DOCUMENTATION

ASSIGNMENT 3

STUDENT NAME: GRAD LAURENȚIU-CĂLIN

GROUP: 30425

Contents

[1. Assignment Objective 3](#_Toc166583968)

[2. Problem Analysis, Modeling, Scenarios, Use Cases 3](#_Toc166583969)

[3. Design 5](#_Toc166583970)

[4. Implementation 5](#_Toc166583971)

[5. Results 5](#_Toc166583972)

[6. Conclusions 6](#_Toc166583973)

[7. Bibliography 6](#_Toc166583974)

# Assignment Objective

Consider an application Orders Management for processing client orders for a warehouse. Relational databases should be used to store the products, the clients, and the orders. 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 application logic.
* Presentation classes – GUI related classes.
* Data access classes – classes that contain access to the database.

*Note:* Other classes and packages can be added to implement the functionality of the application.

# Problem Analysis, Modeling, Scenarios, Use Cases

The assignment asks to implement an application that manages a simple warehouse. It replicates a real-life scenario: any on-line shop must keep track of what products it sells, the available quantity, the clients that regularly buy, their order, and a history of the bills that have been issued. This kind of work is predisposed to creating errors if it is handled by people because of the several things one should keep in mind. However, it is possible to simplify it if an application is designed. A graphical user interface helps even more with the management due to the intuitive interaction between it and an unspecialized user. These are the main guidelines I followed while implementing this assignment. I tried to create an intuitive graphical user interface that does not require computer science knowledge.

The assignment aims to familiarize one with the layered architecture and the reflection techniques provided by the Java programming language. In addition to this, one must work with a relational database management system to ensure data preservation. I chose PostgreSQL.

To store the data (clients, orders, bills, and products), I designed a simple relational database composed of four tables: *client*, *order*, *log*, and *product*. Each table has a primary key called ‘id’. The *order* table has two foreign keys to the *product* and *client* tables because the order is implemented as a one-to-one relation between one client and one product. For simplicity only one product can be part of one specific order, placed by one client, however, the same client can place multiple orders that request different products. This does not resemble a real-life scenario, where one order houses multiple products, but such a functionality is not requested by the text of the assignment.

The *client* relation has four fields: a *name*, a *phone number*, an *address*, and an *id* (unique identification number). These fields are not exhaustive for a client entity, but model accurately what information a shop may require for each person that wants to buy something from them (supposing the shop ships orders remotely).

The *product* relation has four fields, too: a *product name*, a *stock*, a *price per unit*, and an *id* (unique identification number). The fields store only the relevant information that one requires to know when products are inside the warehouse: how they are called, how much of each product is available to sell and what price one unit of that product costs.

The *order* relation has four fields: a *client id*, a *product id*, a *quantity*, and an *id* (unique identification number). The first two fields are linked to the *client* and *product* tables by means of foreign keys. For each order that is placed, an entry in the *log* table isgenerated, that the assignment calls *bill*. The *log* is linked to the *id* field in the order table (also, by means of a foreign key) and it stores the amount of money the order is worth. The order stores the quantity the client wants to buy of a certain product and when the order is placed, the price of the order is computed and, together with the *id* of the related order, stored in the *log* table.

The database used while developing this project is stored in the same directory as this file in the form of an SQL dump file.

The application is developed and designed to run in a Java Integrated Development Environment such as IntelliJ Idea, Eclipse IDE etc. Once the project is run a pop-up window will appear which is called the *Start Panel* (housed inside the *Main Frame* of the application). Four buttons redirect the user to the specific view that executes the create, read, update, and delete operations on the database. The product, the client and the log views have the same structure: on the left side there are buttons that open the input panels for the CRUD operations. In the right part of the window there is the place where input data is inserted, if necessary. In the lower part of the window the *BACK* button redirects the user to the starting view, while the *EXECUTE* button executes the SQL statement. If the data does not comply with the expected format, the execution of the statements fail, and warnings are displayed in the console. If the execution is successful an info message is displayed in the console, as well. The order view is somewhat special because the panel where the order can be created contains two selectors that let the user choose only between existing clients and products. However, a quantity input field is provided. All input fields must respect hard-coded regex patterns.

*NOTE: At any time, the application can be closed if the close button (X) in the top right corner is pressed.*

The user diagram describes the dependencies between the user’s interactions and the system represented by the Warehouse Management System Application. The user can manage the database as described in the above paragraphs.

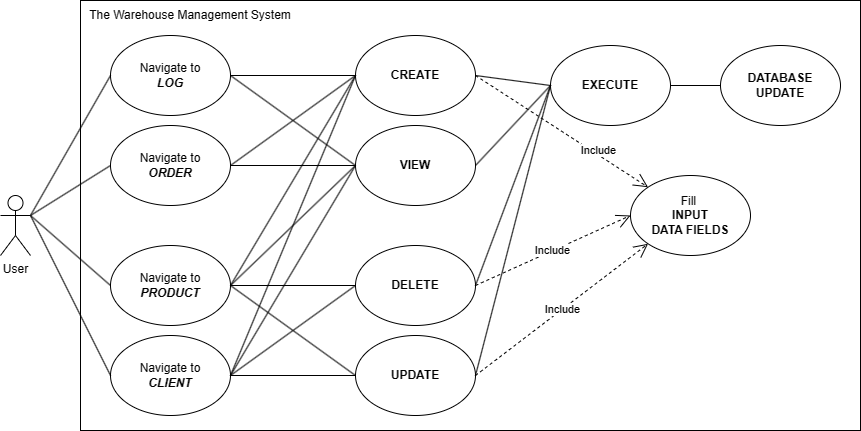


Figure 1. Use case diagram.

# Design

A diagram of a package

Description automatically generated

Figure 2. Package diagram.

# Implementation

The implementation of the classes and of the main methods is discussed in this section.

# Results

# Conclusions

# Bibliography

1. Programming Techniques lectures, laboratories, and support materials
2. Ștefan Tanasă, Ștefan Andrei, Cristian Olaru, Java: de la 0 la expert (2nd edition), Publisher: Polirom, Iași, Romania, ISBN: 978-973-46-2405-8, Published: 2011
3. Paul Deitel, Harvey Deitel, Java How to program (10th edition), Publisher: Pearson Education, Inc., Upper Saddle River, NJ, United States, ISBN: 978-0-13-380780-6, Published: 2015
4. [Use Case Diagrams | Unified Modeling Language (UML) - GeeksforGeeks](https://www.geeksforgeeks.org/use-case-diagram/)
5. [Use Case Diagram Tutorial (Guide with Examples) | Creately](https://creately.com/guides/use-case-diagram-tutorial/)
6. [Unified Modeling Language - Wikipedia](https://en.wikipedia.org/wiki/Unified_Modeling_Language)
7. [Introduction to JDBC | Baeldung](https://www.baeldung.com/java-jdbc)
8. [Layers of a Standard Enterprise Application - DZone](https://dzone.com/articles/layers-standard-enterprise)
9. [Creating PDF Files in Java | Baeldung](https://www.baeldung.com/java-pdf-creation)
10. [Introduction to JavaDoc | Baeldung](https://www.baeldung.com/javadoc)
11. [Java 14 Record Keyword | Baeldung](https://www.baeldung.com/java-record-keyword)
12. [PostgreSQL | IntelliJ IDEA Documentation (jetbrains.com)](https://www.jetbrains.com/help/idea/postgresql.html#reference)
13. [Setting up the JDBC Driver | pgJDBC (postgresql.org)](https://jdbc.postgresql.org/documentation/setup/)
14. [Guide to Java Reflection | Baeldung](https://www.baeldung.com/java-reflection)
15. [What are Java Records and How to Use them Alongside Constructors and Methods? - GeeksforGeeks](https://www.geeksforgeeks.org/what-are-java-records-and-how-to-use-them-alongside-constructors-and-methods/)