Order Management

Documentatie

Nume: Ungurianu Daniel

Grupa 30228 An 2 semestrul al doilea

Disciplina: Tehnici de programare

Profesor indrumator: Cristina Bianca Pop

Cuprins

[1.Obiective 3](file:///C:\Users\danie\Downloads\Order_manager_VATAFU_30228.docx#_Toc70588731)

[1.1 Obiectiv principal 3](file:///C:\Users\danie\Downloads\Order_manager_VATAFU_30228.docx#_Toc70588732)

[1.2 Obiective secundare 3](file:///C:\Users\danie\Downloads\Order_manager_VATAFU_30228.docx#_Toc70588733)

[2.Analiza problemei,modelare,scenarii,cazuri de utilizare 3](file:///C:\Users\danie\Downloads\Order_manager_VATAFU_30228.docx#_Toc70588734)

[2.1 Analiza 3](file:///C:\Users\danie\Downloads\Order_manager_VATAFU_30228.docx#_Toc70588735)

[2.2 Cerințe funcționale 3](file:///C:\Users\danie\Downloads\Order_manager_VATAFU_30228.docx#_Toc70588736)

[2.3 Cerințe non-funcționale 3](file:///C:\Users\danie\Downloads\Order_manager_VATAFU_30228.docx#_Toc70588737)

[2.4 Diagrama Use-Case 4](file:///C:\Users\danie\Downloads\Order_manager_VATAFU_30228.docx#_Toc70588738)

[2.5 Scenarii de funcționare 4](file:///C:\Users\danie\Downloads\Order_manager_VATAFU_30228.docx#_Toc70588739)

[3.Proiectare 5](file:///C:\Users\danie\Downloads\Order_manager_VATAFU_30228.docx#_Toc70588740)

[3.1 Decizii de proiectare 5](file:///C:\Users\danie\Downloads\Order_manager_VATAFU_30228.docx#_Toc70588741)

[3.2 Diagrame de pachete și clase 6](file:///C:\Users\danie\Downloads\Order_manager_VATAFU_30228.docx#_Toc70588742)

[3.2.1. Diagrama de pachete 6](file:///C:\Users\danie\Downloads\Order_manager_VATAFU_30228.docx#_Toc70588743)

[3.2.2.Diagrama de clase 7](file:///C:\Users\danie\Downloads\Order_manager_VATAFU_30228.docx#_Toc70588744)

[3.3 Structuri de date și algoritmi 7](file:///C:\Users\danie\Downloads\Order_manager_VATAFU_30228.docx#_Toc70588746)

[3.4.Interfețe și pachete 7](file:///C:\Users\danie\Downloads\Order_manager_VATAFU_30228.docx#_Toc70588747)

[3.5 Interfața utilizator 8](file:///C:\Users\danie\Downloads\Order_manager_VATAFU_30228.docx#_Toc70588748)

[4.Implementarea 8](file:///C:\Users\danie\Downloads\Order_manager_VATAFU_30228.docx#_Toc70588749)

[4.1 Clasa Client 8](file:///C:\Users\danie\Downloads\Order_manager_VATAFU_30228.docx#_Toc70588750)

[4.2 Clasa Product 8](file:///C:\Users\danie\Downloads\Order_manager_VATAFU_30228.docx#_Toc70588751)

[4.3 Clasa Orders 8](file:///C:\Users\danie\Downloads\Order_manager_VATAFU_30228.docx#_Toc70588752)

[4.4 Clasa ConnectionFactory 8](file:///C:\Users\danie\Downloads\Order_manager_VATAFU_30228.docx#_Toc70588753)

[4.5 Clasa AbstractDAO 10](file:///C:\Users\danie\Downloads\Order_manager_VATAFU_30228.docx#_Toc70588754)

[4.6 Clasele ClientDAO, ProductDAO și OrdersDao 12](file:///C:\Users\danie\Downloads\Order_manager_VATAFU_30228.docx#_Toc70588755)

[4.7 Clasele ClientBLL,ProductBLL,OrderBLL 12](file:///C:\Users\danie\Downloads\Order_manager_VATAFU_30228.docx#_Toc70588756)

[4.8 Interfata grafica 12](file:///C:\Users\danie\Downloads\Order_manager_VATAFU_30228.docx#_Toc70588757)

[4.9 Clasa WriteToFile 13](file:///C:\Users\danie\Downloads\Order_manager_VATAFU_30228.docx#_Toc70588758)

[5.Testarea 14](file:///C:\Users\danie\Downloads\Order_manager_VATAFU_30228.docx#_Toc70588759)

[6.Concluzii 14](file:///C:\Users\danie\Downloads\Order_manager_VATAFU_30228.docx#_Toc70588760)

**1.Obiective**

**1.1 Obiectiv principal**

Obiectivul principal al acestei teme este proiectarea si implementarea unei aplicatii pentru gestionarea comenzilor din partea clientilor.

**1.2 Obiective secundare**

Printre obiectivele secundare ale acestei teme, se enumera:

Analiza problemei si modelarea ce are la baza, cazuri de utilizare, diagrame Use-Case, Proiectarea ce consta in decizii de proiectare, diagrame UML, clase, interfete, structure de date, GUI, algoritmi, packages, implementarea ce se caracterizeaza prin descrierea claselor si a metodelor si in final, Rezultatele ce sunt reprezentate prin seturi de valori.

**2.Analiza problemei, modelare, scenario, cazuri de utilizare**

**2.1 Analiza**

Pentru ducerea la bun sfarsit a acestei teme si implementarea corespunzatoare a aplicatiei, aceasta ar trebui sa indeplineasca cerintele de a modifica, afisa si a tine evidenta comenzilor, produselor si a clientilor, informatii ce vor fi stocate intr-o baza de date MySQL relationala.

**2.2 Cerinte functionale**

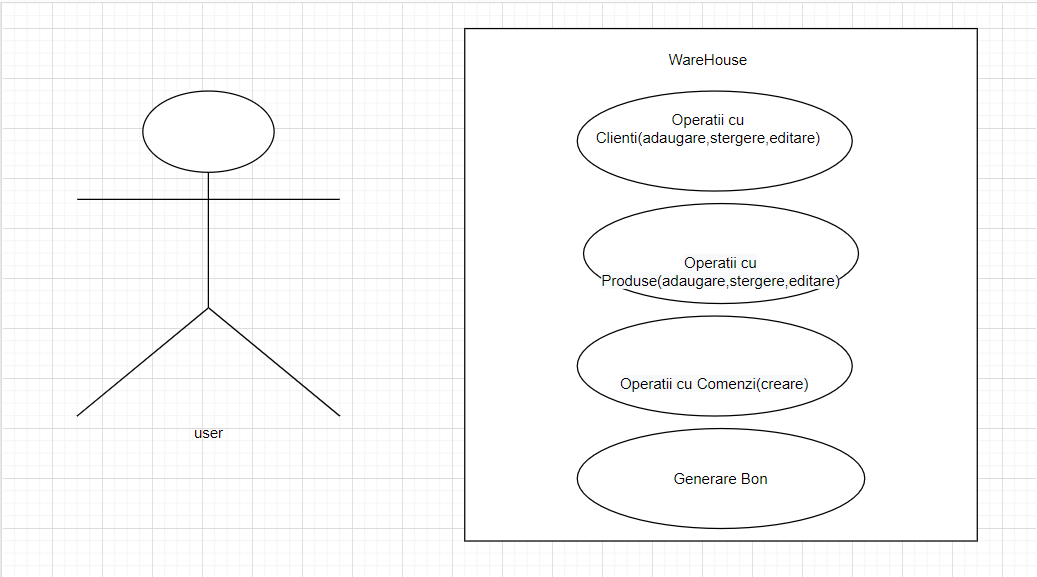
Este necesar ca utilizatorul sa aiba posibilitatea de a :

* Introduce produse in baza de date
* Introduce clienti in baza de date
* Efectua comenzi
* Vizualiza bonul in format text

**2.3 Cerinte non-functionale**

Este necesar sa fie implementata o interfata intuitiva si usor de folosit de catre utilizator, totodata sa se poata observa modificarile aduse bazei de date.

**2.4 Diagrama Use-Case**



Aceasta diagrama use-case evidentiaza principalele facilitate puse la dispozitie user-ului.

**2.5 Scenarii de functionare**

2.5.1) Pentru inceput, user-ul va introduce datele de recunoastere a unui client, dupa care va alege operatia pe care acesta doreste sa o efectueze: insert, update sau delete.

Pentru acest table, exista 2 scenarii posibile:

1. Scenariul cu succes

Valorile introduce de user sunt valide si operatia aleasa de acesta se efectueaza cu success.

1. Scenariul esuat

Valorile introduce de user nu sunt valide, asadar operatia aleasa de acesta va rezulta intr-o eroare

2.5.2) Pentru inceput, user-ul va introduce datele de recunoastere a unui produs, dupa care va alege operatia pe care acesta doreste sa o efectueze: insert, update sau delete.

Pentru acest table, exista 2 scenarii posibile:

1. Scenariul cu succes

Valorile introduce de user sunt valide si operatia aleasa de acesta se efectueaza cu success.

1. Scenariul esuat

Valorile introduce de user nu sunt valide, asadar operatia aleasa de acesta va rezulta intr-o eroare

2.5.3) Pentru inceput, user-ul va introduce datele despre o comanda, dupa care o va introduce in baza de date, iar singura operatie ce poate fi aleasa este create.

Pentru acest table, exista 2 scenarii posibile:

1. Scenariul cu succes

Valorile introduce de user sunt valide si operatia aleasa de acesta se efectueaza cu success.

1. Scenariul esuat

Valorile introduce de user nu sunt valide, asadar operatia aleasa de acesta va rezulta intr-o eroare

**3. Proiectare**

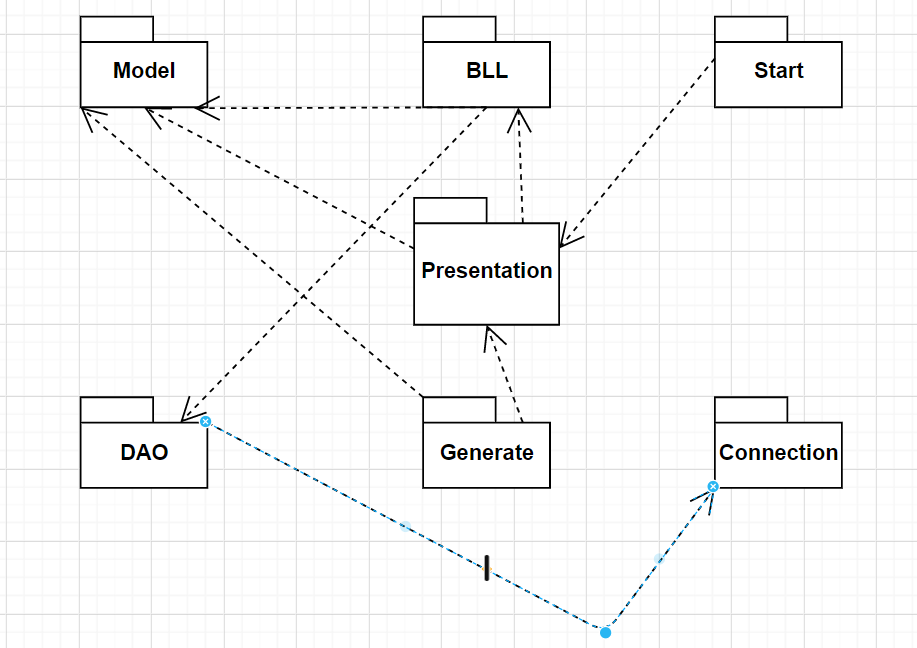
**3.1 Decizii de proiectare**

Sistemul de management al bazei de date a fost creat prin mecanismul de reflection care introduce date furnizate de la interfata de utilizator mai apoi trecute prin controller si ajunse in clasa AbstractDAO care in functie de clasa obiectului face modificarile cerute in baza de date.

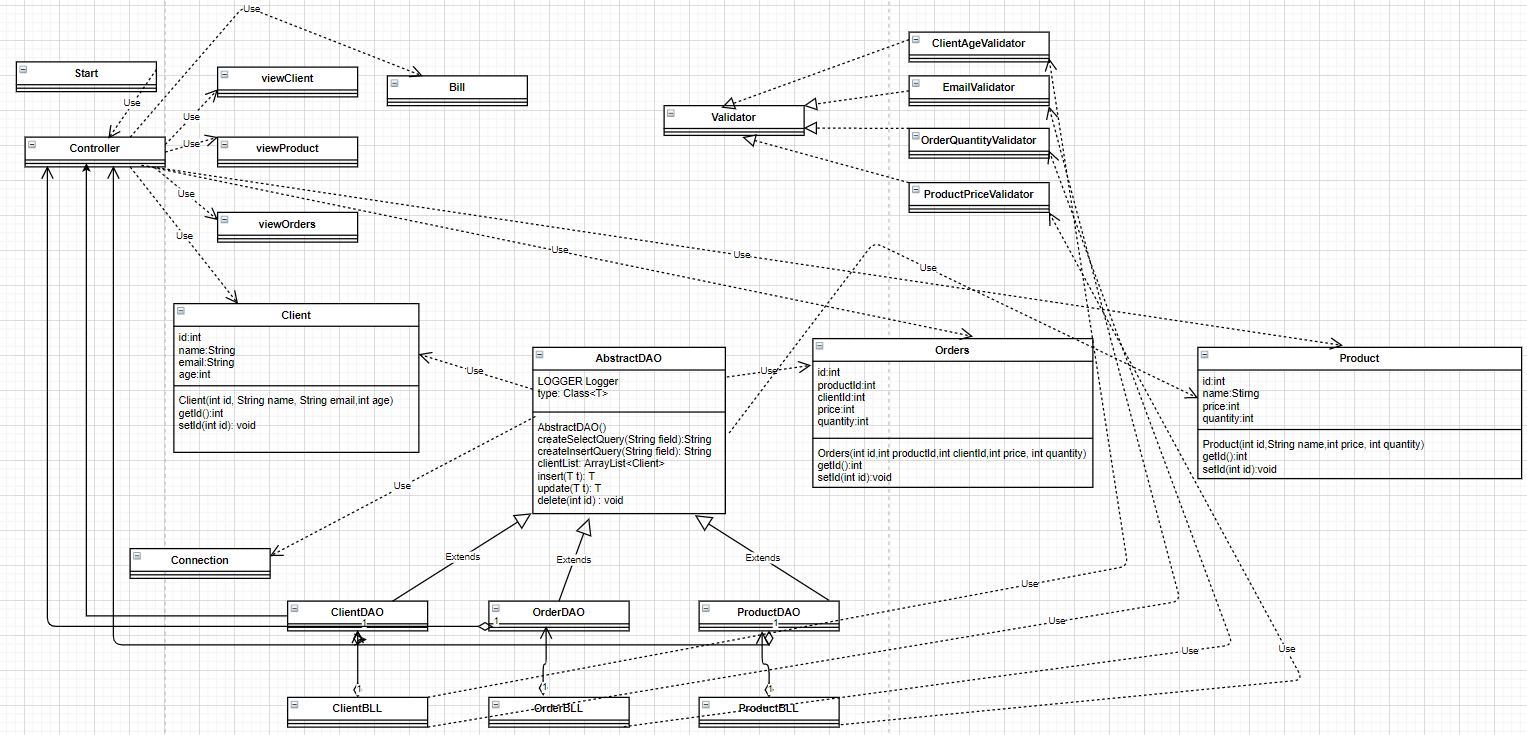
Inainte de a face aceste modificari aplicatia are un sistem de validare a datelor ProductBLL, OrderBLL,ClientBLL care asigura utilizatorul ca nu o sa fie lasat sa insereze lucruri eronate care ar putea sa afecteze sistemul de stocare a bazei de date si implicit altera datele U.C care foloseste aplicatia.

**3.2 Diagrame de pachete si clase**

**3.2.1 Diagrama de pachete**



**3.2.2 Diagrama de clase**



**3.3 Structuri de date si algoritmi**

In realizarea acestei aplicatii, am folosit liste (ce sunt esentiale pentru reflection pentru a lucre cu liste de atribute).

Pe langa elementele cu care am fost deja obisnuit de la temele anterioare, in aceasta tema am folosit un nou element: JTable, necesar pentru formarea tabelului.

**3.4 Interfete si pachete**

Principalele interfete folosite sunt ActionListener.

Pentru a avea posibilitatea de a lucra cu fisiere, am avut nevoie de java.io.FileWriter.

**3.5 Interfata utilizator**

Pentru interfata grafica am folosit un design simplu, usor de intuit.

Utilizatorului i se pun la dispozitie o serie de campuri, ce permin introducerea datelor, si butoane ce ofera posibilitatea realizarii operatiilor folosind datele introduce, cu ajutorul carora se vor seta campurile tabelelor.

**4. Implementarea**

**4.1 Clasa Client**

Atributele folosite pentru descrierea unui Client trebuie sa fie aceleasi cu cele din tabelul corespunzator din baza de date.

De ce este necesar ca aceste atribute sa coincide atat ca denumire, cat si la ordinea declararii? Duce la functionarea eficienta si corecta a tehnicii de reflection.

**4.2 Clasa Product**

Atributele folosite pentru descrierea unui Product trebuie sa fie aceleasi cu cele din tabelul corespunzator din baza de date.

De ce este necesar ca aceste atribute sa coincide atat ca denumire, cat si la ordinea declararii? Duce la functionarea eficienta si corecta a tehnicii de reflection.

**4.3 Clasa Orders**

Atributele folosite pentru descrierea unui Orders trebuie sa fie aceleasi cu cele din tabelul corespunzator din baza de date.

De ce este necesar ca aceste atribute sa coincide atat ca denumire, cat si la ordinea declararii? Duce la functionarea eficienta si corecta a tehnicii de reflection.

Totusi, aici intalnim un caz mai special, cand numele tabelei coincide cu un cuvant cheie din limbajul SQL : Order. Din aceasta cauza am fost nevoit sa aleg o alta denumire pentru acest table.

Pentru toate aceste 3 clase mentionate anterior am folosit cate 3 constructori pentru fiecare. Unul ce are in componenta si atributul id ( ce este folosit in AbstractDAO), unul ce contine restul atributelor, in afara de id(folosit in Controller.Listener) si un constructor fara parametri pentru a scapa de unele erori.

**4.4 Clasa ConnectionFactory**

Aceasta clasa realizeaza conexiunea cu baza de date. Pentru a crea o baza de date, am folosit aplicatia MySQL Workbench in care mi-am creat 3 tabele echivalente claselor mentionate mai sus.

Conexiunea cu baza de date trebuie deschisa si apoi inchisa de fiecare data.

private static ConnectionFactory *singleInstance* = new ConnectionFactory();  
  
 private ConnectionFactory() {  
 try {  
 Class.*forName*(*DRIVER*);  
 } catch (ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
 }  
  
 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();  
 }  
  
 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");  
 }  
 }  
 }  
  
 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");  
 }  
 }  
 }  
  
 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");  
 }  
 }  
 }  
}

**4.5 Clasa AbstractDAO**

In aceasta clasa se implementeaza principalele metode comune accesarii bazei de date, ce pot fi aplicate si pentru tabelele client, product si orders (insert, update, delete si select).

Tehnica reflection a facut posibila aceasta abordare.

Pentru o mai buna structurare am incercat sa separ crearea statement ului de executarea lui:

*/\*\*  
 \* Creaza interogarea de tip SELECT pentru un tabel  
 \** ***@param*** *field campurile care se doresc a fi selectate  
 \** ***@return*** *Este un obiect al clasei string care contine interogarea  
 \*/*private String createSelectQuery(String field) {  
 StringBuilder sb = new StringBuilder();  
 sb.append("SELECT ");  
 sb.append(" \* ");  
 sb.append(" FROM ");  
 sb.append(type.getSimpleName());  
 sb.append(" WHERE " + field + " = ?");  
 //System.out.println(sb.toString());  
 return sb.toString();  
}

*/\*\*  
 \* Gaseste obiectul de tipul T din tabelul corespunzator  
 \** ***@param*** *id dupa el se face cautarea  
 \** ***@return*** *Obiect de tipul T  
 \*/*public T findById(int id) {  
 Connection connection = null;  
 PreparedStatement statement = null;  
 ResultSet resultSet = null;  
 String query = createSelectQuery("id");  
 try {  
 connection = ConnectionFactory.*getConnection*();  
 statement = connection.prepareStatement(query);  
 statement.setInt(1, id);  
 resultSet = statement.executeQuery();  
 return createObjects(resultSet).get(0);  
 } catch (SQLException e) {  
 *LOGGER*.log(Level.*WARNING*, type.getName() + "DAO:findById " + e.getMessage());  
 } finally {  
 ConnectionFactory.*close*(resultSet);  
 ConnectionFactory.*close*(statement);  
 ConnectionFactory.*close*(connection);  
 }  
 return null;  
}

**4.6 Clasele ClientDAO, ProductDAO, OrdersDAO**

Aceste clase mostenesc clasa abstracta AbstractDAO si inlocuiesc tipul generic cu unul explicit.

public class ClientDAO extends AbstractDAO<Client> {  
 public ClientDAO(){};  
}

public class OrderDAO extends AbstractDAO<Orders> {  
public OrderDAO(){};  
}

public class ProductDAO extends AbstractDAO<Product> {  
public ProductDAO(){};  
}

**4.7 Clasele ClientBLL, ProductBLL, OrderBLL**

Sunt clasele corespunzatoare validarii informatiilor introduse si transmiterea lor catre baza de date.

Atribute:

private ArrayList<Validator<Client>> validators;  
private ClientDAO clientDAO;

Cateva metode folosite:

public Client insertClient(Client c) {  
 try {  
 validators.get(0).validate(c);  
 validators.get(1).validate(c);  
 return clientDAO.insert(c);  
 } catch (Exception e) {  
 return null;  
 }  
}  
  
public void delete(int id) {  
 try {  
 clientDAO.delete(id);  
 }  
catch(Exception e){  
 throw new NoSuchElementException("Nu exista clientul acesta!");  
}  
}  
  
public List<Client> findAll() {  
 List<Client> clienti = new ArrayList<Client>();  
 try {  
 clienti = clientDAO.findAll();  
 }  
 catch(Exception e) {  
 throw new NoSuchElementException("Nu sunt clienti deloc :( ");  
 }  
 return clienti;  
}  
  
public Client update(Client c) {  
 try {  
 Client c1 = clientDAO.update(c);  
 return c1;  
 }  
 catch(Exception e) {  
 throw new NoSuchElementException("Nu exista acest client petru a-l updata");  
 }  
}

**4.8 Interfata grafica**

Intrucat am lucrat cu 3 ferestre, am avut nevoie de 3 clase de View si o clasa Controller.

Clasele de View pun la dispozitie textfield-uri si butoane pentru interactiunea cu baza de date.

public class ViewClient extends JFrame {  
  
 public JTextField textFieldNameClient;  
 public JTextField textFieldEmailClient;  
 public JTextField textFieldAgeClient;  
 public JTextField textFieldNameUpdate;  
 public JTextField textFieldEmailUpdate;  
 public JTextField textFieldAgeUpdate;  
 public JTextField textFieldIdUpdate;  
 public JTextField textFieldIdDelete;  
 public JTable tableClients;  
 final JScrollPane scrollPane;  
 final JPanel panelClients;  
  
 public JButton btnAddClient = new JButton("ADD CLIENT");  
 public JButton btnUpdateClient = new JButton("UPDATE CLIENT");  
 public JButton btnDeleteClient = new JButton("DELETE CLIENT");

public class ViewProduct extends JFrame{  
  
 public JTextField textFieldNameAddProduct;  
 public JTextField textFieldPriceAddProduct;  
 public JTextField textFieldQuantityAddProduct;  
 public JTextField textFieldNameUpdateProduct;  
 public JTextField textFieldPriceUpdateProduct;  
 public JTextField textFieldQuantityUpdateProduct;  
 public JTextField textFieldIdUpdateProduct;  
 public JTextField textFieldIdDeleteProduct;  
 public JTable tableProducts;  
 public JButton btnAddProduct;  
 public JButton btnUpdateProduct;  
 public JButton btnDeleteProduct;  
 final JScrollPane scrollPane\_1;  
 final JPanel panelProducts;

public class ViewOrders extends JFrame {  
   
 public JTextField textFieldIdClientOrder;  
 public JTextField textFieldIdProductOrder;  
 public JTextField textFieldQuantityOrder;  
 public JButton btnCreateOrder;  
 final JScrollPane scrollPane\_1\_1;  
 final JPanel panelOrders;  
  
 public JTable tableOrders;

Clasa Controller implementeaza interfata ActionListener ce permite aplicatiei sa reactioneze la apasarea butoanelor, cat si la distingerea acestora .

**4.9 Clasa Bill**

Aceasta clasa se ocupa de generarea unui fisier text in care se afla bonul in urma efectuarii unei comenzi.

public Bill(int id, Client c, Product p, int quantity, int total) throws IOException {  
 StringBuilder output = new StringBuilder();  
 String fileName = "bill"+id+".txt";  
 String del = "------------------------------------------------------\n";  
 new File(fileName).delete();  
 BufferedWriter writer = new BufferedWriter(new FileWriter(fileName,true));  
 output.append(del);  
 output.append("Order ID: ").append(id).append("\nClient Name: ").append(c.getName()).append("\nProduct Name: ").append(p.getName()).append("\nQuantity: ").append(quantity).append("\nTotal price: ").append(total).append("\n").append(del);  
 writer.append(output);  
 writer.flush(); writer.close();  
}

**5. Javadoc**

Pentru aceasta tema am generat documentatia Javadoc. Acest lucru a fost posibil cu ajutorul unei structuri de comentarii speciale. O mare parte din comentarii au fost adaugate in clasa AbstractDAO.

*/\*\*  
 \* Creaza un array list de obiecte din clasa T care reprezinta toate datele din tabelul corespunzator  
 \** ***@return*** *Lista de obiecte de tipul T  
 \*/*public List<T> findAll() {  
 Connection connection = null;  
 PreparedStatement statement = null;  
 ResultSet resultSet = null;  
 ResultSet r = null;  
 String query = createSelectAllQuery();  
 try {  
 connection = ConnectionFactory.*getConnection*();  
 statement = connection.prepareStatement(query);  
 resultSet = statement.executeQuery();  
 return createObjects(resultSet);  
 } catch (SQLException e) {  
 *LOGGER*.log(Level.*WARNING*, type.getName() + "DAO : findall " + e.getMessage());  
 } finally {  
 ConnectionFactory.*close*(resultSet);  
 ConnectionFactory.*close*(statement);  
 ConnectionFactory.*close*(connection);  
 }  
 return null;  
}

**6. Concluzii**

Realizarea acestei teme m-a pus in dificultate intrucat am avut de folosit tabele si o legatura cu baza de date, iar pentru ca acest lucru sa devina posibil, a fost nevoie de documentare.

**7. Surse de documentare**

 Connect to MySQL from a Java application

o https://www.baeldung.com/java-jdbc

o http://www.mkyong.com/jdbc/how-to-connect-to-mysql-with-jdbc-driver-java/

 Layered architectures

o https://dzone.com/articles/layers-standard-enterprise

 Reflection in Java

o http://tutorials.jenkov.com/java-reflection/index.html

 Creating PDF files in Java

o https://www.baeldung.com/java-pdf-creation

 JAVADOC

o https://www.baeldung.com/javadoc

 SQL dump file generation

o https://dev.mysql.com/doc/workbench/en/wb-admin-export-import-management.html

 <https://users.utcluj.ro/~igiosan/teaching_pc.html>

 prezentarile propuse