DOCUMENTATION

ASSIGNMENT *3*

-ORDERS MANAGEMENT-

STUDENT NAME: Pitaru Alexandra-Elena
GROUP: 30423

CONTENTS

1. Assignment Objective

1. Main objective of the assignment:

• Design and implement an application for managing client orders in a warehouse.

A database is a structured collection of data that is organized and stored in a manner that allows for efficient retrieval, management, and manipulation. It serves as a centralized repository where data can be stored, accessed, and modified according to predefined rules and relationships.

By leveraging a database in the warehouse management application, users can efficiently track orders, manage inventory, optimize warehouse operations, and provide superior customer service. The database serves as a critical component that underpins the functionality and effectiveness of the application.

2. The sub-objectives:

   a. Analyze the problem and identify requirements (Section 2)

      i. Gather user requirements for order management.
      ii. Define the data structure for representing orders and clients.
      iii. Identify edge cases

   b. Design the Orders Management Application: (Section 3)

      i. Design the user interface for adding, deleting, editing and viewing clients, products and orders
      ii. Data model design
      iii. Implement strategies for optimizing order fulfillment, such as inventory management.
      iv. Error handling design

   c. Implement the simulation application (Section 4)

      i. Code for user interface components
      ii. Implement back-end logic
      iii. Implement strategies
      iv. Implement error handling

   d. Test the Orders Management Application (Section 5)

2. Problem Analysis, Modeling, Scenarios, Use Cases

- Functional requirements:

  a. The application should allow an employee to add, update, delete a client

  b. The application should allow an employee to add, update, delete a product

  c. The application should allow an employee to add, update, delete a order

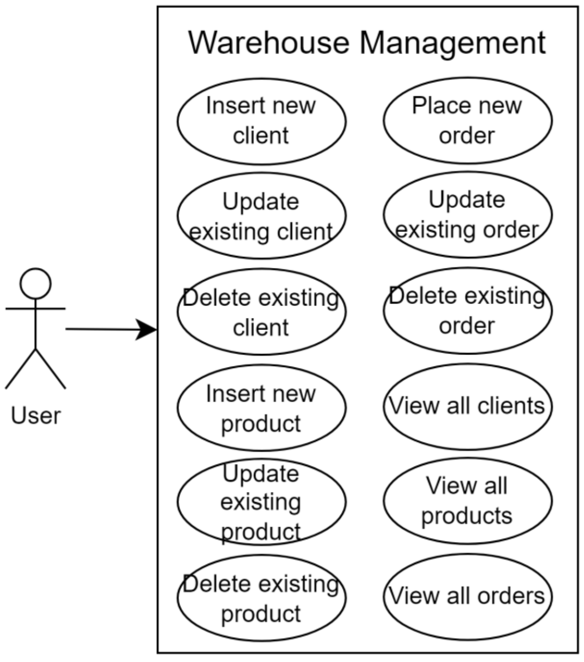
  d. The simulation application should display the data(clients table, products table, orders table).

- Non-Functional Requirements:

  e. Usability: The simulation application should be intuitive and easy to use by the user

  f. User Interface Design: The simulation application's user interface should be visually appealing, with clear and explicit labeling of buttons and controls.

  g. Performance: The application should be responsive and performant, even when handling large datasets or running complex simulations.



- Setup Simulation

User opens the application.

The user selects the desired menu (client, product or order management)

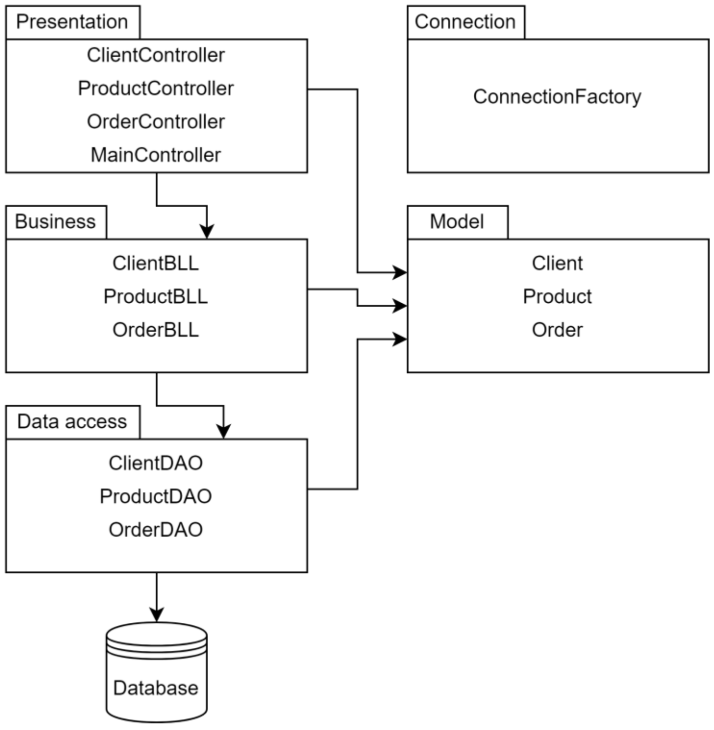
For inserting a new client or product, the user inserts the data in the text fields then presses the "Add new" button.
For placing a new order, the user has to select a client and a product from each table and the quantity of then press the "Place Order" button.
For these three operations, no id should pe inserted, as it will be generated automatically.

For editing or deleting new clients, the id of the existing object will be needed.

For viewing the list of clients, products or orders, the user just needs to press the "View all" button.

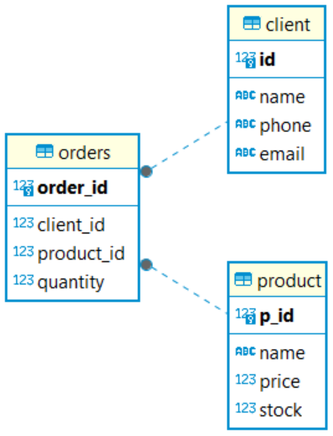
3. Design

- Packages

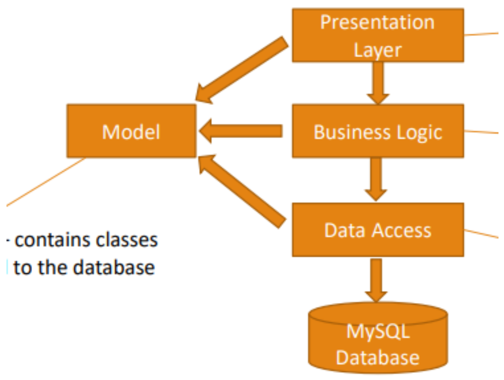


The application is designed according to the layered architecture pattern and uses the following classes:

• 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



Database diagram



Layers scheme

- **Model:**

The Model package contains classes that represent the core data structures and entities used within the warehouse management system. These classes define the structure and attributes of objects such as orders, products and clients.

This package serves as the foundation for storing and manipulating data within the application. It encapsulates the essential components of the business domain and provides a standardized representation of data objects that can be used across different layers of the application.

- **Presentation:**

The Presentation package contains classes responsible for the user interface (UI) components and interaction elements of the warehouse management system. These classes include controllers, views, and UI components built using JavaFX or other UI frameworks.

This package handles the presentation layer of the application, providing the user interface through which users interact with the system. It includes functionalities such as displaying order information, managing inventory, processing user input, and presenting data in a visually appealing and intuitive manner. The presentation layer acts as a bridge between the user and the underlying business logic, facilitating user interaction and feedback.

- **Business Logic:**

The Business Logic package contains classes that implement the business rules, logic, and algorithms required to process client orders, manage inventory, and perform other core operations within the warehouse management system. These classes orchestrate the flow of data and enforce the business rules governing the application.

This package encapsulates the business logic of the application, ensuring that the system operates according to the defined requirements and constraints. It abstracts away the complexities of data processing and manipulation, providing a clean separation between the business rules and the underlying data access layer.

- **Data Access:**

The Data Access package contains classes responsible for interacting with the database and performing CRUD (Create, Read, Update, Delete) operations on the underlying data entities. These classes include data access objects (DAOs), repositories, or data access services that handle database connectivity, query execution, and data manipulation.

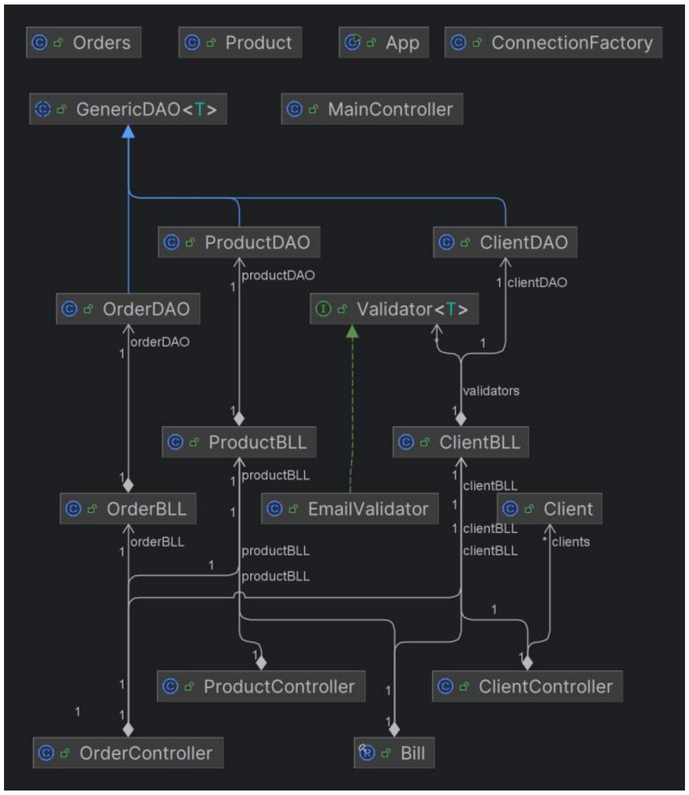
This package provides the necessary infrastructure for accessing and managing data stored in the database. It abstracts away the details of database connectivity and query execution, allowing other parts of the application to interact with the database through a unified interface. By separating data access concerns from business logic, this package promotes modularity, maintainability, and scalability.

- **SQL Database:**

The database used in the warehouse management system is PostgreSQL, a powerful open-source relational database management system (RDBMS) known for its robustness, reliability, and extensibility. The database stores structured data in tables with rows and columns, allowing for efficient storage, retrieval, and manipulation of data.

The database serves as the centralized data storage and management system for the warehouse management application. It stores essential information such as orders, products, customers, and transactional data related to warehouse operations. PostgreSQL provides a secure and scalable platform for storing and retrieving data, ensuring data integrity, consistency, and durability. The database schema is designed to reflect the relationships between different entities in the warehouse domain, facilitating efficient data querying, reporting, and analysis.

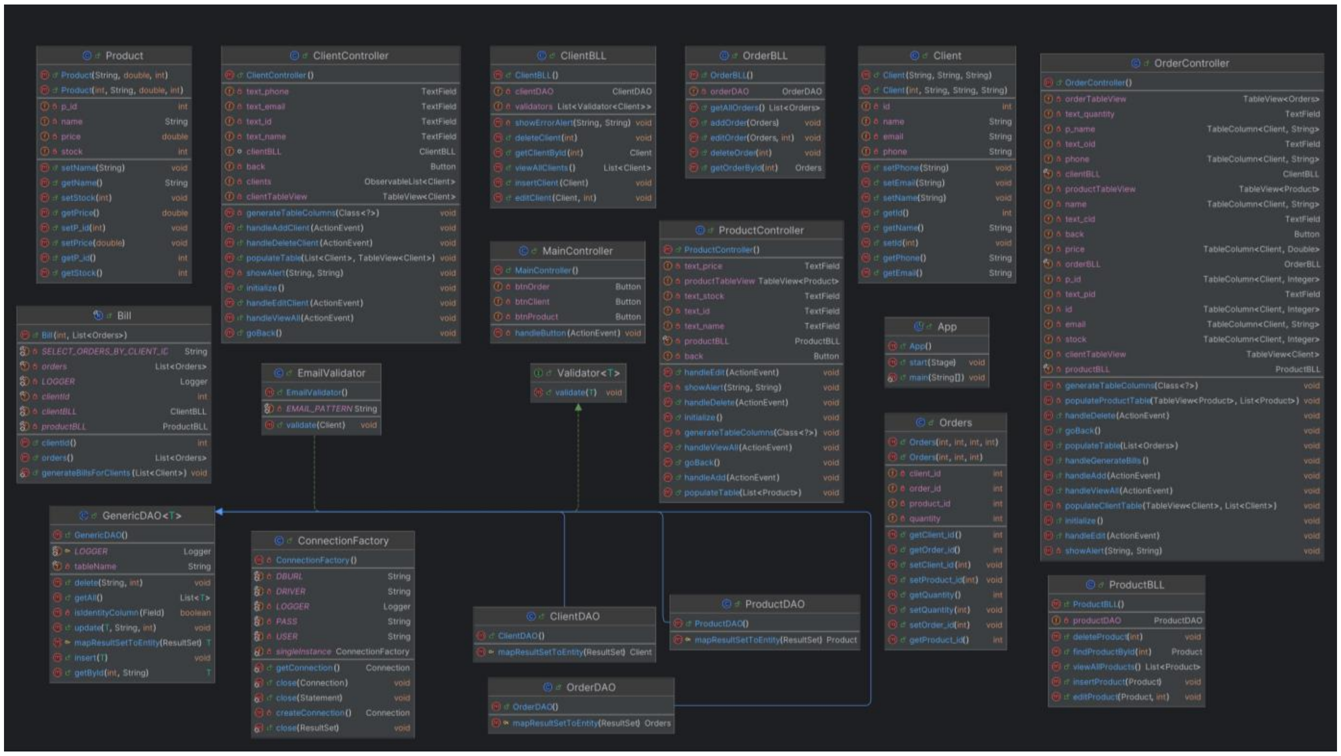
- ## Class Diagram



- ## Data Structures

ArrayList – I chose to use this structure to keep my queues and to temporary hold my clients, because this kind of lists is useful for storing and accessing data

ArrayList internally uses dynamic array to store the elements. Manipulation with ArrayList is slow because it internally uses array. If any element is removed from the array, all the bits are shifted in memory. This class can act as a list only because it implements List only.



4. Implementation

I. Class Description

➢ Application

- Application – starts the application

➢ Connection

- ConnectionFactory- is responsible with the connection with the database, creating a statement. It also has methods to close the connection/ the statement.

➢ DAO

- GenericDAO - contains generic methods to obtain a table from a list of objects, a method to insert in the database a specific object, a method to delete from the database and one to update an existing entry of the database. All the generic methods use reflection techniques.

- ClientDAO – extends the Generic DAO and uses the generic methods to implement CRUD operations on the Client table

- OrderDAO– extends the Generic DAO and uses the generic methods to implement CRUD operations on the Order table

- OrderDetailsDAO– extends the Generic DAO and uses the generic methods to implement CRUD operations on the OrderDetails table
- ProductDAO– extends the Generic DAO and uses the generic methods to implement CRUD operations on the Product table

➢ Model

- Client – models the client table from the database
- Order– models the order table from the database
- Product– models the product table from the database

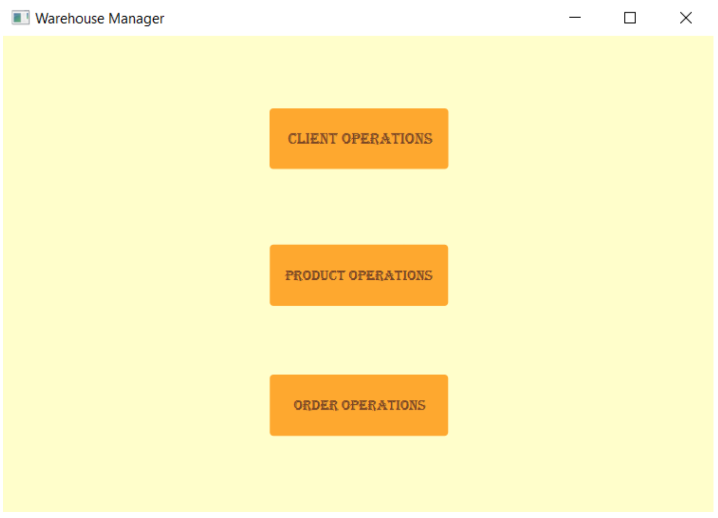
➢ View

- ClientController – Client frame, provides buttons, labels, text fields to interact with the clients table
- OrderController- Order frame, provides buttons, labels, text fields to interact with the order table
- ProductController-Product frame, provides buttons, labels, text fields to interact with the product table
- MainController - the main frame which is the "mother" of all the other frames in this application. Provides access to the other frames

II. Graphical User Interface

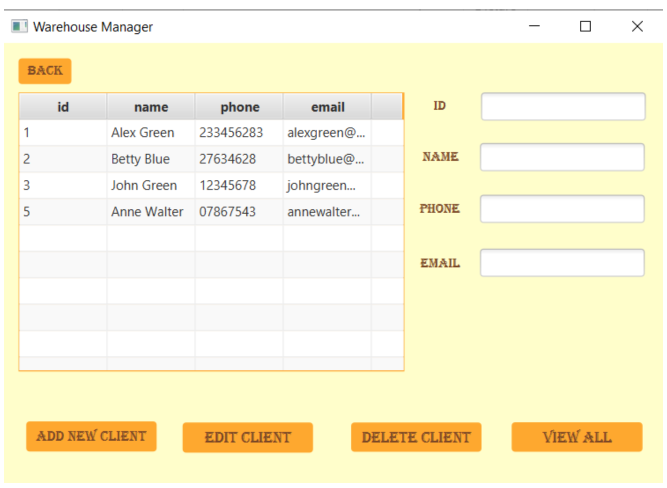
Main Page:

- This page serves as the main navigation hub.
- It contains three buttons to navigate to the next three pages: "Client Page", "Product Page", and "Order Page".

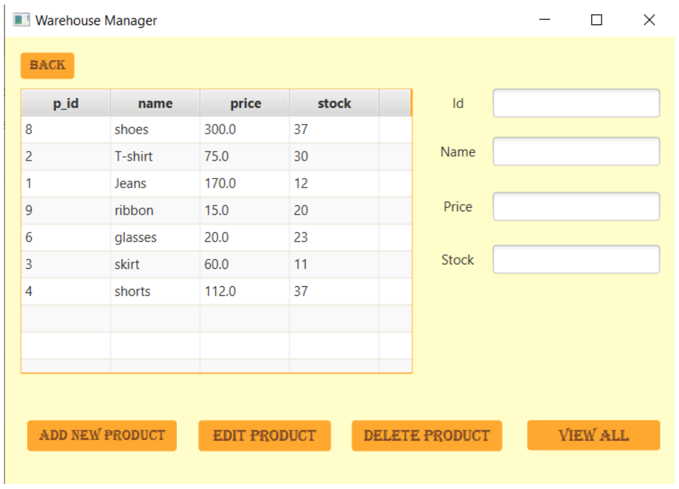


Client Page:

- Allows users to manage client data.
- Displays all existing clients from the database in a table view.
- Provides functionality to insert new clients by entering data into text fields.
- Enables users to edit existing client information.
- Supports deleting clients from the database.



id	name	phone	email
1	Alex Green	233456283	alexgreen@...
2	Betty Blue	27634628	bettyblue@...
3	John Green	12345678	johngreen...
5	Anne Walter	07867543	annewalter...



p_id	name	price	stock
8	shoes	300.0	37
2	T-shirt	75.0	30
1	Jeans	170.0	12
9	ribbon	15.0	20
6	glasses	20.0	23
3	skirt	60.0	11
4	shorts	112.0	37

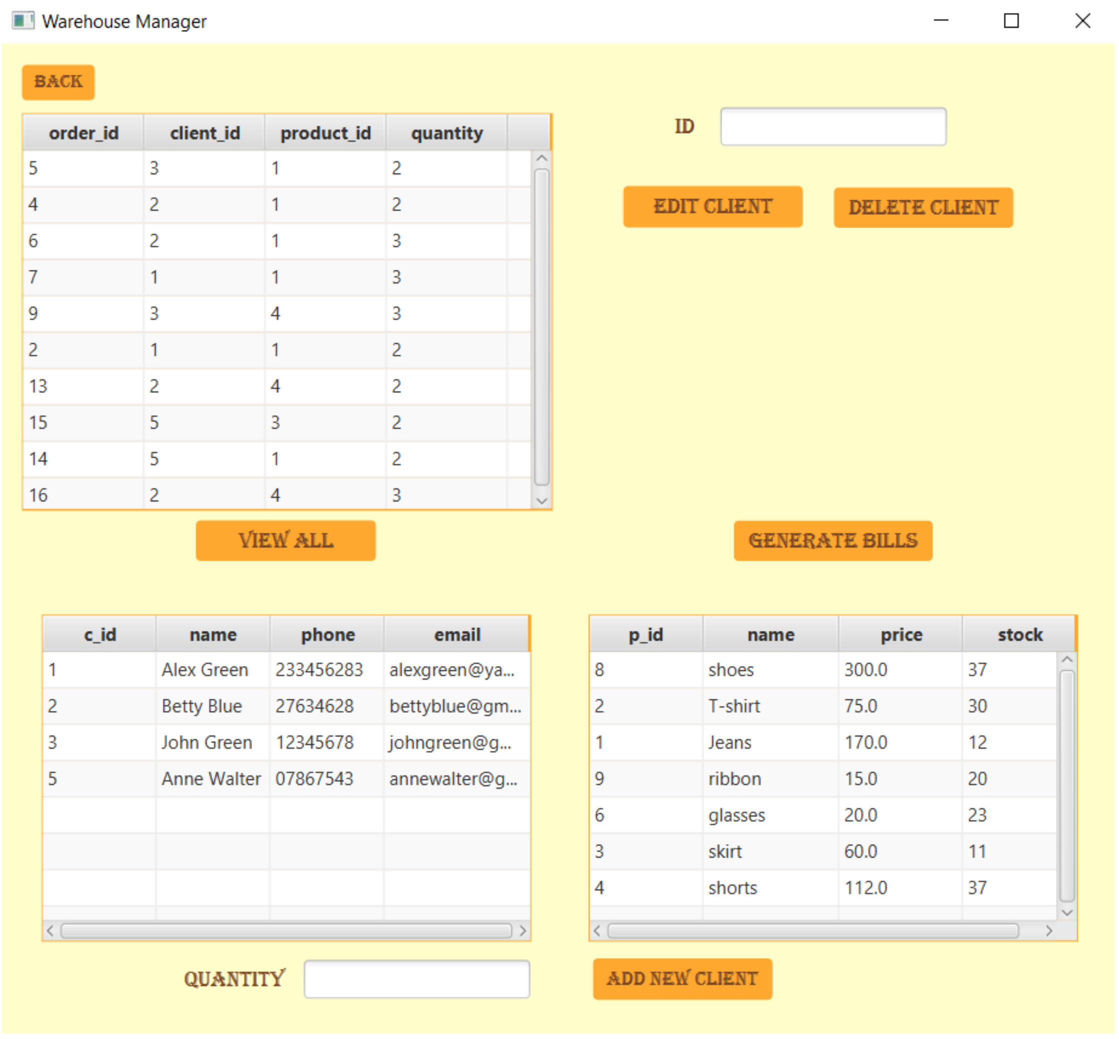
-Client Page- -Product Page-

Product Page:

- Similar to the "Client Page" but for managing product data.
- Displays all existing products from the database in a table view.
- Allows users to insert new products by entering data into text fields.
- Provides functionality to edit existing product information.
- Supports deleting products from the database.

Order Page:

- Displays three tables: one for viewing all orders, and two for viewing existing clients and products respectively.
- Enables users to place a new order by selecting a client from the client table, a product from the product table, and entering the quantity.
- Supports editing and deleting orders from the database.



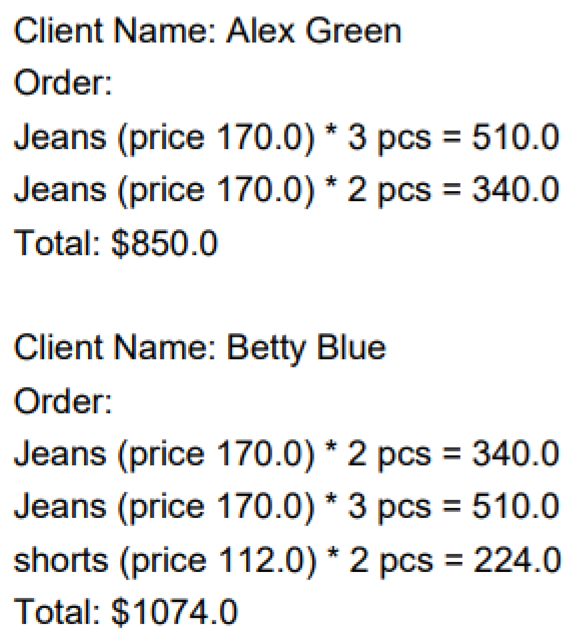
order_id	client_id	product_id	quantity
5	3	1	2
4	2	1	2
6	2	1	3
7	1	1	3
9	3	4	3
2	1	1	2
13	2	4	2
15	5	3	2
14	5	1	2
16	2	4	3

c_id	name	phone	email
1	Alex Green	233456283	alexgreen@ya...
2	Betty Blue	27634628	bettyblue@gm...
3	John Green	12345678	johngreen@g...
5	Anne Walter	07867543	annewalter@g...

p_id	name	price	stock
8	shoes	300.0	37
2	T-shirt	75.0	30
1	Jeans	170.0	12
9	ribbon	15.0	20
6	glasses	20.0	23
3	skirt	60.0	11
4	shorts	112.0	37

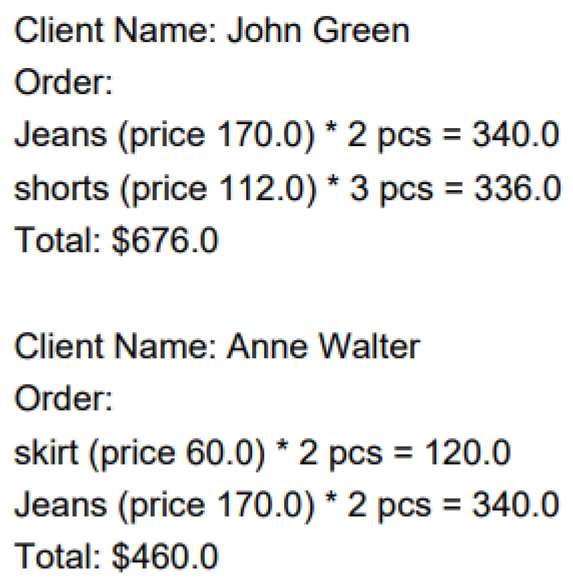
5. Results

In my project I implemented a class named Bill, responsible for generating the bill for each client.



```
Client Name: Alex Green
Order:
Jeans (price 170.0) * 3 pcs = 510.0
Jeans (price 170.0) * 2 pcs = 340.0
Total: $850.0

Client Name: Betty Blue
Order:
Jeans (price 170.0) * 2 pcs = 340.0
Jeans (price 170.0) * 3 pcs = 510.0
shorts (price 112.0) * 2 pcs = 224.0
Total: $1074.0
```



```
Client Name: John Green
Order:
Jeans (price 170.0) * 2 pcs = 340.0
shorts (price 112.0) * 3 pcs = 336.0
Total: $676.0

Client Name: Anne Walter
Order:
skirt (price 60.0) * 2 pcs = 120.0
Jeans (price 170.0) * 2 pcs = 340.0
Total: $460.0
```

Bill Generation Logic:

The Bill class encapsulates the logic for creating bills in PDF format.
It interacts with the client's order data to gather the necessary information for generating the bill, such as the items purchased, their prices, and quantities.
The class likely contains methods to retrieve order details from the database, such as client information, product details, quantities, and prices.

PDF Generation:

The class utilizes a PDF generation library, such as iTextPDF, to create the PDF document.
It sets up the document structure, including page size, margins, and formatting.
The bill contents, including client details, items purchased, prices, quantities, and total amount, are added to the PDF document.

Client and Order Information:

The Bill class retrieves client information from the database, such as client name, address, and contact details, to include on the bill.
It also fetches order details associated with the client, such as the list of items purchased, their prices, and quantities.

PDF Formatting:

The bill layout is structured in a clear and organized manner, typically with sections for client information, order details, and a summary section.
Itemized lists of purchased items along with their respective prices and quantities are included.
Calculations for subtotal, taxes (if applicable), and the total amount are performed and displayed on the bill.

6. Conclusions

Reflecting on the warehouse management project, I've identified several areas for improvement and opportunities for further skill enhancement. This project has been invaluable in reinforcing my knowledge and skills in software development, highlighting both my strengths and areas for growth.

- Javadoc Documentation: Throughout the project, I've gained proficiency in using Javadoc to generate comprehensive documentation for the codebase. Documenting classes, methods, and variables using Javadoc tags such as @param, @return, and @throws has improved code readability and maintainability, facilitating collaboration and understanding among team members.

- Abstract Classes: By incorporating abstract classes in the project's architecture, I've learned to define common behaviors and attributes shared among multiple subclasses. Abstract classes provide a blueprint for concrete classes to implement, promoting code reusability and adhering to the DRY (Don't Repeat Yourself) principle.

- PDF Generation: Implementing PDF generation functionality has been essential for generating bills and reports in the warehouse management system. Leveraging libraries like iTextPDF, I've developed modules to dynamically create PDF documents containing client orders, product details, and billing information. This feature enhances the system's usability by providing users with printable and shareable documents.

- Layered Architecture: Adopting a layered architecture, such as the Model-View-Controller (MVC) pattern, has been instrumental in organizing the project's codebase into distinct layers: presentation, business logic, data access, and model. This architectural approach promotes separation of concerns, modularity, and scalability, enabling easier maintenance and future enhancements.

By leveraging Javadoc documentation, abstract classes, PDF generation, and layered architecture effectively, the warehouse management system demonstrates robustness, scalability, and maintainability. This project has expanded my skill set and deepened my understanding of software development principles, preparing me for more complex and challenging projects in the future.

Future Developments:

- Performance Optimization: Enhance the efficiency of inventory management algorithms to handle larger inventories, increasing transaction throughput and reducing response times.

- Visualization Enhancements: Implement graphical representations of warehouse layouts, stock levels, and order fulfillment processes to provide users with intuitive insights into warehouse operations and inventory status.

- Advanced Inventory Strategies: Introduce advanced inventory management strategies such as demand forecasting, dynamic stock allocation, and automated replenishment to optimize inventory levels and minimize stockouts or excess inventory.

- Integration with External Systems: Enable integration with external systems such as suppliers, logistics providers, and e-commerce platforms to streamline order processing, shipment tracking, and inventory synchronization, ensuring seamless end-to-end operations.

- Automated Alerts and Notifications: Introduce automated alerting mechanisms to notify warehouse managers of inventory shortages, delivery delays, or critical events such as equipment malfunctions or temperature deviations, enabling timely intervention and proactive problem resolution.

- User Authentication and Access Control: Implement user authentication mechanisms and access control features to ensure secure access to the warehouse management system, with role-based permissions for warehouse staff, administrators, and suppliers, enhancing data security and compliance.

7. Bibliography

8. *Teach Yourself Java in 21 days – Laura Lemay : https://www.cs.cmu.edu/afs/cs.cmu.edu/user/gchen/www/download/java/LearnJava.pdf.*
9. *Connecting Postgres with Java using JDBC - https://tembo.io/docs/getting-started/postgres_guides/connecting-to-postgres-with-java*
10. *Layered Architectures - https://dzone.com/articles/layers-standard-enterprise*
11. *Reflection in Java - http://tutorials.jenkov.com/java-reflection/index.html*
12. *Creating PDF files in JAVA- https://www.baeldung.com/java-pdf-creation*
13. *JAVADOC - https://www.baeldung.com/javadoc*
14. *Lectures*