**DOCUMENTATION HOMEWORK 3**

**ORDER MANAGEMENT**

**Chindea Tudor Eugen**

**Grupa 30423**

**Profesor Laborator Assistant Antal Marcel**

Contents

[1. Functional Requirements 3](#_Toc70061222)

[2. Objectives 3](#_Toc70061223)

[2.1. Principal Objective: 3](#_Toc70061224)

[2.2. Secondary Objectives: 4](#_Toc70061225)

[3. Problem Analysis And Use Cases 4](#_Toc70061226)

[3.1 Analysis 4](#_Toc70061227)

[3.2 Use Cases 5](#_Toc70061230)

[4. Design 6](#_Toc70061231)

[4.1. Data structures 6](#_Toc70061232)

[4.2. Class diagram 6](#_Toc70061233)

[4.3. SQL Query 6](#_Toc70061234)

[5. Implementation 7](#_Toc70061235)

[6. Testing 8](#_Toc70061236)

[7. Conclusions and Future Development 8](#_Toc70061237)

[8. Bibliography 8](#_Toc70061238)

# Functional Requirements

Build a store manager which can compute the following operations:

* Read, Update, delete and insert in the user interface elements from the tables Client and Product from a database.
* Insert and display orders in the user interface from the Orders table from the same database.

# Objectives

## Principal Objective:

Propose, build and test a system that is capable of doing basic operations on at least three tables: Orders, Clients, Products. From the user interface we must be able to add elements in all the tables, to modify the content of the Clients and Products tables and to delete Clients and Products.

## Secondary Objectives:

|  |  |  |
| --- | --- | --- |
| **Secondary Objectives** | **Description** | **Chapter** |
| Development of use cases and scenarios | The application should be able to deal with many types of input and many scenarios. The input should be easy and natural for the user to type. | 3 |
| Choosing the data structures | The right data structures enable an easy manipulation of the data stored in the tables. Three basic classes will be implemented and all operations will be done on them. All insert, delete select and update methods are usable on each class | 4 |
| Division into classes | Dividing the methods into separate classes makes the code easier to understand. The AbstractDAO class represents the class in which all sql queryes are written and executed. | 4 |
| Algorithm development | The algorithm implementation was simple once the data was stored properly and the sql code was correct. | 4 |
| Solution implementation | Reflecion techniques were used to generalize the sql statements. | 5 |
| Testing | For each table, insertion, deletion and upate were dnw using the user interface. The table was printed after every iteration. | 6 |

# Problem Analysis And Use Cases

## 3.1 Analysis

Reflection is a feature in the Java programming language. It allows an executing Java program to examine or "introspect" upon itself, and manipulate internal properties of the program. For example, it's possible for a Java class to obtain the names of all its members and display them.The reflection classes, such as Method, are found in java.lang.reflect. There are three steps that must be followed to use these classes. The first step is to obtain a java.lang.Class object for the class that you want to manipulate. java.lang.Class is used to represent classes and interfaces in a running Java program.One way of obtaining a Class object is to say:Class c = Class.forName("java.lang.String"); to get the Class object for String. Another approach is to use: Class c = int.class; or Class c = Integer.TYPE; to obtain Class information on fundamental types. The latter approach accesses the predefined TYPE field of the wrapper (such as Integer) for the fundamental type**.** The second step is to call a method such as getDeclaredMethods, to get a list of all the methods declared by the class.Once this information is in hand, then the third step is to use the reflection API to manipulate the information. For example, the sequence:Class c = Class.forName("java.lang.String"); Method m[] = c.getDeclaredMethods(); System.out.println(m[0].toString());will display a textual representation of the first method declared in String.Once Class information is in hand, often the next step is to ask basic questions about the Class object. For example, the Class.isInstance method can be used to simulate the instanceof operator**.** One of the most valuable and basic uses of reflection is to find out what methods are defined within a class.The program first gets the Class description for method1, and then calls getDeclaredMethods to retrieve a list of Method objects, one for each method defined in the class. These include public, protected, package, and private methods. If you use getMethods in the program instead of getDeclaredMethods, you can also obtain information for inherited methods.

Once a list of the Method objects has been obtained, it's simply a matter of displaying the information on parameter types, exception types, and the return type for each method. Each of these types, whether they are fundamental or class types, is in turn represented by a Class descriptor.

## 3.2 Use Cases

Use Case Name: Place an Order.

Actors: Person using the application.

Triggers: The user presses the button that adds the client id, product id and quantity to the orders table..

Preconditions: The user has typed userid, the product id and the quantity of the product he wants to order..

Post Conditions: The data will be split in designated data structures the insert query will be applied. After the computation of the sql statement, the new orders table will be displayed on the screen and the product quantity will be modified.

Normal Flow:

1. The user types in the first textbox a valid client id for which he wants to make a new order.
2. The user types in the second textbox a valid product id for the order.
3. The user types in the third textbox the quantity of the product he wants to enter.
4. If the quantity typed by the user is larger than the available products on stock, a warning will appear on screen. The data will not be stored until the input is typed in correctly, which means a smaller quantity.
5. The user presses the button for placing the order and the data will appear on the screen.

Use Case Name: Updating a product.

Actors: Person using the application.

Triggers: The user presses the button that updates and displays the product table.

Preconditions: The user has typed the data he wants to update in his product. The user must type in the product id on which he wants the update to be made. He can modify one of more fields from the product name, price and quantity.

Post Conditions: The data will be stored in a new product object. If no data was typed in any of the name, price and quantity fields, the old data will be kept.

Normal Flow:

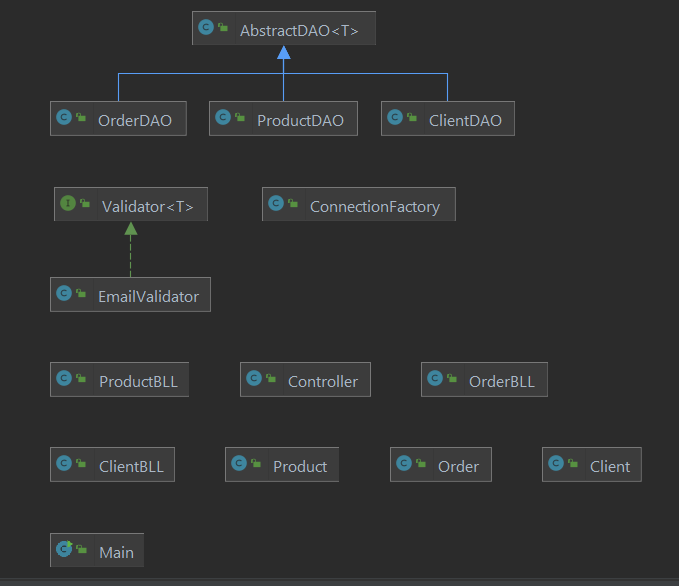
1. The user types in the first textbox the the id of the product he wants to update
2. The user will type in any of the next three textboxes the new data he wants to have for the given product
3. The user presses the button to update the product table.

# Design

## Data structures

The data will be stored in a form such that it would be easy to be interpreted by the sql commands. The three classes on which the insert, update, retrieve and delete querys are performed will have the same name as the tables from the database in which the data is stored. The fields of each class corresponds to the fields in the table, they are written in the same order but upper case and lower case letters can be reversed. This data storing allows for to use the java reflection techniques.

## Class diagram



## SQL Query

The insert query is done using reflection. The name of the table is selected into the query format. The id of the element will not be inserted as it is generated automatically by the database. The field name will be taken from the object fields because the java class and the corresponding database table have the same fields.

The update query is done using reflection. The id of the product or client that is updated must be given by the user. We take the element of the corresponding id and set the fields in which a value was typed to the typed value. The update is done on the modified element.

The delete query is done using reflection. The id of the element to delete must be inserted. The program will know from which table to delete.

The select queries are done using reflection. The first select is one based on the id and will use the objects class name to determine on which table the select is applied. This query is used in the update query. The select \* query just prints all elements from a given table when an object is given. Using reflection we can get the class of the object.

# Implementation

The program is composed of different packages. The model package has the basic classes on which all operations are done. Each one of these classes has two constructors, one that is empty and another one that contains all the fields besides the id which is generated automatically when a new element is added to each table

The dao package contains the most usefull classes. The AbstractDAO class implements select all, select by id, update, insert and delete queries on each table. The methods use java reflection techniques to get the class and the field names of each object. In order to use this data, the classes in the model package must have the same fields with the tables in the database. The findAll function takes an object and returns a list of objects that were stored in the coresponding table. The findById function takes an integer and an object to determine the table and returns the object with the given id from the corresponding table. The insert query does not use an id integer because it is generated by the databse uppon insertion. An object of any type from the model package is given. The class and the fields excluding the id of the object are used to write the mysql query and the values of the fields are given to be inserted in the table. All data is stored in the table. The delete query takes an integer and an object and delets the corresponding element from the table. The update query takes an object fromt the model package and an id integer and update sthe coresponding field in the coresponding table. When an update is done, not all fields must be completed, if a textBox is empty when the update button is pressed, the old values from the element with the given id will remain the same.

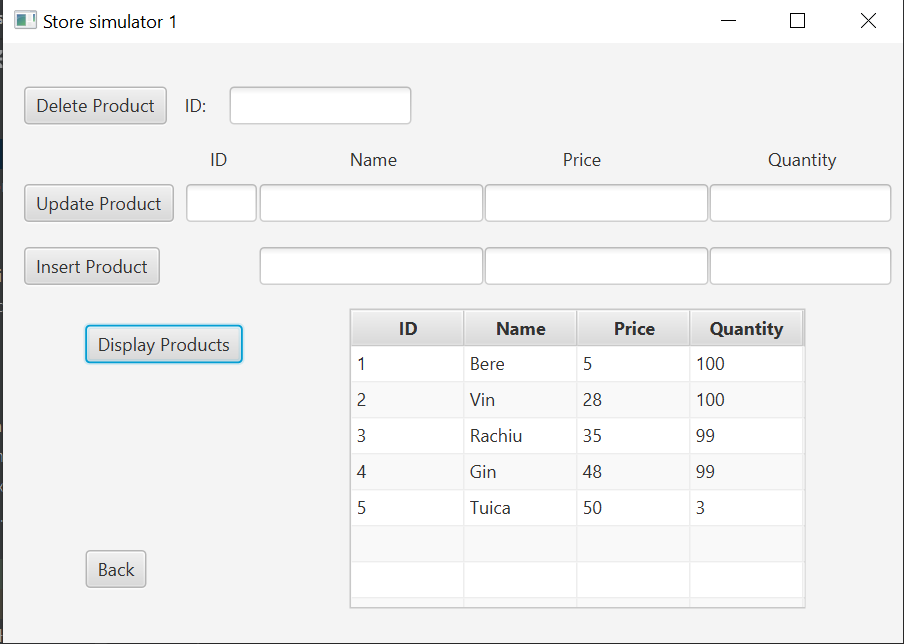
The ClientDAO, ProductDAO, OrderDAO extend the AbstractDAO class and have as parameter the coresponding class from the model package.

ClientBLL,ProductBLL, OrderBLL classes have the methods for selecting all items, selecting items by id, deletion, updating and insertion. They are done on object types from the model package.

The vlidator package contains an email validator that uses a regex to check if the email enterg when storing a new client is ok.

The connection package manages the connection between the java code and the mysql database.

The presentation package includes the controller, in which all button actions are done and the entire user interface is coded.



# Testing

For each class from the package model we can test the queries in the user interface. For the products and clients, update, delete insert and display functions use all the queries in the AbstractDAO class. The order window in the user interface has options only for insert and displaying. Since the methods used are the same, we can assume that is they work on the the orders table as well.

# Conclusions and Future Development

This application is useful for working with java reflection techniques and with databases. We have put the java classes into packages to have a well structured code. The SQL queries written are universal and can be used on any objects. This project structure makes the code portable and easy to understand for anybody. The application can be developed even more by adding other queries and more classes and fields to each class. The user interface might not be tho most intuitive but has all the necessary tools for the user to test all the features of the program.

# Bibliography

1. Laboratory and Course material
2. <https://www.oracle.com/technical-resources/articles/java/javareflection.html>
3. https://www.w3schools.com/mysql/mysql\_ref\_functions.asp