

# DOCUMENTATIE

## TEMA 3

### ORDERS MANAGEMENT

NUME STUDENT: Filip-Dud Cristian Călin  
GRUPA: 30228

## CUPRINS

---

1. Obiectivul temei.....	3
2. Analiza problemei, modelare, scenarii, cazuri de utilizare .....	4
3. Proiectare .....	8
4. Implementare .....	11
5. Rezultate .....	20
6. Concluzii .....	21
7. Bibliografie .....	22

# 1. Obiectivul temei

---

## Obiectivul principal:

Proiectați și implementați o aplicație pentru gestionarea comenzilor clienților pentru un depozit ce conține o diferite produse care pot fi cumparate de clienti prin intermediul unor comenzi.

## Obiective secundare:

- Analizarea problemei si indentificarea cerintelor
- Proiectarea aplicatiei de gestionare a comenzilor
- Implementarea aplicatiei de gestionare a comenzilor
- Testarea aplicatiei de gestionare a comenzilor

## 2. Analiza problemei, modelare, scenarii, cazuri de utilizare

---

### Cerinte functionale:

- Aplicatia trebuie sa permita unui utilizator sa adauge un client nou
- Aplicatia trebuie sa permita unui utilizator sa adauge un produs nou
- Aplicatia trebuie sa permita unui utilizator sa adauge o comanda noua

### Cerinte nonfunctionale:

- Aplicatia trebuie sa fie intuitiva si usor de folosit de catre angajat si client

### Cazul de utilizare:

**Use Case:** operatii de adaugare, stergere, actualizare, cautare de produs

**Actor principal:** utilizator

**Scenariul principal de succes:**

1. Utilizatorul alege optiunea de a adauga un produs nou
2. Aplicatia va afisa o noua fereastră cu campurile necesare pentru introducerea informatiei
3. Utilizatorul introduce datele necesare in acele campuri (nume, pret, producator, cantitate, si id mai putin in cazul de inserare, id-ul fiind generat automat de catre baza de date)
4. Utilizatorul selecteaza operatia dorita apasand butonul aferent acesteia

5. Datele se actualizeaza si afiseaza in tabel

**Secvența alternativă:** Datele introduse sunt invalide

- Utilizatorul a introdus date invalide (e.g. caractere text in campul cantitate)
- Aplicația afișează un mesaj de eroare si cere utilizatorului sa introducă date valide;
- Scenariul se întoarce la pasul 3.

**Use Case:** operatii de adaugare, stergere, actualizare, cautare de client

**Actor principal:** utilizator

**Scenariul principal de succes:**

1. Utilizatorul alege optiunea de a adauga un client nou
2. Aplicatia va afisa o noua fereastră cu campurile necesare pentru introducerea informatiei
3. Utilizatorul introduce datele necesare in acele campuri (nume, adresa, telefon, si id mai putin in cazul de inserare, id-ul fiind generat automat de catre baza de date)
4. Utilizatorul selecteaza operatia dorita apasand butonul aferent acesteia
5. Datele se actualizeaza si afiseaza in tabel

**Secvența alternativă:** Datele introduse sunt invalide

- Utilizatorul a introdus date invalide (e.g. caractere text in campul telefon)
- Aplicația afișează un mesaj de eroare si cere utilizatorului sa introducă date valide;
- Scenariul se întoarce la pasul 3.

**Use Case:** operatii de adaugare, stergere, actualizare, cautare de comenzi

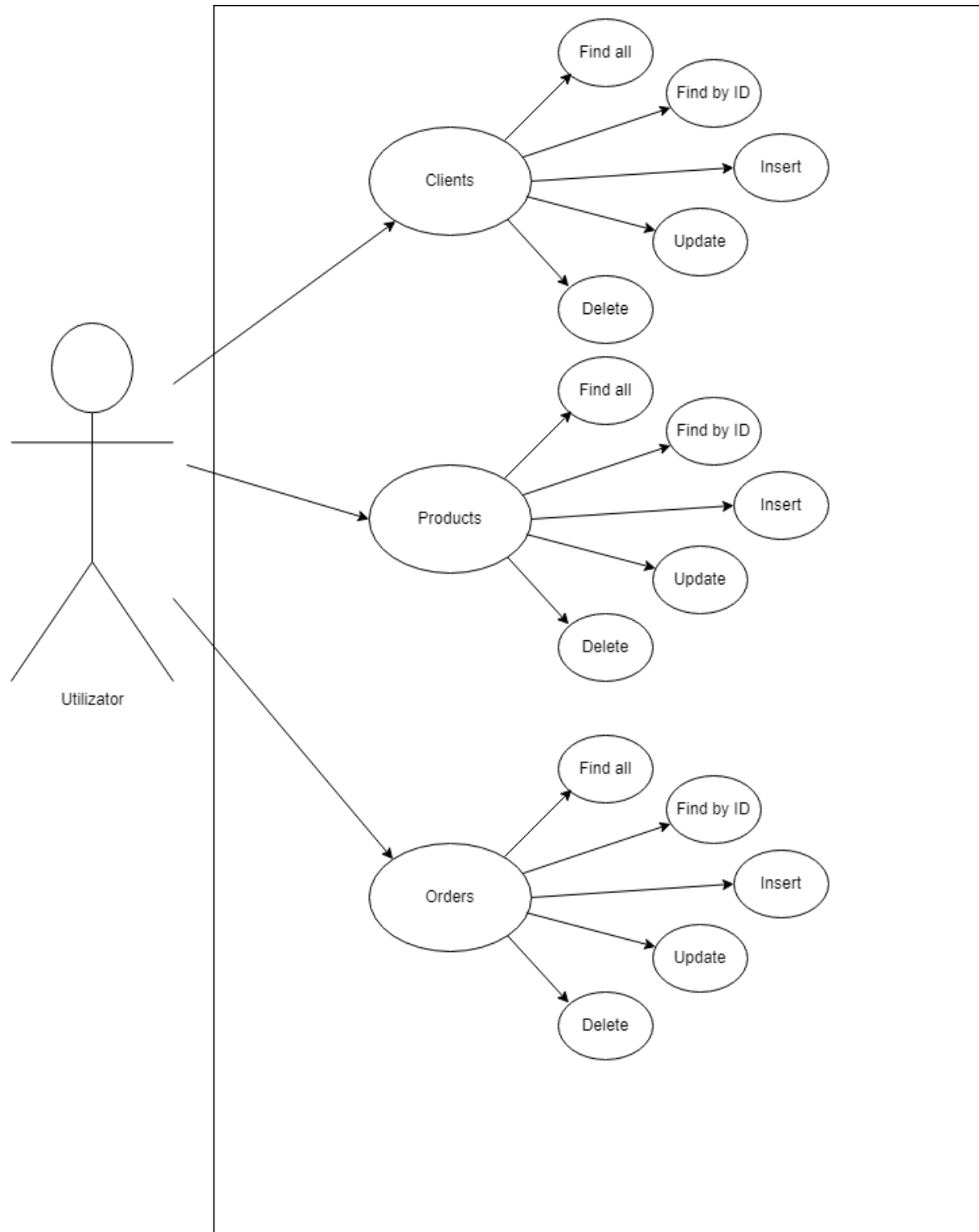
**Actor principal:** utilizator

**Scenariul principal de succes:**

1. Utilizatorul alege opțiunea de a adauga o comanda noua
2. Aplicatia va afisa o noua fereastră cu campurile necesare pentru introducerea informatiei
3. Utilizatorul introduce datele necesare in acele campuri (id client, id produs , cantitate si id mai putin in cazul de inserare, id-ul fiind generat automat de catre baza de date)
4. Utilizatorul selecteaza operatia dorita apasand butonul aferent acesteia. Daca s-a dorit inserarea unei comenzi, o factura va fi generata in baza de date.
5. Datele se actualizeaza si afiseaza in tabel

**Secvența alternativă:** Datele introduse sunt invalide

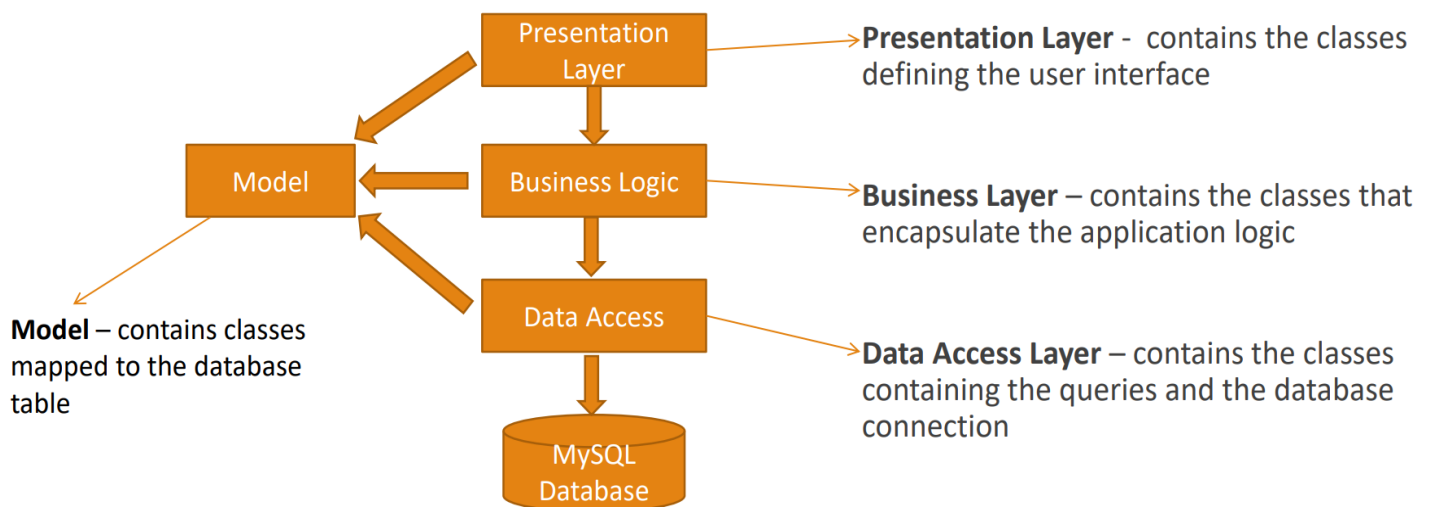
- Utilizatorul a introdus date invalide (e.g. caractere text in campul telefon)
- Aplicația afișează un mesaj de eroare si cere utilizatorului sa introducă date valide;
- Scenariul se întoarce la pasul 3.



## 3. Proiectare

---

### Proiectarea POO



Pachetele in care este structurata aplicatia sunt: Presentation (implementeaza interfata GUI a aplicatiei), DAO (implementeaza modurile de interogare si extragere a datelor din tabele), BLL (implementeaza logica aplicatiei), Connection (face conexiunea cu baza de date) si Model (contine clasele Client, Product, Product\_orders si Bill care simuleaza depozitul).

Interfata contine 4 ferestre, campuri de text pentru a introduce datele necesare introducerii de date noi, butoane si tabele pentru afisarea datelor.



— □ ×

Clients

Products

Orders

— □ ×

Name  ID

Find All

Address  Insert

Find by ID

Update

Telephone  Delete

— □ ×

Name  ID

Find All

Price  Insert

Find by ID

Update

Producer  Delete

Quantity

— □ ×

ClientID

Find All

ProductID  Insert order

Find by ID

Update

Quantity  Delete

Diagrama UML de pachete:

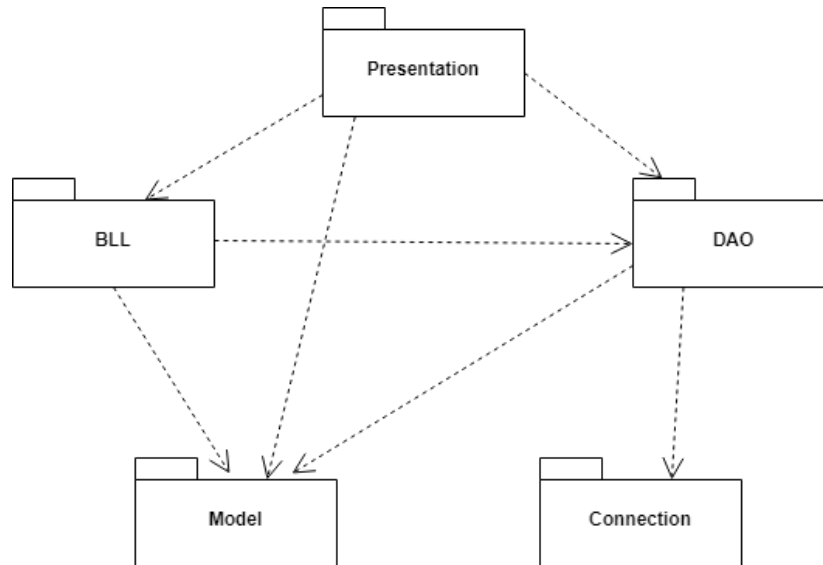
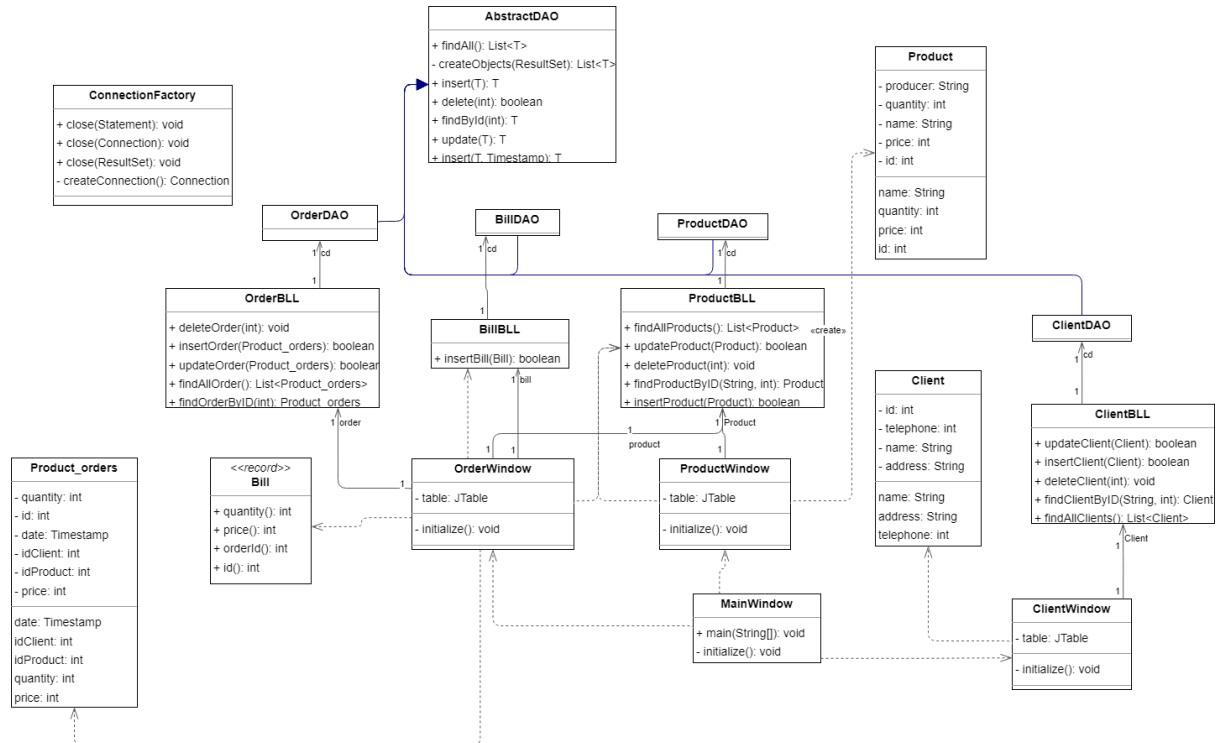


Diagrama UML de clase:



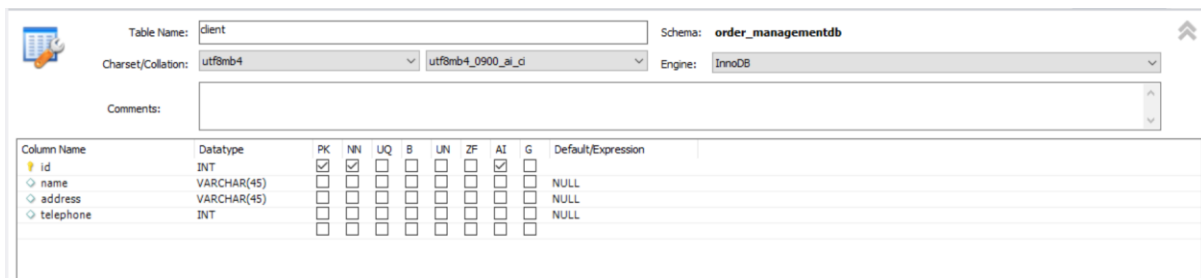
## 4. Implementare

Implementarea aplicatiei are la baza paradigma de programare orientata pe obiecte aplicata in concordanta cu aplicarea operatiilor pe o baza de date relationala stocata intr-un server MySQL.

Pachetul Model conține clasele Client, Product, Product\_orders si inregistrarea Bill, acestea se mapează direct cu tabelele din BD.

Exemplu:

```
public class Client {
    private int id;
    private String name;
    private String address;
    private int telephone;
}
```



Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
id	INT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
name	VARCHAR(45)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
address	VARCHAR(45)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
telephone	INT	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL

Id-ul fiecărei clase și tabele se generează automat, autoincrementându-se, ceea ce garantează lipsa randurilor duplicate din baza de date. Clasele conțin constructori, gettere și settere.

Exemplu:

```
public Client(int id, String name, String address, int telephone) {
    this.id = id;
    this.name = name;
    this.address = address;
    this.telephone = telephone;
}

public Client() {
}

public int getId() {
    return id;
}
```

```
}  
  
public void setId(int id) {  
    this.id = id;  
}
```

Pentru inregistrarea Bill am declarat in felul urmator:

```
public record Bill(int id, int orderId, int quantity, int price) {  
  
}
```

Ceea ce face ca datele unui obiect de tipul Bill sa nu poate fi modificate ulterior initializarii acestuia.

Pentru pachetul DAO am creat o clasaa AbstractDAO care utilizează tehnicile de reflexie pentru a efectua operațiile de find, insert, update, delete pentru orice obiect care se dă ca parametru; ClientDAO, ProductDAO, OrderDAO, BillDAO extinzand aceasta clasa.

```
package dao;  
  
import java.beans.IntrospectionException;  
import java.beans.PropertyDescriptor;  
import java.lang.reflect.*;  
import java.sql.*;  
import java.util.ArrayList;  
import java.util.List;  
import java.util.logging.Level;  
import java.util.logging.Logger;  
  
import connection.ConnectionFactory;  
  
public class AbstractDAO<T> {  
    protected static final Logger LOGGER =  
        Logger.getLogger(AbstractDAO.class.getName());  
  
    private final Class<T> type;  
  
    @SuppressWarnings("unchecked")  
    public AbstractDAO() {  
        this.type = (Class<T>) ((ParameterizedType)  
            getClass().getGenericSuperclass()).getActualTypeArguments()[0];  
    }  
  
    /**  
     * Creates and calls the query to return all data from a table  
     * @return a list that contains all rows of data of a given object  
     */  
    public List<T> findAll() {  
        Connection connection = null;  
        PreparedStatement statement = null;  
        ResultSet rs = null;  
        String query = new String("SELECT * FROM " + type.getSimpleName() +
```

```

";");
    try {
        connection = ConnectionFactory.getConnection();
        statement = connection.prepareStatement(query);
        rs = statement.executeQuery();

        return createObjects(rs);
    } catch (SQLException e) {
        LOGGER.log(Level.WARNING, type.getName() + "DAO:findAll " +
e.getMessage());
    } finally {
        ConnectionFactory.close(rs);
        ConnectionFactory.close(statement);
        ConnectionFactory.close(connection);
    }
    return null;
}

/**
 * Creates and calls the query to return the row with the given id
 * @param id
 * @return an object that has the corresponding id in the table
 */
public T findById(int id) {
    Connection connection = null;
    PreparedStatement statement = null;
    ResultSet resultSet = null;
    String query = new String("SELECT * FROM " + type.getSimpleName() +
" WHERE 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;
}

/**
 * Creates the list with all the rows from a table
 * @param resultSet
 * @return a list with all the rows from a table
 */
private List<T> createObjects(ResultSet resultSet) {
    List<T> list = new ArrayList<T>();
    try {
        while (resultSet.next()) {
            T instance = type.getDeclaredConstructor().newInstance();

            for (Field field : type.getDeclaredFields()) {
                Object value = resultSet.getObject(field.getName());
                PropertyDescriptor propertyDescriptor = new

```

```

PropertyDescriptor(field.getName(), type);
        Method method = propertyDescriptor.getWriteMethod();

        method.invoke(instance, value);
    }
    list.add(instance);
}
} catch (IllegalAccessException | SecurityException |
InvocationTargetException | IllegalArgumentException | SQLException |
IntrospectionException e) {
    e.printStackTrace();
} catch (InstantiationException e) {
    System.out.println(e.getMessage());
} catch (NoSuchMethodException e) {
    throw new RuntimeException(e);
}

return list;
}

/**
 * Inserts a new row in a table with data from the parameter object
 * @param t
 * @return the object that has been inserted in the table
 */
public T insert(T t) {

    Connection connection = null;
    PreparedStatement statement = null;
    ResultSet resultSet = null;
    String query = new String("INSERT INTO " + type.getSimpleName() + "
VALUES ( ");

    for (Field field : t.getClass().getDeclaredFields()) {
        field.setAccessible(true);
        Object value;

        try {
            value = field.get(t);
            query = query + "'" + value + "', ";
        } catch (IllegalArgumentException | IllegalAccessException e) {
            e.printStackTrace();
        }

    }

    String queryNew = query.substring(0, query.length() - 2) + ");";
    System.out.println(queryNew);

    try {
        connection = ConnectionFactory.getConnection();
        statement = connection.prepareStatement(queryNew,
Statement.RETURN_GENERATED_KEYS);
        statement.executeUpdate();
        resultSet = statement.getGeneratedKeys();
        return t;
    } catch (SQLException e) {
        LOGGER.log(Level.WARNING, type.getName() + "DAO:insert " +
e.getMessage());
    } finally {
        ConnectionFactory.close(resultSet);
    }
}

```

```

        ConnectionFactory.close(statement);
        ConnectionFactory.close(connection);
    }
    return null;
}

/**
 * Inserts a new row in a table with data from the parameter object and
 * a date
 * @param t
 * @param date
 * @return the object that has been inserted in the table
 */
public T insert(T t, Timestamp date) {

    Connection connection = null;
    PreparedStatement statement = null;
    ResultSet resultSet = null;
    String query = new String("INSERT INTO " + type.getSimpleName() + "
VALUES ( ");

    for (Field field : t.getClass().getDeclaredFields()) {
        field.setAccessible(true);
        Object value;

        try {
            value = field.get(t);
            query = query + "'" + value + "', ";
        } catch (IllegalArgumentException | IllegalAccessException e) {
            e.printStackTrace();
        }
    }

    query = query + "'" + date + "', ";

    String queryNew = query.substring(0, query.length() - 2) + ");";
    System.out.println(queryNew);

    try {
        connection = ConnectionFactory.getConnection();
        statement = connection.prepareStatement(queryNew,
Statement.RETURN_GENERATED_KEYS);
        statement.executeUpdate();
        resultSet = statement.getGeneratedKeys();
        return t;
    } catch (SQLException e) {
        LOGGER.log(Level.WARNING, type.getName() + "DAO:insert " +
e.getMessage());
    } finally {
        ConnectionFactory.close(resultSet);
        ConnectionFactory.close(statement);
        ConnectionFactory.close(connection);
    }
    return null;
}

/**
 * Updates a row with new given data
 * @param t
 * @return the object that has been inserted in the table

```

```

    */
    public T update(T t) {

        Connection connection = null;
        PreparedStatement statement = null;
        ResultSet resultSet = null;
        String query = new String("UPDATE " + type.getSimpleName() + " SET

");

        for (Field field : t.getClass().getDeclaredFields()) {
            field.setAccessible(true);
            Object value;
            try {
                value = field.get(t);
                query = query + field.getName() + "=" + value + " , ";
            } catch (IllegalArgumentException | IllegalAccessException e) {
                e.printStackTrace();
            }
        }

        String queryNew = query.substring(0, query.length() - 2) + "WHERE

";

        for (Field field : t.getClass().getDeclaredFields()) {
            field.setAccessible(true);
            Object value;

            try {
                value = field.get(t);
                queryNew = queryNew + field.getName() + "=" + value + "

";
            } catch (IllegalArgumentException | IllegalAccessException e) {
                e.printStackTrace();
            }
            break;
        }

        System.out.println(queryNew);
        try {
            connection = ConnectionFactory.getConnection();
            statement = connection.prepareStatement(queryNew,
Statement.RETURN_GENERATED_KEYS);
            statement.executeUpdate();
            return t;
        } catch (SQLException e) {
            LOGGER.log(Level.WARNING, type.getName() + "DAO:update " +
e.getMessage());
        } finally {
            ConnectionFactory.close(resultSet);
            ConnectionFactory.close(statement);
            ConnectionFactory.close(connection);
        }
        return null;
    }

    /**
     * Removes from the table the row with the id given as parameter
     * @param id
     * @return true if the object has been found and deleted, otherwise
false
    */
    public boolean delete(int id) {

```



```
Connection connection = null;
PreparedStatement statement = null;
ResultSet resultSet = null;

String query = new String("DELETE FROM " + type.getSimpleName() + "
WHERE " + "id" + "=" + id + ";");

try {
    connection = ConnectionFactory.getConnection();
    statement = connection.prepareStatement(query,
Statement.RETURN_GENERATED_KEYS);
    statement.executeUpdate();
    return true;
} catch (SQLException e) {

    LOGGER.log(Level.WARNING, type.getName() + "DAO:delete " +
e.getMessage());
    return false;
} finally {
    ConnectionFactory.close(resultSet);
    ConnectionFactory.close(statement);
    ConnectionFactory.close(connection);
}
}

}

public class ClientDAO extends AbstractDAO<Client> {
}
public class ProductDAO extends AbstractDAO<Product> {
}
public class OrderDAO extends AbstractDAO<Product_orders>{
}
public class BillDAO extends AbstractDAO<Bill> {
}
```

În pachetul Connection, avem implementată clasa ConnectionFactory care realizează conexiunea cu BD:

```
package connection;

import java.sql.*;
import java.util.logging.Level;
import java.util.logging.Logger;

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/order_managementdb";
    private static final String USER = "root";
    private static final String PASS = "Ogreisbae31";

    private static ConnectionFactory singleInstance = new
ConnectionFactory();
}
```

```
private ConnectionFactory() {
    try {
        Class.forName(DRIVER);
    } catch (ClassNotFoundException e) {
        System.out.println("FUCK YOFOFA");
        e.printStackTrace();
    }
}

private Connection createConnection() {
    Connection connection = null;
    try {
        connection = DriverManager.getConnection(DBURL, USER, PASS);
    } catch (SQLException e) {
        LOGGER.log(Level.WARNING, "An error occurred while trying to
connect to the database");
        e.printStackTrace();
    }
    return connection;
}

public static Connection getConnection() {
    return singleton.createConnection();
}

public static void close(Connection connection) {
    if (connection != null) {
        try {
            connection.close();
        } catch (SQLException e) {
            LOGGER.log(Level.WARNING, "An error occurred 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 occurred 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 occurred while trying to
close the ResultSet");
        }
    }
}
}
```

Clasele din pachetul BLL reprezintă clasele de business pentru aplicația noastră. De exemplu, pentru tabelul Client, avem:

```
package bll;

import dao.ClientDAO;
import model.Client;

import java.util.List;

/**
 * Class that has the methods to validate if the queries executed correctly
 */
public class ClientBLL {
    private ClientDAO cd = new ClientDAO();

    /**
     * Searches in the table if a client with the given id exists
     * @param s
     * @param id
     * @return the client
     * @throws Exception if the client with the given id does not exist
     */
    public Client findClientByID(String s, int id) throws Exception {
        Client client = cd.findById(id);
        if (client == null) {
            throw new Exception("The student with id =" + id + " was not found!");
        }
        return client;
    }

    /**
     * Retrieves all clients from the table
     * @return a list with all the clients in the table
     * @throws Exception if no clients exist
     */
    public List<Client> findAllClients() throws Exception {
        List<Client> c = cd.findAll();
        if (c == null) {
            throw new Exception("There are no Clients in the database!");
        }
        return c;
    }

    /**
     * Inserts a new client in the table
     * @param client
     * @return if the operation was successful
     * @throws Exception if the client could not be inserted in the table
     */
    public boolean insertClient(Client client) throws Exception {
        if (cd.insert(client) == null) {
            throw new Exception("The Client could not be inserted in the database!");
        }
        return true;
    }

    /**
     * Updates the data of a client from the table
     */
}
```

```
* @param client
* @return if the operation was successful
* @throws Exception if the client could not be updated
*/
public boolean updateClient (Client client) throws Exception{
    if(cd.update(client) == null)
    {
        throw new Exception("The client could not be updated");
    }
    return true;
}

/**
 * Deletes the client with the given id from the table
 * @param id
 * @throws Exception if the client was not found
 */
public void deleteClient(int id) throws Exception {
    boolean r = cd.delete(id);
    if (r == false) {
        throw new Exception("The Client was not found!");
    }
}
}
```

## 5. Rezultate

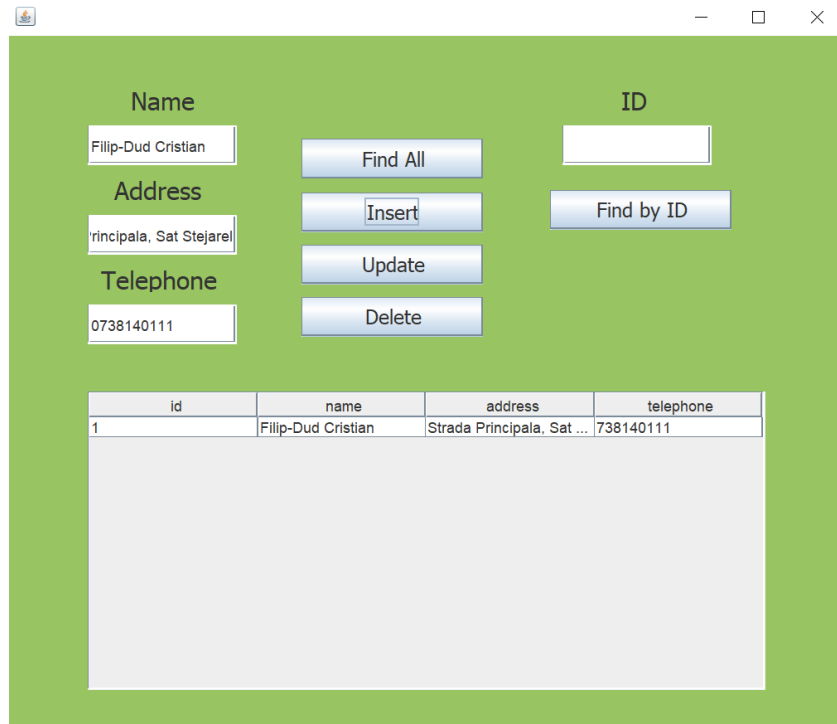
Testarea aplicatiei se realizeaza prin efectuarea operatiilor asupra bazei de date folosind interfata data.



The screenshot shows a Java Swing application window titled "Client Management". The window has a green background and contains the following elements:

- Name:** A text input field containing "Filip-Dud Cristian".
- Address:** A text input field containing "Bajarel, jud. Hunedoara".
- Telephone:** A text input field containing "0738140111".
- ID:** An empty text input field.
- Buttons:** A vertical stack of buttons: "Find All", "Insert", "Update", "Delete", and "Find by ID".
- Table:** A large, empty white rectangular area at the bottom of the window, likely intended for displaying a list of clients.

La apăsarea butonului de inserare, datele sunt introduse în tabel:



id	name	address	telephone
1	Filip-Dud Cristian	Strada Principala, Sat ...	738140111

## 6. Concluzii

---

Dezvoltarea acestei teme m-a ajutat să învăț despre modul de creare a metodelor folosind reflexii, utilizarea înregistrărilor și comunicarea cu o bază de date mysql folosind concepte POO.

În ceea ce privește modalitățile de dezvoltare ulterioară a aplicației, acestea ar putea fi:

- O interfață mai interactivă și cu mai multe elemente.
- Crearea unor ferestre specifice unui angajat și a unui client.
- Adăugarea de noi parametri pentru mai multe detalii legate de tabele.

## 7. Bibliografie

---

- ❖ [https://gitlab.com/utcn\\_dsrl/pt-layered-architecture/-/tree/master](https://gitlab.com/utcn_dsrl/pt-layered-architecture/-/tree/master)
- ❖ <https://stackoverflow.com/>