DOCUMENTATION

ASSIGNMENT 3

STUDENT NAME: Muntian Mariana-Ionela
GROUP: 30423

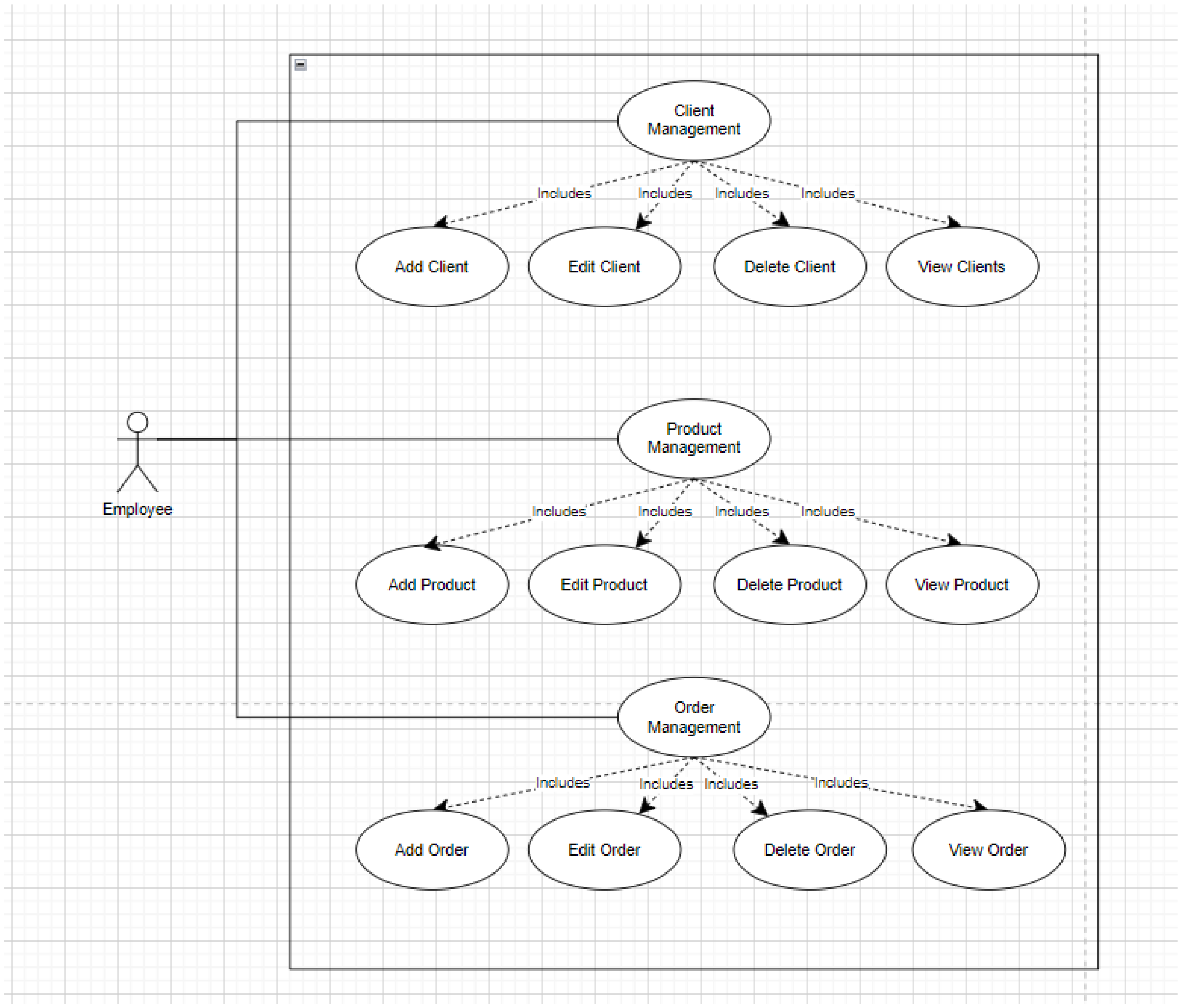
CONTENTS

1. Assignment Objective

Design and implement an application for managing the client orders for a warehouse, using reflection techniques and following a standard layered architecture.

Sub-objective	***Description***	***Section***
1. Analyze the problem and identify requirements	*The problem is analyzed from the point of view of the use cases and the functional requirements.*	*2. Problem Analysis, Modeling, Scenarios, Use Cases*
2. Design the orders management application	*UML packages, UML diagram, the data structures and the algorithms that are used are presented.*	*3. Design*
3. Implement the simulation application	*The implementation of the interface and the structure of each class are described.*	*4. Implementation*
4. Test the simulation application	*The testing results are checked by analyzing the interface and the database.*	

2. Problem Analysis, Modeling, Scenarios, Use Cases



Reference: [Lecture_2_UML_Diagrams.pdf]()

Use Case 1: Client Management	
Brief Description	*This use case involves choosing the option: insert/edit/delete/show.*
Parent Scenario	-
Actor(s)	*The User*
Priority	*High*
Trigger	*Login*
Pre-conditions	-
Post-conditions	*Start Simulation*
Basic Flow	*Step 1: Press Client Button* *Step 2: Choose one option: add/edit/delete/show* *Step 3: Press Save Button*
Exception Flow(s)	*Step 1: If an error occurs during the setup process, handle the error accordingly, changing the parameters accordingly.* *Step 2: Review the configured parameters.* *Step 3: Press Save again.*

Use Case 2: Porduct Management	
Brief Description	*This use case involves choosing the option: insert/edit/delete/show.*
Parent Scenario	*-*
Actor(s)	*The User*
Priority	*High*
Trigger	*-*
Pre-conditions	*-*
Post-conditions	*Start Simulation*
Basic Flow	*Step 1: Press Product Button* *Step 2: Choose one option: add/edit/delete/show* *Step 3: Press Save Button*
Exception Flow(s)	*Step 1: If an error occurs during the setup process, handle the error accordingly, changing the parameters accordingly.* *Step 2: Review the configured parameters.* *Step 3: Press Save again.*

Use Case 3: Order Management	
Brief Description	*This use case involves choosing the option: insert/edit/delete/show.*
Parent Scenario	*-*
Actor(s)	*The User*
Priority	*High*
Trigger	*-*
Pre-conditions	*-*
Post-conditions	*Start Simulation*
Basic Flow	*Step 1: Press Orders Button* *Step 2: Choose one option: add/edit/delete/show* *Step 3: Press Save Button*
Exception Flow(s)	*Step 1: If an error occurs during the setup process, handle the error accordingly, changing the parameters accordingly.* *Step 2: Review the configured parameters.* *Step 3: Press Save again.*

Use Case 4: Add Client	
Brief Description	*This use case involves choosing the data about the client: name, age, email.*
Parent Scenario	*Client Management*
Actor(s)	*The User*
Priority	*High*
Trigger	*Client Management*
Pre-conditions	*-*
Post-conditions	*Press Save*
Basic Flow	*Step 1: Press insert Button* *Step 2: Insert the data: name, age, email.* *Step 3: Press Save Button*
Exception Flow(s)	*Step 1: If an error occurs during the setup process, handle the error accordingly, changing the parameters accordingly.* *Step 2: Review the configured parameters.* *Step 3: Press Save again.*

Use Case 5: Edit Client	
Brief Description	*This use case involves choosing the data about the client: name, age, email, id. The data are modified based on the id.*

Parent Scenario	*Client Management*
Actor(s)	*The User*
Priority	*High*
Trigger	*Client Management*
Pre-conditions	*-*
Post-conditions	*Press Save*
Basic Flow	*Step 1: Press edit Button* *Step 2: Insert the data: id, name, age, email.* *Step 3: Press Save Button*
Exception Flow(s)	*Step 1: If an error occurs during the setup process, handle the error accordingly, changing the parameters accordingly.* *Step 2: Review the configured parameters.* *Step 3: Press Save again.*

Use Case 6: Delete Client	
Brief Description	*This use case involves choosing the data about the client: id, name, age, email.*
Parent Scenario	*Client Management*
Actor(s)	*The User*
Priority	*High*
Trigger	*Client Management*
Pre-conditions	*-*
Post-conditions	*Press Save*
Basic Flow	*Step 1: Press Delete Button* *Step 2: Insert the data: id, name, age, email.* *Step 3: Press Save Button*
Exception Flow(s)	*Step 1: If an error occurs during the setup process, handle the error accordingly, changing the parameters accordingly.* *Step 2: Review the configured parameters.* *Step 3: Press Save again.*

Use Case 7: Show Table Client	
Brief Description	*This use case will show to the client/user all the data inserted about the clients.*
Parent Scenario	*Client Management*
Actor(s)	*The User*
Priority	*High*
Trigger	*Client Management*
Pre-conditions	*-*
Post-conditions	*Press Save*
Basic Flow	*Step 1: Press Show Table Button*
Exception Flow(s)	*-*

Functional requirements:

- *The application should allow an employee to add a new client*
- *The application should allow an employee to edit client*
- *The application should allow an employee to delete client*
- *The application should allow an employee to show a table with clients*
- *The application should allow an employee to add a new product*
- *The application should allow an employee to edit product*

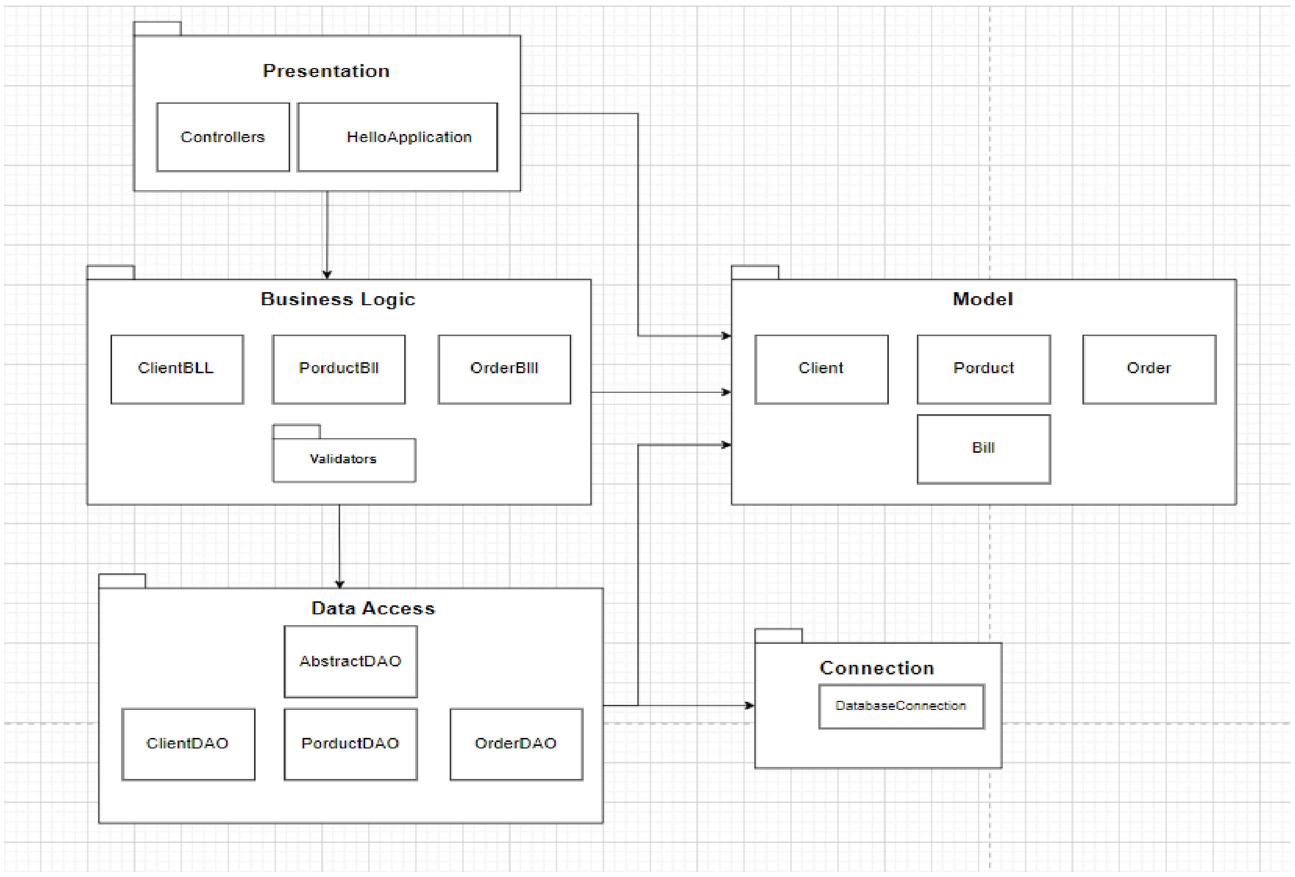
- *The application should allow an employee to delete product*
- *The application should allow an employee to show a table with product*

Non-Functional requirements:

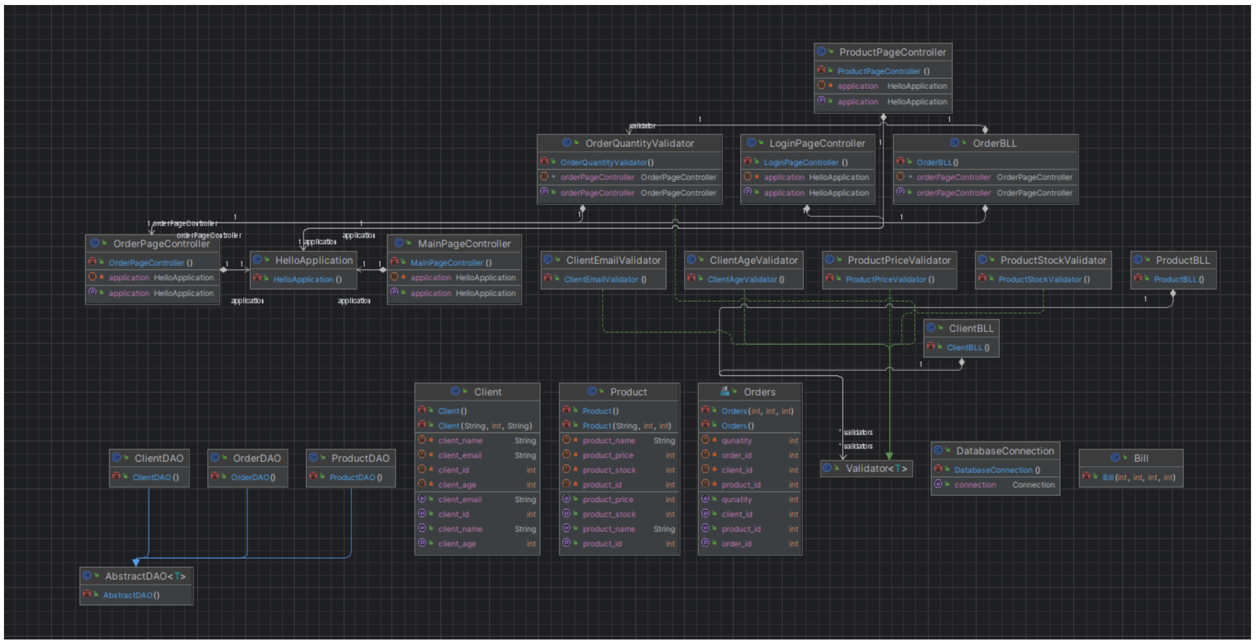
- *The simulation application should be intuitive and easy to use by the user*
- *Response times for updating the interface and dispatching tasks should be minimal.*
- *The system should be scalable to accommodate future enhancements and modifications.*

3. Design

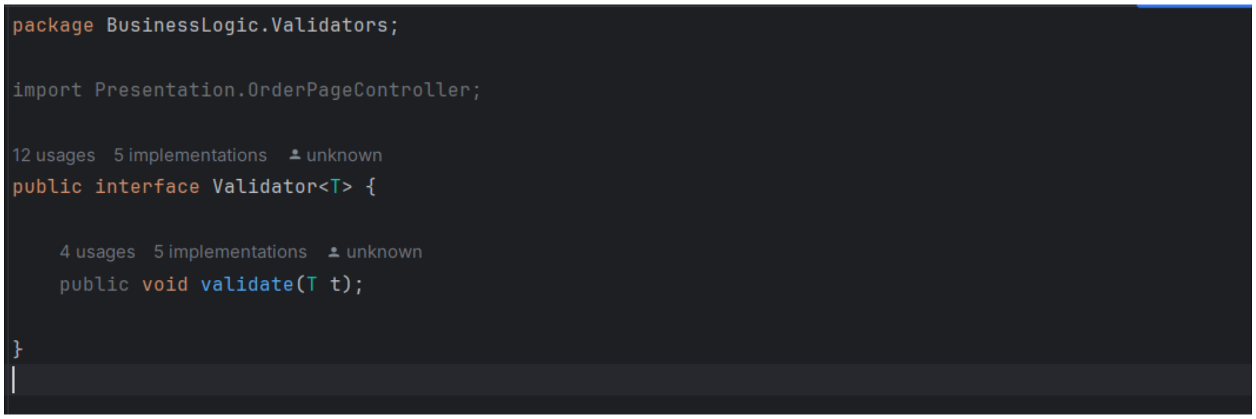
- ***UML packages***



- ***Class Diagram***



- ***Defined Interfaces***



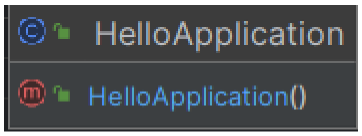
```
package BusinessLogic.Validators;

import Presentation.OrderPageController;

12 usages   5 implementations   unknown
public interface Validator<T> {

    4 usages   5 implementations   unknown
    public void validate(T t);

}
```

The Validator interface defines a contract for classes that perform validation on objects of type T. Implementing classes should provide a concrete implementation of the validate method to perform specific validation logic tailored to the type T. There are several classes that implement this interfaces, because all the information inserted by the user regarding the client, order or product bust be validated before any changes in the database are made.

4. Implementation

I. Presentation Package

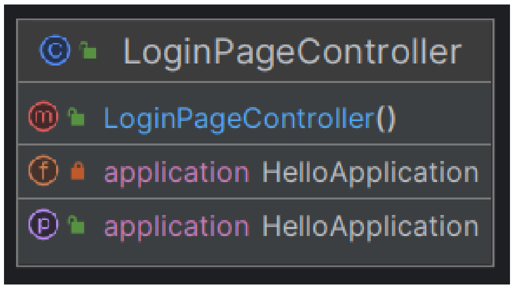
a) HelloApplication



HelloApplication

HelloApplication()

b) LoginPageController



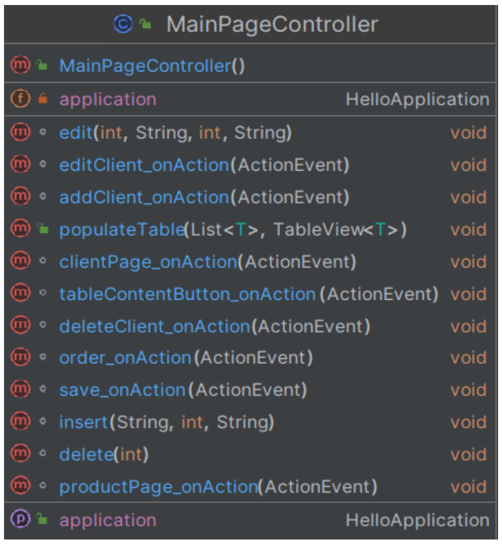
LoginPageController

LoginPageController()

application HelloApplication

application HelloApplication

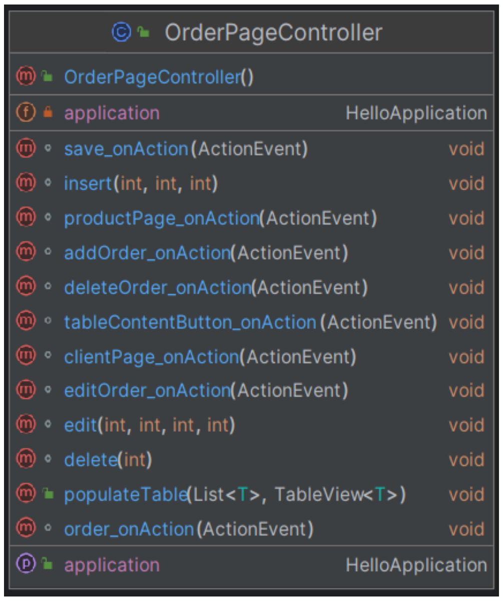
c) MainPageController



MainPageController

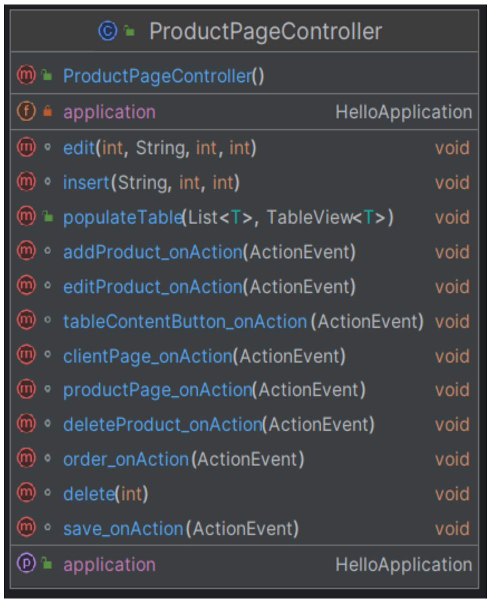
MainPageController()	
application	HelloApplication
edit(int, String, int, String)	void
editClient_onAction(ActionEvent)	void
addClient_onAction(ActionEvent)	void
populateTable(List<T>, TableView<T>)	void
clientPage_onAction(ActionEvent)	void
tableContentButton_onAction(ActionEvent)	void
deleteClient_onAction(ActionEvent)	void
order_onAction(ActionEvent)	void
save_onAction(ActionEvent)	void
insert(String, int, String)	void
delete(int)	void
productPage_onAction(ActionEvent)	void
application	HelloApplication

d) OrderPageController



```
OrderPageController
OrderPageController()
application                                HelloApplication
save_onAction(ActionEvent)                             void
insert(int, int, int)                                  void
productPage_onAction(ActionEvent)                      void
addOrder_onAction(ActionEvent)                         void
deleteOrder_onAction(ActionEvent)                      void
tableContentButton_onAction(ActionEvent)               void
clientPage_onAction(ActionEvent)                       void
editOrder_onAction(ActionEvent)                        void
edit(int, int, int, int)                               void
delete(int)                                            void
populateTable(List<T>, TableView<T>)                   void
order_onAction(ActionEvent)                            void
application                                HelloApplication
```

e) ProductPageController

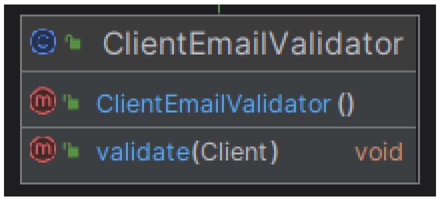


```
ProductPageController
ProductPageController()
application                                HelloApplication
edit(int, String, int, int)                            void
insert(String, int, int)                               void
populateTable(List<T>, TableView<T>)                   void
addProduct_onAction(ActionEvent)                       void
editProduct_onAction(ActionEvent)                      void
tableContentButton_onAction(ActionEvent)               void
clientPage_onAction(ActionEvent)                       void
productPage_onAction(ActionEvent)                      void
deleteProduct_onAction(ActionEvent)                    void
order_onAction(ActionEvent)                            void
delete(int)                                            void
save_onAction(ActionEvent)                             void
application                                HelloApplication
```

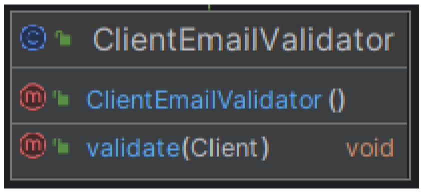
II. Business Logic Package

a) Validators Package

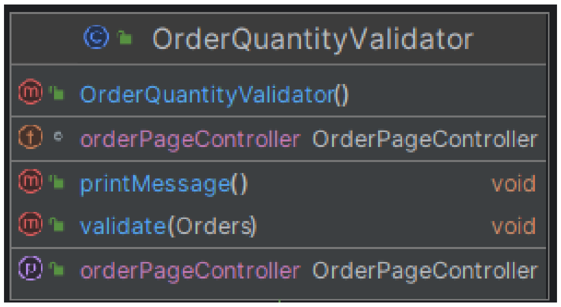
- **ClientAgeValidator**



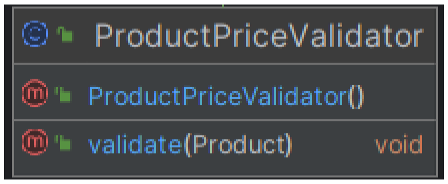
- **ClientEmailValidator**



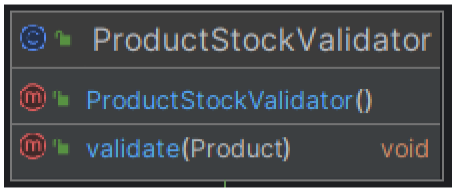
- **OrderQuantityValidator**



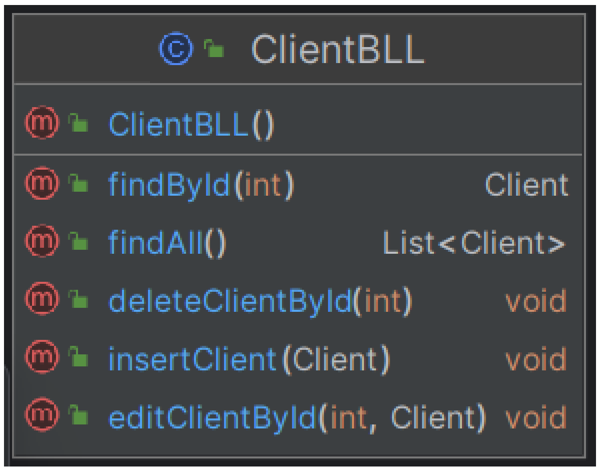
- **ProductPriceValidator**



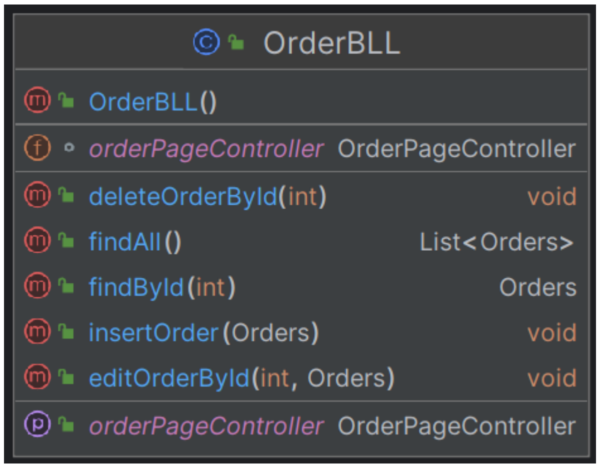
- **ProductStockValidator**



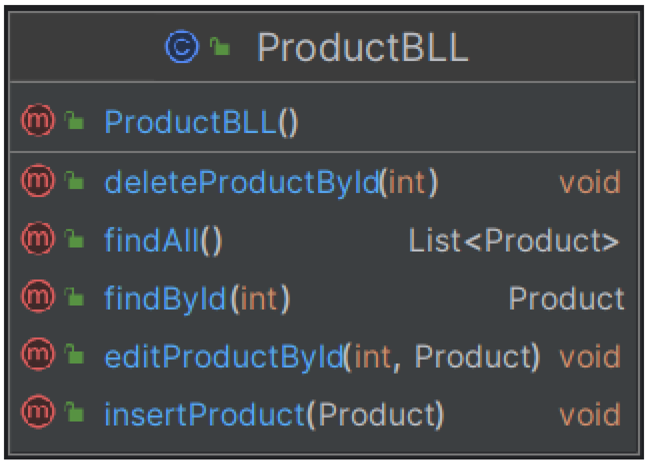
b) **ClientBLL**


ClientBLL
ClientBLL()
findById(int) Client
findAll() List<Client>
deleteClientById(int) void
insertClient(Client) void
editClientById(int, Client) void

c) **OrderBLL**

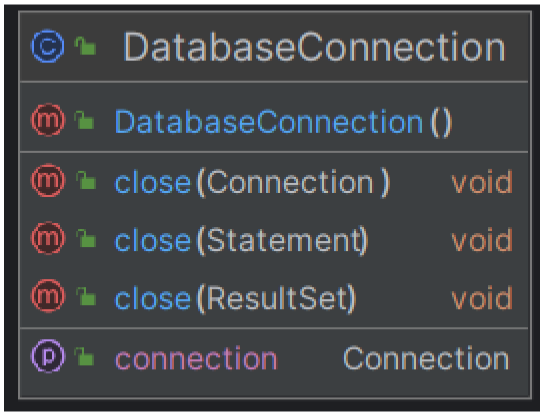

OrderBLL
OrderBLL()
orderPageController OrderPageController
deleteOrderById(int) void
findAll() List<Orders>
findById(int) Orders
insertOrder(Orders) void
editOrderById(int, Orders) void
orderPageController OrderPageController

d) **ProductBLL**


ProductBLL
ProductBLL()
deleteProductById(int) void
findAll() List<Product>
findById(int) Product
editProductById(int, Product) void
insertProduct(Product) void

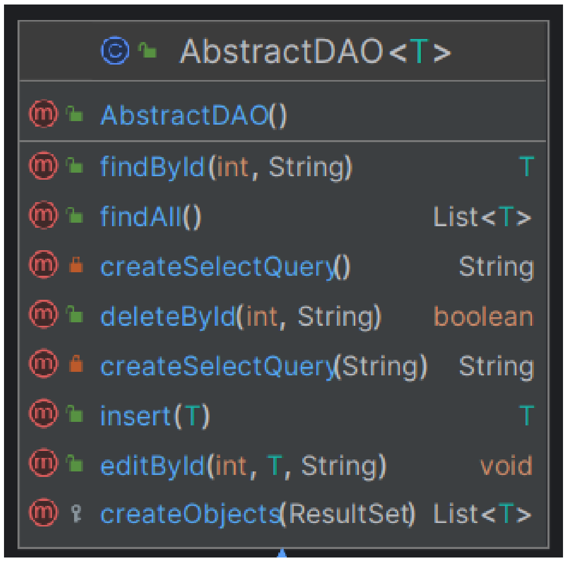
III. Connection Package

a) **DatabaseConnection**

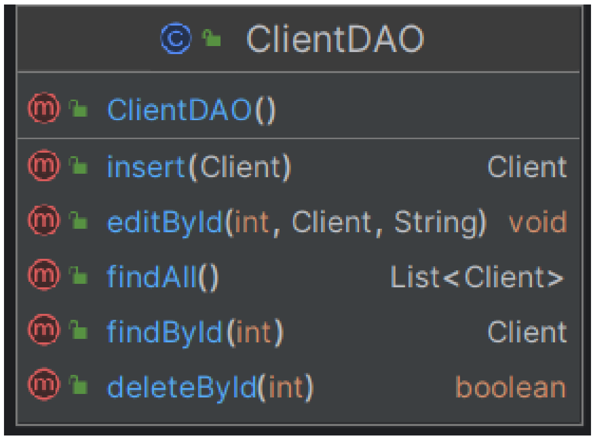

```
DatabaseConnection
DatabaseConnection()
close(Connection)        void
close(Statement)         void
close(ResultSet)         void
connection               Connection
```

IV. **Data Access Package**

a) **AbstractDAO<T>**

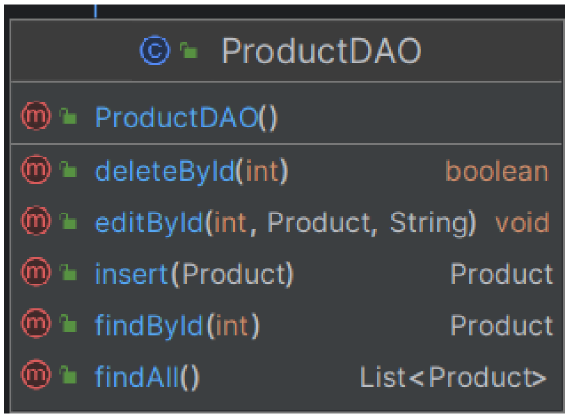

```
AbstractDAO<T>
AbstractDAO()
findById(int, String)             T
findAll()                         List<T>
createSelectQuery()               String
deleteById(int, String)           boolean
createSelectQuery(String)         String
insert(T)                         T
editById(int, T, String)          void
createObjects(ResultSet)          List<T>
```

b) **ClientDAO**



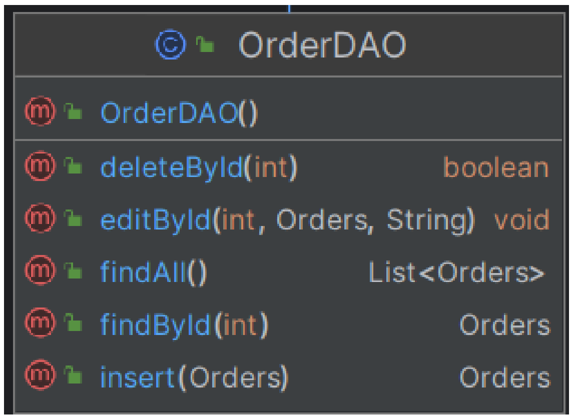
```
ClientDAO
ClientDAO()
insert(Client)                  Client
editById(int, Client, String)   void
findAll()                       List<Client>
findById(int)                   Client
deleteById(int)                 boolean
```

c) **ProductDAO**



```
ProductDAO
ProductDAO()
deleteById(int)                  boolean
editById(int, Product, String)   void
insert(Product)                  Product
findById(int)                    Product
findAll()                        List<Product>
```

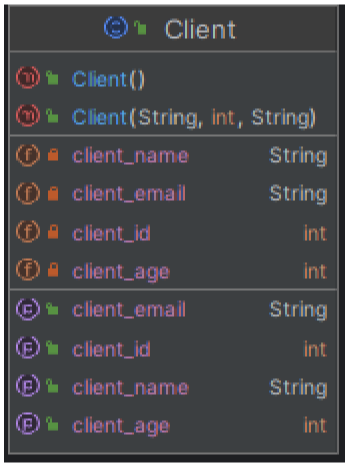
d) **OrderDAO**



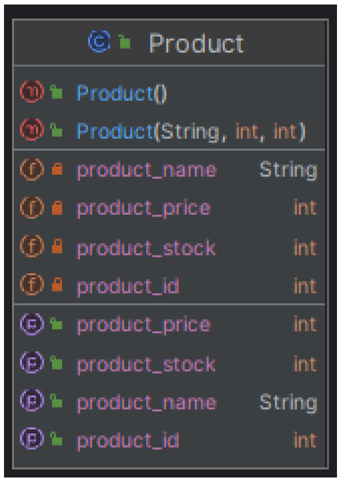
```
OrderDAO
OrderDAO()
deleteById(int)                 boolean
editById(int, Orders, String)   void
findAll()                       List<Orders>
findById(int)                   Orders
insert(Orders)                  Orders
```

V. Model Package

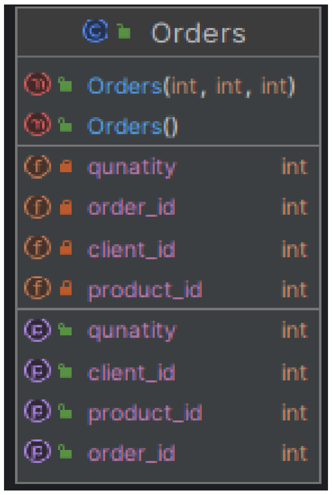
a) **Client**


```
Client
Client()
Client(String, int, String)
client_name     String
client_email    String
client_id          int
client_age         int
client_email    String
client_id          int
client_name     String
client_age         int
```

b) **Product**


```
Product
Product()
Product(String, int, int)
product_name    String
product_price      int
product_stock      int
product_id         int
product_price      int
product_stock      int
product_name    String
product_id         int
```

c) **Orders**


```
Orders
Orders(int, int, int)
Orders()
qunatity        int
order_id        int
client_id       int
product_id      int
qunatity        int
client_id       int
product_id      int
order_id        int
```

5. Conclusions

The project concludes that following the conventional architecture (with distinct modules for models, business logic, connections, and data access) demands more effort during the initial development phase. However, it significantly streamlines the process of debugging and implementing modifications later on. Additionally, employing reflection demonstrates notable efficiency gains in code reuse.

6. Bibliography

Connect to MySql from a Java application:
https://www.baeldung.com/java-jdbc
http://www.mkyong.com/jdbc/how-to-connect-to-mysql-with-jdbc-driver-java/

Layered architectures
https://dzone.com/articles/layers-standard-enterprise

Reflection in Java
http://tutorials.jenkov.com/java-reflection/index.html

Creating PDF files in Java o
https://www.baeldung.com/java-pdf-creation

JAVADOC
https://www.baeldung.com/javadoc

SQL dump file generation
https://dev.mysql.com/doc/workbench/en/wb-admin-export-import-management.html