TECHNICAL UNIVERSITY
OF CLUJ-NAPOCA

Orders Management Application

Assignment 3

- Documentation -

Student: Alexandra Ilovan
Group: 30422
Laboratory Professor: Andreea-Valeria Vesa

TABLE OF CONTENTS

1. Assignment Objectives.....3
2. Problem Analysis. Modeling. Scenarios and Use Cases.....4
3. Design.....6
4. Implementation.....9
5. Results.....11
6. Conclusions.....13
7. Bibliography.....14

1. Assignment Objectives

1.1. Main Objective

The main objective of the given assignment is to correctly design and implement an application for managing the client orders for a warehouse. The application uses relational databases in order to store data concerning the clients of the warehouse, the available products, and the orders that are placed by the clients.

The order management is done depending on the product stock and the quantity of products ordered by the client, an order being able to be created only when the warehouse has enough products in stock.

1.2. Secondary Objectives

Among the secondary objectives of the assignment are the following:

- Creating a class responsible with managing the database connection (opening/ closing the connection, the statements, the result set);
- Using relational databases for storing the data;
- Generating a database in MySQL, creating its tables and establishing the needed relationships among the tables. A minimum of three tables is required for the logistics of the application to be met, those being: a Clients table, a Products table and an Orders table;
- The definition of classes as real world transposed objects (clients, products, orders);
- Establishing the classes' attributes in such a way that they mimic the concepts from reality;
- The definition of object validating rules, so that an object will be inserted into the database only under the condition that its attributes are in correlation with said rules;
- The creation and utilization of data access classes (which are part of the Data Access Layer);
- The definition of an abstract class that provides data access through the usage of Reflection Techniques;
- Using JavaDoc files for documenting the classes;
- Getting accustomed to using a Layered Architecture;
- Providing an intuitive graphical user interface (GUI) through which the user can easily interact with the database. The GUI has to include windows that facilitate the management of clients, products and orders individually.

2. Problem Analysis. Modeling. Scenarios and Use Cases

2.1. Problem Analysis & Modeling

Stating the problem: Managing the clients, the products and the orders for a warehouse using hand-written registries is difficult and time consuming.

In real life, a warehouse's role is to deposit merchandise in huge amounts, which can be ordered by one or more clients. In the context of this application, the warehouse is represented by a relational database in which all the clients, products and orders provided and managed by the user (represented by an employee of the warehouse) will be included.

2.2. Scenarios and Use Cases

Upon starting the application, the user will be met with a small Menu Window, where he/ she is provided with 4 options: either to open the Clients Management Window, open the Products Management Window, open the Orders Management Window, or exit the application.

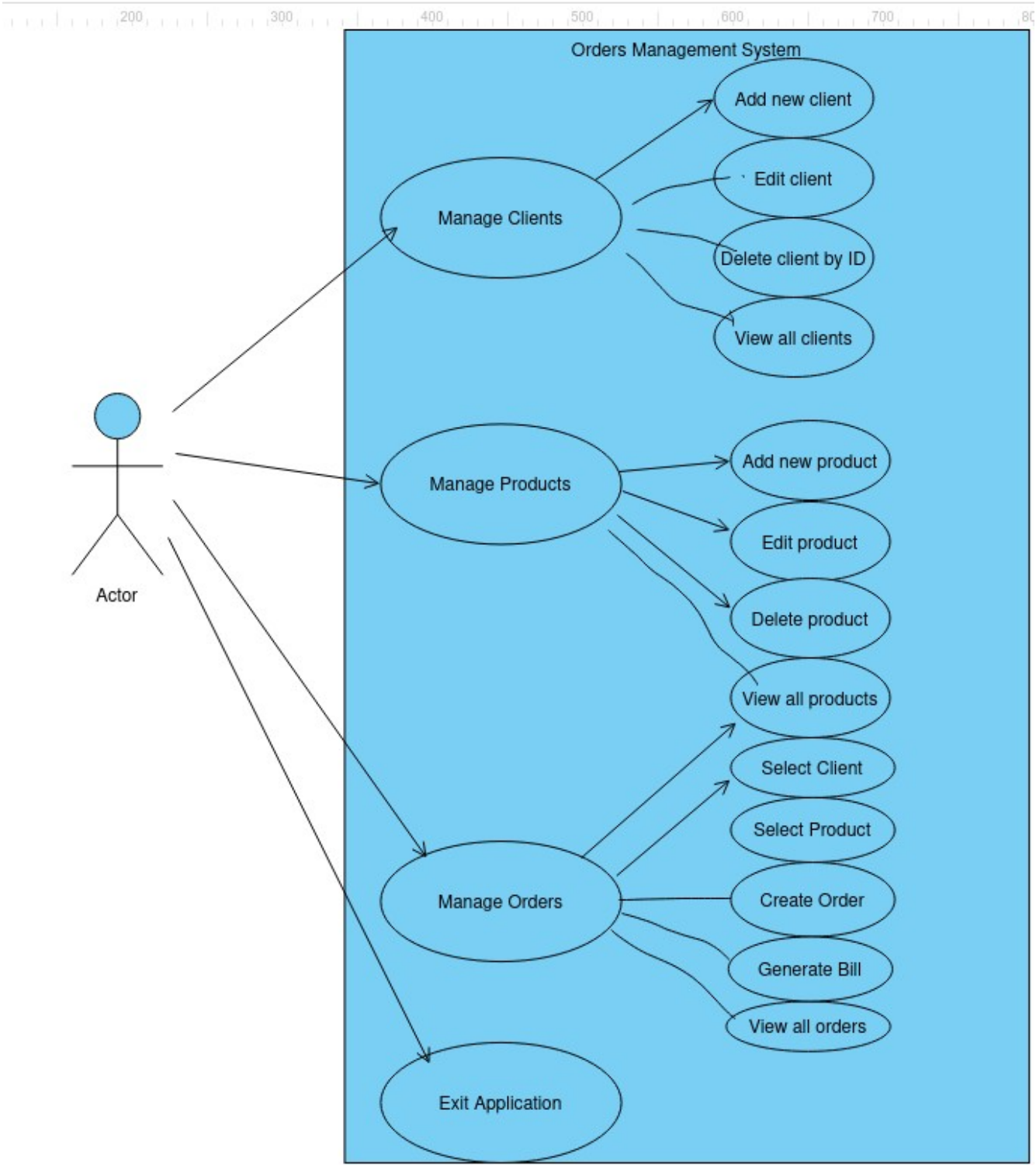
If the user chooses to press the "Manage Clients" button, a new window destined for clients management will open. The newly opened window will give the user the option to either add a new client to the database, to edit an existing user, to remove a user from the database at a specificied ID, or to view all the existing clients from the database (in which case, all the data will be displayed in a Jtable).

If the user presses the "Manage Producs" button, a similar outcome will occur. The user will be able to add a new product to the database, to edit a product, to delete a product or to display all the existing products.

However, if the user chooses "Manage Orders", upon the opening of the Order Management Window, the user will be able to select an existing client and an existing product from two different drop-down lists, enter the desired product quantity, and if the warehouses has a big enough stock of the selected product, then an order can be created.

Otherwise, an understock warning message will be displayed and no order will be placed. This latter window, also offers the user the option to select an order ID, and depending on the chosen number, if the "Get Order Bill" button is pressed, a text file of the bill corresponding to the selected order will be custom generated and saved. Besides these, much like the other two management windows, this window also provides the functionality of displaying all of the existing orders for the user to see.

Use Case Diagram:



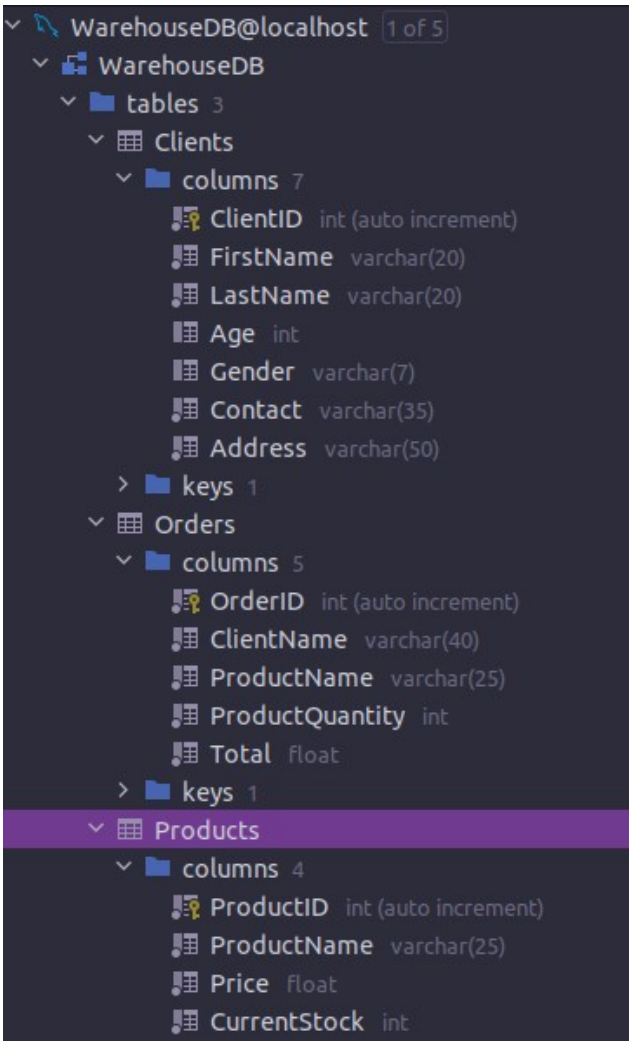
3. Design

3.1. Design Pattern. Design Decisions

The usage of object-oriented design, as well as following the Layered Architecture model represents the basis of this application's design. Object-oriented design is the process of planning a system of interacting objects for the purpose of solving a software problem. Put in simpler terms, it is a paradigm of transposing objects that are part of the real world into packages and classes. As far as the layered architecture is concerned, this implies that the application will contain at least four packages: Data Access Layer, Business Layer, model and presentation. However, in this implementation, along the four previously mentioned packages, there are also controller, validators and connection packages provided. This architecture model implicitly incorporates elements of a slightly modified version of the MVC (Model View Controller) architecture since it contains the packages model, controller and presentation (equivalent to the view package).

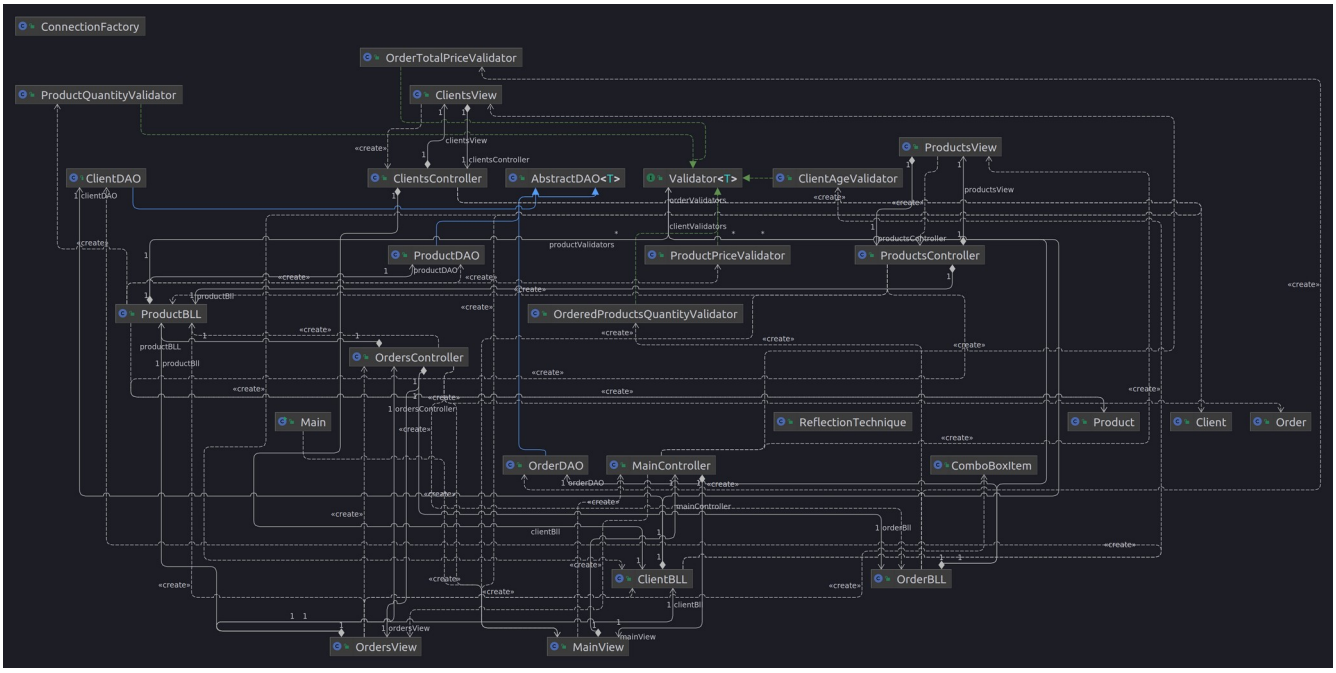
For the database driven side of the application, it includes three tables: Clients, Products, Orders. Each table has an ID as primary key (a unique ID specific to the table) and they each contain detailed information about their contents.

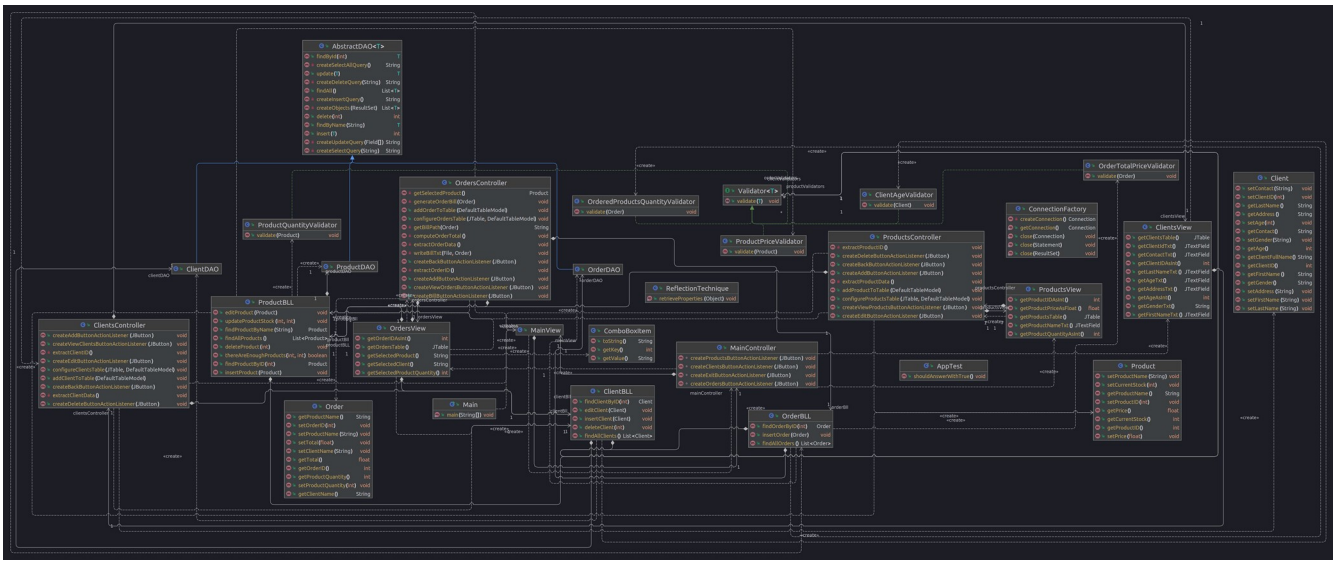
An illustration of the priorly mentioned table is given below:



3.2. UML Diagram

A more simplistic version of the UML Class Diagram generated by Intellij is presented below:





3.3. Package Distribution and Classes

The packages of the application are: start, presentation, model, connection, dao, bll and validators. The Main method is located inside the start package and serves for launching the program towards execution.

The presentation package contains four classes responsible for creating and displaying the windows of the application: MainView, ClientsView, ProductsView, OrdersView, along with four controller classes (MainController, ClientsController, ProductsController, OrdersController) that contain the implement the actual functionality of the view classes and are what actually gives the user the possibility to interact with the GUI.

The model package contains real life-like data, transposed into different objects: Client (the class that retains the data about clients), Product (the transposal of the products), Order (the class that creates the connection between the clients and their desired products/ products they want to order).

The dao package contains the definition of classes responsible with accessing the data from the database tables. This package contains an abstract generic class that implements several database data management operations, such as: inserting data into tables, updating existing data, removing data from tables and selecting data from tables depending on several criterias.
The connection package realizes the connection with the database, in order to make the execution of the operations from dao possible.
The bll package contains the classes that make the interaction between the operations implemented in the dao region and the user interface possible, since the methods from bll are the ones called inside the controller classes.
The validators package contains the definition of multiple validation classes, each table having at least one corresponding validator. The bll classes validate the information entered by the user by using those validators, and if the input passes, it will be inserted into the database of dao type objects.

The tables of the warehouse database share the same structure (in terms of name of the class, name of the attributes and attributes type) with the classes inside the model package.

4. Implementation

start package: Main Class Implementation

The Main class only contains the main() method, in which an instance of the MainView class is being instantiated in order for the menu window to be generated at the beginning of the program's execution.

connection package: ConnectionFactory Class Implementation

The ConnectionFactory class contains as static attributes the data necessary for connecting to the database (the driver, the user, the password, and the url of the database). The constructor of the class creates an object based on the driver initialized with the given values for these connection attributes.

The createConnection() method establishes the connection with the database or throws an exception if the connection cannot be created. The class also contains three additional close() methods, that are overloaded and only differ through the transmitted parameters. These methods ensure that the statement, result set and connection to the database are safely closed, so that no other issues are encountered as a result.

model package:
Client Class Implementation

The Client class contains the following attributes: clientID, firstName, lastName, age, gender, contact and address. Those attributes represent the data that will be stored in the database. The clientID attribute is used for the unique identification of the clients, and the rest of the attributes are quite self explanatory.

Products Class Implementation

The Products class contains the following attributes: productID, productName, price, currentStock. The productID attribute is used for the unique identification of the products in the database, productName stores the name of the product, price stores the price of one product unit, and currentStock represents the quantity of a product that is currently available (in stock) in the warehouse.

Orders Class Implementation

The Orders class contains the following attributes: orderID (used for the unique identification of the orders), clientName (the name of the client who is placing the order), productName (the name of the desired product), productQuantity (the number of product units the user wants to buy) and total (representing the total price of the order). The clientName and productName fields are the ones through which the Orders table communicates with the Clients and Products tables (by using the findByName() method).

All the classes inside the model package contain constructors that receive as parameters the values for their attributes, as well as getters and setters for each field.

bll package:

The classes contained in this package are the following: ClientBLL, ProductBLL and OrderBLL. These classes each contain lists of validators specifically tailored for each data model and an instance of their corresponding dao class as attributes and they are basically some intermediary classes. Their purpose is to validate the data before it is inserted into the database, or intercept the possible errors by throwing exceptions otherwise.

validators package: Validator Interface Implementation

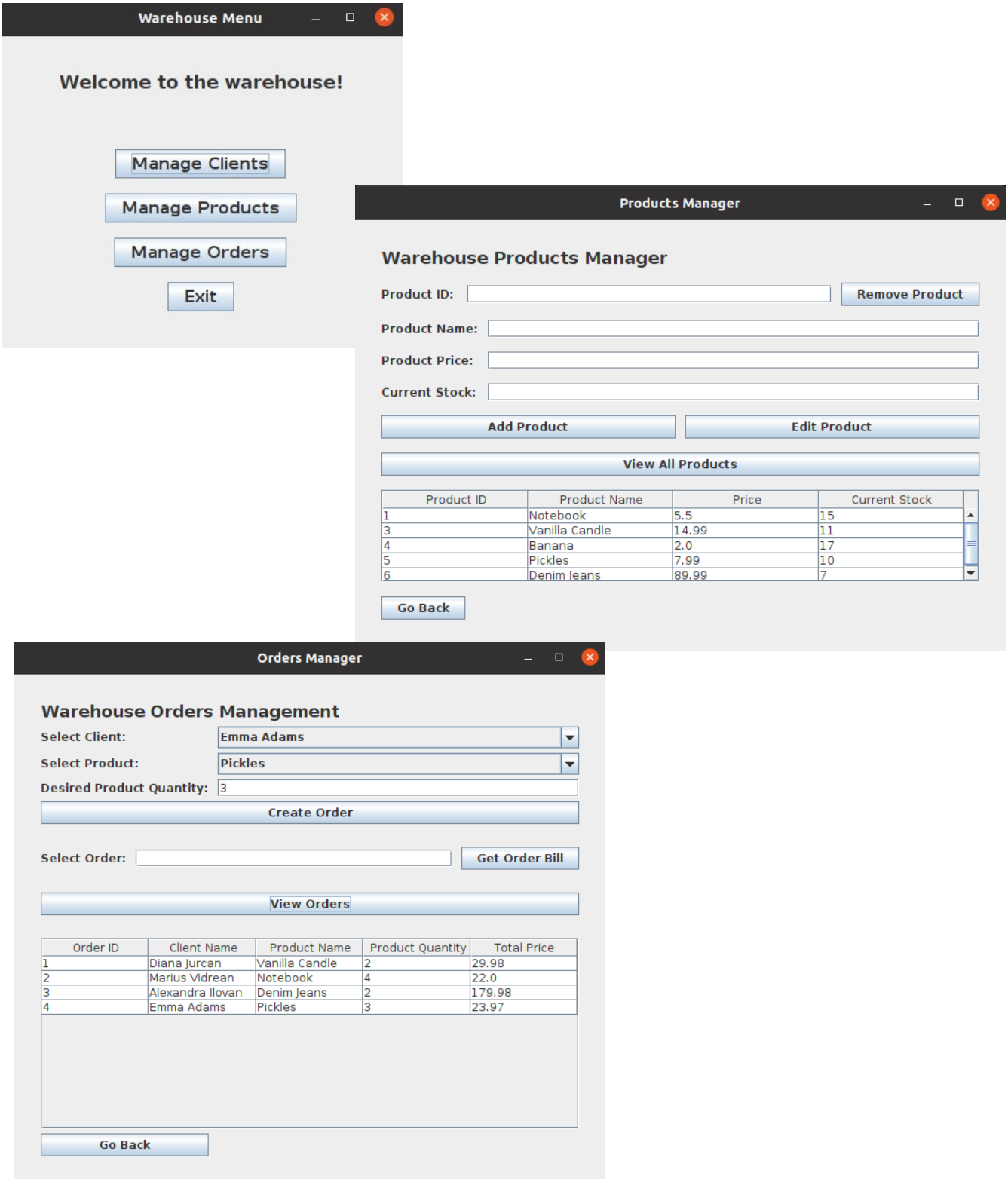
The Validator interface only contains the validate(T t) method, which receives as argument a generic object of type T. This interface is implemented by the other classes in the package, each class oferring a different implementation of the method depending on their specific validating rules. In this application, validators for the client's age, the product's price and quantity and for the orders' total and ordered products were defined.

dao package:

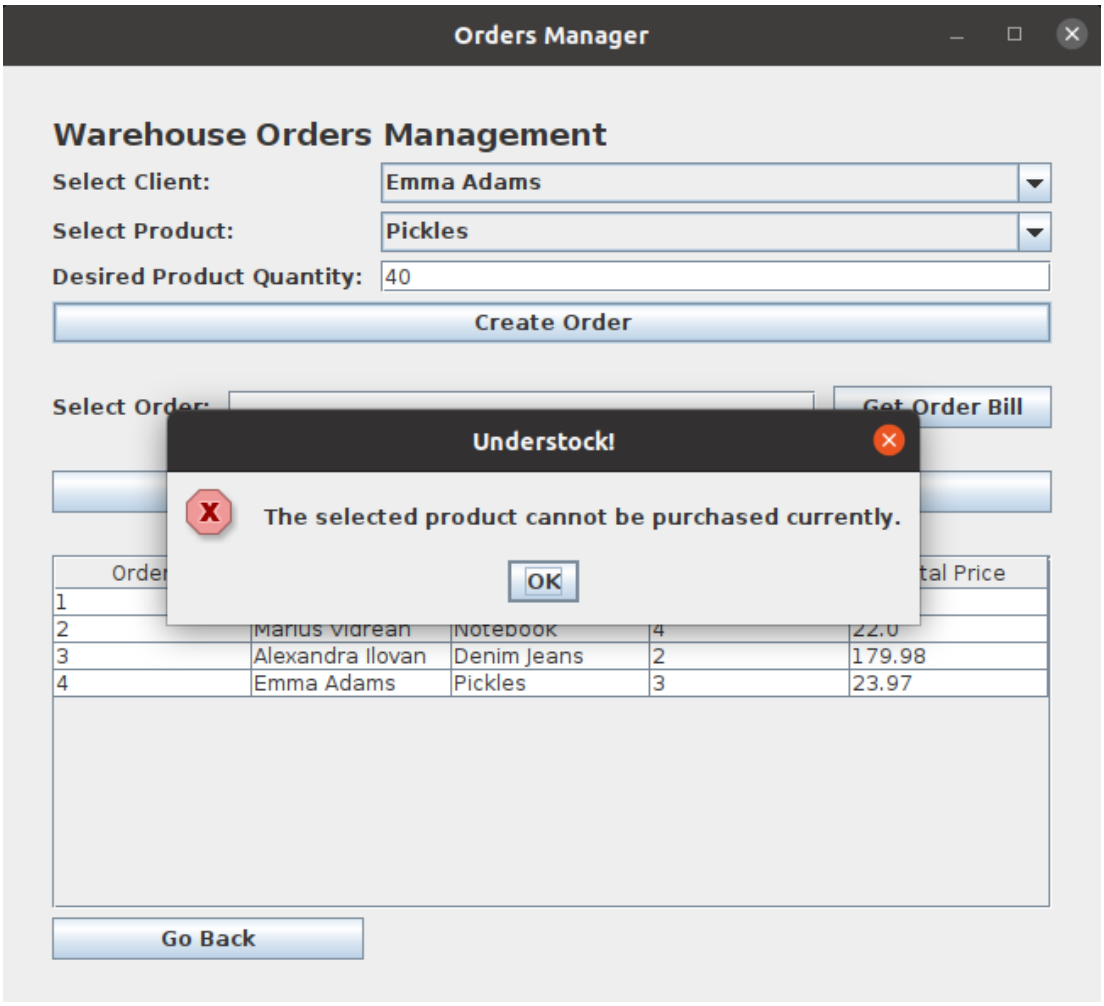
This package contains four classes, one for each data type (ClientDAO, ProductDAO, OrderDAO) and one abstract class. The first three classes don't contain any attributes nor methods, but they extend the class AbstractDAO<T>, T being a generic type that can be replaced by the right type of each class when the inheritance is done.

5. Results

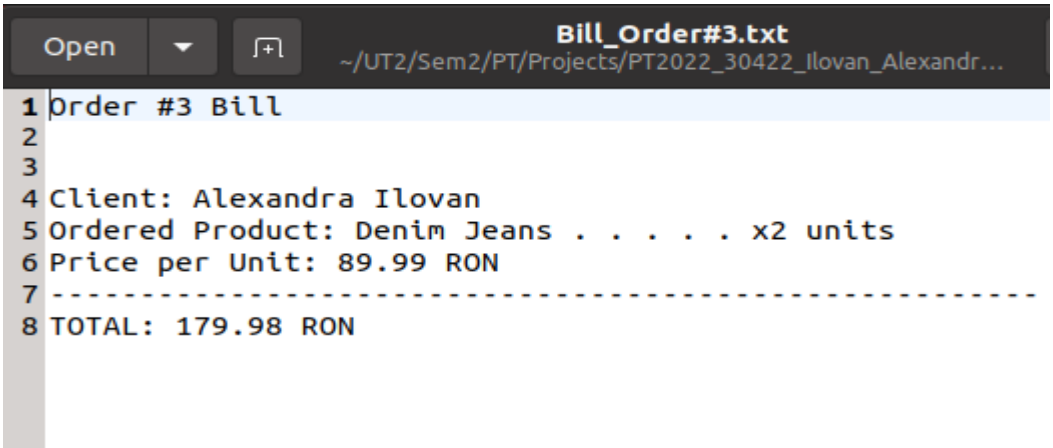
A short presentation of the application at work:



An example of a case when an order cannot be created due to a desired product quantity that is larger than the stock of the warehouse:



And below is an example of a generated order bill file:



6. Conclusions

By working on this assignment, I have learned how to establish a connection with databases in Java, and improved my skills of managing relational databases significantly. I also got accustomed to working with reflection techniques in Java, as well as implementing and making use of generic classes and attributes. This was also my first time hearing about JavaDoc files and trying to generate such files, and I was amazed by how easy and efficient it is to produce detailed documentations of one's code.

7. Bibliography

- the provided support presentation
- the sample code provided on GitLab:
https://gitlab.com/utcn_dsrl/pt-layered-architecture
https://gitlab.com/utcn_dsrl/pt-reflection-example

https://www.jetbrains.com/help/idea/working-with-code-documentation.html#generate-javadoc
https://www.jetbrains.com/help/idea/export-data.html#create-a-full-data-dump-for-mysql-and-postgresql
https://www.baeldung.com/javadoc
https://stackoverflow.com/questions/17887927/adding-items-to-a-jcombobox