DOCUMENTATION

Order Management System

STUDENT NAME: Risteiu Ioana-Stefania
GROUP: 30421

CONTENTS

1. Assignment Objective

1. Main Objective**:**
    - Design and implement an application for managing client orders for a warehouse.

2. Sub-objectives:

Sub-Objective	Description
Analyze the problem and identify requirements	Understand the needs of the business and how orders are currently managed. Gather requirements from stakeholders to define what the software solution should accomplish.
Design the order management system	Create a blueprint for the software solution, detailing its structure, features, and user interface. Determine how data will be stored and accessed and outline the functionality of each component.
Implement the order management system	Develop the software solution based on the design specifications. Write the code to create the necessary functionality, including features for adding, editing, and processing orders.
Test the order management system	Validate the functionality and performance of the software solution through thorough testing. Conduct various tests to ensure that it operates correctly under different conditions and scenarios.

2. Problem Analysis, Modeling, Scenarios, Use Cases

1. Functional Requirements:
    - The application should allow a user to add a new client.
    - The application should allow a user to add a new product.
    - The application should allow a user to update client information.
    - The application should allow a user to update product information.
    - The application should allow a user to place an order for a client.
    - The application should allow a user to view all clients.
    - The application should allow a user to view all products.

2. Non-Functional Requirements:
    - The application should have a user-friendly interface, with intuitive navigation and clear instructions.
    - The application should be responsive and provide quick feedback to user actions.
    - The application should be reliable, with minimal downtime and robust error handling.
    - The application should be compatible with various devices and browsers, ensuring accessibility for users with different preferences.
    - The application should have efficient performance, with fast loading times and minimal latency in data retrieval and processing.
    - The application should comply with relevant regulations and standards for data privacy and security.
    - The application should have adequate documentation, including user manuals and technical guides, to assist users and developers.

3. Use Case Description:

Use Case: add client **Primary Actor**: Employee **Main Success Scenario:** 1. The employee selects the option to add a new client 2. The application will display a form in which the client details should be inserted 3. The employee inserts the name of the client, email, address and age 4. The employee clicks on the "Submit" button 5. The application stores the client data in the database and	**Use Case**: add product **Primary Actor**: Employee **Main Success Scenario:** 1. The employee selects the option to add a new product 2. The application will display a form in which the product details should be inserted 3. The employee inserts the name of the product, details, its price and current stock 4. The employee clicks on the "Submit" button 5. The application stores the product data in the database and

displays an acknowledge message. **Alternative Sequence:** Invalid values for the client's data → The user inserts an invalid value for the age or an email with a wrong format → The application displays an error message and requests the user to insert a valid stock → The scenario returns to step 3	displays an acknowledge message. **Alternative Sequence:** Invalid values for the product's data → The user inserts a negative value for the stock of the product → The application displays an error message and requests the user to insert a valid stock → The scenario returns to step 3
Use Case: edit client **Primary Actor**: Employee **Main Success Scenario:** 1. The employee selects the option to edit a client 2. The application will display a view in which the employee can choose a client 3. The employee clicks on the "Submit" button 4. The application updates the client data in the database and displays an acknowledge message.	**Use Case**: edit product **Primary Actor**: Employee **Main Success Scenario:** 1. The employee selects the option to edit a product 2. The application will display a view in which the employee can choose a product 3. The employee clicks on the "Submit" button 4. The application updates the client data in the database and displays an acknowledge message.
Use Case: delete client **Primary Actor**: Employee **Main Success Scenario:** 1. The employee selects the option to delete a client 2. The application will display a view in which the employee can choose a client 3. The employee clicks on the "Submit" button 4. The application deletes the client data in the database and displays an acknowledge message.	**Use Case**: delete product **Primary Actor**: Employee **Main Success Scenario:** 1. The employee selects the option to delete a product 2. The application will display a view in which the employee can choose a product 3. The employee clicks on the "Submit" button 4. The application deletes the client data in the database and displays an acknowledge message.
Use Case: view all clients **Primary Actor**: Employee **Main Success Scenario:** 1. The employee selects the option to view all clients 2. The application will create a view in which all the clients will be displayed in a table 3. The employee clicks on the "Back" button to go back to the option view	**Use Case**: view all products **Primary Actor**: Employee **Main Success Scenario:** 1. The employee selects the option to view all products 2. The application will display a view in which the employee can choose a product 3. The employee clicks on the "Back" button to go back to the option view
Use Case: create an order **Primary Actor**: Employee **Main Success Scenario:** 1. The employee selects the option to create an order 2. The application will display a view in which the employee can choose a client 3. The employee clicks on the "Submit" button 4. The application will display a view in which the employee can choose a client 5. The employee clicks on the "Submit" button 6. The application will display a view in which the employee can choose the quantity wanted 7. The employee clicks on the "Submit" button 8. The application creates an order data in the database and displays an acknowledge message.	

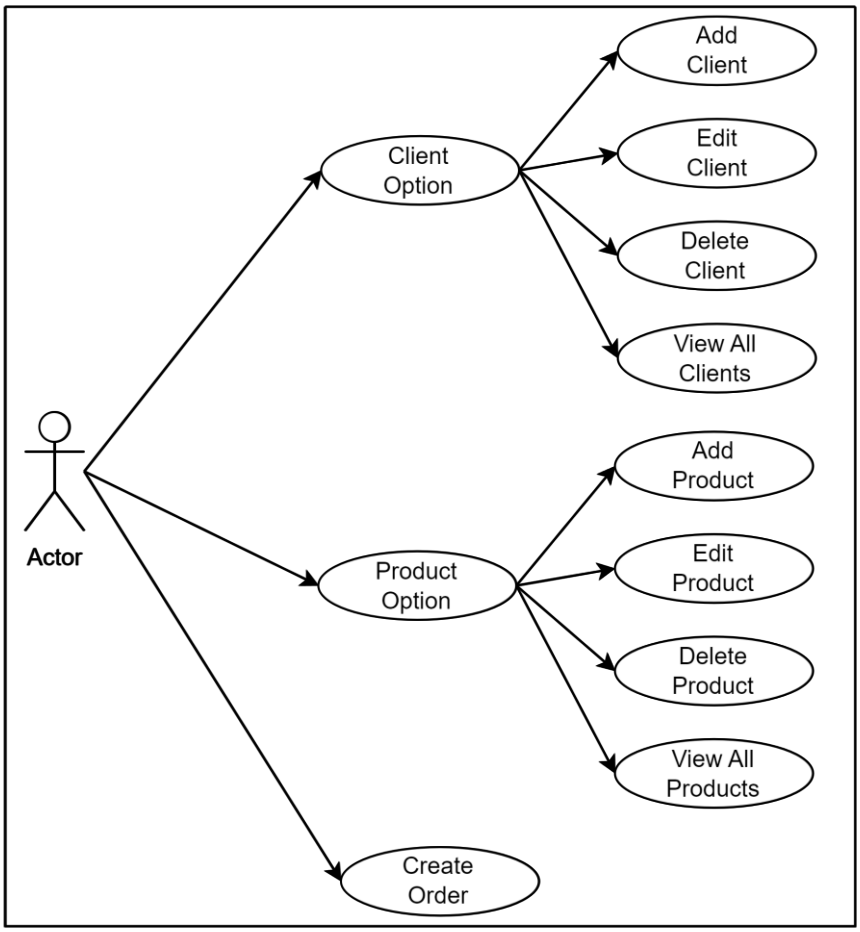


Fig 1. Use Case Diagram

3. Design

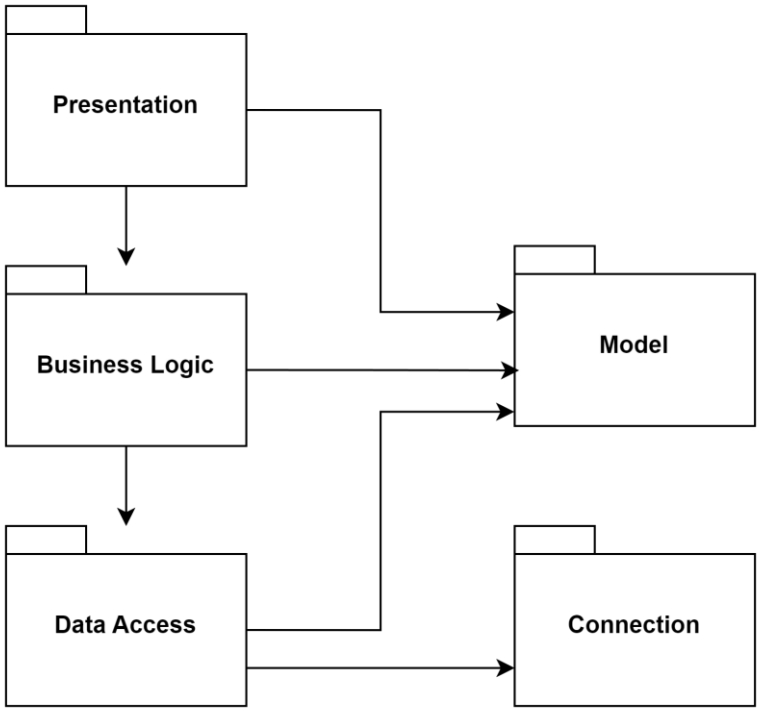
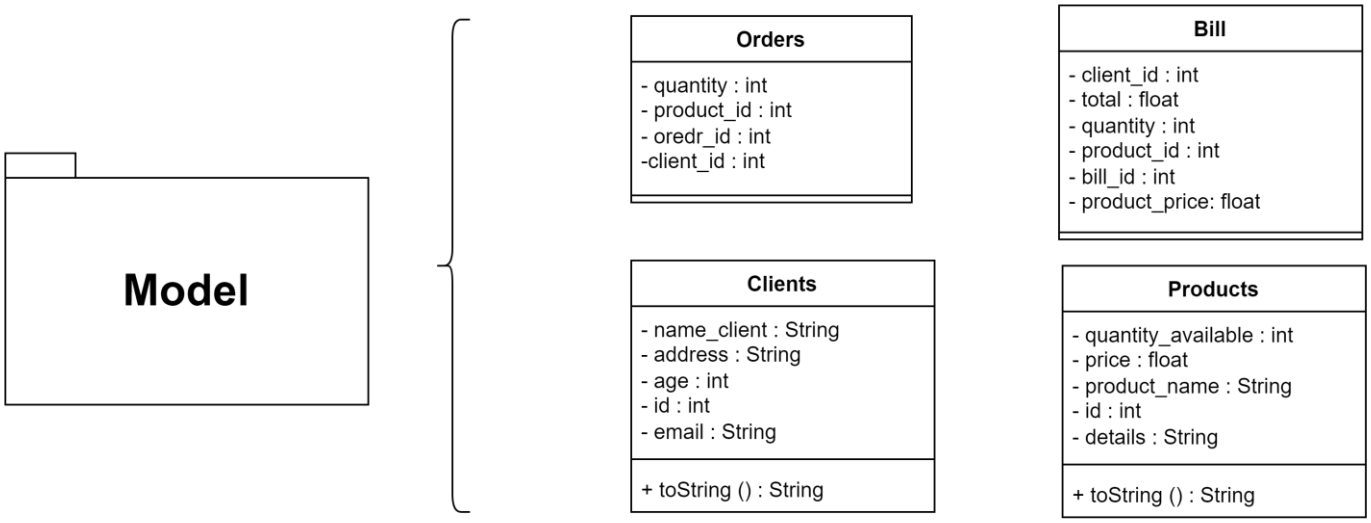
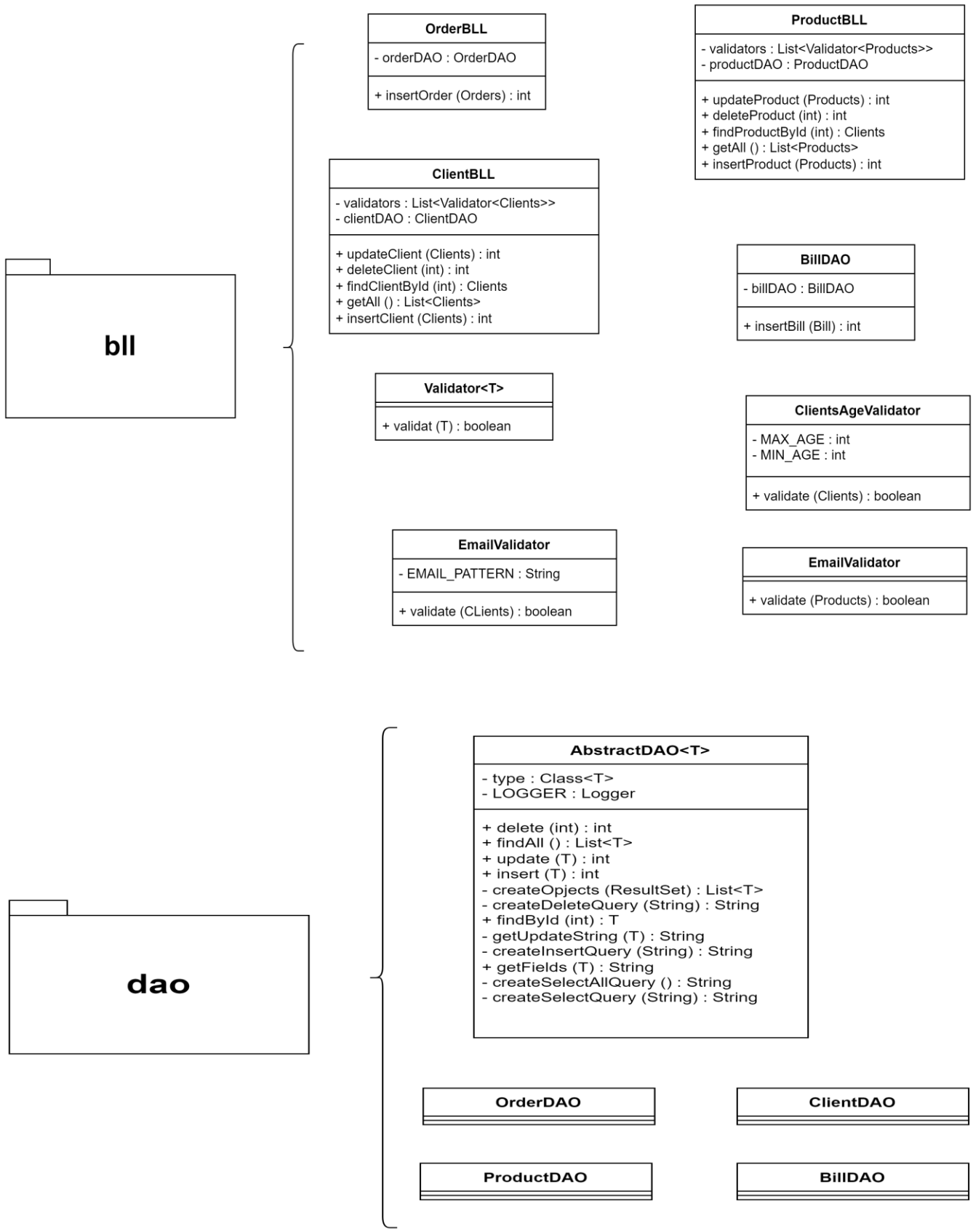


Fig 2. Package Diagram

Division Into Packages




bll
OrderBLL
- orderDAO : OrderDAO
+ insertOrder (Orders) : int
ProductBLL
- validators : List<Validator<Products>>
- productDAO : ProductDAO
+ updateProduct (Products) : int
+ deleteProduct (int) : int
+ findProductById (int) : Clients
+ getAll () : List<Products>
+ insertProduct (Products) : int
ClientBLL
- validators : List<Validator<Clients>>
- clientDAO : ClientDAO
+ updateClient (Clients) : int
+ deleteClient (int) : int
+ findClientById (int) : Clients
+ getAll () : List<Clients>
+ insertClient (Clients) : int
BillDAO
- billDAO : BillDAO
+ insertBill (Bill) : int
Validator<T>
+ validat (T) : boolean
ClientsAgeValidator
- MAX_AGE : int
- MIN_AGE : int
+ validate (Clients) : boolean
EmailValidator
- EMAIL_PATTERN : String
+ validate (CLients) : boolean
EmailValidator
+ validate (Products) : boolean
dao
AbstractDAO<T>
- type : Class<T>
- LOGGER : Logger
+ delete (int) : int
+ findAll () : List<T>
+ update (T) : int
+ insert (T) : int
- createOpjects (ResultSet) : List<T>
- createDeleteQuery (String) : String
+ findById (int) : T
- getUpdateString (T) : String
- createInsertQuery (String) : String
+ getFields (T) : String
- createSelectAllQuery () : String
- createSelectQuery (String) : String
OrderDAO
ClientDAO
ProductDAO
BillDAO

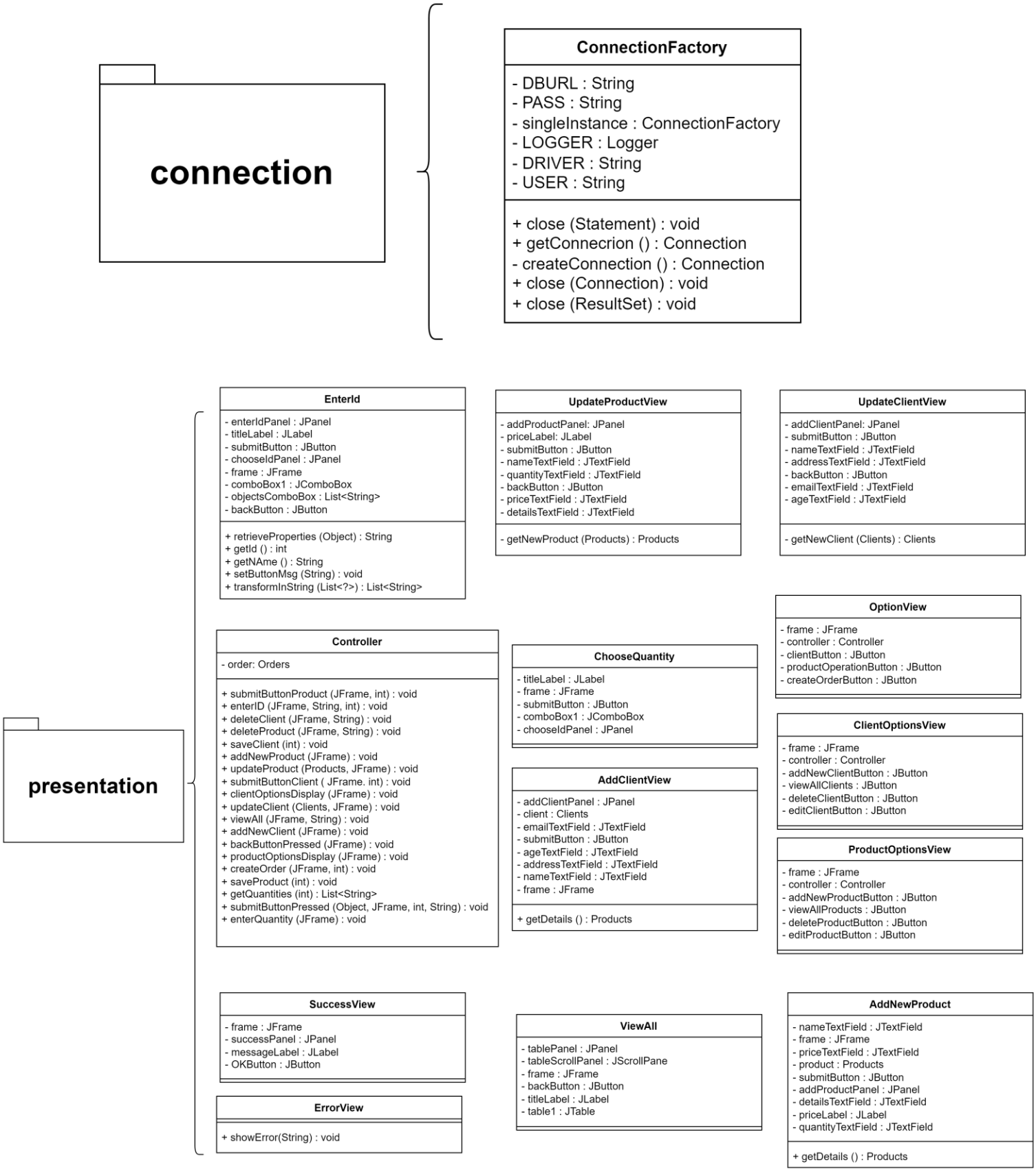


Fig 3. Division Into Packages

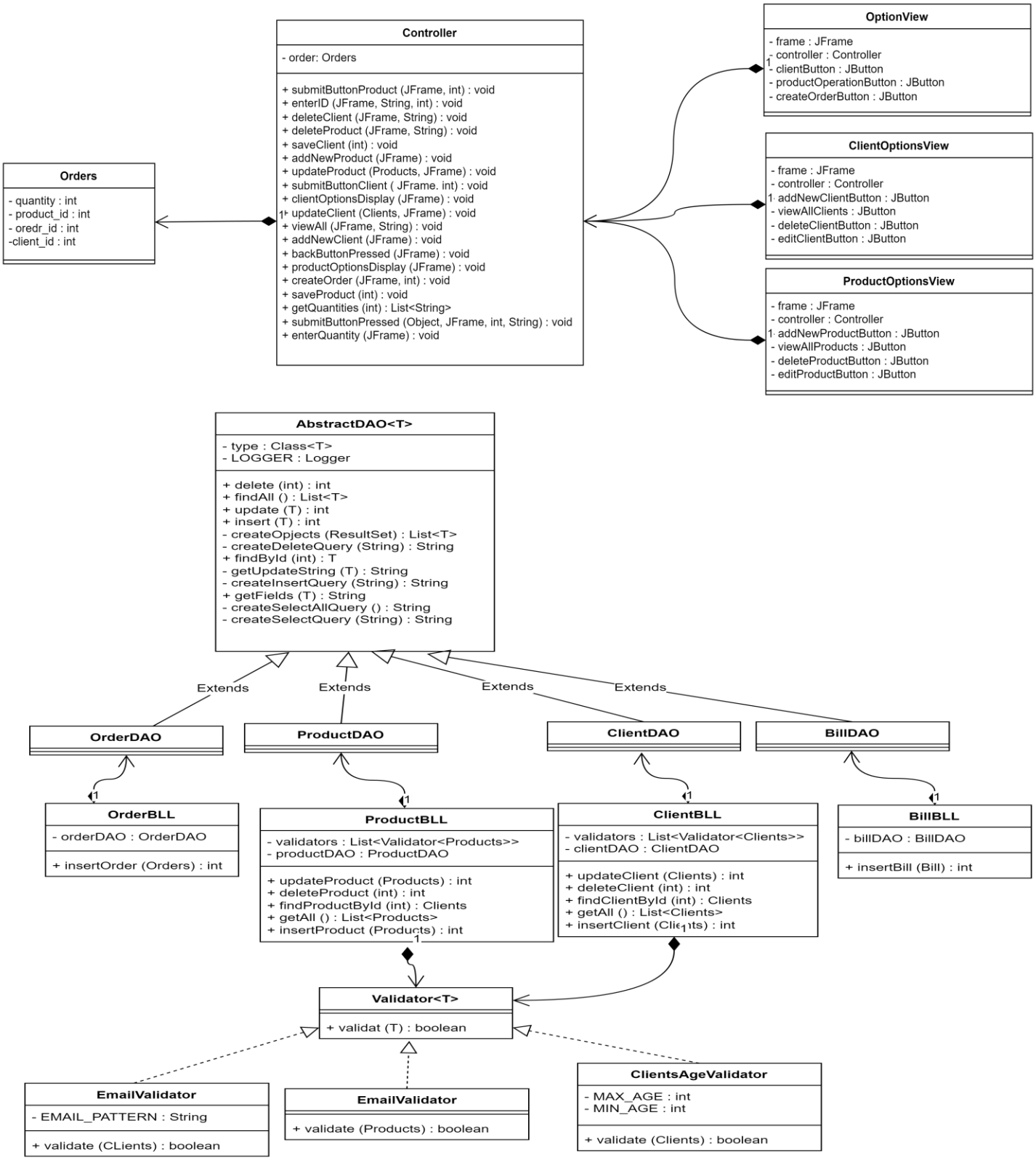


Fig 4. Class Diagram

4. Implementation

Clients Class

→ Fields:
- *id* - The unique identifier for the client.
- *name_client*; - The name of the client.
- *address*; - The address of the client.
- *email*; - The email address of the client.
- *age*; - The age of the client.

Products Class

→ Fields:
- *id*; - The unique identifier for the product.
- *product*_name; - The name of the product.
- *details*; - The details of the product.
- *quantity_available*; - The quantity available of the product.
- *price*; - The price of the product.

Orders Class

→ Fields
- *order_id*; - Represents the unique identifier for the order.
- *product_id*; - Represents the identifier of the product that is ordered.
- *client_id*; - Represents the identifier of the client who made the order.
- *quantity*; - Represents the quantity of the product that has been ordered.

Bill Record

The Bill record represents a bill entity that encapsulates information about a specific bill. It is an immutable data structure that includes various attributes related to billing details.

→ Fields
- *bill_id*; - Represents the unique identifier for the bill.
- *client_id*; - Represents the identifier of the client associated with the bill.
- *product_id*; - Represents the identifier of the product included in the bill.
- *product_price*; - Represents the price of the product at the time of billing.
- *quantity*; - Represents the quantity of the product billed.
- *total*; - Represents the total amount for the bill, calculated as product_price multiplied by quantity.

AbstractDAO Class

The AbstractDAO<T> class is a generic Data Access Object (DAO) that provides basic CRUD (Create, Read, Update, Delete) operations using reflection and generics to map Java objects to database tables. This class simplifies database interactions by automatically generating SQL queries based on the entity's structure.

→ Fields
- *LOGGER* : A logger for logging messages.
- *type* : The class type of the entity T being managed by this DAO.

→ Private Methods
- *createSelectQuery*(String field) : Creates a SQL SELECT query string for the specified field. Returns the SQL SELECT query string.
- *createSelectAllQuery*() : Creates a SQL SELECT query string to select all records. Returns the SQL SELECT query string.
- *createInsertQuery*(String fields) : Creates a SQL INSERT query string with the specified fields. Returns the SQL INSERT query string.
- *createUpdateQuery*(String values) : Creates a SQL UPDATE query string with the specified values. Returns the SQL UPDATE query string.
- *createDeleteQuery*() : Creates a SQL DELETE query string to delete a record by ID. Returns the SQL DELETE query string.

- *findById*(int id) : Finds an entity by its ID.
- *findAll*() : Finds all entities.
- *insert*(T t) : Inserts a new entity into the database.
- *update*(T t) : Updates an existing entity in the database.
- *delete*(int id) : Deletes an entity by its ID.

- *createObjects*(ResultSet resultSet) : Creates a list of entities from the given ResultSet.
- getUpdateString(T t) : Constructs a SQL UPDATE query string with placeholders for the entity's fields, except for the first field, which is typically the ID.
- getFields(T t) : Constructs a SQL INSERT query string with placeholders for the entity's fields, except for the first field, which is typically the ID.

ProductBLL Class

The ProductBLL class is part of the Business Logic Layer (BLL) and is responsible for handling operations related to products. This class encapsulates the business logic for managing products and interacts with the ProductDAO to perform CRUD operations.

→ Fields
- *List<Validator<Products>> validators* : A list of validators for Products objects.
- *productDAO :* An instance of ProductDAO.

→ Important Methods
- *findProductById(int id)* : Finds a product by its ID.
- *getAll ()* : Retrieves all products from the database.
- *insertProduct(Products)* : Inserts a new product into the database.
- *updateProduct(Products)* : Updates an existing product in the database.
- *deleteProduct (int id)*: Deletes a product from the database by its ID.

ClientBLL Class

The ClientBLL class is part of the Business Logic Layer (BLL) and is responsible for handling operations related to clients. This class encapsulates the business logic for managing clients and interacts with the ClientDAO to perform CRUD operations.

→ Fields
- *List<Validator<Clients>> validators* : A list of validators for Clients objects.
- *clientDAO :* An instance of ClientDAO.

→ Important Methods
- *findProductById(int id)* : Finds a client by its ID.
- *getAll ()* : Retrieves all clients from the database.
- *insertClient(Clients)* : Inserts a new client into the database.
- *updateClient(Clients)* : Updates an existing client in the database.
- *deleteClient (int id)*: Deletes a client from the database by its ID.

OrderBLL Class

The OrderBLL class is part of the Business Logic Layer (BLL) and is responsible for handling operations related to orders. This class encapsulates the business logic for managing orders and interacts with the OrderDAO to perform CRUD operations.

→ Fields
- *orderDAO :* An instance of OrderDAO.

→ Important Methods
- *insertOrder(Order)* : Inserts a new order into the database.

BillBLL Class

The BillBLL class is part of the Business Logic Layer (BLL) and is responsible for handling operations related to bills. This class encapsulates the business logic for managing orders and interacts with the BillDAO to perform CRUD operations.

→ Fields
- *billDAO :* An instance of BillDAO.

→ Important Methods
- *insertBill(Bill)* : Inserts a new bill into the database.

Reflection Class

The Reflection class is a utility class designed for performing reflection operations on objects. It provides methods to retrieve property names, details (values), and convert objects to their string representations.

→ Important Methods
- *retrieveProperties (Object object)* : Retrieves the names of properties (fields) of an object.
- *getDetails (Object object)* : Retrieves the details (values) of properties of an object.

- *getAllDetails(Object object)* : Retrieves details of multiple objects as a 2D array.
- *toString (Object object)* : Converts an object to its string representation.

Controller Class

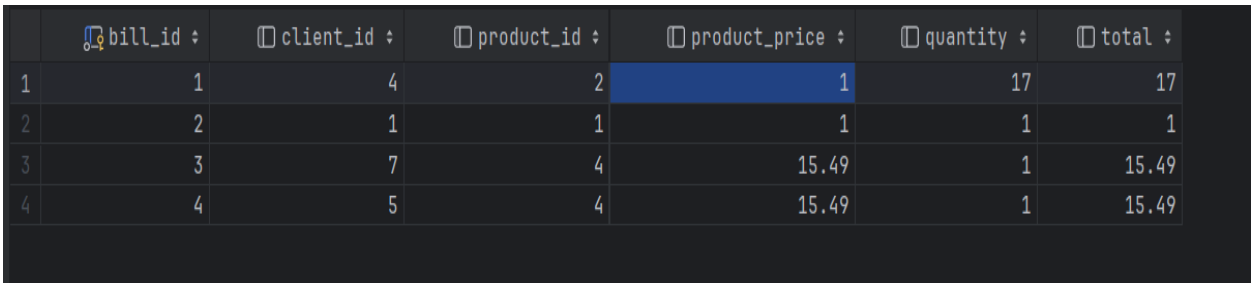
The Controller class serves as the central component that manages user interactions and coordinates actions between the presentation layer (views) and the business logic layer (BLL). It contains methods for handling various user actions, such as displaying views, submitting data, updating information, and creating orders.

→ Important Methods

- *clientOptionsDisplay* : Displays the client options view.
- *addNewClient* : Displays the add new client view.
- *addNewProduct* : Displays the add new product view.
- *submitButtonPressed* : Handles the submit button pressed event for adding clients or products.
- *enterID* : Displays the EnterId view based on the message and cases provided.
- *submitButtonClient* : Retrieves a client by ID and displays the UpdateClientView if found.
- *updateClient* : Updates a client's information and displays a success message if successful.
- *updateProduct* : Updates a product's information and displays a success message if successful.
- *deleteClient* : Displays the EnterId view for deleting a client or a product.
- *deleteButtonPressed* : Handles the deletion of a client or a product based on the object type.
- *backButtonPressed* : Displays the option view.
- *viewAll* : Displays a view to show all clients or all products.
- *productOptionsDisply* : Displays the product options view.
- *deleteProduct* : Displays the EnterId view for deleting a product.
- *submitButtonProduct* : Retrieves a product by ID and displays the UpdateProductView if found.
- *saveClient* : Saves the client ID for the current order being processed.
- *saveProduct* : Saves the product ID for the current order being processed.
- *enterQuantity* : Displays a view to enter the quantity of a product.
- *createOrder* : Creates an order for the selected product and displays a success message if successful.

5. Results

After clients and products are added to the database, the employee can create an order by selecting a client, a product, and the desired quantity. Upon pressing the submit button, a new entry will be added to the Orders table in the database, and the bill will be automatically generated.



	bill_id	client_id	product_id	product_price	quantity	total
1	1	4	2	1	17	17
2	2	1	1	1	1	1
3	3	7	4	15.49	1	15.49
4	4	5	4	15.49	1	15.49

Fig 5. Bill Table

6. Conclusions

In this assignment, I have successfully designed and implemented a system that integrates client and product management with order processing and automatic bill generation. This project has provided valuable insights into the following areas:

1. **Database Interaction**: This project has deepened my understanding of database operations, including SQL querying and JDBC integration. By mastering these fundamentals, I am better equipped to handle database interactions in future projects.

2. **Modular Structure**: The implementation of distinct business logic layers, such as ClientBLL, ProductBLL, and OrderBLL, has proven effective in organizing code and separating concerns. This modular approach enhances maintainability and scalability.

3. **Reflection Usage**: Exploring reflection techniques has been enlightening. Leveraging reflection to dynamically retrieve object properties has streamlined code and minimized redundancy. This dynamic behavior enhances flexibility and adaptability.

4. **Immutable Classes**: Embracing immutable classes has underscored the importance of data integrity and thread safety. By ensuring that objects remain unmodifiable once created, I have mitigated potential bugs and improved code reliability.

7. Bibliography

1. https://www.baeldung.com/java-jdbc
2. http://www.mkyong.com/jdbc/how-to-connect-to-mysql-with-jdbc-driver-java/
3. https://dzone.com/articles/layers-standard-enterprise
4. http://tutorials.jenkov.com/java-reflection/index.html
5. https://www.baeldung.com/java-pdf-creation
6. https://www.baeldung.com/javadoc
7. https://dev.mysql.com/doc/workbench/en/wb-admin-export-import-management.html
8. https://www.digitalocean.com/community/tutorials/how-to-create-immutable-class-in-java
9. https://www.geeksforgeeks.org/reflection-in-java/
10. https://www.geeksforgeeks.org/generic-class-in-java/