DOCUMENTATION

ASSIGNMENT *3*

STUDENT NAME: Apostol Horia Dan

GROUP: 30227

# CONTENTS

1.Assignment Objective…………………………………………………………………………………..3

2.Problem Analysis, Modeling,Scenarios, Use Cases……………………………………………………4

3.Design…………………………………………………………………………………………………..5

4.Implementation…………………………………………………………………………………………6

5.Results…………………………………………………………………………………………………11

6.Conclusions……………………………………………………………………………………………11

7.Bibliography…………………………………………………………………………………………...12

# Assignment Objective

The objective of this assignment is to develop an object-oriented program that connects to a database and utilizes tables to store and manage data related to products, clients, and orders. The program should provide functionality for adding, updating, and removing clients, products, and orders. Additionally, it should allow for searching based on unique identifiers and generate bills for clients based on their order history.

Specific requirements for the assignment include:

1. Database Connectivity: Implement a database connection mechanism to establish communication between the program and the database. Ensure that the program can establish a reliable connection and perform operations on the database tables.
2. Functionality for Adding: Develop functionality to add new clients, products, and orders to their respective tables. Allow users to input relevant details for each entity, such as client information, product details, and order specifications. Ensure the data is stored correctly in the corresponding tables.
3. Functionality for Updating: Implement functionality to update existing client and product records. Enable users to modify client details and product information specifications as needed. Update the respective tables in the database with the modified data.
4. Functionality for Adding: Develop functionality to add new clients, products, and orders to their respective tables. Allow users to input relevant details for each entity, such as client information, product details, and order specifications. Ensure the data is stored correctly in the corresponding tables.
5. Functionality for Updating: Implement functionality to update existing client and product records. Enable users to modify client details, product information, and order specifications as needed. Update the respective tables in the database with the modified data.

# Problem Analysis, Modeling, Scenarios, Use Cases

A picture containing screenshot, diagram, design

Description automatically generated

**2.1 Use Case Diagram**

The objective of this assignment is to develop an object-oriented program with a user-friendly interface that connects to a database and allows users to manage data related to clients, products, and orders. The program will provide a menu with three options: Clients, Products, and Orders.

Clients:

When the user selects "Clients," a ClientsView window will appear, displaying a table of clients.

The window will have options to add, remove, and update clients.

At the top of the window, there will be a search option to find clients based on their ID.

Additionally, the ClientsView will include a "Bill" option, which, upon selection, will open a new window showing the products ordered by the client and the total amount to be paid.

Products:

Choosing the "Products" option will open a ProductsView window, displaying a table of products.

Similar to the ClientsView, the ProductsView window will have options to add, remove, and update products.

The window will also include a search option at the top based on the product ID.

Orders:

The "Orders" option will allow the user to search for specific orders based on their ID.

Additionally, the user can add new orders, which will open a new window with a selection of clients.

After selecting a client, a ProductsView window will open, allowing the user to choose a product for the order.

The user will need to specify the quantity, and the program will validate if it is available in the store before adding it to the order.

For this project, I utilized XAMPP as my local server to establish a connection with the database. By launching XAMPP, I was able to set up a local development environment that included Apache as the web server and MySQL as the database management system.

Once XAMPP was up and running, I accessed the database management tool, phpMyAdmin, by navigating to "localhost/phpmyadmin" in my web browser. PhpMyAdmin provided a user-friendly interface to manage the MySQL database associated with my project.

# Design

A picture containing text, diagram, plan, parallel

Description automatically generated

3.1. UML of the project

# Implementation

Bll package

The **ClientBLL** class acts as an intermediary between the user interface and the data access layer, providing validation and insertion/updating functionality for Client objects. It employs a list of validators, such as EmailValidator and PhoneNumberValidator, to ensure data integrity. The insertClient() method validates and inserts a client, while updateClient() validates and updates an existing client in the database. Overall, ClientBLL ensures reliable and accurate client data management.

Bll.validators package

The **EmailValidato**r class is an implementation of an email validator in Java. It uses regular expressions to match and validate email addresses.

**PhoneNumberValidator** is a class, which validates the phone number of a Client object based on a specified pattern, throwing an exception if the phone number does not match the pattern.

The connection package contains the **ConnectionFactory** class, which provides methods for establishing a connection to a MySQL database. It creates a connection using the specified database URL, username, and password. The class also includes methods for closing the connection, statement, and result set objects to ensure proper resource management.

The dao package contains the **AbstractDAO** class, which serves as a superclass for other DAO (Data Access Object) classes in the project. It provides various methods for executing database operations, including creating SQL queries, retrieving data, updating and deleting rows, and generating tables.

The key methods in the AbstractDAO class are as follows:

createSelectQuery(String field): Creates a SELECT query string for retrieving data from a table based on a specified field.

createDeleteQuery(): Creates a DELETE query string for deleting a row from a table based on the ID.

showTableQuery(): Creates a query string for retrieving all data from a table.

createInsertQuery(Object o): Creates an INSERT query string for inserting data into a table based on an object.

createUpdateQuery(Object o): Creates an UPDATE query string for updating a row in a table based on an object.

getIdsFromTable(): Retrieves the IDs from a table as a list of strings.

generateTable(List<Object> list): Generates a table in the console based on a list of objects.

createObjects(ResultSet resultSet): Creates a list of objects by populating them with data from a ResultSet.

showTable(): Retrieves a ResultSet of all data from a table.

deleteFromTable(Choice choice): Deletes a row from a table based on the selected choice.

retrieveProperties(T object): Retrieves the properties (field names and values) of an object as a map.

updateFromTable(T obj): Updates a row in a table with the values provided in the object.

Overall, the AbstractDAO class provides a set of reusable methods that can be used by other DAO classes to perform common database operations in a flexible manner.

The **ClientDAO** class is a data access object (DAO) specifically designed for client-related operations. It extends the AbstractDAO class, indicating that it works with the Client model.

It provides methods for client-related database operations such as finding a client by ID and inserting a new client.

The class uses the ConnectionFactory class to establish a connection to the database.

It logs any warnings or exceptions using the Logger class.

Methods in the ClientDAO class:

findById(int clientId, Client client): This method retrieves a client from the database based on the provided client ID. It returns a ResultSet containing the row with the matching ID. The client object passed as a parameter is updated with the retrieved values.

insert(Client client): This method inserts a new client into the database. It takes a Client object as a parameter and returns the ID of the inserted client. The method uses a prepared statement to execute an insert query and retrieves the generated ID using Statement.RETURN\_GENERATED\_KEYS.

Overall, the ClientDAO class provides an interface to interact with the client data in the database by encapsulating the database operations specific to clients.

The **OrderDAO** class is a data access object (DAO) specifically designed for managing orders. It provides methods and queries for performing operations related to orders in the database. The class extends the AbstractDAO class.

The class contains specialized methods and complex queries tailored for order management.

It utilizes the ConnectionFactory class to establish a database connection.

The class includes SQL statements for inserting orders, retrieving orders by ID, displaying order details, and generating a bill for a specific client.

Methods in the OrderDAO class:

findById(int orderId, Order order): This method retrieves an order from the database based on the provided order ID. It returns a ResultSet containing the row with the matching ID. The order object passed as a parameter is updated with the retrieved values.

insert(Client c, Product p, int quantity): This method inserts a new order into the database. It takes a client, product, and quantity as parameters and returns the ID of the inserted order. Before inserting, it checks if the quantity requested is available in the product's inventory.

showTable(): This method returns a ResultSet containing the inner join of the orders, clients, and products tables. It provides a comprehensive view of the orders with additional details such as client name, product name, quantity, client address, and client phone number.

showBill(int id): This method returns a ResultSet containing the orders and product details for a specific client ID. It generates a bill by retrieving the product name, product price, and order quantity.

Overall, the OrderDAO class encapsulates the database operations specific to orders and provides an interface to interact with order data in the database.

The **ProductDAO** class is a data access object (DAO) that handles product-related operations in the application. It provides methods for retrieving and inserting products into the database. The class extends the AbstractDAO class.

The class contains methods specific to product-related operations, such as finding a product by ID and inserting a new product.

It utilizes the ConnectionFactory class to establish a database connection.

The class interacts with the Product model class to retrieve and update product data in the database.

Methods in the ProductDAO class:

findById(int productId, Product product): This method retrieves a product from the database based on the provided product ID. It returns a ResultSet containing the row with the matching ID. The Product object passed as a parameter is updated with the retrieved values.

insert(Product product): This method inserts a new product into the database. It takes a Product object as a parameter and returns the ID of the inserted product. The method uses the createInsertQuery method from the AbstractDAO class to generate the SQL insert statement.

Overall, the ProductDAO class encapsulates the database operations specific to products and provides an interface to interact with product data in the database.

The **Client**, **Order**, and **Product** classes belong to the "model" package and represent the objects that correspond to the tables in the database. These classes have attributes that mirror the columns in their respective tables.

Summary of the classes in the "model" package:

Client:

Attributes: id, name, address, email, phoneNumber

Represents the client object that corresponds to the "clients" table in the database.

Order:

Attributes: order\_id, client\_id, product\_id, quantity

Represents the order object that corresponds to the "orders" table in the database.

Product:

Attributes: id, name, description, price, quantity

Represents the product object that corresponds to the "products" table in the database.

These classes serve as models for the database tables, and their attributes reflect the columns in the tables. By using these classes, you can create instances of client, order, and product objects and interact with the corresponding data in the database.

The GUIs are located in the "**presentation**" package. There are several views within this package that serve different purposes:

**AddClientView** and **AddProductView**:

These views have a similar layout but are associated with different tables.

Users can input data into the provided fields to add new entries into the respective tables.

The data entered by the user is inserted into the database.

**BillView**:

This view displays a table of objects ordered by the selected client.

Users can choose a client, and the view shows the corresponding orders for that client.

**ClientsView** and **ProductsView**:

These views display tables of clients and products, respectively.

Users can view the data in the tables and have limited options for interacting with the data.

Both views have a similar layout.

**OrdersView**:

This view presents a table of orders.

Users can view the order data and have limited options for further actions.

**MainView**:

This view serves as the main menu interface.

It provides three options: clients, orders, and products.

Users can select one of these options to navigate to the corresponding view.

**UpdateClientView** and **UpdateProductView**:

These views are similar to the add GUIs but with pre-filled fields containing existing data.

Users can update any of the displayed fields and save the changes to the respective tables.

Overall, the presentation package contains various GUI views that allow users to interact with the database. The views differ in terms of the tables they are associated with, the data they display, and the available options for users.

The **Start1** class in the “start” package instantiates a new MainView object and assigns a controller to it. The MainView serves as the main menu interface for the application. It is responsible for displaying the available options to the user and handling user interactions.

By creating a MainView object and assigning a controller to it, the Start1 class establishes the initial user interface and sets up the functionality for the main menu.

# Results

The project incorporated Abstract DAO, generics, and reflection, leading to the following outcomes:

Abstract DAO: The usage of Abstract DAO facilitated the creation of a generic data access layer. This allowed for code reusability and streamlined database operations by providing a common set of methods for different entities.

Generics: The implementation of generics enabled the development of flexible and type-safe data access methods. It eliminated the need for duplicate code by allowing the same set of operations to be applied to multiple entity types, promoting modularity and flexibility.

Reflection: The project utilized reflection to dynamically retrieve and manipulate object properties. This automated tasks such as populating object attributes from the database result set, enhancing efficiency and adaptability.

By incorporating Abstract DAO, generics, and reflection, the project achieved a more flexible and scalable architecture, promoting code reuse, type safety, and dynamic object manipulation.

# Conclusions

In conclusion, the project successfully developed a Java application that implemented a database management system for clients, orders, and products. It employed various software design patterns and concepts such as MVC (Model-View-Controller), DAO (Data Access Object), and generics to achieve modularity, code reusability, and maintainability.

Throughout the project, several key features were implemented, including CRUD (Create, Read, Update, Delete) operations for managing clients, orders, and products. The application utilized a graphical user interface (GUI) built with Swing to provide a user-friendly experience for interacting with the system.

Notable achievements of the project include the effective use of abstraction through the Abstract DAO and generics, allowing for generic data access methods that could be applied to multiple entities. Reflection was also leveraged to automate tasks such as object property retrieval and manipulation.

Through the development of this project, valuable knowledge and skills were gained in areas such as database management, software design patterns, GUI development, and the use of Java libraries and frameworks. The project demonstrated the ability to build a functional and efficient application that can effectively handle database operations and provide a seamless user experience.

Overall, the project was a valuable learning experience that showcased the application of software engineering principles and techniques in developing a practical database management system.

# Bibliography

<https://app.diagrams.net/>

https://dsrl.eu/courses/pt/