**Homework 3**

1. **Objective**

The main objective of this homework is to propose, design and implement an application OrderManagement for processing customer orders for a warehouse. Relational databases are used to store the products, the clients and the orders. Furthermore, the application uses (minimally) the following classes:

• Model classes - represent the data models of the application

• Business Logic classes - contain the application logic

• Presentation classes – classes that contain the graphical user interface

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

The secondary objectives are:

1. **Designing a model for Client.** This is used to store information about the clients. This is detailed in cap 2 and the implementation can be found in cap 4;
2. **Designing a model for Products.** This is used to store information about the products. This is detailed in cap 2 and the implementation can be found in cap 4.
3. **Designing a model for Order.** This is used to store information about the orders. This is detailed in cap 2 and the implementation can be found in cap 4.
4. **Designing a model for Stock.** This is used to store the quantity of each product in the warehouse. This is detailed in cap 2 and the implementation can be found in cap 4.
5. **Designing a model for Order Type.** This is used to store the types of orders that can be made. This is detailed in cap 2 and the implementation can be found in cap 4.
6. **Implementing the connection to the database.** We need this to have access to the database. The implementation can be found in cap 4;
7. **Implementing the DAO through reflection.** Needed to gain access to individual tables from the database. The implementation can be found in cap 4;
8. **Implementing validators for each table.** The input must be checked before it alters the database. The implementation can be found in cap 4;
9. **Implementing a BLL for each model.** These will be used directly in the interface as they are validated. The implementation can be found in cap 4;
10. **Implementing the Graphical User Interface (GUI).** A simple gui to interact with. This discussed in cap 3 and cap 4.
11. **Implementing a bill for each order.** A pdf file created to certify an order. The implementation can be found in cap 4;
12. **Problem analysis, modeling, scenarios, use cases**

There are several use cases for this problem. The “actor” is the user who would like to interact with the database through this application. The system in this case is the Warehouse Management System (WMS). Let us see the use cases:

1. Use case (Clients)

-The user presses the “Clients” button;

-The WMS pops the Clients window;

-The user chooses one of the operations Find, Delete, Add, Update;

1. Use case (Orders)

-The user presses the “Orders” button;

-The WMS pops the Orders window;

-The user chooses one of the operations Find, Delete, Add, Update;

1. Use case (Products)

-The user presses the “Products” button;

-The WMS pops the Products window;

-The user chooses one of the operations Find, Delete, Add, Update;

1. Use case (Stock)

-The user presses the “Stock” button;

-The WMS pops the Stock window;

-The user chooses one of the operations: Find, Update;

1. Use case (Order Types)

-The user presses the “Order Types” button;

-The WMS pops the Order Types window;

-The user can see the allowed orders;

1. Use case (Find)

-Once on one of the windows that allow this feature, the user introduces in the first text box under id the id to be searched

-The WMS fills the rest of the text boxes with the information of that object or it displays a wrong id message;

1. Use case (Add)

-Once on one of the windows that allow this feature, the user adds the info of the objects to be inserted in the text boxes;

-The user presses the “Add” button;

-The WMS adds the new object to the database and the table displayed is refreshed or it displays an error message;

1. Use case (Update)

-Once on one of the windows that allow this feature, the user first searches for the wished object (Find use case);

-The user alters one of the text boxes as he sees fit;

-The user presses the “Update” button;

-The WMS alters the object from the database and the table displayed is refreshed or it displays an error message;

1. Use case (Delete)

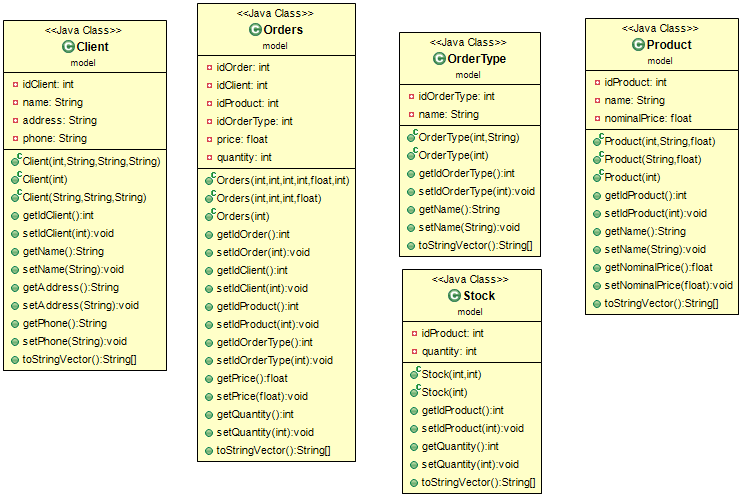
-Once on one of the windows that allow this feature, the first searches for the wished object (Find use case) or introduces the id of the wised object;

-The user presses the “Delete” button;

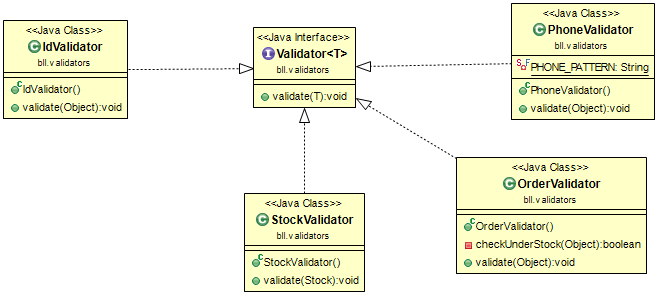
-The WMS deletes the new object to the database and the table displayed is refreshed or it displays an error message;

1. **Design (design decisions, UML diagrams, data structures, class design, interfaces, relationships, packages, algorithms, user interface)**

Firstly, we must design the warehouse in such a way as to correspond to the required conditions. I have used 5 tables in MySql to represent the full functionality of the warehouse: client, orders (because order is a command in SQL), product, ordertype and stock. After this, I created the model in java: for each table, a class with the same name and fields has been created, also, getters and setters.



Secondly, the connection has been created and the DAO implemented (reflection for each operation has been used).

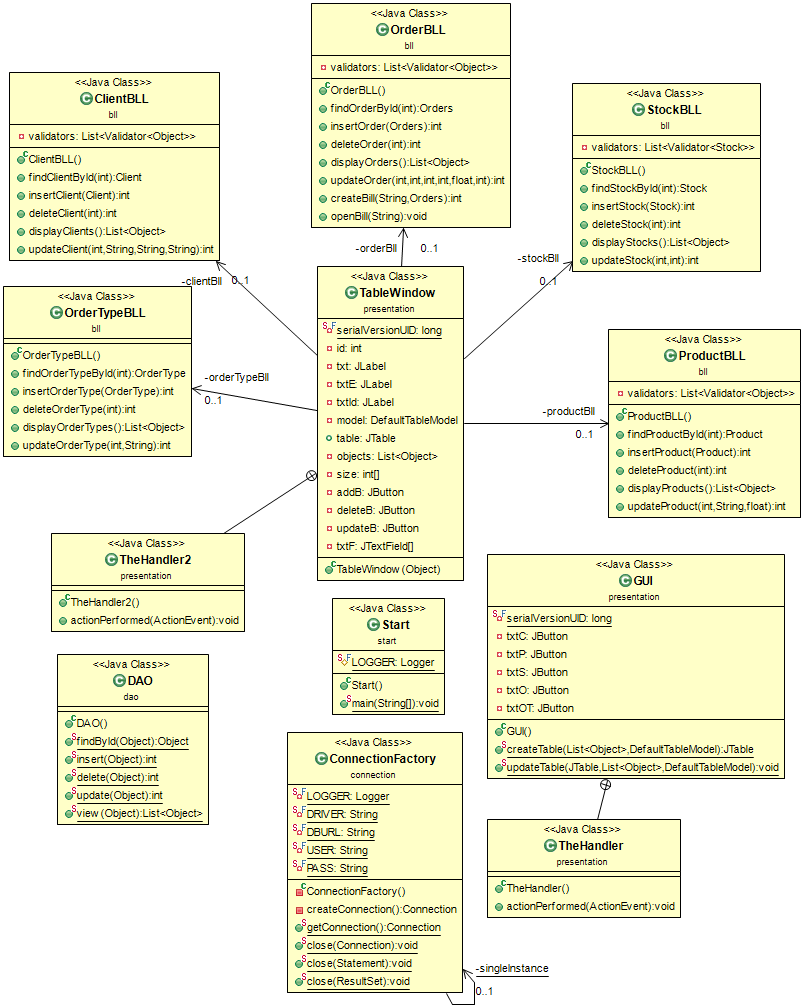


Next, the validators have been implemented. There is an interface Validator that has one function called validate. There are 4 validators: IdValidator – for duplicate ids, OrderValidator – to check that we are not understocked in case of a sell operation and to see if the ordertype and the id of the client exist, PhoneValidator – used to check the phone of the client and StockValidator – used to check we want to create a stock for in inexistent product.

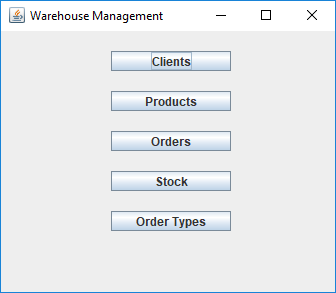
The Bll for each model has been next. Here we use the validators to check the introduced objects and to handle the other actions like update, add, find and delete.

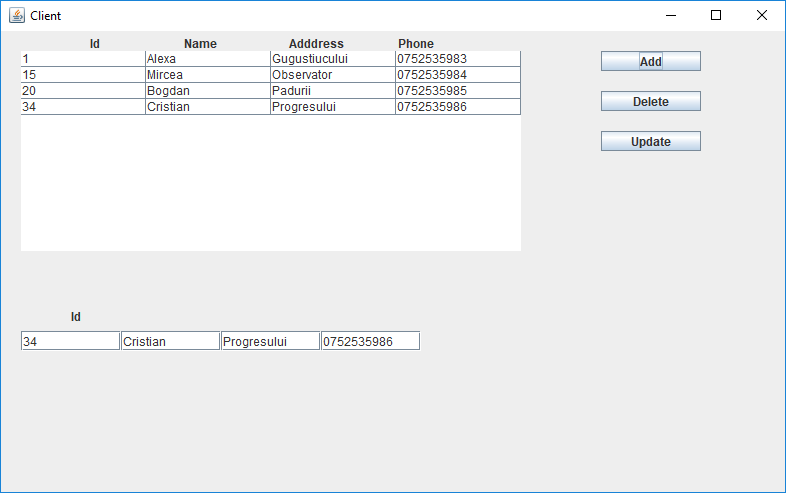
Next the gui has been created. It consists of a central window and the TableWindow for each table. Also, the pdf after a successful order placement implemented.

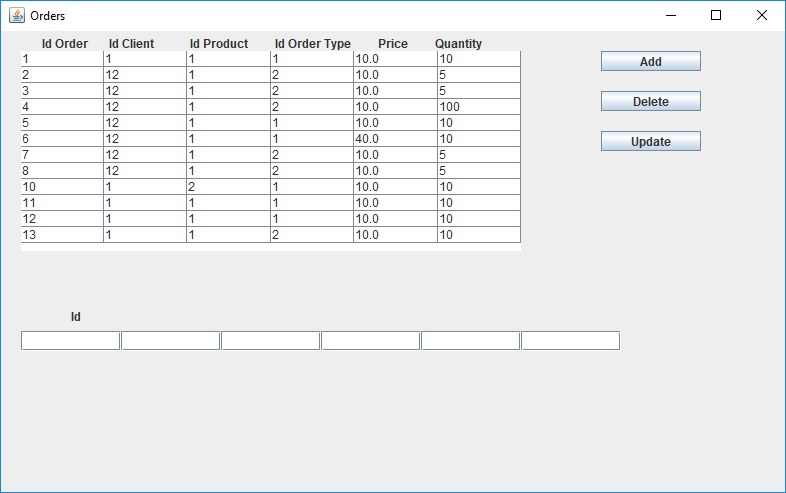
The project has 7 packages: bll, bll.validators, dao, start, presentation, connection and model.

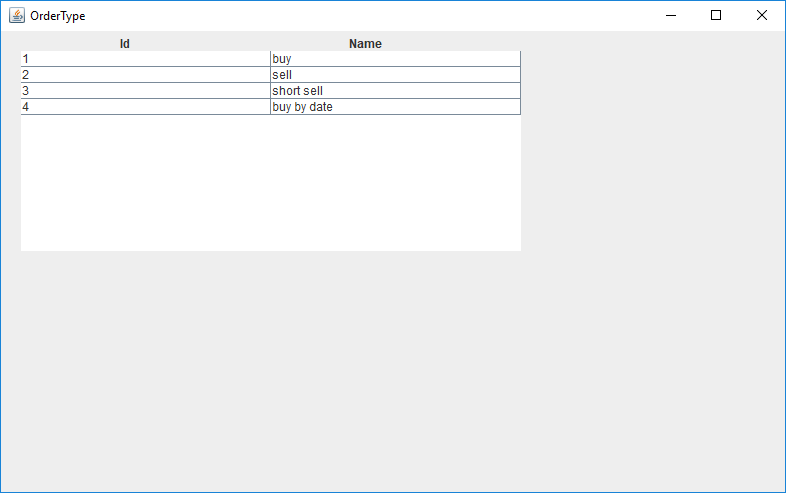


Some pictures of the gui:

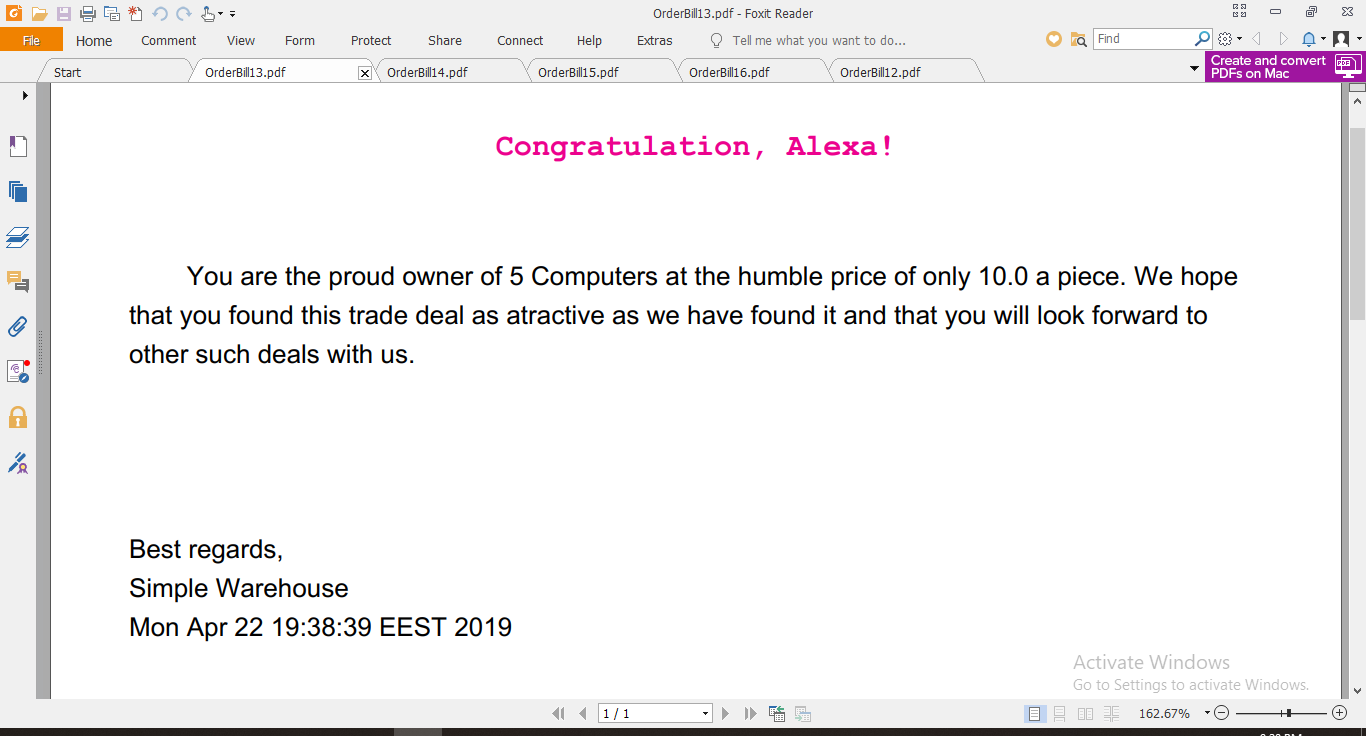








After a successful order placement.



1. **Implementation**

The most important classes used were:

1. Model classes. I will give only one example.

**package** model;

/\*\*

\* **@author** IndreBogdan

\*

\*/

**public** **class** Client {

**private** **int** idClient;

**private** String name;

**private** String address;

**private** String phone;

**public** Client(**int** idClient, String name, String address, String phone) {

**super**();

**this**.idClient = idClient;

**this**.name = name;

**this**.address = address;

**this**.phone = phone;

}

**public** Client(**int** idClient) {

**super**();

**this**.idClient = idClient;

}

**public** Client(String name, String address, String phone) {

**super**();

**this**.name = name;

**this**.address = address;

**this**.phone = phone;

}

/\*\*

\* **@return** the idClient

\*/

**public** **int** getIdClient() {

**return** idClient;

}

/\*\*

\* **@param** idClient

\* the idClient to set

\*/

**public** **void** setIdClient(**int** idClient) {

**this**.idClient = idClient;

}

/\*\*

\* **@return** the name

\*/

**public** String getName() {

**return** name;

}

/\*\*

\* **@param** name

\* the name to set

\*/

**public** **void** setName(String name) {

**this**.name = name;

}

/\*\*

\* **@return** the address

\*/

**public** String getAddress() {

**return** address;

}

/\*\*

\* **@param** address

\* the address to set

\*/

**public** **void** setAddress(String address) {

**this**.address = address;

}

/\*\*

\* **@return** the phone

\*/

**public** String getPhone() {

**return** phone;

}

/\*\*

\* **@param** phone

\* the phone to set

\*/

**public** **void** setPhone(String phone) {

**this**.phone = phone;

}

**public** String[] toStringVector() {

String[] a = **new** String[4];

a[0] = String.*valueOf*(idClient);

a[1] = **this**.name;

a[2] = **this**.address;

a[3] = **this**.phone;

**return** a;

}

}

1. The DAO is used to access the database and get the information from it, also to alter the database.

**public** **class** DAO {

**public** **static** Object findById(Object o) {

String findStatementString = "SELECT \* FROM ";

String table = o.getClass().getSimpleName().toLowerCase();

**int** id = 0;

String idName = **new** String();

Field f = o.getClass().getDeclaredFields()[0];

f.setAccessible(**true**);

**try** {

id = f.getInt(o);

idName = f.getName();

} **catch** (IllegalArgumentException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

} **catch** (IllegalAccessException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

} **catch** (SecurityException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

Object toReturn = **null**;

Connection dbConnection = ConnectionFactory.*getConnection*();

PreparedStatement findStatement = **null**;

ResultSet rs = **null**;

findStatementString = findStatementString.concat(table + " where " + idName + " = " + id);

**try** {

findStatement = dbConnection.prepareStatement(findStatementString);

rs = findStatement.executeQuery();

rs.next();

**if** (table.equals("client")) {

String name = rs.getString("name");

String address = rs.getString("address");

String phone = rs.getString("phone");

toReturn = **new** Client(id, name, address, phone);

}

**if** (table.equals("orders")) {

**int** idClient = rs.getInt("idClient");

**int** idProduct = rs.getInt("idProduct");

**int** idOrderType = rs.getInt("idOrderType");

**float** price = rs.getFloat("price");

**int** quantity = rs.getInt("quantity");

toReturn = **new** Orders(id, idClient, idProduct, idOrderType, price, quantity);

}

**if** (table.equals("ordertype")) {

String name = rs.getString("name");

toReturn = **new** OrderType(id, name);

}

**if** (table.equals("product")) {

String name = rs.getString("name");

**float** nominalPrice = rs.getFloat("nominalPrice");

toReturn = **new** Product(id, name, nominalPrice);

}

**if** (table.equals("stock")) {

**int** quantity = rs.getInt("quantity");

toReturn = **new** Stock(id, quantity);

}

} **catch** (SQLException e) {

} **finally** {

ConnectionFactory.*close*(rs);

ConnectionFactory.*close*(findStatement);

ConnectionFactory.*close*(dbConnection);

}

**return** toReturn;

}

**public** **static** **int** insert(Object o) {

String insertStatementString = "INSERT INTO ";

Connection dbConnection = ConnectionFactory.*getConnection*();

PreparedStatement insertStatement = **null**;

**int** insertedId = -1;

String table = o.getClass().getSimpleName().toLowerCase();

insertStatementString = insertStatementString.concat(table + " (");

**for** (Field f : o.getClass().getDeclaredFields()) {

// if (!f.getName().equals(o.getClass().getDeclaredFields()[0].getName()))

insertStatementString = insertStatementString.concat(f.getName() + ",");

}

insertStatementString = insertStatementString.substring(0, insertStatementString.length() - 1);

insertStatementString = insertStatementString.concat(") VALUES (");

**for** (Field f : o.getClass().getDeclaredFields()) {

f.setAccessible(**true**);

**try** {

// if (!f.getName().equals(o.getClass().getDeclaredFields()[0].getName())) {

**if** (f.getType() == String.**class**) {

insertStatementString = insertStatementString.concat("'" + f.get(o) + "'" + ",");

} **else**

insertStatementString = insertStatementString.concat(f.get(o) + ",");

// }

} **catch** (IllegalArgumentException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

} **catch** (IllegalAccessException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

insertStatementString = insertStatementString.substring(0, insertStatementString.length() - 1);

insertStatementString = insertStatementString.concat(")");

**try** {

insertStatement = dbConnection.prepareStatement(insertStatementString);

insertStatement.executeUpdate();

insertedId = 1;

} **catch** (SQLException e) {

} **finally** {

ConnectionFactory.*close*(insertStatement);

ConnectionFactory.*close*(dbConnection);

}

**return** insertedId;

}

**public** **static** **int** delete(Object o) {

String deleteStatementString = "DELETE FROM ";

String table = o.getClass().getSimpleName().toLowerCase();

**int** id = 0;

String idName = **new** String();

Field f = o.getClass().getDeclaredFields()[0];

f.setAccessible(**true**);

**try** {

id = f.getInt(o);

idName = f.getName();

} **catch** (IllegalArgumentException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

} **catch** (IllegalAccessException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

} **catch** (SecurityException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

Connection dbConnection = ConnectionFactory.*getConnection*();

PreparedStatement deleteStatement = **null**;

deleteStatementString = deleteStatementString.concat(table + " where " + idName + " = " + id);

**try** {

deleteStatement = dbConnection.prepareStatement(deleteStatementString);

deleteStatement.executeUpdate();

**return** 1;

} **catch** (SQLException e) {

**return** -1;

} **finally** {

ConnectionFactory.*close*(deleteStatement);

ConnectionFactory.*close*(dbConnection);

}

}

**public** **static** **int** update(Object o) {

String updateStatementString = "UPDATE ";

String table = o.getClass().getSimpleName().toLowerCase();

**int** id = 0;

String idName = **new** String();

Field f = o.getClass().getDeclaredFields()[0];

f.setAccessible(**true**);

**try** {

id = f.getInt(o);

idName = f.getName();

} **catch** (IllegalArgumentException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

} **catch** (IllegalAccessException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

} **catch** (SecurityException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

Connection dbConnection = ConnectionFactory.*getConnection*();

PreparedStatement updateStatement = **null**;

updateStatementString = updateStatementString.concat(table + " SET ");

**for** (Field fi : o.getClass().getDeclaredFields()) {

fi.setAccessible(**true**);

**try** {

**if** (fi.get(o) != **null**) {

**if** (!(fi.getType().isInstance(**new** String()) && ((String) fi.get(o)).equals("")))

updateStatementString = updateStatementString

.concat(fi.getName() + " = " + "'" + fi.get(o) + "'" + ",");

}

} **catch** (IllegalArgumentException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

} **catch** (IllegalAccessException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

updateStatementString = updateStatementString.substring(0, updateStatementString.length() - 1);

updateStatementString = updateStatementString.concat(" where " + idName + " = " + id);

**try** {

updateStatement = dbConnection.prepareStatement(updateStatementString);

updateStatement.executeUpdate();

**return** 1;

} **catch** (SQLException e) {

**return** -1;

} **finally** {

ConnectionFactory.*close*(updateStatement);

ConnectionFactory.*close*(dbConnection);

}

}

**public** **static** List<Object> view(Object o) {

String viewStatementString = "SELECT \* FROM ";

List<Object> l = **new** ArrayList<Object>();

String table = o.getClass().getSimpleName().toLowerCase();

viewStatementString = viewStatementString.concat(table);

Connection dbConnection = ConnectionFactory.*getConnection*();

PreparedStatement viewStatement = **null**;

ResultSet rs = **null**;

**try** {

viewStatement = dbConnection.prepareStatement(viewStatementString);

rs = viewStatement.executeQuery();

**while** (rs.next()) {

**if** (table.equals("client")) {

**int** id = rs.getInt("idClient");

String name = rs.getString("name");

String address = rs.getString("address");

String phone = rs.getString("phone");

l.add(**new** Client(id, name, address, phone));

}

**if** (table.equals("orders")) {

**int** id = rs.getInt("idOrder");

**int** idClient = rs.getInt("idClient");

**int** idProduct = rs.getInt("idProduct");

**int** idOrderType = rs.getInt("idOrderType");

**float** price = rs.getFloat("price");

**int** quantity = rs.getInt("quantity");

l.add(**new** Orders(id, idClient, idProduct, idOrderType, price, quantity));

}

**if** (table.equals("ordertype")) {

**int** id = rs.getInt("idOrderType");

String name = rs.getString("name");

l.add(**new** OrderType(id, name));

}

**if** (table.equals("product")) {

**int** id = rs.getInt("idProduct");

String name = rs.getString("name");

**float** nominalPrice = rs.getFloat("nominalPrice");

l.add(**new** Product(id, name, nominalPrice));

}

**if** (table.equals("stock")) {

**int** id = rs.getInt("idProduct");

**int** quantity = rs.getInt("quantity");

l.add(**new** Stock(id, quantity));

}

}

} **catch** (SQLException e) {

} **finally** {

ConnectionFactory.*close*(rs);

ConnectionFactory.*close*(viewStatement);

ConnectionFactory.*close*(dbConnection);

}

**return** l;

}

}

1. BLLs. I will give only one example.

**public** **class** OrderBLL {

**private** List<Validator<Object>> validators;

**public** OrderBLL() {

validators = **new** ArrayList<Validator<Object>>();

validators.add(**new** IdValidator());

validators.add(**new** OrderValidator());

}

**public** Orders findOrderById(**int** id) {

Orders a = **new** Orders(id);

Orders st = (Orders) DAO.*findById*(a);

**if** (st == **null**) {

**throw** **new** NoSuchElementException("The order with id =" + id + " was not found!");

}

**return** st;

}

**public** **int** insertOrder(Orders order) {

**for** (Validator<Object> v : validators) {

v.validate(order);

}

**int** id = DAO.*insert*(order);

**if** (id != -1) {

**int** quant = order.getQuantity();

OrderType s = (OrderType) DAO.*findById*(**new** OrderType(order.getIdOrderType()));

**if** (s.getName().contains("sell")) {

quant = -quant;

}

Stock a = **new** Stock(order.getIdProduct());

a = (Stock) DAO.*findById*(a);

a.setQuantity(a.getQuantity() + quant);

DAO.*update*(a);

**return** 1;

}

**return** -1;

}

**public** **int** deleteOrder(**int** id) {

Orders s = findOrderById(id);

**if** (s != **null**) {

**return** DAO.*delete*(s);

} **else** {

**return** -1;

}

}

**public** List<Object> displayOrders() {

Orders o = **new** Orders(0);

**return** DAO.*view*(o);

}

**public** **int** updateOrder(**int** idO, **int** idC, **int** idP, **int** idOT, **float** price, **int** quantity) {

Orders c = **new** Orders(idO, idC, idP, idOT, price, quantity);

**return** DAO.*update*(c);

}

**public** **int** createBill(String id, Orders order) {

Document document = **new** Document();

**try** {

PdfWriter writer = PdfWriter.*getInstance*(document, **new** FileOutputStream("OrderBill" + id + ".pdf"));

document.open();

Client c = (Client) DAO.*findById*(**new** Client(order.getIdClient()));

Font redFont = FontFactory.*getFont*(FontFactory.***COURIER***, 30, Font.***BOLD***, **new** CMYKColor(40, 40, 40, 100));

Paragraph title = **new** Paragraph("Congratulation, " + c.getName() + "!\n\n\n", redFont);

title.setAlignment(Element.***ALIGN\_CENTER***);

document.add(title);

StringBuilder a = **new** StringBuilder();

a.append(" You");

OrderType ot = (OrderType) DAO.*findById*(**new** OrderType(order.getIdOrderType()));

**if** (ot.getName().contains("sell")) {

a.append(" are the proud owner of ");

} **else** {

a.append(" have sold us ");

}

a.append(order.getQuantity() + " ");

Product p = (Product) DAO.*findById*(**new** Product(order.getIdProduct()));

a.append(p.getName() + "s at the humble price of only " + order.getPrice()

+ " a piece. We hope that you found this trade deal as atractive as we have found it and that you will look forward to other such deals with us.");

document.add(**new** Paragraph(a.toString()));

Paragraph ending = **new** Paragraph("\n\n\n\nBest regards,\nSimple Warehouse\n" + **new** Date());

document.add(ending);

document.addAuthor("Indre Bogdan");

document.addCreationDate();

document.addTitle("OrderBill" + id);

document.close();

writer.close();

} **catch** (DocumentException e) {

e.printStackTrace();

} **catch** (FileNotFoundException e) {

**return** -1;

}

**return** 1;

}

**public** **void** openBill(String id) **throws** IOException {

File file = **new** File("C:\\Users\\Asus\\Desktop\\oop\\HW3\\HW3\\OrderBill" + id + ".pdf");

Desktop desktop = Desktop.*getDesktop*();

**if** (file.exists())

desktop.open(file);

}

}

1. GUI
2. **Results**

There are no JUnit tests required in this homework.

1. **Conclusions**

An interesting exercise in reflection and database connections, also, pdf writing. There is room for many improvements such as a more extended database, an automatization of the warehouse (being able to simulate order placement and deals while remaining on profit) and a better GUI for the user.

1. **Biography**

-the pdf given in the laboratory;

<https://howtodoinjava.com/apache-commons/read-generate-pdf-java-itext/> for the pdf writing.