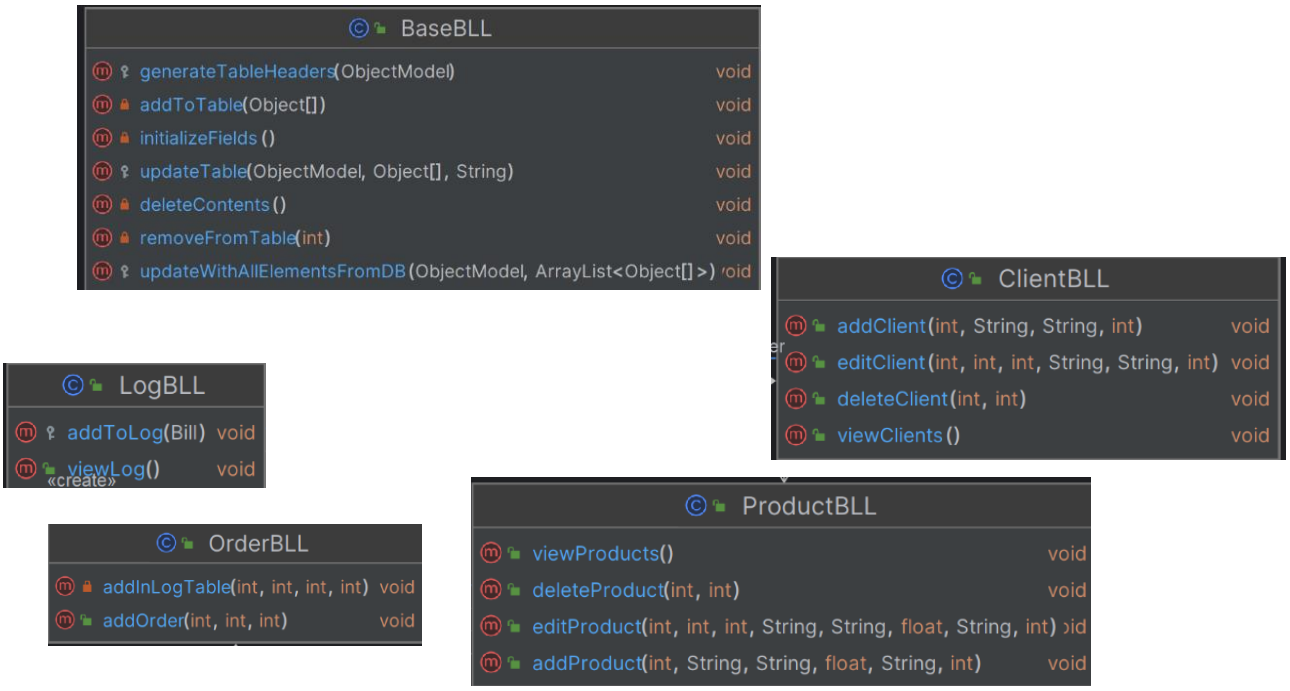




ConnectionFactory is the class which creates the connection with the PostgreSQL server. It has two methods: loadProperties() and getConnection(). getConnection takes the parameters from the db.properties.

```
private void loadProperties() {
        properties = new Properties();
        try (InputStream input = new
FileInputStream("src/main/java/Connection/db.properties")) {
            properties.load(input);
        } catch (IOException e) {
            JOptionPane.showMessageDialog(null, "Error trying to connect to the
database");
        }
    }
```

The classes from the BusinessLogic package implement the methods from the AccessData package. Their main role is to update the elements from the tableModel and access the DAO classes.

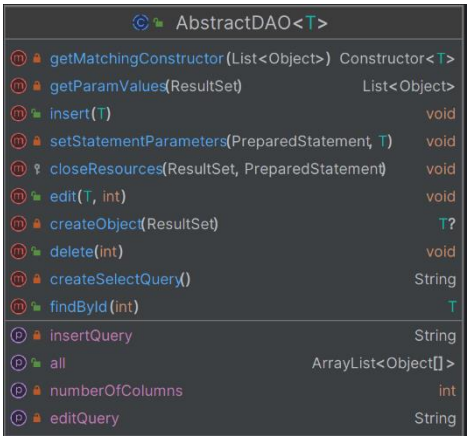


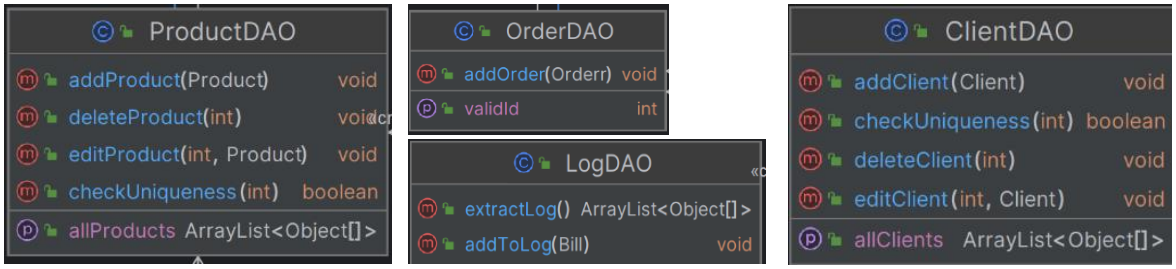
The updateTable method from BaseBLL is called from every child class from this package and is used to update the JTable created in Controller.

```
    /**
     * Updates the tableModel and objectsInTable according to parameters
     * @param objectModel Either Bill, Client or Product
     * @param item objectModel in Object[] form
     * @param chooser A string used to access add, edit or delete
     * @throws NoSuchFieldException  In case it does not find a field => returns an
error
     * @throws IllegalAccessException  In case it is impossible to access => returns an
error
     */
    protected void updateTable(ObjectModel objectModel, Object[] item, String chooser)
throws NoSuchFieldException, IllegalAccessException {
        initializeFields();

        Object[] newItem = new Object[item.length - 1];
```

```
        System.arraycopy(item, 0, newItem, 0, item.length - 1);

        generateTableHeaders(objectModel); //if empty, create header

        switch (chooser) {
            case "Add" -> addToTable(newItem);
            case "Edit" -> {
                int firstId = (int) item[item.length - 1];
                addToTable(newItem);
                removeFromTable(firstId);
            }
            case "Delete" -> {
                int newId = (int) item[item.length - 1];
                removeFromTable(newId);
            }
        }
    }
```

Finally, the DataAccess classes are the ones who update the tables from the database, using an object of type ConnectionFactory. The methods are used by the classes from the BusinessLogic package. They use a parent class called AbstractDAO which has certain functions that are implemented using reflection techniques. They are called from the child classes.





RESULTS

The final results are shown in the JTables. For each order the Log table is updated.

Log Operations

id	clientName	productName	totalPrice	timestamp
1	Cosmin Tianu	Washing machine Whirlpool	1699.0	2024-05-12 14:54:37.22.
2	Jimmy McGill	Laptop Gaming	17998.0	2024-05-15 12:54:26.26.
3	OceanMan	Shoes Killian Mbappe	7998.12	2024-05-15 12:54:33.62.

CONCLUSION

In conclusion, the orders management application project provides a user-friendly tool for updating clients, products, and orders in the database. Through encapsulation and modular design, the project ensures clean code organization and ease of maintenance.

This project helped in understanding the concepts of reflection by creating general classes.

BIBLIOGRAPHY

Lecture materials and other resources: https://dsrl.eu/courses/pt/

Layered architecture: https://dzone.com/articles/layers-standard-enterprise

Understanding Reflection:
https://jenkov.com/tutorials/java-reflection/index.html