**DOCUMENTATION**

**ASSIGNMENT *3***

**CONTENTS**

1. [Assignment Objective](file:///C:\Users\Tudor\Downloads\PT2023_Documentation_Template_EN%20(1).doc%23_Toc128043139) [3](file:///C:\Users\Tudor\Downloads\PT2023_Documentation_Template_EN%20(1).doc%23_Toc128043139)

2. [Problem Analysis, Modeling, Scenarios, Use Cases](file:///C:\Users\Tudor\Downloads\PT2023_Documentation_Template_EN%20(1).doc%23_Toc128043140) [3](file:///C:\Users\Tudor\Downloads\PT2023_Documentation_Template_EN%20(1).doc%23_Toc128043140)

3. [Design](file:///C:\Users\Tudor\Downloads\PT2023_Documentation_Template_EN%20(1).doc%23_Toc128043141) [5](file:///C:\Users\Tudor\Downloads\PT2023_Documentation_Template_EN%20(1).doc%23_Toc128043141)

4. [Implementation](file:///C:\Users\Tudor\Downloads\PT2023_Documentation_Template_EN%20(1).doc%23_Toc128043142) [6](file:///C:\Users\Tudor\Downloads\PT2023_Documentation_Template_EN%20(1).doc%23_Toc128043142)

5. [Results](file:///C:\Users\Tudor\Downloads\PT2023_Documentation_Template_EN%20(1).doc%23_Toc128043143) [15](file:///C:\Users\Tudor\Downloads\PT2023_Documentation_Template_EN%20(1).doc%23_Toc128043143)

6. [Conclusions](file:///C:\Users\Tudor\Downloads\PT2023_Documentation_Template_EN%20(1).doc%23_Toc128043144) [18](file:///C:\Users\Tudor\Downloads\PT2023_Documentation_Template_EN%20(1).doc%23_Toc128043144)

7. [Bibliography](file:///C:\Users\Tudor\Downloads\PT2023_Documentation_Template_EN%20(1).doc%23_Toc128043145) [18](file:///C:\Users\Tudor\Downloads\PT2023_Documentation_Template_EN%20(1).doc%23_Toc128043145)

1. **Assignment Objective**
2. ***The main objective***

The assignments’s main objective is to create an application that that is managing the clients, products and orders from a warehouse.

1. ***The sub-objectives***

• Analyze the problem and identify requirements

• Design the orders management application

• Implement the orders management application

• Test the orders management application

**2.Problem Analysis, Modeling, Scenarios, Use Cases**

1. ***Problem Analysis***

The application should perform the operations of inserting a client(id, name, address, age), a product(id, name, quantity, price), and placing an order(id, client id, product id, quantity).

1. ***Modeling***

We will have a main interface from which we can go to client operations(add, edit, delete client and view all inserted clients), product operations(add, edit, delete product and view all inserted products), order operations, where we place the orders, this operations will be three buttons. In client operations and product operations we will have 1 button for each operation. In add we will have one text field for each parameter. In edit we will have a combobox, in which we can select the id of the client or product that will suffer an edit, three textfields for the parameters that will be changed. In delete we will have one combobox which will give the id of the client that we will delete. In view product/clients we will have a Jtable that will store all clients/products that had been inserted. In order operations we will have a textfield for id, one combobox for the id of the client that will place the order and one combobo for the id of the product that will be bought, and one textfield for the quantity.

1. ***Scenarios***

We have only one scenario: the scenario where the user doesn’t insert numbers on the text fields where numbers should be inserted(id, age, quantity, price) and inserts other special characteres.

A picture containing drawing, text, diagram, design

Description automatically generated ***(IV) Use cases***

1. **Design**

The main class in the applications is the AbstractDao and the ConnectionFactory classes.

The ConnectionFactory is designed using singleton design pattern.

The ConnectionFactory class:  
package connection;  
  
import java.sql.Connection;  
import java.sql.DriverManager;  
import java.sql.ResultSet;  
import java.sql.SQLException;  
import java.sql.Statement;  
import java.util.logging.Level;  
import java.util.logging.Logger;  
  
*/\*\*  
 \* <p>ConnectionFactory is a class used to establish the connection between the database and the program</p>  
 \*/*public class ConnectionFactory {  
 private static final Logger *LOGGER* = Logger.*getLogger*(ConnectionFactory.class.getName());  
 private static final String *DRIVER* = "com.mysql.cj.jdbc.Driver";  
 private static final String *DBURL* = "jdbc:mysql://localhost:3306/warehouse"; //jdbc:mysql://localhost:3306  
 private static final String *USER* = "root";  
 private static final String *PASS* = "root";  
  
 private static ConnectionFactory *singleInstance* = new ConnectionFactory();  
  
 private ConnectionFactory() {  
 try {  
 Class.*forName*(*DRIVER*);  
 } catch (ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
 }  
  
 */\*\*  
 \* <p>It uses a Connection object which gets the value of the static method getConnection() from the DriverManager</p>  
 \** ***@return*** *a Connection object that establishes the connection to the database  
 \*/* private Connection createConnection() {  
 Connection connection = null;  
 try {  
 connection = DriverManager.*getConnection*(*DBURL*, *USER*, *PASS*);  
 } catch (SQLException e) {  
 *LOGGER*.log(Level.*WARNING*, "An error occured while trying to connect to the database");  
 e.printStackTrace();  
 }  
 return connection;  
 }  
  
 public static Connection getConnection() {  
 return *singleInstance*.createConnection();  
 }  
  
 */\*\*  
 \* <p>It closes the connection to the database</p>  
 \** ***@param*** *connection the connection to the database  
 \*/* public static void close(Connection connection) {  
 if (connection != null) {  
 try {  
 connection.close();  
 } catch (SQLException e) {  
 *LOGGER*.log(Level.*WARNING*, "An error occured while trying to close the connection");  
 }  
 }  
 }  
  
 */\*\*  
 \* It closes the statement  
 \** ***@param*** *statement the statement used in extracting/updating/removing data  
 \*/* public static void close(Statement statement) {  
 if (statement != null) {  
 try {  
 statement.close();  
 } catch (SQLException e) {  
 *LOGGER*.log(Level.*WARNING*, "An error occured while trying to close the statement");  
 }  
 }  
 }  
  
 */\*\*  
 \* <p>It closes the result set resulted from the execution of the statements</p>  
 \** ***@param*** *resultSet the corresponding result set  
 \*/* public static void close(ResultSet resultSet) {  
 if (resultSet != null) {  
 try {  
 resultSet.close();  
 } catch (SQLException e) {  
 *LOGGER*.log(Level.*WARNING*, "An error occured while trying to close the ResultSet");  
 }  
 }  
 }  
  
}

AbstractDao class stores the methods for inserting, removing, finding, updating items from the database. All function are based on the reflexive principle.

package dao;  
  
import connection.ConnectionFactory;  
  
import javax.swing.\*;  
import javax.swing.table.DefaultTableModel;  
import java.beans.IntrospectionException;  
import java.beans.PropertyDescriptor;  
import java.lang.reflect.\*;  
import java.security.PublicKey;  
import java.sql.Connection;  
import java.sql.PreparedStatement;  
import java.sql.ResultSet;  
import java.sql.SQLException;  
import java.util.ArrayList;  
import java.util.List;  
import java.util.logging.Level;  
import java.util.logging.Logger;  
  
public abstract class AbstractDAO<T> {  
 protected static final Logger *LOGGER* = Logger.*getLogger*(AbstractDAO.class.getName());  
 protected int identifier;  
 private final Class<T> type;  
  
 protected AbstractDAO() {  
 this.type = (Class<T>) ((ParameterizedType) getClass().getGenericSuperclass()).getActualTypeArguments()[0];  
 }  
  
 */\*\*  
 \*<p>It creates a select query based on a field and returns it.</p>  
 \** ***@param*** *field the field that is being tested inside the query.  
 \** ***@return*** *the query that is to be executed on the database  
 \*/* private String createSelectQuery(String field,int id) {  
 StringBuilder sb = new StringBuilder();  
 sb.append("SELECT \* FROM ");  
 sb.append(type.getSimpleName());  
 sb.append(" WHERE ").append(field).append(" = ").append(id);  
 return sb.toString();  
 }  
 private String createDeleteQuery(String field, int id){  
 StringBuilder sb = new StringBuilder();  
 sb.append("DELETE FROM ");  
 sb.append(type.getSimpleName());  
 sb.append(" WHERE ").append(field).append(" = ").append(id);  
 return sb.toString();  
 }  
 protected abstract String createInsertStatement(T t);  
 protected abstract String createUpdateStatement(T t);  
 private List<T> createObjects(ResultSet resultSet) {  
 List<T> list = new ArrayList<T>();  
 Constructor[] ctors = type.getDeclaredConstructors();  
 Constructor ctor = null;  
 for (int i = 0; i < ctors.length; i++) {  
 ctor = ctors[i];  
 if (ctor.getGenericParameterTypes().length == 0)  
 break;  
 }  
 try {  
 while (resultSet.next()) {  
 ctor.setAccessible(true);  
 T instance = null;  
 if(identifier == 1) {  
 instance = (T) ctor.newInstance(0, null, null, 0);  
 }  
 if (identifier == 2){  
 instance = (T) ctor.newInstance(0, "", 0, 0d);  
 }  
 if (identifier==3){  
 instance = (T) ctor.newInstance(0, 0, 0,0, 0d);  
 }  
  
 for (Field field : type.getDeclaredFields()) {  
 String fieldName = field.getName();  
 Object value = resultSet.getObject(fieldName);  
 PropertyDescriptor propertyDescriptor = new PropertyDescriptor(fieldName, type);  
 Method method = propertyDescriptor.getWriteMethod();  
 method.invoke(instance, value);  
 }  
 list.add(instance);  
 }  
 } catch (InstantiationException | IntrospectionException | SQLException | InvocationTargetException | IllegalArgumentException | SecurityException | IllegalAccessException e) {  
 e.printStackTrace();  
 }  
 return list;  
 }  
 */\*\*  
 \* <p>It creates a query based on the createSelectQuery() with the field "id" and sets the unknown parameter ? into the integer id and then creates an object of type T</p>  
 \** ***@param*** *id the id that is to be inserted into the query  
 \** ***@return*** *an object of type T with the id equal to the parameter  
 \*/* public T findById(int id) {  
 Connection connection = null;  
 PreparedStatement statement = null;  
 ResultSet resultSet = null;  
 String query = createSelectQuery("id",id);  
  
 try {  
 connection = ConnectionFactory.*getConnection*();  
 statement = connection.prepareStatement(query);  
 resultSet = statement.executeQuery();  
 return createObjects(resultSet).get(0);  
 } catch (SQLException | IndexOutOfBoundsException e) {  
 *LOGGER*.log(Level.*WARNING*, type.getName() + "DAO:findById " + e.getMessage());  
 } finally {  
 ConnectionFactory.*close*(resultSet);  
 ConnectionFactory.*close*(statement);  
 ConnectionFactory.*close*(connection);  
 }  
 return null;  
 }  
 */\*\*  
 \* <p>Based on the result set, it creates objects of type T using reflection, by calling constructors, PropertyDescriptors and Methods. It uses the instance variable identifier to create a specific kind of Instance</p>  
 \** ***@return*** *the list of newly created objects  
 \*/* public List<T> findALL(){  
 Connection connection = null;  
 PreparedStatement statement = null;  
 ResultSet resultSet = null;  
 String query = "SELECT \* from "+ this.type.getSimpleName();  
  
 try {  
 connection = ConnectionFactory.*getConnection*();  
 statement = connection.prepareStatement(query);  
 resultSet = statement.executeQuery();  
 return createObjects(resultSet);  
 } catch (SQLException | IndexOutOfBoundsException e) {  
 *LOGGER*.log(Level.*WARNING*, type.getName() + "DAO:findById " + e.getMessage());  
 } finally {  
 ConnectionFactory.*close*(resultSet);  
 ConnectionFactory.*close*(statement);  
 ConnectionFactory.*close*(connection);  
 }  
 return null;  
 }  
 public void remove(int id){  
 Connection connection = null;  
 PreparedStatement statement = null;  
 ResultSet resultSet = null;  
 String query=createDeleteQuery("id",id);  
 try {  
 connection = ConnectionFactory.*getConnection*();  
 statement = connection.prepareStatement(query);  
 statement.executeUpdate();  
  
 } catch (SQLException e) {  
 *LOGGER*.log(Level.*WARNING*, type.getName() + "DAO:remove " + e.getMessage());  
 } finally {  
 ConnectionFactory.*close*(resultSet);  
 ConnectionFactory.*close*(statement);  
 ConnectionFactory.*close*(connection);  
 }  
  
 }  
  
 public void upate(T t){  
 Connection connection = null;  
 PreparedStatement statement = null;  
 ResultSet resultSet = null;  
 String query=createUpdateStatement(t);  
 try {  
 connection = ConnectionFactory.*getConnection*();  
 statement = connection.prepareStatement(query);  
 statement.executeUpdate();  
  
 } catch (SQLException e) {  
 *LOGGER*.log(Level.*WARNING*, type.getName() + "DAO:update " + e.getMessage());  
 } finally {  
 ConnectionFactory.*close*(resultSet);  
 ConnectionFactory.*close*(statement);  
 ConnectionFactory.*close*(connection);  
 }  
 }  
  
 public void insert(T t){  
 Connection connection = null;  
 PreparedStatement statement = null;  
 ResultSet resultSet = null;  
 String query=createInsertStatement(t);  
 try{  
 connection=ConnectionFactory.*getConnection*();  
 statement=connection.prepareStatement(query);  
 statement.executeUpdate();  
 }catch (SQLException e) {  
 *LOGGER*.log(Level.*WARNING*, type.getName() + "DAO:insert " + e.getMessage());  
 } finally {  
 ConnectionFactory.*close*(resultSet);  
 ConnectionFactory.*close*(statement);  
 ConnectionFactory.*close*(connection);  
 }  
 }  
  
 public JTable createTable(List<T> t) throws IllegalAccessException {  
 JTable jTable;  
 Class<T> tClass= (Class<T>) t.get(0).getClass();  
  
 Field[] fields=tClass.getDeclaredFields();  
 String[] column=new String[fields.length];  
  
 for(int i=0;i<fields.length;i++)  
 {  
 column[i]=fields[i].getName();  
 }  
  
 Object[][] objects=new Object[t.size()][fields.length];  
 for(int i=0;i<t.size();i++){  
 Object obj=t.get(i);  
 for(int j=0;j<fields.length;j++){  
 Field field=fields[j];  
 field.setAccessible(true);  
 Object value=field.get(obj);  
 objects[i][j]=value;  
 }  
 }  
 DefaultTableModel tableModel=new DefaultTableModel(objects,column);  
  
 jTable=new JTable(tableModel);  
 return jTable;  
 }  
  
}

The uml diagram:

A screenshot of a computer screen

Description automatically generated with low confidence

1. **Implementation**
2. ***Clients Class***

package model;  
  
public class Clients {  
 private Integer id;  
 private String name;  
 private String address;  
 private Integer age;  
  
 public Clients(Integer id, String name, String address, Integer age) {  
 this.id = id;  
 this.name = name;  
 this.address = address;  
 this.age = age;  
 }  
  
 public void setId(Integer id) {  
 this.id = id;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
  
 public void setAddress(String address) {  
 this.address = address;  
 }  
  
 public void setAge(Integer age) {  
 this.age = age;  
 }  
  
 public Integer getId() {  
 return id;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public String getAddress() {  
 return address;  
 }  
  
 public Integer getAge() {  
 return age;  
 }  
}

1. ***Products Class***

package model;  
  
public class Products {  
 private Integer id;  
 private String name;  
 private Integer quantity;  
 private Double price;  
  
 public Products(Integer id, String name, Integer quantity, Double price) {  
 this.id = id;  
 this.name = name;  
 this.quantity = quantity;  
 this.price = price;  
 }  
  
 public void setId(Integer id) {  
 this.id = id;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
  
 public void setQuantity(Integer quantity) {  
 this.quantity = quantity;  
 }  
  
 public void setPrice(Double price) {  
 this.price = price;  
 }  
  
 public Integer getId() {  
 return id;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public Integer getQuantity() {  
 return quantity;  
 }  
  
 public Double getPrice() {  
 return price;  
 }  
}

1. ***Order Class***

package model;  
  
public class Orders {  
 private Integer id;  
  
 private Integer IDclient;  
  
 private Integer IDproduct;  
 private Integer quantity;  
 private Double price;  
  
 public Orders(Integer id, Integer IDclient, Integer IDproduct, Integer quantity, Double price) {  
 this.id = id;  
 this.IDclient = IDclient;  
 this.IDproduct = IDproduct;  
 this.quantity = quantity;  
 this.price = price;  
 }  
  
 public void setId(Integer id) {  
 this.id = id;  
 }  
  
 public void setIDclient(Integer IDclient) {  
 this.IDclient = IDclient;  
 }  
  
 public void setIDproduct(Integer IDproduct) {  
 this.IDproduct = IDproduct;  
 }  
  
 public void setQuantity(Integer quantity) {  
 this.quantity = quantity;  
 }  
  
 public void setPrice(Double price) {  
 this.price = price;  
 }  
  
 public Integer getId() {  
 return id;  
 }  
  
 public Integer getIDclient() {  
 return IDclient;  
 }  
  
 public Integer getIDproduct() {  
 return IDproduct;  
 }  
  
 public Integer getQuantity() {  
 return quantity;  
 }  
  
 public Double getPrice() {  
 return price;  
 }  
}

1. ***Bill record***

package model;  
  
public record Bill(Integer id, Integer IDclient, Integer IDproduct, Integer quantity, Double price) {  
  
}

1. ***Clients DAO***

package dao;  
  
import model.Clients;  
  
public class ClientsDAO extends AbstractDAO<Clients> {  
 public ClientsDAO(){  
 this.identifier=1;  
 }  
 @Override  
 protected String createInsertStatement(Clients clients) {  
 StringBuilder sb=new StringBuilder();  
 sb.append("insert into clients values( ");  
 sb.append(clients.getId()).append(",");  
 sb.append("'").append(clients.getName()).append("',");  
 sb.append("'").append(clients.getAddress()).append("',");  
 sb.append(clients.getAge()).append(")");  
  
 return sb.toString();  
 }  
  
 @Override  
 protected String createUpdateStatement(Clients clients) {  
 StringBuilder sb=new StringBuilder();  
 sb.append("UPDATE clients SET name='").append(clients.getName()).append("'");  
 sb.append(", address='").append(clients.getAddress()).append("'");  
 sb.append(", age=").append(clients.getAge());  
 sb.append(" where id=").append(clients.getId());  
 return sb.toString();  
 }  
}

1. ***Products DAO***

package dao;  
  
import model.Orders;  
  
public class OrdersDAO extends AbstractDAO<Orders>{  
  
 public OrdersDAO(){  
 this.identifier=3;  
 }  
 @Override  
 protected String createInsertStatement(Orders orders) {  
 StringBuilder sb=new StringBuilder();  
 sb.append("insert into orders values( ");  
 sb.append(orders.getId()).append(",");  
 sb.append(orders.getIDclient()).append(",");  
 sb.append(orders.getIDproduct()).append(",");  
 sb.append(orders.getQuantity()).append(",");  
 sb.append(orders.getPrice()).append(")");  
 return sb.toString();  
 }  
  
 @Override  
 protected String createUpdateStatement(Orders orders) {  
 return null;  
 }  
}

1. ***Orders DAO***

package dao;  
  
import model.Orders;  
  
public class OrdersDAO extends AbstractDAO<Orders>{  
  
 public OrdersDAO(){  
 this.identifier=3;  
 }  
 @Override  
 protected String createInsertStatement(Orders orders) {  
 StringBuilder sb=new StringBuilder();  
 sb.append("insert into orders values( ");  
 sb.append(orders.getId()).append(",");  
 sb.append(orders.getIDclient()).append(",");  
 sb.append(orders.getIDproduct()).append(",");  
 sb.append(orders.getQuantity()).append(",");  
 sb.append(orders.getPrice()).append(")");  
 return sb.toString();  
 }  
  
 @Override  
 protected String createUpdateStatement(Orders orders) {  
 return null;  
 }  
}

1. ***Clients BLL***

package bll;  
  
import dao.ClientsDAO;  
import gui.AddNewClient;  
import gui.DeleteClient;  
import gui.EditClient;  
import model.Clients;  
  
import javax.swing.\*;  
import javax.swing.table.DefaultTableModel;  
import java.util.ArrayList;  
import java.util.List;  
  
public class ClientsBLL {  
  
 private ClientsDAO clientsDAO;  
  
 public ClientsBLL() {  
 this.clientsDAO = new ClientsDAO();  
 }  
  
 public static int checkInput(String tocheck1, String tocheck2, String tocheck3, String tocheck4, AddNewClient addNewClient){  
  
 if(!tocheck1.matches("\\d+") || !tocheck4.matches("\\d+")){  
 JOptionPane.*showMessageDialog*(addNewClient,"AGE and ID fields have to be integers","Try Again",JOptionPane.*ERROR\_MESSAGE*);  
 return 0;  
 }  
 if(tocheck1.isEmpty() || tocheck2.isEmpty() || tocheck3.isEmpty() || tocheck4.isEmpty()){  
 JOptionPane.*showMessageDialog*(addNewClient,"All fields have to be inserted","Try Again",JOptionPane.*ERROR\_MESSAGE*);  
 return 0;  
 }  
 return 1;  
 }  
 public static int checkEditInput(String tocheck1, String tocheck2, String tocheck3, EditClient editClient){  
  
 if( !tocheck3.matches("\\d+") && !tocheck3.isEmpty()){  
 JOptionPane.*showMessageDialog*(editClient,"AGE field has to be integers","Try Again",JOptionPane.*ERROR\_MESSAGE*);  
 return 0;  
 }  
 if(tocheck1.isEmpty() && tocheck2.isEmpty() && tocheck3.isEmpty() ){  
 JOptionPane.*showMessageDialog*(editClient,"At least one field has to changed","Try Again",JOptionPane.*ERROR\_MESSAGE*);  
 return 0;  
 }  
 return 1;  
 }  
 public void insertClient(Clients clients,AddNewClient addNewClient){  
 //Clients clients=new Clients(id,name,address,age);  
 Clients clientExists;  
 clientExists=clientsDAO.findById(clients.getId());  
 if(clientExists==null) {  
 clientsDAO.insert(clients);  
 JOptionPane.*showMessageDialog*(addNewClient,"Client added");  
 }  
 else {  
 JOptionPane.*showMessageDialog*(addNewClient,"This ID already exists","Try Again",JOptionPane.*ERROR\_MESSAGE*);  
  
 }  
 }  
 public ArrayList<Integer> getClientsIDs(){  
 List<Clients> clients=clientsDAO.findALL();  
 ArrayList<Integer> clientsIDs=new ArrayList<>();  
 for(Clients clients1: clients){  
 clientsIDs.add(clients1.getId());  
 }  
 return clientsIDs;  
 }  
  
 public void deleteClient(int id, DeleteClient deleteClient){  
 clientsDAO.remove(id);  
 JOptionPane.*showMessageDialog*(deleteClient,"Client successfully deleted");  
 }  
  
 public Clients findClient(int id){  
 Clients clients=clientsDAO.findById(id);  
 return clients;  
 }  
 public void updateClient(Clients clients, EditClient editClient){  
 clientsDAO.upate(clients);  
 JOptionPane.*showMessageDialog*(editClient,"Client suffered a change");  
 }  
  
 public JTable viewProducts() throws IllegalAccessException {  
 List<Clients> clients=clientsDAO.findALL();  
 ClientsDAO clientsDAO1=new ClientsDAO();  
  
 return clientsDAO1.createTable(clients);  
 }  
  
}

1. ***Products BLL***

package bll;  
  
import dao.ClientsDAO;  
import dao.ProductsDAO;  
import gui.\*;  
import model.Clients;  
import model.Products;  
  
import javax.swing.\*;  
import java.util.ArrayList;  
import java.util.List;  
  
public class ProductsBLL {  
 private ProductsDAO productsDAO;  
  
 public ProductsBLL() {  
 productsDAO=new ProductsDAO();  
 }  
 public static int checkInput(String tocheck1, String tocheck2, String tocheck3, String tocheck4, AddNewProduct addNewProduct){  
  
 if(!tocheck1.matches("\\d+") || !tocheck3.matches("\\d+") || !tocheck4.matches("\\d+(\\.\\d+)?$")){  
 JOptionPane.*showMessageDialog*(addNewProduct,"ID and Quantity fields have to be integers and price have to be double or integer","Try Again",JOptionPane.*ERROR\_MESSAGE*);  
 return 0;  
 }  
 if(tocheck1.isEmpty() || tocheck2.isEmpty() || tocheck3.isEmpty() || tocheck4.isEmpty()){  
 JOptionPane.*showMessageDialog*(addNewProduct,"All fields have to be inserted","Try Again",JOptionPane.*ERROR\_MESSAGE*);  
 return 0;  
 }  
 return 1;  
 }  
  
 public static int checkEditInput(String tocheck1, String tocheck2, String tocheck3, EditProduct editProduct){  
  
 if( !tocheck2.matches("\\d+") && !tocheck2.isEmpty()){ //|| !tocheck3.matches("\\d+(\\.\\d+)?$")){  
 JOptionPane.*showMessageDialog*(editProduct," Quantity field has to be integers and price have to be double or integer","Try Again",JOptionPane.*ERROR\_MESSAGE*);  
 return 0;  
 }  
 if(!tocheck3.matches("\\d+(\\.\\d+)?$") && !tocheck3.isEmpty()){  
 JOptionPane.*showMessageDialog*(editProduct," Quantity field has to be integers and price have to be double or integer","Try Again",JOptionPane.*ERROR\_MESSAGE*);  
 return 0;  
 }  
 if(tocheck1.isEmpty() && tocheck2.isEmpty() && tocheck3.isEmpty() && tocheck3.isEmpty()){  
 JOptionPane.*showMessageDialog*(editProduct,"At least one field has to be inserted","Try Again",JOptionPane.*ERROR\_MESSAGE*);  
 return 0;  
 }  
 return 1;  
 }  
 public void insertProduct(Products products, AddNewProduct addNewProduct){  
 //Clients clients=new Clients(id,name,address,age);  
 Products productsExists;  
 productsExists=productsDAO.findById(products.getId());  
 if(productsExists==null) {  
 productsDAO.insert(products);  
 JOptionPane.*showMessageDialog*(addNewProduct,"Product added added");  
 }  
 else {  
 JOptionPane.*showMessageDialog*(addNewProduct,"This ID already exists","Try Again",JOptionPane.*ERROR\_MESSAGE*);  
  
 }  
 }  
 public ArrayList<Integer> getProductIDs(){  
 List<Products> products=productsDAO.findALL();  
 ArrayList<Integer> productsIDs=new ArrayList<>();  
 for(Products products1: products){  
 productsIDs.add(products1.getId());  
 }  
 return productsIDs;  
 }  
  
  
  
 public void deleteProduct(int id, DeleteProduct deleteProduct){  
 productsDAO.remove(id);  
 JOptionPane.*showMessageDialog*(deleteProduct,"Product successfully deleted");  
 }  
 public Products findProduct(int id){  
 Products products=productsDAO.findById(id);  
 return products;  
 }  
  
 public void updateProduct(Products products, EditProduct editProduct){  
 productsDAO.upate(products);  
 JOptionPane.*showMessageDialog*(editProduct,"Product suffered a change");  
 }  
 public void updateProduct2(Products products, OrdersOperations ordersOperations){  
 productsDAO.upate(products);  
 JOptionPane.*showMessageDialog*(ordersOperations,"The stock of the product has been updated");  
 }  
 public JTable viewProducts() throws IllegalAccessException {  
 List<Products> clients=productsDAO.findALL();  
 ProductsDAO productsDAO1=new ProductsDAO();  
  
 return productsDAO1.createTable(clients);  
 }  
}

1. ***Orders BLL***

package bll;  
  
import dao.OrdersDAO;  
import gui.AddNewClient;  
import gui.DeleteClient;  
import gui.OrdersOperations;  
import model.Clients;  
import model.Orders;  
  
import javax.swing.\*;  
import java.util.List;  
  
public class OrdersBLL {  
 OrdersDAO ordersDAO;  
  
 public OrdersBLL() {  
 this.ordersDAO = new OrdersDAO();  
 }  
  
 public static int checkInput(String toCheck1, String tocheck2, OrdersOperations ordersOperations){  
 if(toCheck1.isEmpty() || tocheck2.isEmpty()){  
 JOptionPane.*showMessageDialog*(ordersOperations,"All fields have to be inserted","try again",JOptionPane.*ERROR\_MESSAGE*);  
 return 0;  
 }  
 if(!toCheck1.matches("\\d+") || !tocheck2.matches("\\d+")){  
 JOptionPane.*showMessageDialog*(ordersOperations,"ID and quantity fields have to be integers","Try Again",JOptionPane.*ERROR\_MESSAGE*);  
 return 0;  
 }  
 return 1;  
 }  
 public int insertOrder(Orders orders, OrdersOperations ordersOperations){  
  
 Orders ordersExsists;  
 ordersExsists=ordersDAO.findById(orders.getId());  
 if(ordersExsists==null) {  
 ordersDAO.insert(orders);  
 JOptionPane.*showMessageDialog*(ordersOperations,"Order placed");  
 return 1;  
 }  
 else {  
 JOptionPane.*showMessageDialog*(ordersOperations,"This ID already exists","Try Again",JOptionPane.*ERROR\_MESSAGE*);  
 return 0;  
 }  
 }  
  
 public void deleteOrderFromClient(int id){  
 List<Orders> ordersList=ordersDAO.findALL();  
 for(Orders orders: ordersList){  
 if(orders.getIDclient()==id){  
 ordersDAO.remove(orders.getId());  
 }  
 }  
 }  
 public void deleteOrderFromProduct(int id){  
 List<Orders> ordersList=ordersDAO.findALL();  
 for(Orders orders: ordersList){  
 if(orders.getIDproduct()==id){  
 ordersDAO.remove(orders.getId());  
 }  
 }  
 }  
}

1. ***The GUI***

A picture containing text, screenshot, font, design

Description automatically generated

A screenshot of a computer

Description automatically generated with low confidence

A picture containing text, screenshot, font, number

Description automatically generated

A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a login form

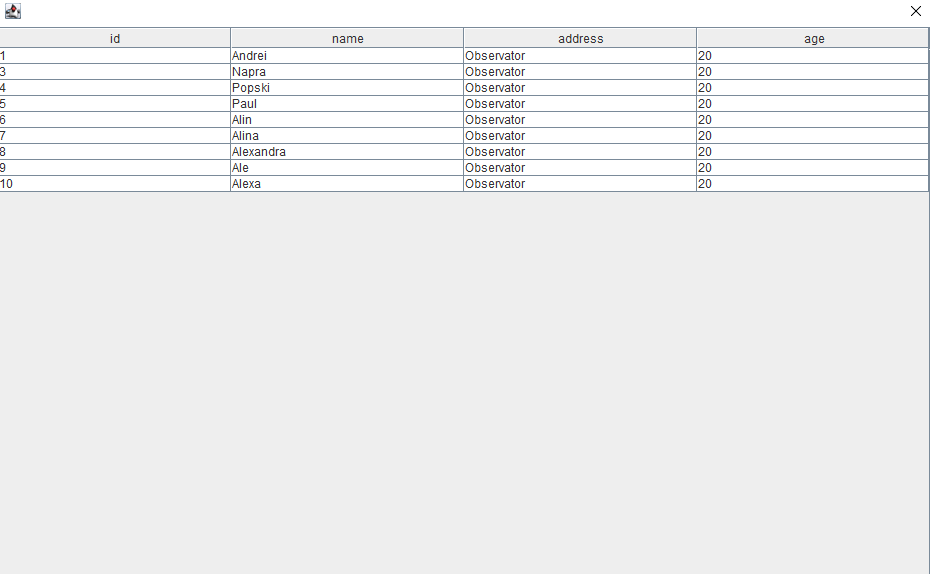
Description automatically generated with low confidence

A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with medium confidence



A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with medium confidence

Some messages

A screenshot of a computer error

Description automatically generated with low confidence

A screen shot of a computer error

Description automatically generated with low confidence

A screenshot of a computer error

Description automatically generated with medium confidence

1. **Results**

The results can be seen in the data base after in which insertion:

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with medium confidence

1. **Conclusions**

In conclusion, from this assignment, I learned about reflection techniques and how useful can be to not use repetitive code. I also learned about singleton pattern design and database connection.