UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

PROGRAMMING TECHNIQUES
ASSIGNMENT 3

ORDERS MANAGEMENT

STUDENT: GYARMATHY CSABA
UNIVERSITY: TECHNICAL UNIVERSITY OF CLUJ-NAPOCA
FACULTY: COMPUTER SCIENCE
TEACHER: CRISTINA BIANCA POP
TEACHER ASSISTANT: ANDREEA VALERIA VESA
YEAR: 2
ACADEMIC YEAR: 2021/2022
GROUP: 30424

Contents

Assignment Objective

Consider an application Orders Management for processing client orders for a warehouse. Relational databases should be used to store the products, the clients, and the orders. The application should be designed according to the layered architecture pattern and should use (minimally) the following classes:

- Model classes - represent the data models of the application
- Business Logic classes - contain the application logic
- Presentation classes – GUI related classes
- Data access classes - classes that contain the access to the database

Project analysis, scenario, approach and use cases

Analysis

This project consists of creating a very application that is self-explanatory and easy to understand. Behind all the code lies database that holds all the user information.

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

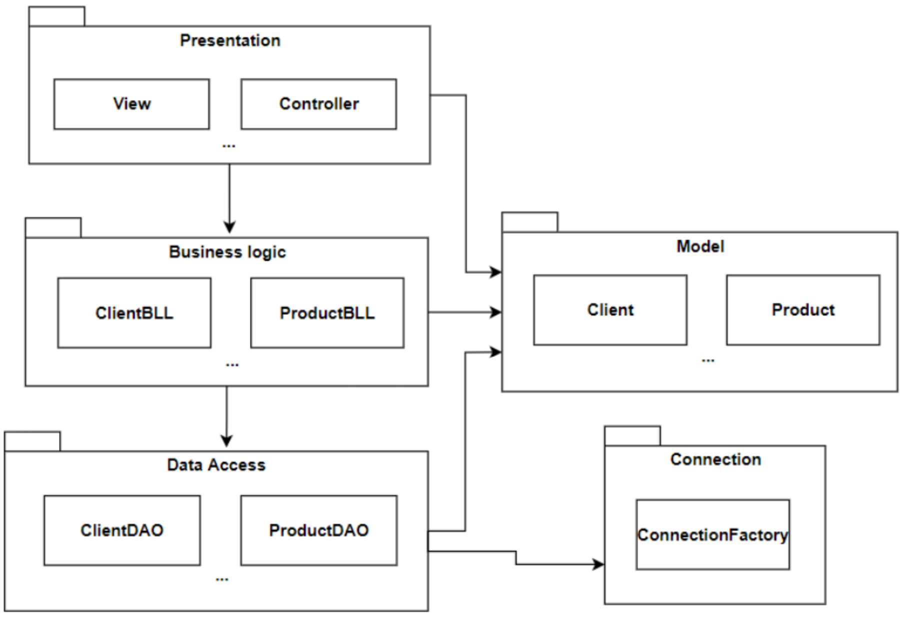
What is CRUD? CRUD is an acronym that comes from the world of computer programming and refers to the four functions that are considered necessary to implement a persistent storage application: create, read, update and delete. Persistent storage refers to any data storage device that retains power after the device is powered off, such as a hard disk or a solid-state drive. In contrast, random access memory and internal caching are two examples of volatile memory - they contain data that will be erased when they lose power.

Several SQL servers could have been used to create a connection to the database. Instead of choosing to connect to a database server, I used SQLite. SQLite is an embedded SQL database engine. Unlike most other SQL databases, SQLite does not have a separate server process. SQLite reads and writes directly to ordinary disk files. A complete SQL database with multiple tables, indices, triggers, and views, is contained in a single disk file. The database file format is cross-platform - you can freely copy a database between 32-bit and 64-bit systems or between big-endian and little-endian architectures. These features make SQLite a popular choice as an Application File Format. SQLite database files are a recommended storage format by the US Library of Congress. Think of SQLite not as a replacement for Oracle but as a replacement for fopen().

Scenario and modelling

The user is required to choose which part of the database they wish to operate in. Our database contains 3 tables: one for clients, one for products and one for orders. Upon opening the application, the user can choose one of these branches and perform CRUD operations in it. the very basic graphical user interface (GUI) guides the user through this process.

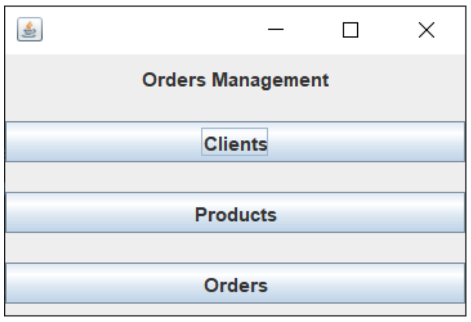
Unlike previous projects, this project is not based on a Model-View-Controller (MVC) architecture and was instead built around a Layered Architecture.



The layered architecture style is one of the most common architectural styles. The idea behind Layered Architecture is that modules or components with similar functionalities are organized into horizontal layers. As a result, each layer performs a specific role within the application.

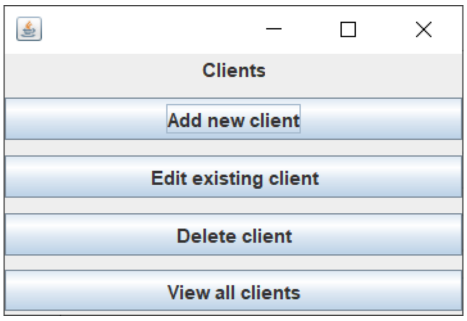
The layered architecture style does not have a restriction on the number of layers that the application can have, as the purpose is to have layers that promote the concept of separation of concerns. The layered architecture style abstracts the view of the system, while providing enough detail to understand the roles and responsibilities of individual layers and the relationship between them.

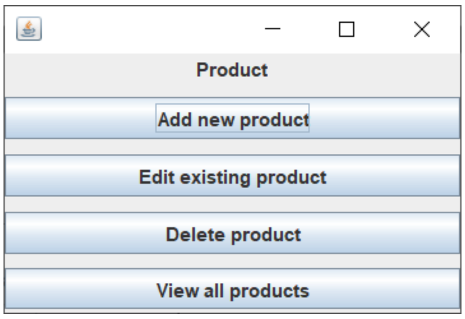
Upon application start, the user is greeted with the following window from which they can choose where to continue:



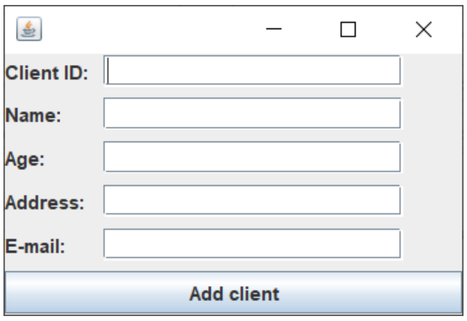
Each button will open a new window, therefore in order to go back to the main menu, the user must close all open windows except this one.

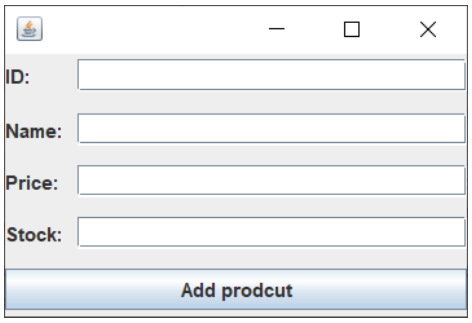
The Clients and Products branches are very similar to each other. The user can perform the CRUD operations for either clients or products.





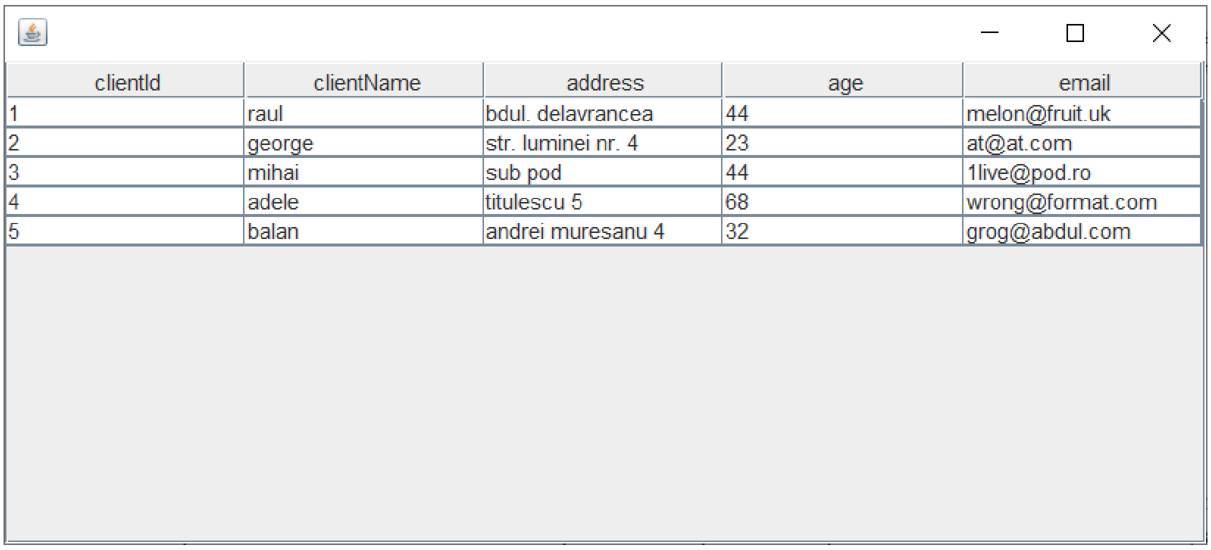
The only difference is that when adding a new entry, for clients, beside the ID and the name, the user must specify the home address, email and age, while for products the price and stock must be specified beside the ID and the name, as shown in the following screenshots.





When editing either a client, product or order, the application first checks if there exists an entity with the given ID. If not, then a corresponding error will be displayed. Similarly, when deleting an entity an ID must be specified.

If the user wishes to display all rows of data in the database table corresponding to either clients, products or orders, using reflection techniques, a JTable will be created. Here the user can shift the columns and rows as they desire.



clientId	clientName	address	age	email
1	raul	bdul. delavrancea	44	melon@fruit.uk
2	george	str. luminei nr. 4	23	at@at.com
3	mihai	sub pod	44	1live@pod.ro
4	adele	titulescu 5	68	wrong@format.com
5	balan	andrei muresanu 4	32	grog@abdul.com

The JTable class is a part of Java Swing Package and is generally used to display or edit two-dimensional data that is having both rows and columns. It is similar to a spreadsheet. This arranges data in a tabular form.

Use case scenarios

A use case is a methodology used in system analysis to identify, clarify, and organize system requirements. The use case is made up of a set of possible sequences of interactions between systems and users in a particular environment and related to a particular goal. The use cases are strongly connected with the user steps.

As mentioned before, the user must choose which branch of the application they wish to access and then select which operation they wish to perform: add, delete, edit or view. In order to do any of these, except you, the input is checked is it corresponds according to the logic behind the operation.

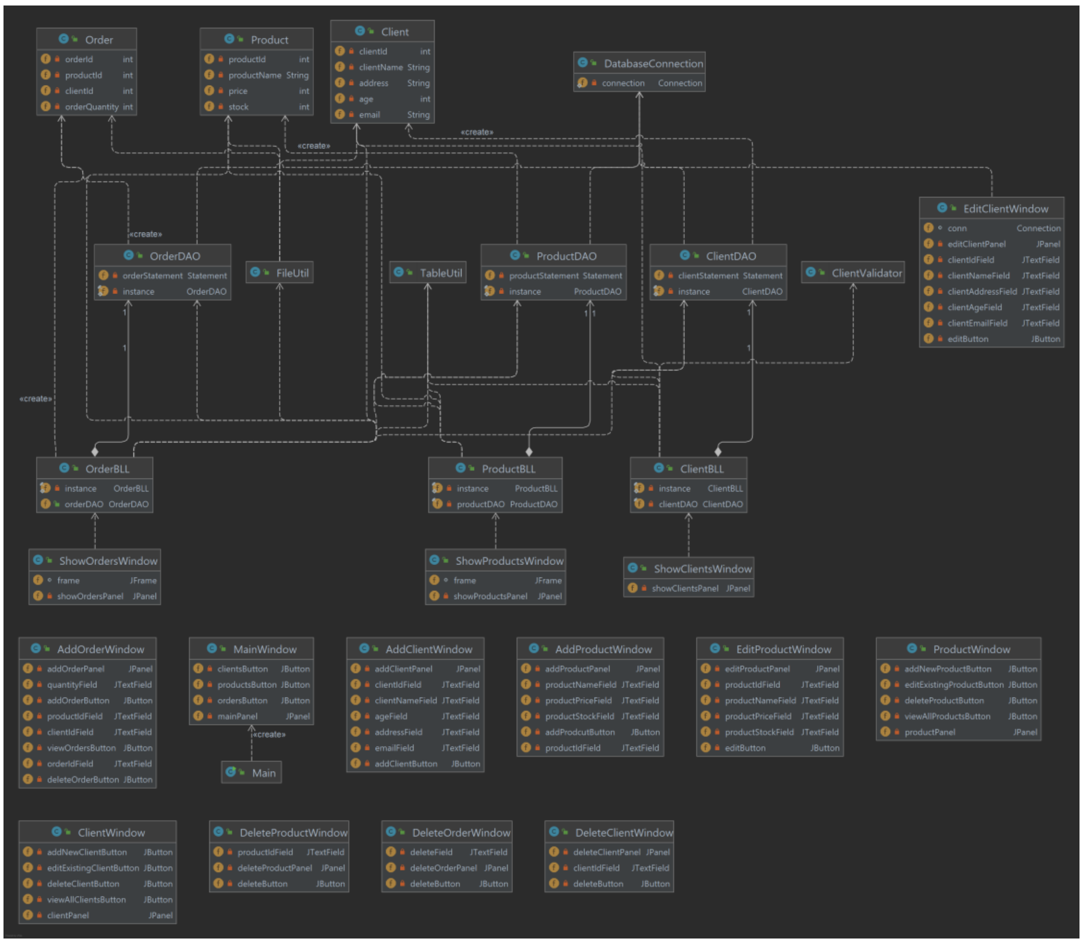
Implementation

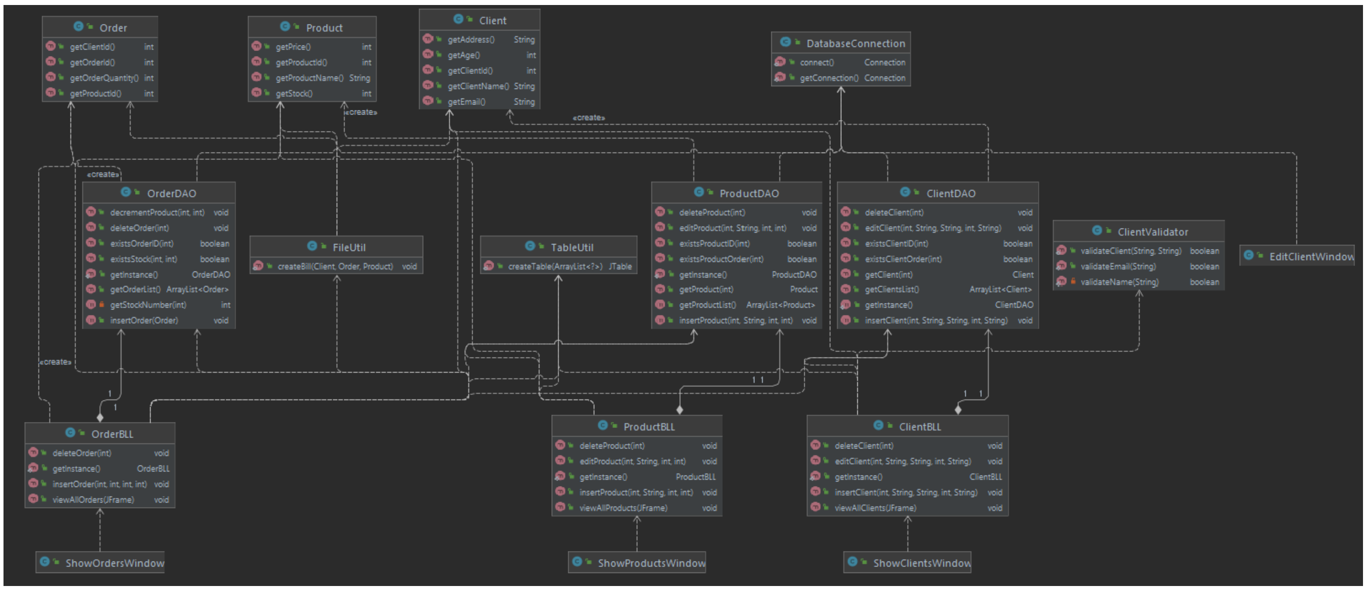
Diagrams

Class diagram

In the course of creating the project, several diagrams were made to help with the development.

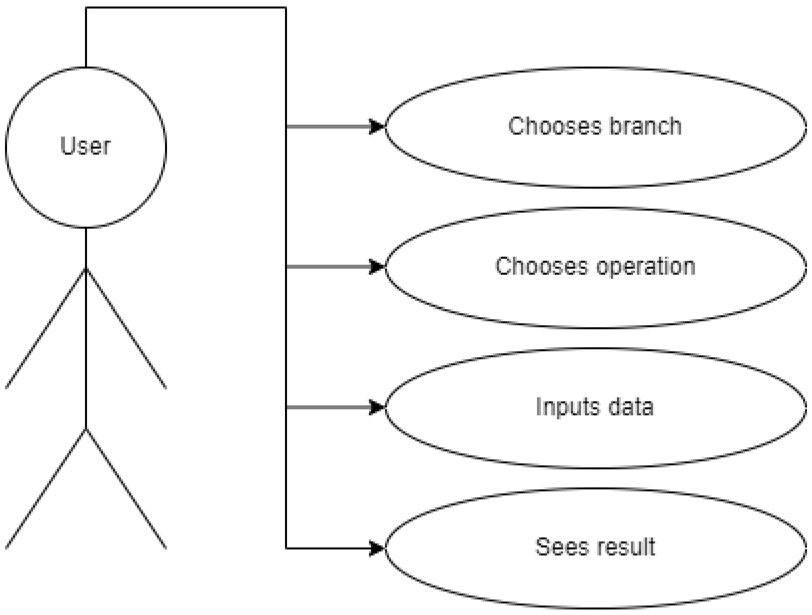
Below you will see a UML (Unified Modelling Language) diagram, used to visualize, specify, and document the software system. This diagram contains all the Java classes used in the implementation of this project, as well as their dependencies and relations with each other. This diagram is known as the class diagram.

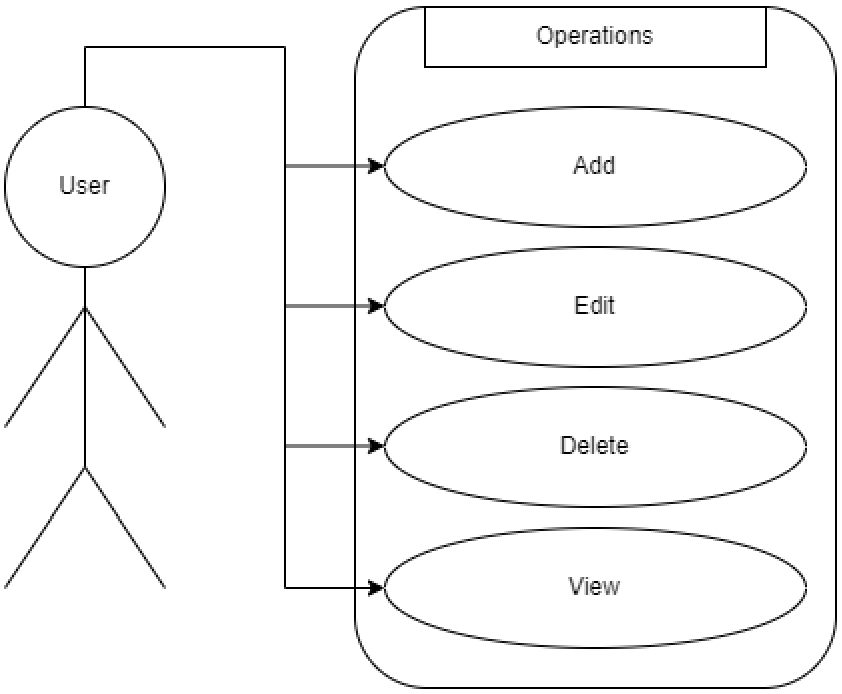



Order
getClientId() int
getOrderId() int
getOrderQuantity() int
getProductId() int
Product
getPrice() int
getProductId() int
getProductName() String
getStock() int
Client
getAddress() String
getAge() int
getClientId() int
getClientName() String
getEmail() String
DatabaseConnection
connect() Connection
getConnection() Connection
«create»
OrderDAO
decrementProduct(int, int) void
deleteOrder(int) void
existsOrderID(int) boolean
existsStock(int, int) boolean
getInstance() OrderDAO
getOrderList() ArrayList<Order>
getStockNumber(int) int
insertOrder(Order) void
FileUtil
createBill(Client, Order, Product) void
TableUtil
createTable(ArrayList<?>) JTable
ProductDAO
deleteProduct(int) void
editProduct(int, String, int, int) void
existsProductID(int) boolean
existsProductOrder(int) boolean
getInstance() ProductDAO
getProduct(int) Product
getProductList() ArrayList<Product>
insertProduct(int, String, int, int) void
ClientDAO
deleteClient(int) void
editClient(int, String, String, int, String) void
existsClientID(int) boolean
existsClientOrder(int) boolean
getClient(int) Client
getClientsList() ArrayList<Client>
getInstance() ClientDAO
insertClient(int, String, String, int, String) void
ClientValidator
validateClient(String, String) boolean
validateEmail(String) boolean
validateName(String) boolean
EditClientWindow
OrderBLL
deleteOrder(int) void
getInstance() OrderBLL
insertOrder(int, int, int, int) void
viewAllOrders(JFrame) void
ProductBLL
deleteProduct(int) void
editProduct(int, String, int, int) void
getInstance() ProductBLL
insertProduct(int, String, int, int) void
viewAllProducts(JFrame) void
ClientBLL
deleteClient(int) void
editClient(int, String, String, int, String) void
getInstance() ClientBLL
insertClient(int, String, String, int, String) void
viewAllClients(JFrame) void
ShowOrdersWindow
ShowProductsWindow
ShowClientsWindow

Use case diagram

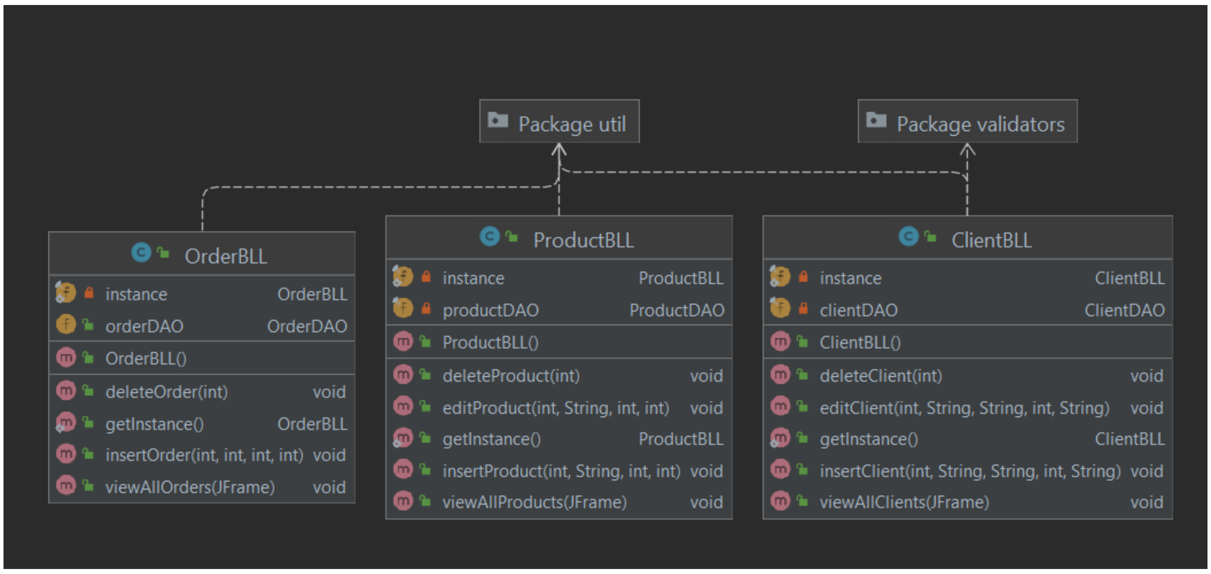
The following diagrams are the use case diagrams, depicting the different cases a user finds itself in.





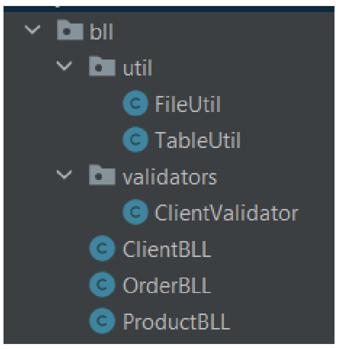
Packages

Business Logic Layer (BLL)



This package contains the main logic of the application and connects the Data Access Objects (DAOs) with the GUI. Each model class has its own BLL and performs the same operations as in its DAO but with access to the user interface through creating an instance of the DAO in the BLL and calling its CRUD methods.

The business logic layer is the business components that provide Open Applications Group Integration Specification (OAGIS) services to return data or start business processes. The presentation layer uses these OAGIS services to display data, or to invoke a business process. The business logic provides data required by the presentation layer. The business logic layer exists because more than just fetching and updating data is required by an application; there is also additional business logic independent of the presentation layer.



The two inner packages, Util and Validators, contain helper classes.

Util contains a class TableUtil which is the class that creates the JTable when the view operation is selected. It's using reflection technique to receive an ArrayList list of objects and create the table accordingly.

Reflection helps programmers make generic software libraries to display data, process different formats of data, perform serialization or deserialization of data for communication, or do bundling and unbundling of data for containers or bursts of communication. In object-oriented programming languages such as Java, reflection allows inspection of classes, interfaces, fields and methods at runtime without knowing the names of the interfaces, fields, methods at compile time. It also allows instantiation of new objects and invocation of methods.

```
public static JTable createTable(ArrayList<?> myList) {
    int tableSize = myList.get(0).getClass().getDeclaredFields().length;
    String[] columnNames = new String[tableSize];
    int columnIndex = 0;
    for (java.lang.reflect.Field currentField :
myList.get(0).getClass().getDeclaredFields()) {
        currentField.setAccessible(true);
        try {
            columnNames[columnIndex] = currentField.getName();
            columnIndex++;
        } catch (IllegalArgumentException e) {
            e.printStackTrace();
        }
    }
    DefaultTableModel myModel = new DefaultTableModel() {
        @Override
        public boolean isCellEditable(int row, int column) {
            return false;// all cells false
        }
    };
    myModel.setColumnIdentifiers(columnNames);
    for (Object o : myList) {
        Object[] obj = new Object[tableSize];
        int col = 0;
        for (java.lang.reflect.Field currentField :
o.getClass().getDeclaredFields()) {
            currentField.setAccessible(true);
            try {
                obj[col] = currentField.get(o);
                col++;
            } catch (IllegalArgumentException | IllegalAccessException e) {
                e.printStackTrace();
            }
        }
        myModel.addRow(obj);
    }
    return new JTable(myModel);
```

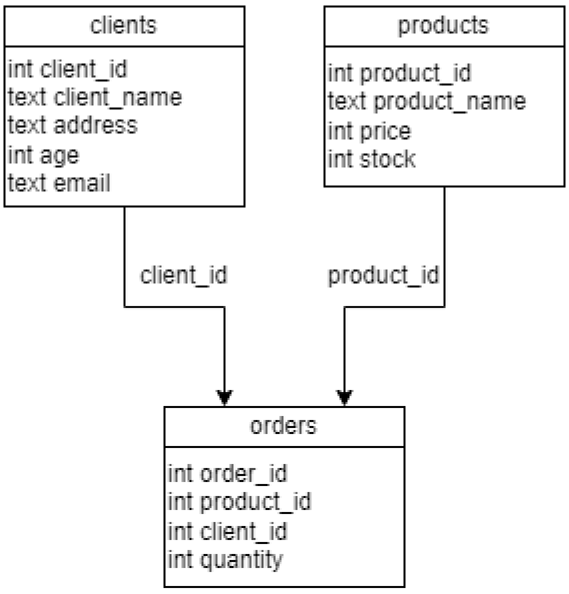
The other class FileUtil is used to create the bills for each placed order and to get a bonus point for the project lolz. It's using a PrintWirter to build a text which is then output in a text file whose name is according to the ID of the order that was successfully placed.

Connection

This package contains a single class that creates the database connection to the local SQLite database. I’ve chosen SQLite mainly because the database is a single .db file and with thew SQLite command prompt it’s very easy to create and edit a database.

```
        public static Connection connect() {
    Connection conn = null;
    try {
        Class.forName("org.sqlite.JDBC");
        conn = DriverManager.getConnection("jdbc:sqlite:schooldb.db");
        System.out.println("Connection to SQLite has been established.");
    } catch (SQLException | ClassNotFoundException e) {
        System.out.println(e.getMessage());
    } finally {
        try {
            if (conn != null) {
                conn.close();
            }
        } catch (SQLException ex) {
            System.out.println(ex.getMessage());
        }
    }
    return conn;
}
```

As mentioned before, our database contains 3 tables with the following attributes (columns):



The relation between the tables is shown by the arrows that are representing the foreign keys of the Orders table, which means that, e.g., a client with ID 6 that does not exists in the Clients table cannot exist in the Orders table either, etc. This relation is good as a security measure to prevent false and incorrect data from being input into the database.

Data Access Object (DAO)

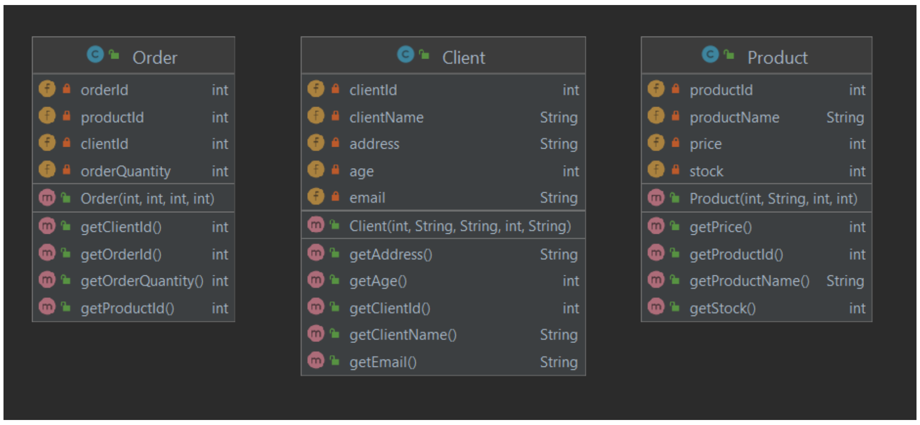
In software, a data access object (DAO) is a pattern that provides an abstract interface to some type of database or other persistence mechanism. By mapping application calls to the persistence layer, the DAO provides some specific data operations without exposing details of the database.

Each model class has its own DAO and corresponding CRUD methods created with SQL queries. All three classes are very similar to each other, only differences being the queries, and, of course, the variables that we work with.

This isolation supports the single responsibility principle. It separates what data access the application needs, in terms of domain-specific objects and data types (the public interface of the DAO), from how these needs can be satisfied with a specific DBMS, database schema, etc. (the implementation of the DAO).

Model

This package has the model classes, each corresponding to a table of the database, having their attributes as variables. These objects were mainly used to create the bill for the orders, where we need to know the credentials of the client through a given ID, the name and price of a product through a given ID, and the quantity of the order to therefore calculate the final price of the order. Each class has a constructor and getters.



Order	
orderId	int
productId	int
clientId	int
orderQuantity	int
Order(int, int, int, int)	
getClientId()	int
getOrderId()	int
getOrderQuantity()	int
getProductId()	int

Client	
clientId	int
clientName	String
address	String
age	int
email	String
Client(int, String, String, int, String)	
getAddress()	String
getAge()	int
getClientId()	int
getClientName()	String
getEmail()	String

Product	
productId	int
productName	String
price	int
stock	int
Product(int, String, int, int)	
getPrice()	int
getProductId()	int
getProductName()	String
getStock()	int

Presentation

Finally, we have the presentation package. This package contains only graphical user interface elements (classes and forms) automatically generated by Java Swing.

Swing is a set of program component s for Java programmes that provide the ability to create graphical user interface (GUI) components, such as buttons and scroll bars, that are independent of the windowing system for specific operating system. Swing components are used with the Java Foundation Classes (JFC).

Consider that each window that appears when the application is being used was made with a new Swing form GUI editor for easy placement of components and ActionListener creation. Because that project is not based on the MVC architecture, I’ve decided that the ActionListener stay in the interface classes, as they should, instead of being in the controller.

Further development

A bonus requirement of the project was to create and abstract/generic DAO which is then used to create the DAOs of the models – this project does not contain such class. Therefore, to lower the risk of query error, it would be better if all DAOs were created by reflecting this abstract class.

Instead of having a different window for each button, it would be nicer if the GUI consisted of one single window that kept updating with each button press, reducing the cluster of the bazillion windows this way. Also, it would be better if instead of Swing, JavaFX was used, as it looks much more professional and high-end.

As a final note, this project could also contain an MVC architecture on top of the layered architecture.

Conclusion

This project was a good exercise in remembering the OOP concepts learned in the first semester, but also learning new ones, which I found very useful and challenging at first. It was a good exercise to learn about reflection and how to connect to a database in Java. The most challenging part was creating the queries with a ResultSet. I was stuck for a week on the project trying to figure out how the ResultSet works and what it return with the right methods, but in the end I managed and understood what they do and how they work.

I arrived at the conclusion that facing problems with your code and trying to make it work by yourself, through the mean of research, has the benefit of learning new concepts and a better use of the known ones

Bibliography

https://dsrl.eu/courses/pt/
https://stackoverflow.com/
https://www.youtube.com/
https://www.geeksforgeeks.org/

Data access object - Wikipedia
Reflective programming - Wikipedia
Business logic layer (hcltechsw.com)
asd.drawio - diagrams.net
Java Swing | JTable - GeeksforGeeks
Layered.pdf (uwaterloo.ca)