DOCUMENTATION

ASSIGNMENT *3*

STUDENT NAME: TUDOR ALEXIA-CRISTINA
GROUP: 30424

CONTENTS

1. Assignment Objective

The main objective of the assignment is designing and implementing an application managing the products, the clients and the orders for a warehouse using handwritten registries is difficult and time consuming.

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

2. Problem Analysis, Modeling, Scenarios, Use Cases

- Problem Analysis

The problem at hand is to develop an Orders Management application for processing client orders for a warehouse. The application needs to store information about products, clients, and orders in a relational database. The application should follow a layered architecture pattern and utilize the following classes: Model classes, Business Logic classes, Presentation classes, and Data access classes.

- Modeling

The user can simulate the application by entering the valid inputs for the client, order or product into the graphical user interface. For example, the client GUI has 5 Text Fields: id, name, address, email, age.

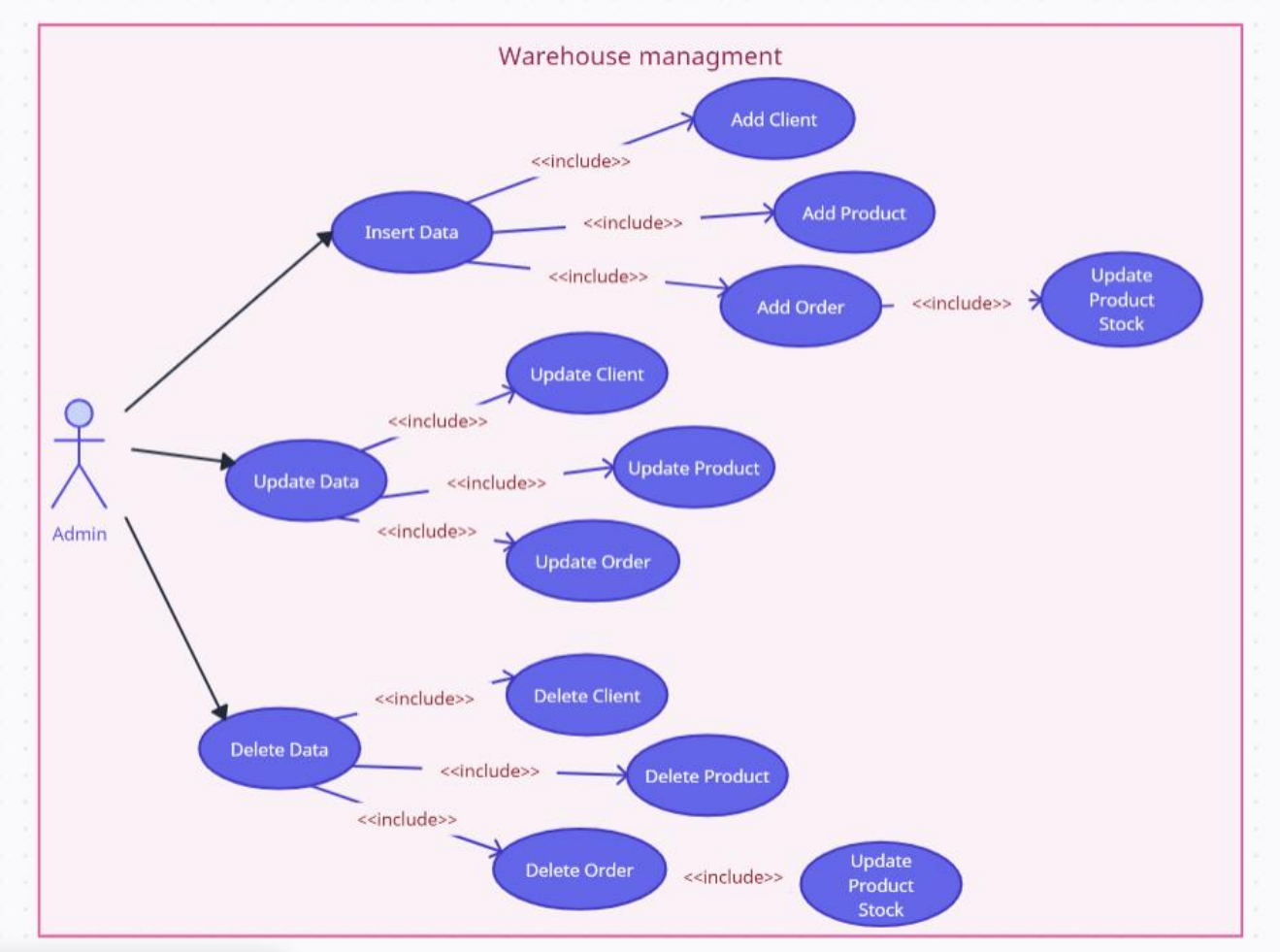
By pressing the “Add” button, the given inputs are checked in the validator. If they are valid, they would be stored in the database: storedb and the table would be updated with the new added object.

By pressing the “Update” button, the given inputs are checked in the validator. If they are valid, the already existing data would be updated in the database: storedb and the table would be updated with the new updated object.

By pressing the “Delete” button, the stored given id in the database: storedb and the table would be deleted.

- Scenarios and Use Cases

The user should insert valid information in the Text Fields and select one of the buttons: Add, Update, Delete, Bill (in case of order) to modify the database data and to update the JTable in real-time.



3. Design

- Packages

In the layered architecture pattern, the application is divided into different layers, each with its specific responsibilities. Here is a breakdown of the layers and their corresponding classes in the Orders Management application:

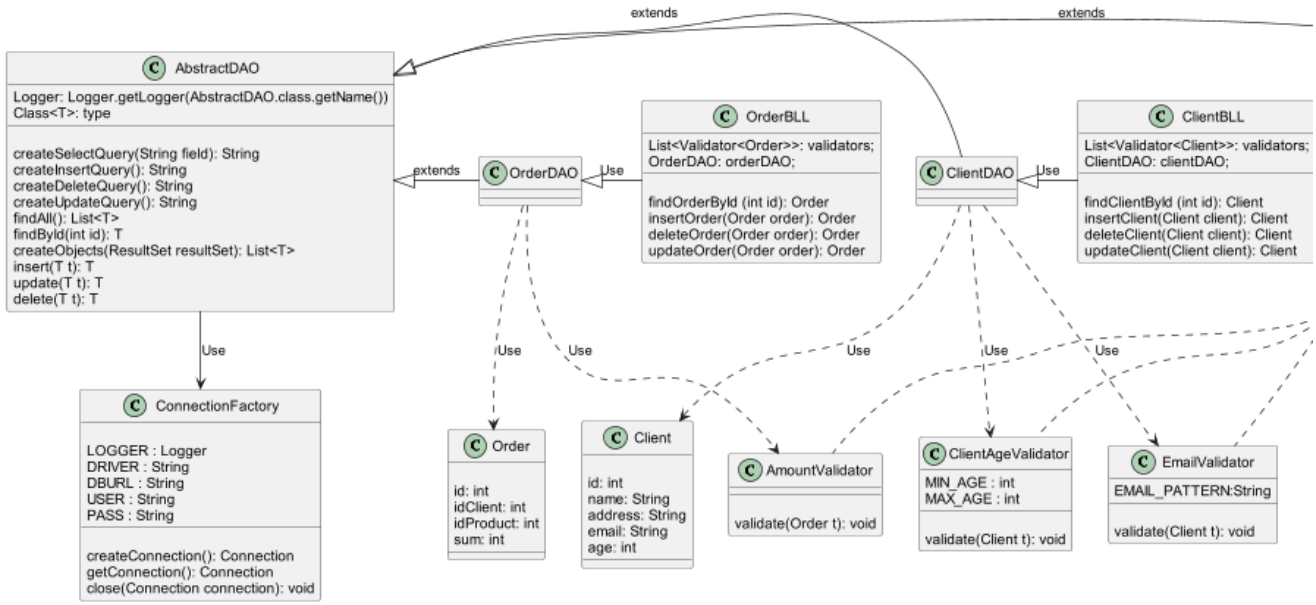
- DataAccess: contains the classes containing the queries and the database connection
- Model: contains classes mapped to the database table
- BusinessLogic: contains the classes that encapsulate the application logic and the validators, enforcing business rules, and coordinating interactions between the presentation layer and the data access layer

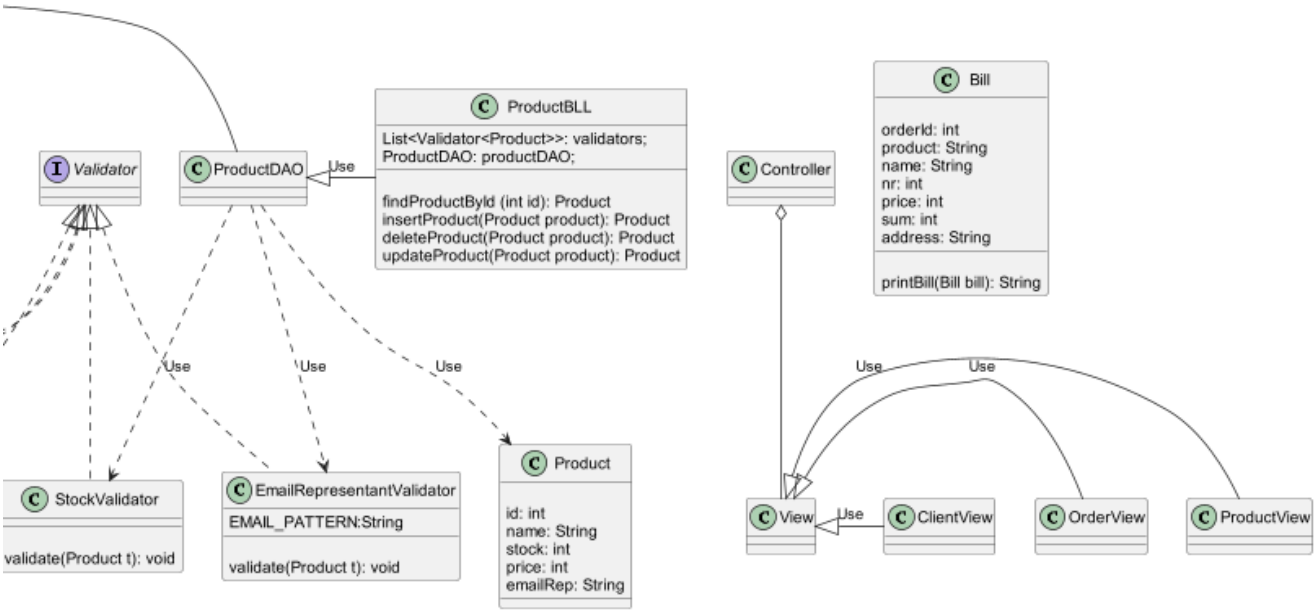
-Validators: validate the data or objects based on specific rules or conditions before performing certain operations or actions

- Connection: contains a connection to a relational database and provides methods for database interaction and transaction management

-Presentation: - contains the classes defining the user interface

- Class Diagram





- Data Structures

The data structures that I've used are primitive data types: integers, Strings, booleans; List of objects such as: ArrayList. I have created new objects such as: Client, Product, Order, Bill.

- Algorithms

I have not used specific algorithms, but I've developed approaches to assist me in improving code efficiency. As a result, I chose appropriate algorithms for the problems, used efficient data structures, and avoided redundancy and unused code. In addition, I used object-oriented programming principles to make my code easier to read and maintain.

4. Implementation

- Model

A) Product

Constructor:
public Product(int id,String name,int stock,int price,String emailRep): constructs a Product object with the specified ID, name, stock, price, emailrep

public Product(String name,int stock,int price,String emailRep): constructs a Product object with the specified name, stock, price, emailrep
public Product(): default constructor

Methods:
Getters and Setters

B) Order

Constructor:
public Order(int id,int idClient,int idProduct,int sum): constructs an Order object with the specified ID, client ID, product ID, and nr bought products.
public Order(int idClient,int idProduct,int sum): constructs an Order object with the client ID, product ID, and nr bought products.
public Order(): default constructor

Methods:
Getters and Setters

C) Client

Constructor:
public Client(int id,String name,String address,String email,int age): constructs a Client object with the specified ID, name, address, email, and age
public Client(String name,String address,String email,int age): constructs a Client object with the specified name, address, email, and age
public Client(): default constructor

Methods:
Getters and Setters

D) Bill

Constructor:
public Bill(int orderId,String product,String name,int nr,int price,String address): constructs a Bill object with the specified orderId,product, name, nr, price, address
public Bill(): default constructor

Methods:
public String printBill(Bill bill): builds into a StringBuilder the order's details about the product and the client

- DataAccess

A) ProductDAO extends AbstractDAO<Product>

Constructor:
none

Methods:
none

B) OrderDAO extends AbstractDAO<Order>

Constructor:
none

Methods:
none

C) ClientDAO extends AbstractDAO<Client>

Constructor:
none

Methods:
none

D) AbstractDAO

Constructor:
public AbstractDAO():

Methods:
private String createSelectQuery(String field): generates a SQL SELECT query string to retrieve data from a table based on a specified field
private String createInsertQuery(): builds a SQL INSERT query string to insert data into a table
private String createDeleteQuery(): builds a SQL DELETE query string to delete a row from a table based on the ID
private String createUpdateQuery(): builds a SQL UPDATE query string to update specific fields of a row in a table, excluding the ID field
public List<T> findAll(): finds all objects of the specified type
public T findById(int id): retrieves an object of type T from the database based on the given ID.
private List<T> createObjects(ResultSet resultSet): transforms the ResultSet obtained from a database query into a list of objects of type T, using reflection to set the values of the object's fields based on the column values in the ResultSet

public T insert(T t): inserts an object T into the database
public T update(T t): updates an object T from the database
public T delete(T t): deletes an object T from the database

- BusinessLogic
  - ♦ Validators
    i. Validator

Constructor:
none

Methods:
public void validate(T t): defines a contract for validating objects of a specific type

ii. StockValidator

Constructor:
none

Methods:
public void validate(Product t): validates the stock of a product

iii. EmailValidator
Constructor:
none

Methods:
public void validate(Client t): validates the email of a client

iv. EmailRepresentantValidator

Constructor:
none

Methods:
public void validate(Product t): validates the email representation of a product's representative

v. ClientAgeValidator

Constructor:
none

Methods:
public void validate(Client t): validates the age of a Client object

vi. AmountValidator

Constructor:

none

Methods:

public void validate(Order t): validates the quantity of a product in an Order, such that it doesn't order more than available in stock

A) ProductBLL

Constructor:

public ProductBLL(): initializes the validators list with specific validators and creates an instance of the ProductDAO class

Methods:

public Product findProductById (int id): finds and retrieves a product by its ID
public Product insertProduct(Product product): inserts a new product into the database after validating it using a set of validators
public Product deleteProduct(Product product): deletes a product from the database
public Product updateProduct(Product product): updates an existing product in the database after validating it using a set of validators

B) OrderBLL

Constructor:

public OrderBLL(): initializes the validators list with specific validators and creates an instance of the OrderDAO class

Methods:

public Order findOrderById (int id): finds and retrieves an order by its ID
public Order insertOrder(Order order): inserts a new order into the database after validating it using a set of validators and updates the corresponding product's stock
public Order deleteOrder(Order order): deletes an order from the database and updates the corresponding product's stock
public Order updateOrder(Order order): updates an existing order in the database after validating it using a set of validators

C) ClientBLL

Constructor:

public ClientBLL(): initializes the validators list with specific validators and creates an instance of the ClientDAO class

Methods:

public Client findClientById(int id): finds and retrieves a client by their ID
public Client insertClient(Client client): inserts a new client into the database after validating them using a set of validators
public Client deleteClient(Client client): deletes a client from the database
public Client updateClient(Client client): updates an existing client in the database after validating them using a set of validators

- Connection

A) ConnectionFactory

Constructor:
private ConnectionFactory()

Methods:
private Connection createConnection(): creates a new database connection using the specified URL, username, and password
public static Connection getConnection(): returns a database connection by invoking the createConnection() method on the single instance of ConnectionFactory
public static void close(Connection connection): closes the given database connection

- Presentation

A) View

Constructor:
public View():

Methods:
private void initComponents(): initializes the graphical user interface components and sets up the layout and properties of the main frame
private void exitAction(java.awt.event.MouseEvent evt): handles the event when the user clicks on the "X" button to exit the application
private void clientButtonActionPerformed(java.awt.event.ActionEvent evt): handles the event when the user clicks on the "Clients" button, disposing the current view and opening the ClientView window
private void productButtonActionPerformed(java.awt.event.ActionEvent evt): handles the event when the user clicks on the "Products" button, disposing the current view and opening the ProductView window
private void orderButtonActionPerformed(java.awt.event.ActionEvent evt): handles the event when the user clicks on the "Orders" button, disposing the current view and opening the OrderView window

B) ProductView

Constructor:
public ProductView():

Methods:
public void setOrderDetailsToTable(): retrieves product details from the database and populates them into a table
public boolean addOrder (): adds a new product to the database based on the user input
public boolean updateOrder(): updates an existing product in the database based on the user input
public boolean deleteOrder(): deletes a product from the database based on the user input
public void clearTable(): clears the contents of the table
private void initComponents(): initializes and configures the graphical user interface components of the product management system
private void backAction(java.awt.event.MouseEvent evt): performs an action (navigating back) when the back button is clicked
private void addButtonActionPerformed(java.awt.event.ActionEvent evt): performs an action (adding a product) when the add button is clicked
private void updateButtonActionPerformed(java.awt.event.ActionEvent evt): performs an action (updating a product) when the update button is clicked
private void deleteButtonActionPerformed(java.awt.event.ActionEvent evt): performs an action (deleting a product) when the delete button is clicked

C) ClientView

Constructor:
public ClientView ():

Methods:
public void setOrderDetailsToTable(): retrieves client details from the database and populates them into a table
public boolean addOrder (): adds a new client to the database based on the user input
public boolean updateOrder(): updates an existing client in the database based on the user input
public boolean deleteOrder(): deletes a client from the database based on the user input
public void clearTable(): clears the contents of the table
private void initComponents(): initializes and configures the graphical user interface components of the client management system
private void backAction(java.awt.event.MouseEvent evt): performs an action (navigating back) when the back button is clicked
private void addButtonActionPerformed(java.awt.event.ActionEvent evt): performs an action (adding a client) when the add button is clicked
private void updateButtonActionPerformed(java.awt.event.ActionEvent evt): performs an action (updating a client) when the update button is clicked
private void deleteButtonActionPerformed(java.awt.event.ActionEvent evt): performs an action (deleting a client) when the delete button is clicked

D) OrderView

Constructor:

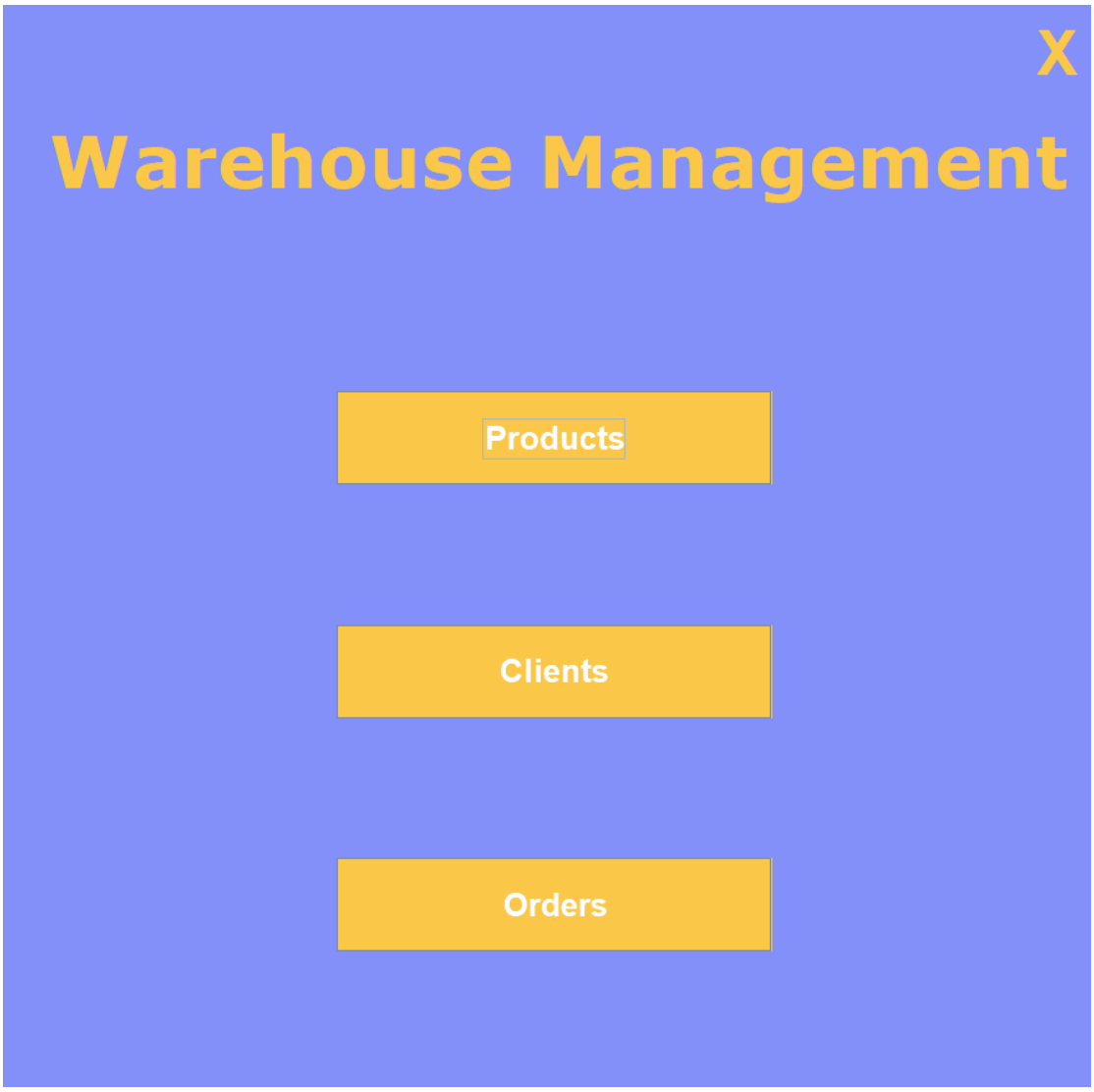
public OrderView ():

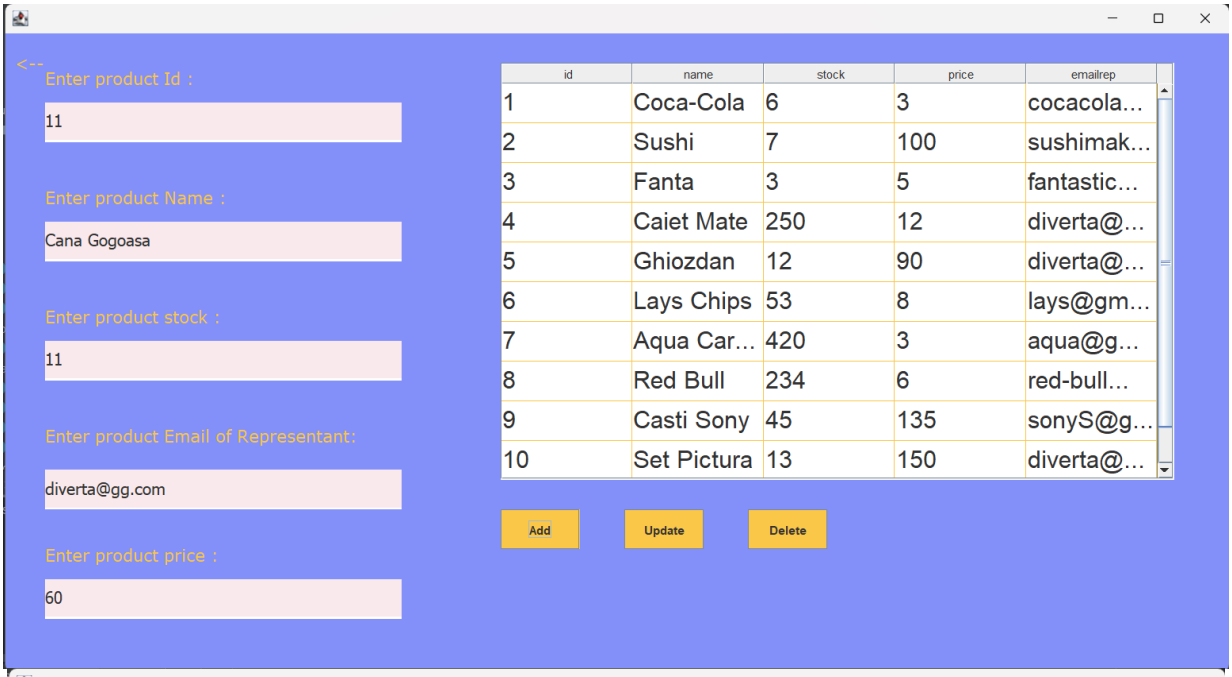
Methods:

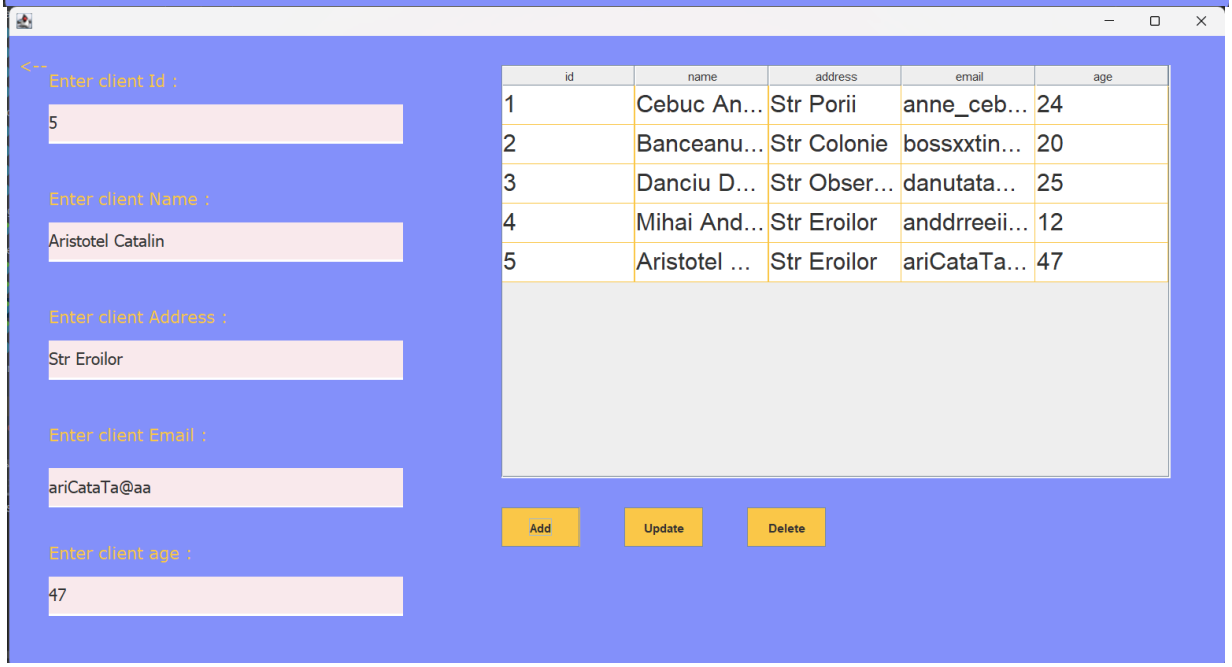
public void setOrderDetailsToTable(): retrieves order details from the database and populates them into a table
public boolean addOrder (): adds a new order to the database based on the user input
public boolean updateOrder(): updates an existing order in the database based on the user input
public boolean deleteOrder(): deletes a order from the database based on the user input
public void clearTable(): clears the contents of the table
private void initComponents(): initializes and configures the graphical user interface components of the order management system
private void backAction(java.awt.event.MouseEvent evt): performs an action (navigating back) when the back button is clicked
private void addButtonActionPerformed(java.awt.event.ActionEvent evt): performs an action (adding a order) when the add button is clicked
private void updateButtonActionPerformed(java.awt.event.ActionEvent evt): performs an action (updating a order) when the update button is clicked
private void deleteButtonActionPerformed(java.awt.event.ActionEvent evt): performs an action (deleting a order) when the delete button is clicked

E) Controller

public static void main(String args[]): starts the application, intializing the View Class.


X
Warehouse Management
Products
Clients
Orders


<--
Enter product Id :
11
Enter product Name :
Cana Gogoasa
Enter product stock :
11
Enter product Email of Representant:
diverta@gg.com
Enter product price :
60
id
name
stock
price
emailrep
1
Coca-Cola
6
3
cocacola...
2
Sushi
7
100
sushimak...
3
Fanta
3
5
fantastic...
4
Caiet Mate
250
12
diverta@...
5
Ghiozdan
12
90
diverta@...
6
Lays Chips
53
8
lays@gm...
7
Aqua Car...
420
3
aqua@g...
8
Red Bull
234
6
red-bull...
9
Casti Sony
45
135
sonyS@g...
10
Set Pictura
13
150
diverta@...
Add
Update
Delete


<--
Enter client Id :
5
Enter client Name :
Aristotel Catalin
Enter client Address :
Str Eroilor
Enter client Email :
ariCataTa@aa
Enter client age :
47
id
name
address
email
age
1
Cebuc An...
Str Porii
anne_ceb...
24
2
Banceanu...
Str Colonie
bossxxtin...
20
3
Danciu D...
Str Obser...
danutata...
25
4
Mihai And...
Str Eroilor
anddrreeii...
12
5
Aristotel ...
Str Eroilor
ariCataTa...
47
Add
Update
Delete

5. Results

The generated bill and the testing of the Add Button.

BillGenerated.txt

File Edit View

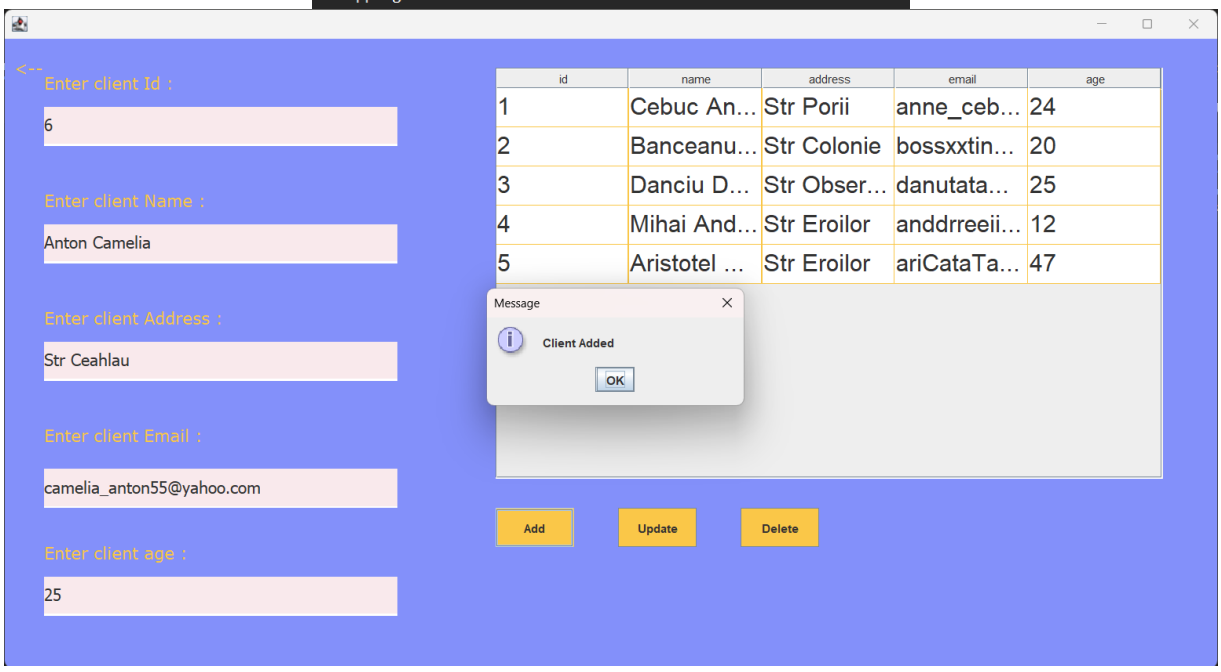
```
Bill order: 1
Product Coca-Cola
Price 3
Quantity: 6
Total Sum: 18
Bought by: Cebuc Anneliese
Shipping to: Str Porii

Bill order: 2
Product Coca-Cola
Price 3
Quantity: 3
Total Sum: 9
Bought by: Cebuc Anneliese
Shipping to: Str Porii

Bill order: 3
Product Coca-Cola
Price 3
Quantity: 3
Total Sum: 9
Bought by: Cebuc Anneliese
Shipping to: Str Porii

Bill order: 4
Product Sushi
Price 100
Quantity: 3
Total Sum: 300
Bought by: Banceanu Tina
Shipping to: Str Colonie

Bill order: 6
Product Ghiozdan
Price 90
Quantity: 2
Total Sum: 180
Bought by: Aristotel Catalin
Shipping to: Str Eroilor
```



6. Conclusions

This assignment provided a good challenge for me, helping me understand and further improve my OOP skills. Along with fundamental OOP knowledge and programming skills, these kinds of projects also necessitate excellent time management skills and a good understanding of the problem's requirements and constraints. It was challenging, but it was impressive how I learned about project management - designing, implementing, and showing real time evolution - in a short time.

Working with the database proved to be one of the most challenging parts of the assignment. Nevertheless, I was able to gain a lot of knowledge and insight into its use and implementation.

Future Development:
The functionality and usability of the client views are two areas that could be improved. These views are currently not intuitive and are primarily intended for administrative purposes. To better serve customers, the interface should be updated with elements such as product images, pricing information, a shopping cart, and a simplified checkout process. This will improve the overall user experience and make it more user-friendly.

7. Bibliography

1. *JavaDoc - https://www.baeldung.com/javadoc*
2. *Java JDBC - https://www.baeldung.com/java-jdbc*
3. *Layers of a Standard Enterprise Application - https://dzone.com/articles/layers-standard-enterprise*
4. *Java Reflection - https://www.oracle.com/technical-resources/articles/java/javareflection.html*
5. *Java Reflection - https://www.geeksforgeeks.org/reflection-in-java/*
6. *JTable - https://www.codingninjas.com/codestudio/library/jtable-in-java*
7. *Java Model Object - https://www.geeksforgeeks.org/object-model-in-java/*
8. *Java Main Frame - https://docs.oracle.com/javase/tutorial/uiswing/components/frame.html*