

**Fundamental Programming Techniques**

**Assignment 3: Orders Management**

**Student:** Diana-Maria Jurcan

**Group:** 30422

**Laboratory professor:** Marcel Antal

**Course professor:** Cristina Bianca Pop

# TABLE OF CONTENS

1. Assignment objective……………………………………….3
2. Problem analysis, modelling, scenarios, usage cases……….3
3. Design……………………………………………………….5
4. Implementation………………………………………………7
5. Results……………………………………………....……...10
6. Conclusions………………………………………………...12
7. Bibliography……………………………………………......12
8. **Assignment objective**

The objective of the current topic is to propose, design and implement an application in order to improve the processing of orders and products available inside a warehouse. In other words, it is an application designed to automate both the processing of orders from customers (including the creation of the receipt) and the updating of the menu and products available in the warehouse.

The main purpose of this topic is to get acquainted with working with the database. Both learning the connection with the database and processing all the information it offers us. From the beginning, we work with visualizing the information, adding new information, updating and deleting information.

In my opinion, it is very important to be as familiar as possible with the database connection because this is the principle on which almost everything around us works, for example: cash registers in stores, lists of students from a certain faculty. , etc. Retrieving information from a well-defined place is a necessity nowadays, as well as its manipulation and processing.

In addition to this familiarization, this topic also aims to create a connection of the project maker with a network connection such as this project. It consists of adding customers, adding products, orders, deleting customers, products, updating certain products and viewing all of them, respectively. After the completion of the project, the person who implemented the project will have the necessary knowledge to work with certain products, information, and some orders.

1. **Analysis**

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 the application logic

• Presentation classes – GUI related classes

• Data access classes - classes that contain the access to the database

The project should contain:

Object-oriented programming design, classes with maximum 300 lines, methods with maximum 30 lines, Java naming conventions

• Use javadoc for documenting classes and generate the corresponding JavaDoc files.

• Use relational databases for storing the data for the application, minimum three tables: Client, Product and Order.

• Create a graphical user interface including:

A window for client operations: add new client, edit client, delete client, view all clients in a table (JTable)

A window for product operations: add new product, edit product, delete product, view all product in a table (JTable)

A window for creating product orders - the user will be able to select an existing product, select an existing client, and insert a desired quantity for the product to create a valid order. In case there are not enough products, an under-stock message will be displayed.

After the order is finalized, the product stock is decremented.

• Use reflection techniques to create a method that receives a list of objects and generates the header of the table by extracting through reflection the object properties and then populates the table with the values of the elements from the list.

• Good quality documentation covering the sections from the documentation template.

• Use reflection techniques to create a generic class that contains the methods for accessing the DB: create object, edit object, delete object and find object. The queries for accessing the DB for a specific object that corresponds to a table will be generated dynamically through reflection.

* **Use cases**

The requirements are descriptions of the services provided by the system and the constraints that are generated by during the requirements engineering process. The specific functional requirements for this issue are as follows: Designing an application designed to improve the operation of the warehouse, including both order management and real-time updates of menus and - implicitly - products in the warehouse.

1. **Design**

* **Use case diagram**

Diagram

Description automatically generated

* **UML Diagram**

Unified Modeling Language or UML for short is a standard language for describing models and software specifications. UML was originally developed to represent the complexity of object-oriented programs, the foundation of which is the structuring of programs into classes, and their instances (also called objects). However, due to its efficiency and clarity in the representation of abstract elements, UML is used beyond the IT domain. On the next page you can see the UML diagrams generated with IntelliJ.

Diagram

Description automatically generated

1. **Implementation**

**Data structures:**

The most important data structures that have been used are the following:

- AbstractDAO <> -generic structure necessary to access the database for different types of data such as: client, order and product.

- List <T> - the list for operations on a generic type of object T.

- List<Validator< >> - list of validators for objects of any type.

**Database:**

**Text

Description automatically generated**

Components within the layered architecture pattern are organized into horizontal layers, each layer performing a specific role within the application (e.g., presentation logic or business logic).

Our application consists of four layers: Business Layer, Data Access Layer, model layer and presentation. However, the project implements will contain several other packages with many other classes, like controller, view, validators and connection.

The database contains three tables: for clients, for products, for orders. Each table has the primary key the ID specific to the contents of the table and each other columns contains specific information regarding the object presented there. For example for client we have ID, name and address. For product we have ID, name, quantity and price. For orders we have ID, client name, product name, order quantity and order total.

**Packages:**

* **Package businessLogicLayer:**

- Class ClientBLL: represents the abstraction of the logic block in the implementation of the access to the database for the Client table.

- Class OrderBLL: represents the abstraction of the logic block in the implementation of the database access for the Order table.

- Class ProductBLL: represents the abstraction of the logic block in the implementation of the access to the database for the product table.

* + **Package validators:**
    - Validator interface: 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.
    - OrderQuantityValidator
    - OrderTotalValidator
    - ProductPriceValidator
    - ProductQuantityValidator
* **Package connection:**

- Class ConnectionFactory:makes the actual connection to the database

Packagedao:

- Class AbstractDAO: contains all generic methods for database operations, extended class of all other classes in the package

* findAll:

This method queries the database to obtain a list of all available objects in the database. First of all, it starts by declaring objects such as Connection, PreparedStatement and ResultSet, necessary for processing the data obtained from the query according to the existing application. The createSelectAllQuery () method executes a string of the form "SELECT \* FROM tableName ; ”,string that will be transmitted to the database for querying it by calling the statement.executeQuery () method. Finally, if all went well, a list of objects such as those extracted from the database will be created and returned.

* findById:

The method presented above has the same behavior as that of the first method described, with the difference that the query on the database will be filtered, returning a single object. Specifically, the query string will be in the form "SELECT name FROM tableName WHERE id =?”.

* findByName:

It has exactly the same behavior as described above, with the difference that the filtering will be done by name, more specifically: "SELECT name FROM tableName WHERE name =?”.

* Insert: This method ensures the insertion of the data of a generic object in the database. The failure of this will be signaled by a message from the treatment of exceptions that may occur.
* Delete: This method ensures the deletion of the data of a generic object in the database. The failure of this will be signaled by a message from the treatment of exceptions that may occur.
* Update: This method will update the data in the database, such as updating the quantity of products available, the price or the menu items.

- Class ClientDAO: extends the class AbstractDAO

- Class ProductDAO: extends the class AbstractDAO

- Class OrderDAO: extends the classAbstractDAO

The last three classes do not contain any attributes or methods but because they extends the abstract class and because T is an object of generic type, it can be replaced in each class by any type of object like client, product or order.

* **Package model:**

- Class Client: contains attributes like client ID, client name and address. The attributes will later be stored in the database. The identification will often be made by ID or name.

- Class Product: contains attributes like product ID, product name, price, quantity. The attributes will later be stored in the database. The identification will often be made by ID or name.

- Class Order: contains attributes like order ID, client name, product name, order quantity and order total. The attributes will later be stored in the database. The identification will often be made by ID.

* **Package start:**

- Class Start: contains the main method, by default this is where the program starts running

* **Package presentation:** contains the view and controller classes:
* Class ClientSettingsController: contains the functionality of the add, edit, delete, view all clients buttons, along with the back and exit buttons.
* Class ProductSettingsController: contains the functionality of the add, edit, delete, view all products buttons, along with the back and exit buttons.
* Class OrderSettingsController: contains the functionality of the add order button, along with the back and exit buttons.
* Class MainMenuController: contains the functionality of all the windows: Client Settings, Product Settings, Order Settings.
* Classes ClientSettingsView, ProductSettingsView, OrderSettingsVies, MainMenuView.

1. **Results**

The main menu:

Contains 4 buttons: Client Settings, Product Settings, Order Settings and exit application.

**Graphical user interface, application

Description automatically generated**

Upon pressing the Client Settings button we are met with the following menu:

Graphical user interface, website

Description automatically generated

Here, we can enter an ID, a name or an address. By these fields, we can add a client, edit a client, or delete a client from the database. We can also view all clients from our database.

By pressing the back button we return to the Main menu and by pressing the exit button we exit the application.

Upon pressing the Product Settings button we are met with the following menu:

Timeline

Description automatically generated

Here, we can enter an ID, a name, quantity and a price. By these fields, we can add a product, edit a product, or delete a product from the database. We can also view all products from our database.

By pressing the back button we return to the Main menu and by pressing the exit button we exit the application.

Upon pressing the Order Settings button we are met with the following menu:

Graphical user interface, application

Description automatically generated

Here, we can choose an existing client from a combo box and we can also choose in the same manner a product from a combo box. We can enter a desired quantity of the product and by pressing the add order button we shall see that the order will be added to the database.

By pressing the back button we return to the Main menu and by pressing the exit button we exit the application.

1. **Conclusions**

By working on this assignment, the following skills were learned in this topic: connecting a Java application with databases, operations generic with database, Layered Architecture pattern, Database Structure, reflection technique and java doc files.

Further developments:

- the possibility of logging in with the password and username of the clients.

-the possibility of creating an order bill after each order has been created.

-the possibility of the client to obtain a list with his own orders.

1. **Bibliography**

* <https://stackoverflow.com/>
* <https://docs.oracle.com/javase/tutorial/uiswing/>
* <https://www.vogella.com/tutorials/JUnit/article.html>
* <https://www.baeldung.com/junit-5>
* <https://www.geeksforgeeks.org/>
* <https://en.wikipedia.org/wiki/Use_case>
* <https://google.github.io/styleguide/javaguide.html>
* <https://www.lucidchart.com/>
* <https://docs.oracle.com/javase/tutorial/essential/concurrency/index.html>
* <https://www.tutorialspoint.com/java/util/timer_schedule_period.htm>
* <https://www.javacodegeeks.com/2013/01/java-thread-pool-example-using-executors-and-threadpoolexecutor.html>
* Courses, laboratories, homework, and OOP project from year 2, semester 1.