Logo, company name

Description automatically generated

Programming Techniques

Laboratory - Assignment 3

ORDERS MANAGEMENT

Student: Onaci Andreea-Maria

Group: 30421

Contents

[1. Objective 3](#_Toc135330707)

[2. Analysis of the problem 3](#_Toc135330708)

[a) Databases in Java 3](#_Toc135330709)

[b) CRUD queries 3](#_Toc135330710)

[c) Scenarios & Cases 3](#_Toc135330711)

[d) Cases & Case Diagram 4](#_Toc135330712)

[3. Design 5](#_Toc135330713)

[OOP Design 5](#_Toc135330714)

[UML Package Diagram 5](#_Toc135330715)

[5](#_Toc135330716)

[Classes Diagram 6](#_Toc135330717)

[Approach of implementation 7](#_Toc135330718)

[4. Implementation 7](#_Toc135330719)

[AbstractDAO 7](#_Toc135330720)

[Client, Product, Order 7](#_Toc135330721)

[NameValidators, PriceValidators, QuantityOrderValidators, QuantityValidators and Validators Interface 8](#_Toc135330722)

[ClientBLL, ProductBLL, OrderBLL 8](#_Toc135330723)

[ClientDAO, ProductDAO, OrderDAO and Fields Interface 8](#_Toc135330724)

[Bill and LogFile 8](#_Toc135330725)

[ControllerMenu, ControllerClient, ControllerProduct, ControllerOrder 9](#_Toc135330726)

[ViewMenu, ViewClient, ViewBill, ViewProduct, ViewOrders 9](#_Toc135330727)

[5. Results 9](#_Toc135330728)

[6. Conclusions 10](#_Toc135330729)

[7. Bibliography 10](#_Toc135330730)

1. Objective

Design and implement an application for orders management in a warehouse.

1. Analysis of the problem
2. Databases in Java

When using PostgreSQL in Java, developers can utilize the JDBC API to establish connections, execute queries, and manage data efficiently. By including the PostgreSQL JDBC driver and providing connection details, a connection object can be obtained. SQL statements can be executed using interfaces such as Statement or PreparedStatement, with data processed through ResultSet. Proper resource handling, including closing connections and statements, is crucial. The combination of Java and PostgreSQL offers a reliable and powerful solution for building database-driven applications.

1. CRUD queries

CRUD queries, an acronym for Create, Read, Update, and Delete, are fundamental operations in database management. They allow users to create new records, retrieve existing data, update or modify records, and delete unwanted data, providing a comprehensive set of functionalities for data manipulation and management.

CRUD queries are essential components in database systems, enabling seamless data management. The Create operation facilitates the insertion of new records into a database, while Read enables the retrieval of specific data based on user-defined criteria. The Update operation allows users to modify existing records, ensuring data accuracy and relevance, while the Delete operation enables the removal of unwanted data, maintaining data integrity. These four operations collectively form the foundation of data manipulation and ensure efficient data handling in various applications and systems.

1. Scenarios & Cases

When running the application, it will display a GUI. The user have 3 buttons representing the 3 databases: clients, products and orders.

The Client GUI provides functionality to add new clients to the database, delete or update a client's name based on their ID. Additionally, it displays a table that represents the existing records in the database, automatically updating the table after each operation is performed, ensuring real-time data visibility and management.

The Product GUI allows users to add new products to the database, delete or update a product's details using its ID. It also presents a table that displays the existing product records, dynamically updating the table after each operation, ensuring accurate and up-to-date information regarding the products stored in the database.

The Order GUI facilitates the creation of orders by allowing users to select an existing product and client, specifying the desired quantity. If the requested quantity exceeds the available stock, an error message is displayed. The GUI includes tables for orders and products, which dynamically update after each order is placed. Additionally, there is a button to access the bills table, providing a comprehensive overview of the billing information associated with the orders.

However, in the case where the user inputs invalid values for any of the setup parameters, the program will throw specific exceptions, informing the user of the invalid input values. The application will then request that the user input valid values before proceeding. At this point, the scenario returns to step 1, where the user must input the required parameters again.

Once the user has successfully validated the input data, the application should work properly, providing the desired result.

d) Cases & Case Diagram

Press the button in the main interface which redirects the user to the interface corresponding to that button.

Enter the input and press the corresponding button

See the database updating accordingly to the request button.

User

The case diagram depicts a user interacting with a main interface. The user can press various buttons on the interface, each corresponding to a specific functionality. Upon pressing a button, the user is redirected to the corresponding interface where they can enter input relevant to their request. After entering the input, the user can press the corresponding button to initiate the requested action. As a result, the database associated with the system updates accordingly, reflecting the changes made based on the user's request.

1. Design

OOP Design

The Orders Management respects the OOP concepts. The application provides the encapsulation and abstractization concepts and the manipulation of input data is hidden in the background, so that the user can see the desired output in a very readable manner. The AbstractDAO class is implementing all the CRUD operations, and the others DAO classes are inherited this class. The implementation contains interfaces, so that many methods are overrided.

UML Package Diagram

This Orders Management Application is divided in 4 large packages, *BusinessLogic* which contains the classes *ClientBLL, OrderBLL* and *ProductBLL* and a subpackage *Validators* with the classes *NameValidators, PriceValidators, QuantityOrderValidators, QuantityValidator* and the interface *Validators*, the package *DataAccess* with the big class *AbstractDAO* and the other classes *ClientDAO, OrderDAO, ProductDAO* and the interface *Fields,* the package *Model* which contains the classes *Client, Ordeer, LogFile, Orders, Product* and the record *Bill* and the package *Presentation* which contains the *ControllerClient, ControllerMain, ControllerOrder, ControllerProduct, ViewBills, ViewClients, ViewMenu, ViewOrders* and *ViewProduct*.

Package Presentation

ControllerClient

ControllerMain

ControllerOrder

ViewBills

ViewClients

ViewMenu

ViewOrders

ViewProduct

Package DataAccess

AbstractDAO

ClientDAO

OrderDAO

ProductDAO

Fields

Package BusinessLogic

Validators

NameValidators

PriceValidators

QuantityOrderValidators

QuantityValidators

Validators

ClientBLL

OrderBLL

ProductBLL

Package Model

Bill

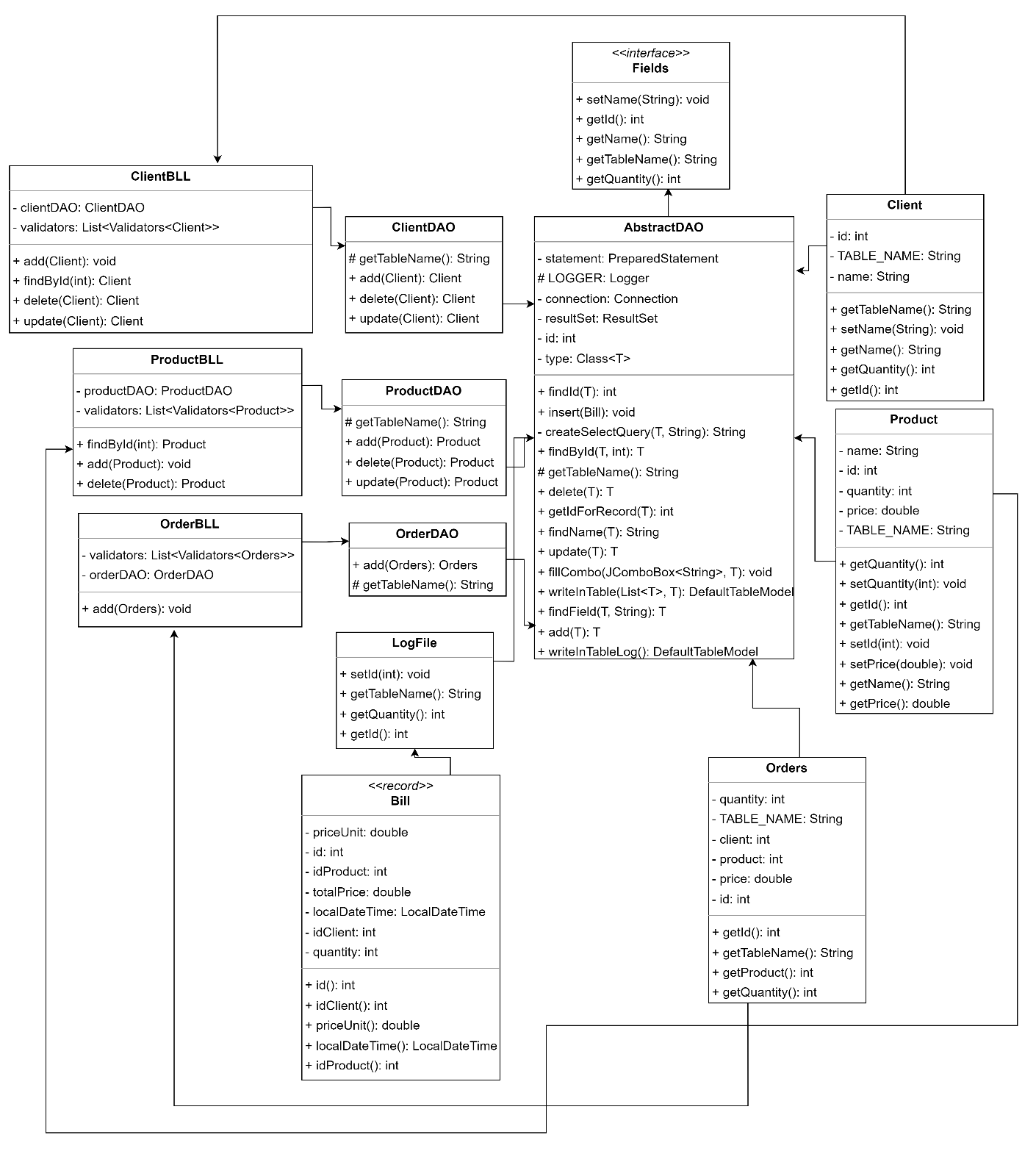
Client

LogFile

Orders

Product

Classes Diagram



The diagram contains all the classes implemented for computing the Orders Management Application. *View* is the class which contains the elements from the GUI, *Controller* classes which realize the relation between the GUI and the classes that implements the program functionality, *AbstractDAO* is the class which implements the database queries and the other classes assures the relation between AbstractDAO and Controller.

Approach of implementation

Reflection is a powerful feature in Java that enables programmers to inspect and manipulate classes, methods, and fields at runtime. It allows for the examination of an object's structure, as well as the invocation of its methods and access to its fields, even if they are private. Reflection provides great flexibility and is extensively, so it allows the configuration of objects dynamically based on runtime information. On the other hand, dynamic queries in Java refer to the ability to construct and execute database queries at runtime. This approach is commonly used when the query structure or conditions need to be determined dynamically based on user input or changing requirements. Dynamic queries facilitate the creation of flexible and adaptable applications that can adjust their data retrieval based on various runtime factors.

1. Implementation

AbstractDAO

The AbstractDAO class is a generic abstract class that provides methods for database access and manipulation. It extends the JFrame class, indicating that it is used for creating graphical user interfaces (GUI).

The class has several methods for performing common database operations. The *add* method inserts a given object into the corresponding database table. The *delete* method removes an object from the database. The *update* method updates the values of an existing object in the database. The *findById* method retrieves an object from the database based on its ID. The *findField* method retrieves a specific field of an object from the database. The *findName* method retrieves the name of an object from the database. The *findId* method retrieves the ID of an object from the database. The *writeInTable* method retrieves data from the database and creates a DefaultTableModel object that can be used to populate a GUI table.

The class also contains utility methods for creating SQL queries, establishing database connections, and handling exceptions. The database connection is established using the PostgreSQL JDBC driver. The class makes use of Java reflection to dynamically access and manipulate object fields and properties.

Client, Product, Order

The Client class is a package model representing a client entity. It extends the *AbstractDAO* class and implements the Fields interface. It has fields for the client's name and ID, along with corresponding getters and setters. It also provides methods for retrieving the table name and quantity of records associated with the client. The class is designed to store and manipulate client data in a database. The Product and Order classes are implemented in the same manner.

NameValidators, PriceValidators, QuantityOrderValidators, QuantityValidators and Validators Interface

The PriceValidators class in the BusinessLogic.Validators package is responsible for validating the price of a product. It implements the Validators interface for the Product model. The validate method checks if the price is negative and not absolute, throwing an IllegalArgumentException if the validation fails. The other Validators are implemented with the same approach.

The BusinessLogic.Validators package provides an interface Validators that defines a generic validate method for validating fields from GUI.

ClientBLL, ProductBLL, OrderBLL

The ClientBLL class is a business logic layer that handles client operations. It validates input using validators and interacts with the ClientDAO for data access. It includes methods to find, add, delete, and update clients, as well as write client data into a table model. The ProductBLL and OrderBLL are created in the same manner.

ClientDAO, ProductDAO, OrderDAO and Fields Interface

The ClientDAO class is responsible for data access operations related to clients. It extends the AbstractDAO class and implements CRUD (Create, Read, Update, Delete) operations. These operations are utilized in the ClientBLL class. The class provides methods for deleting, updating, and adding clients to the database. The ProductDAO and OrderDAO are created in the same manner.

The DataAccess package defines an interface Fields with methods for retrieving and setting common fields like ID, name, table name, and quantity, and it can handle SQLException.

Bill and LogFile

The Bill class is a record that represents a bill. It has several data fields including id, localDateTime, idProduct, idClient, quantity, priceUnit, and totalPrice. Each field corresponds to specific information related to a bill, such as its unique identifier, the date and time of the bill, the product ID, client ID, quantity, unit price, and total price (calculated as the product of quantity and price unit).

The LogFile class is responsible for handling the insertion of bills into a log table. It extends the AbstractDAO class and implements the Fields interface. The constructor of LogFile takes the necessary parameters (id, localDateTime, idProduct, idClient, quantity, priceUnit, totalPrice) to create a Bill object, and then inserts the bill into the appropriate table. Additionally, the LogFile class overrides methods from the Fields interface to provide functionality for retrieving the ID, setting the ID, getting the table name, and obtaining the quantity.

Together, these classes in the Model package facilitate the representation and management of bill-related data in the system.

ControllerMenu, ControllerClient, ControllerProduct, ControllerOrder

The ControllerMain class in the Presentation package connects the GUI with the selected interface. It handles button clicks in the main menu and creates the corresponding views and controllers for clients, orders, and products.

The ControllerClient class in the Presentation package acts as a mediator between the GUI and client operations. It manages events triggered by buttons in the GUI, such as adding, deleting, and updating clients. It also handles the navigation back to the main menu. The class utilizes the ClientBLL class for performing client-related business logic and interacts with the ViewClients class for displaying and updating the GUI. The same approach is used for ControllerProduct.

The ControllerOrder class in the Presentation package establishes the connection between the GUI and the order operations. It handles events related to orders, such as adding a new order, navigating back to the main menu, and viewing bills. The class utilizes the OrderBLL and ProductBLL classes for performing order-related and product-related business logic, respectively. It also interacts with various model classes (Orders, Product, Client, LogFile, Bill) and the ViewOrders and ViewBills GUI classes.

ViewMenu, ViewClient, ViewBill, ViewProduct, ViewOrders

The ViewClients class in the Presentation package represents the GUI for managing clients. It extends JFrame and provides methods for setting up the window dimensions, adding event listeners to buttons, accessing input fields, and displaying messages. The other View classes are implemented in the same manner.

1. Results

The user inputs the data which is validated. If the data is correct, the operation chosen is computed and a Success message is displayed. In error case, the program will throw a specific error in the console. The user can add, delete or update a specific client, add delete or update a specific product and add an order choosing an existing product and an existing client and introducing a specific quantity. If the quantity is greater than the stock for the chosen product, an *Insufficient stock!* Message will be displayed. In the Success case, the order will be added and it will be created a bill for this order, added also in the *Bills* table.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated with medium confidence

1. Conclusions

This assignment used the OOP and DB concepts learned in the first semester, so it was a good way for revising what I learned in the first semester. Also, it was a challenge to manage my time in order to have time for implementing it.

I learned about how to use reflection and dynamically techniques for DB queries, how to operate with them and also I learned how to use JTable. It was a good practice of learning Java Record, JavaDoc for documenting my code and how to create a dump file.

7. Bibliography

<https://forums.oracle.com/ords/apexds/post/update-a-database-table-from-jtable-6602>

<https://jenkov.com/tutorials/java-reflection/index.html>

<https://gitlab.com/utcn_dsrl/pt-reflection-example>