DOCUMENTATION

**ASSIGNMENT *3***

## STUDENT NAME: Mocoi Ioan-Victor

## GROUP: 30224

**CONTENTS**

1. [Assignment Objective 3](#_bookmark0)
2. [Problem Analysis, Modeling, Scenarios, Use Cases 3](#_bookmark1)
3. [Design 3](#_bookmark2)
4. [Implementation 6](#_bookmark3)
5. [Results 8](#_bookmark4)
6. [Conclusions 8](#_bookmark5)
7. [Bibliography 8](#_bookmark6)
   1. **ASSIGNMENT OBJECTIVE**

The primary aim of this project is to develop, deploy, and evaluate an Orders Management Application by utilizing reflection mechanisms. To accomplish this goal, the following steps were undertaken:

* Problem analysis, solution modeling, and scenario building were thoroughly described in the second section.
* The application's object-oriented design, including UML package and class diagrams, were outlined in section three.
* The fourth section encompasses the implementation details of each class and significant methods within the application.
* Section five is dedicated to presenting the outcomes and test cases of the application.
* Lastly, a concise conclusion and potential future enhancements will be discussed in the final section.

# PROBLEM ANALYSIS, MODELING, SCENARIOS, USE CASES

The objective of the orders management application is to represent and handle clients, products, orders, and bills. The application enables users to add new entities, which are then stored in a persistence layer.

To model a client, the application requires the following information: client ID, age, name, address, and identity card ID. For a product, it should have a name, quantity, price, manufacturer, and an ID. A bill is represented by an ID and a total price. Lastly, an order consists of a client ID, a product ID, an amount, and a bill.

Several use scenarios exist within the application. Users can add valid clients or products, but they can also input invalid data, such as a negative quantity for a product or an invalid age for a client. In such cases, the application will display an error dialog. When inserting an order, it is considered valid only if a valid product, a valid client, and a correct amount are selected. The amount should not be negative or exceed the total quantity of the product.

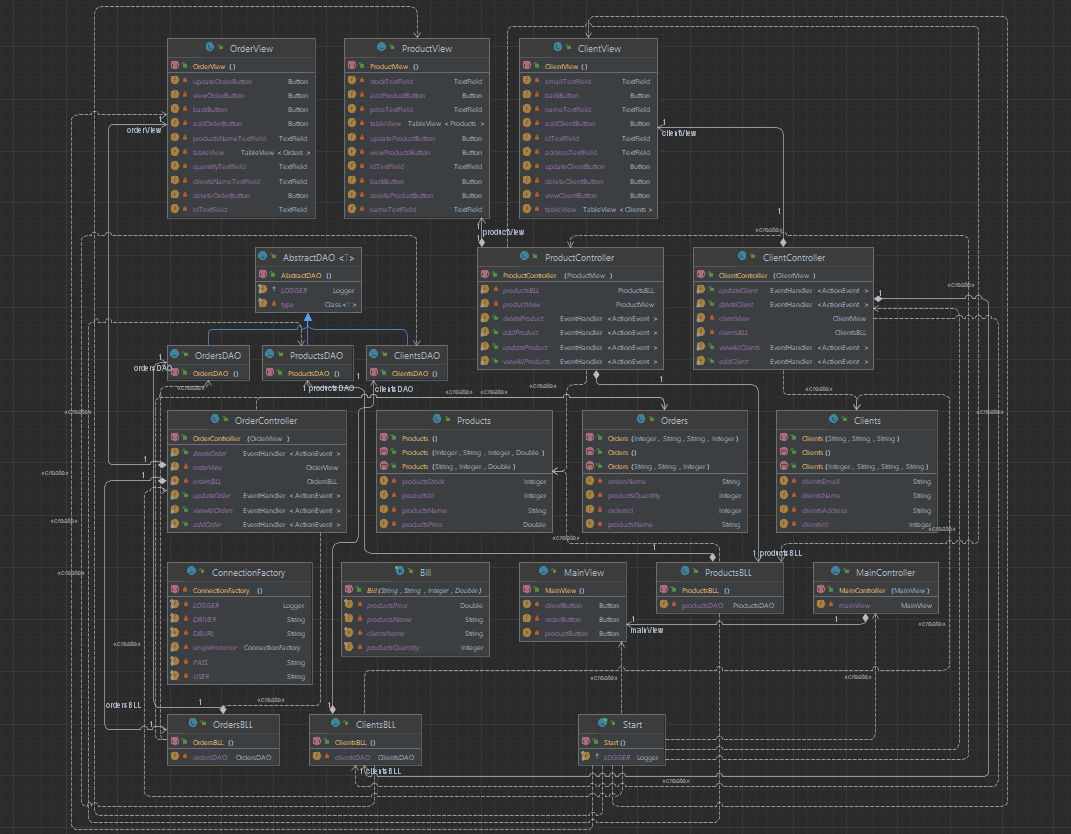
# DESIGN

The application's object-oriented design comprises five packages. The View package is responsible for implementing the graphical user interface and includes the View classes and Controller classes

The Model package is dedicated to describing the data models used in the application. The Logic package handles the implementation of the application's business logic.

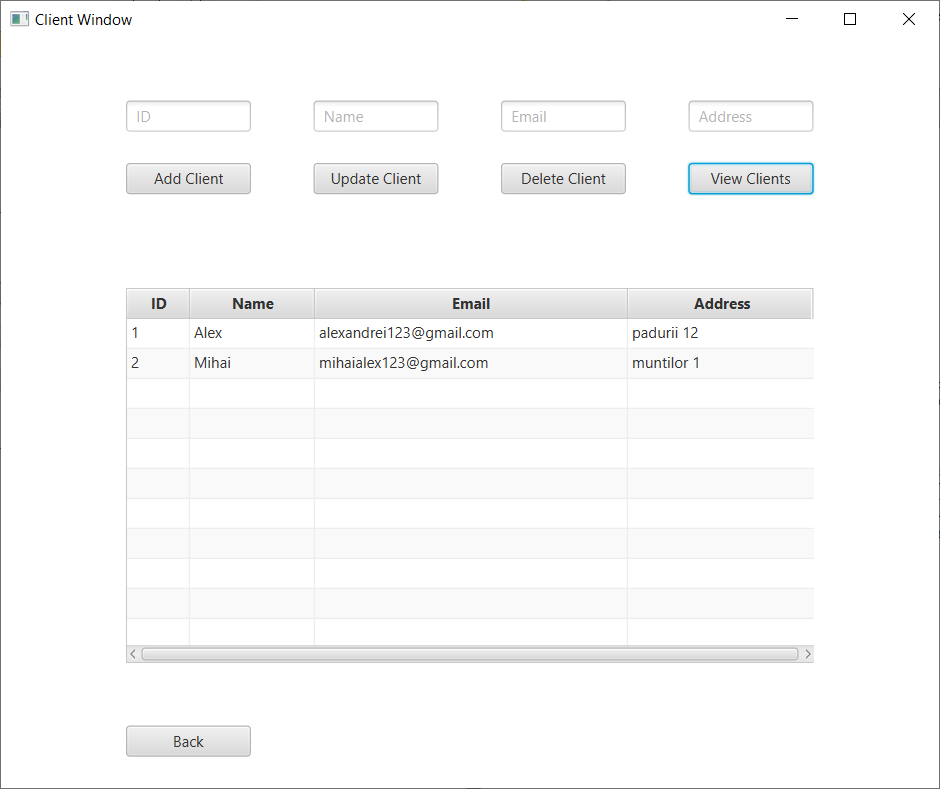
The Connection package consists of a helper class, ConnectionFactory, which facilitates the retrieval of connections with the database.

The Data Access package is responsible for implementing objects that facilitate database access.

****

UML Diagram

### 



GUI

# IMPLEMENTATION

The following classes are part of the application:

1. ConnectionFactory class:

- This helper class facilitates the retrieval and control of connections to the database.

- It implements a getter method for obtaining a connection and methods for closing the connection, statement, and result set.

2. AbstractDAO class:

- This class implements various methods for accessing the database using reflection techniques.

- It holds a type variable for the class T.

- The main methods include finding an object by its ID, retrieving all objects from a table, updating an object in the database, inserting and deleting objects into the database.

- It also implements a method for creating objects, which returns a list of objects.

3. ProductDAO class:

- This is the data access class for the Product model.

- It extends the AbstractDAO class.

4. ClientDAO class:

- This is the data access class for the Client model.

- It extends the AbstractDAO class.

5. OrderDAO class:

- This is the data access class for the Order model.

- It extends the AbstractDAO class.

6. ProductBLL class:

- This class implements the business logic for the Product model.

7. OrderBLL class:

- This class implements the business logic for the Order model.

8. ClientBLL class:

- This class implements the business logic for the Client model.

9. Bill class:

- This class models the bills.

10. Product class:

- This class models the products.

11. Clients class:

- This class models the clients.

12. Orders class:

- This class models the orders.

13. View classes and Controller classes:

- These classes implement the graphical user interface.







# RESULTS

In addition to manual testing, SQL dumps were exported to facilitate testing the application on an existing generated database. The source code also incorporates JavaDoc documentation.

# CONCLUSIONS

Conclusion:

- The application has been successfully developed and is fully functional, capable of performing all the required operations such as adding new clients, products, and placing orders.

Learned skills:

- Through this project, I gained valuable knowledge and experience in utilizing Reflection, a powerful tool for software development. It allowed me to explore new software architecture techniques and enhanced my ability to organize and structure my code effectively.

# BIBLIOGRAPHY

1. *Regex:* [Tutorials List - Javatpoint](https://www.javatpoint.com/)
2. [Stack Overflow - Where Developers Learn, Share, & Build Careers](https://stackoverflow.com/)
3. [*www.baeldung.com/java-reflection#6-implemented-interfaces*](http://www.baeldung.com/java-reflection#6-implemented-interfaces) *- Reflection*
4. [*www.baeldung.com*](http://www.baeldung.com/)