

DOCUMENTATIE

TEMA 3

NUME STUDENT: PEPELEA IOANA-ADINA

GRUPA: 30228

CUPRINS

1.	Obiectivul temei	3
2.	Analiza problemei, modelare, scenarii, cazuri de utilizare	3
3.	Proiectare	7
4.	Implementare	9
5.	Rezultate	21
6.	Concluzii	22
7.	Bibliografie	22

1.Obiectivul temei

Obiectivul principal al temei este de a realiza o aplicatie pentru gestionarea comenzilor clientilor pentru un depozit. Programul contine 3 tabele: Client, Product si Order. Aplicatia se va lega la o baza de date, unde se vor pastra informatiile legate de clienti, produse si comenzi.

Obiectivele secundare sunt urmatoarele:

- Analiza problemei si identificarea cerintelor
- Realizarea design-ului aplicatiei de gestionare a comenzilor
- Implementarea aplicatiei de gestionare a comenzilor
- Testarea aplicatiei pe baza unor input-uri prestabilite.

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

Analizand cerinta problemei, se pot deduce o serie de cerinte functionale si cerinte non-functionale care trebuie indeplinite.

Cerintele functionale sunt indeplinite de:

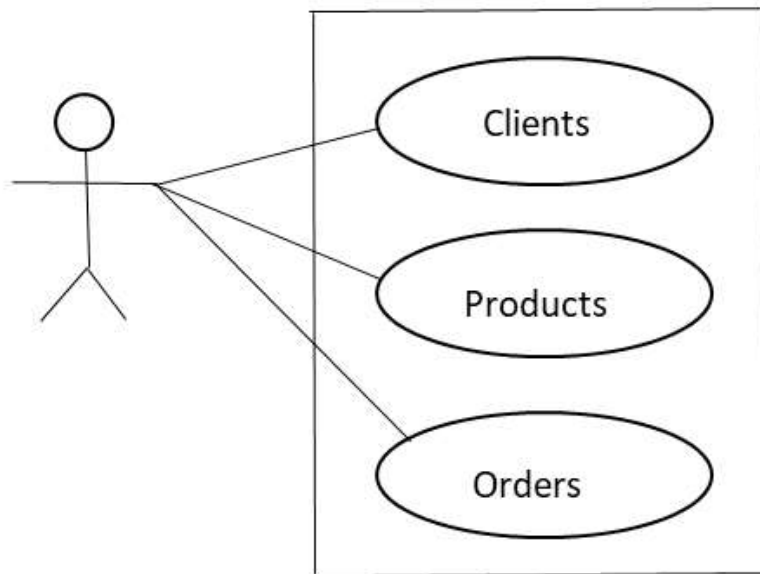
- Aplicatia ar trebui sa permita utilizatorilor sa aleaga un tabel in care sa efectueze operatii
- Aplicatia ar trebui sa permita utilizatorilor sa adauge un client
- Aplicatia ar trebui sa permita utilizatorilor sa modifice un client
- Aplicatia ar trebui sa permita utilizatorilor sa stearga un client
- Aplicatia ar trebui sa permita utilizatorilor sa vada tabelul de clienti
- Aplicatia ar trebui sa permita utilizatorilor sa revina la pagina principala
- Aplicatia ar trebui sa permita utilizatorilor sa adauge un produs
- Aplicatia ar trebui sa permita utilizatorilor sa modifice un produs
- Aplicatia ar trebui sa permita utilizatorilor sa stearga un produs
- Aplicatia ar trebui sa permita utilizatorilor sa vada tabelul de produse
- Aplicatia ar trebui sa permita utilizatorilor sa adauge o comanda
- Aplicatia ar trebui sa permita utilizatorilor sa stearga o comanda
- Aplicatia ar trebui sa permita utilizatorilor sa vada tabelul comenzilor

Cerintele non-functionale sunt:

-Aplicatia de simulare ar trebui sa fie intuitiva si usor de realizat.

Utilizarea aplicatiei in lumea reala, poate da nastere mai multor scenarii, care trebuie prevazute inca din faza de proiectare. Aceste cazuri de utilizare pot fi descrise cu ajutorul unor documente use-case, in care e descrisa evolutia sistemului si interactiunea utilizatorului cu acesta.

Use-case interfata View:



Use-case: Alegerea tabelului

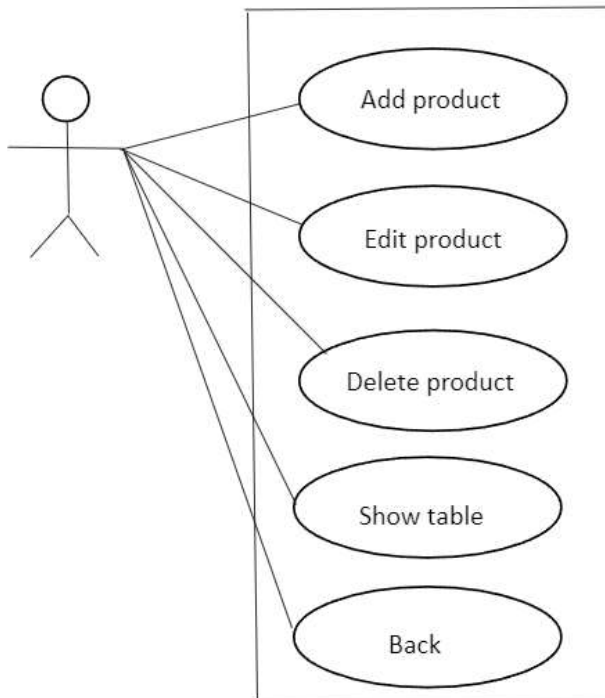
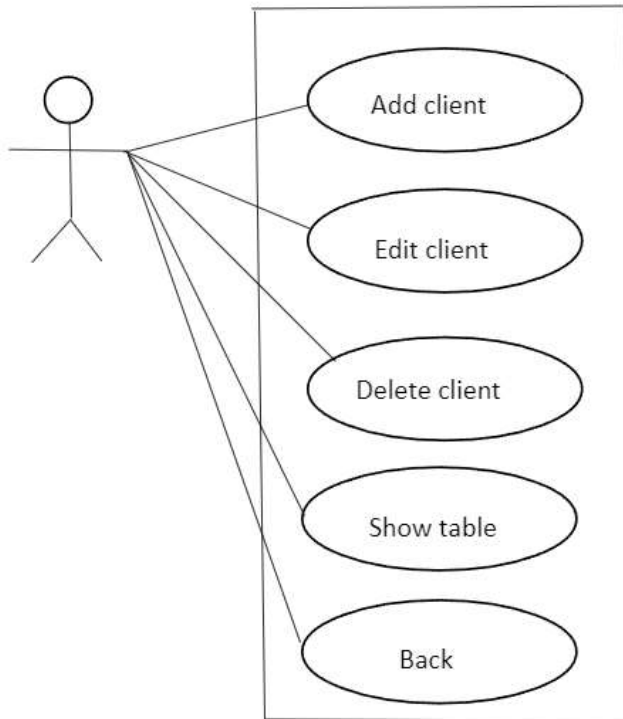
Primary actor: User

Main success scenario:

- 1) User-ul apasa pe unul dintre cele 3 butoane: "CLIENTS", "PRODUCTS", "ORDERS".

Alternative sequence: -

Use-case interfata ViewClient si ViewProduct



Use-case: Alegerea operatiei

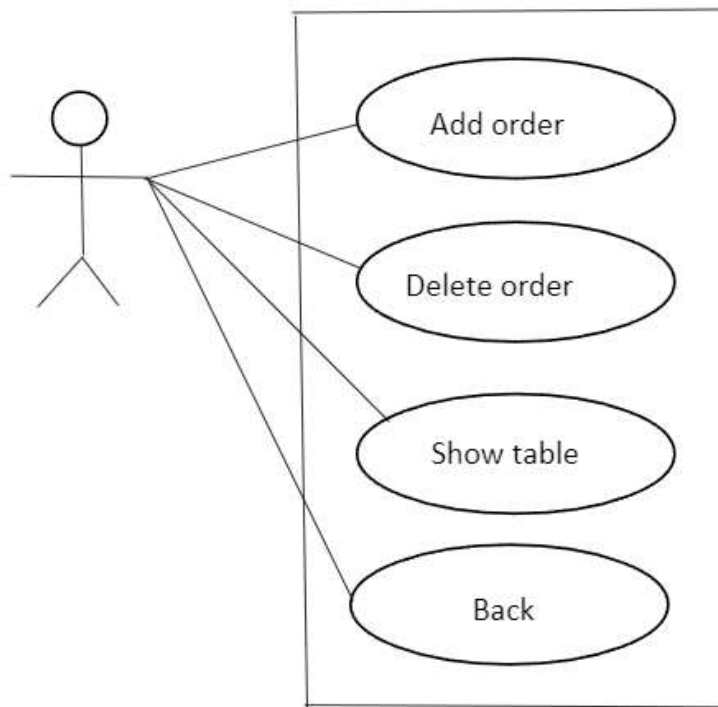
Primary actor: User

Main success scenario:

- 1) User-ul introduce datele reprezentative in campurile din interfata
- 2) User-ul face click pe unul din butoanele: "ADD CLIENT", "EDIT CLIENT", "DELETE CLIENT", "ADD PRODUCT", "EDIT PRODUCT", "DELETE PRODUCT"
- 3) Daca datele introduse au fost corecte, acestea vor fi stocate in baza de date
- 4) Daca user-ul face click pe butonul "SHOW TABLE", acesta va putea vedea datele pe care le-a introdus in baza de date
- 5) Daca user-ul apasa pe butonul "BACK", aceasta va reveni la pagina principala.

Alternative sequence: User-ul introduce date despre client sau produs invalide. Aplicatia afiseaza un mesaj de eroare, prin care este semnalat motivul erorii, si prin care se cere introducerea altor date referitoare la simulare.

Use-case interfata ViewOrder:



Use-case: Alegerea operatiei

Primary actor: User

Main success scenario:

- 6) User-ul introduce datele reprezentative in campurile din interfata
- 7) User-ul face click pe unul din butoanele: "ADD ORDER", "DELETE ORDER"
- 8) Daca datele introduse au fost corecte, acestea vor fi stocate in baza de date

- 9) Daca user-ul face click pe butonul "SHOW TABLE", acesta va putea vedea datele pe care le-a introdus in baza de date
- 10) Daca user-ul apasa pe butonul "BACK", aceasta va reveni la pagina principala.

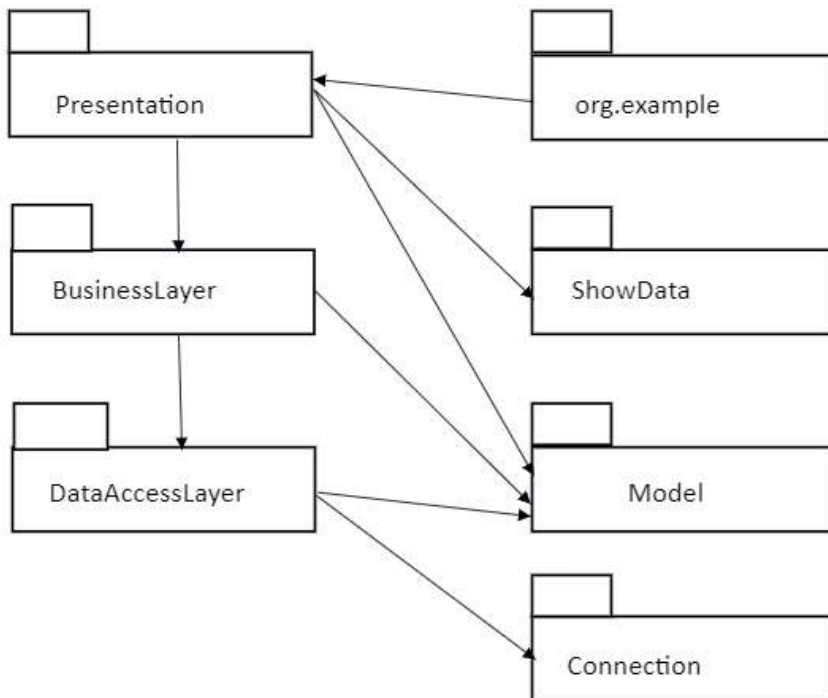
Alternative sequence: User-ul introduce date despre comanda invalida. Aplicatia afiseaza un mesaj de eroare, prin care este semnalat motivul erorii, si prin care se cere introducerea altor date referitoare la simulare.

3. Proiectare

Luam in considerare cerintele functionale si cele non-functionale, proiectul va fi impartit in sapte pachete: BusinessLayer, Connection, DataAccessLayer, Model, org.example, Presentation si ShowData.

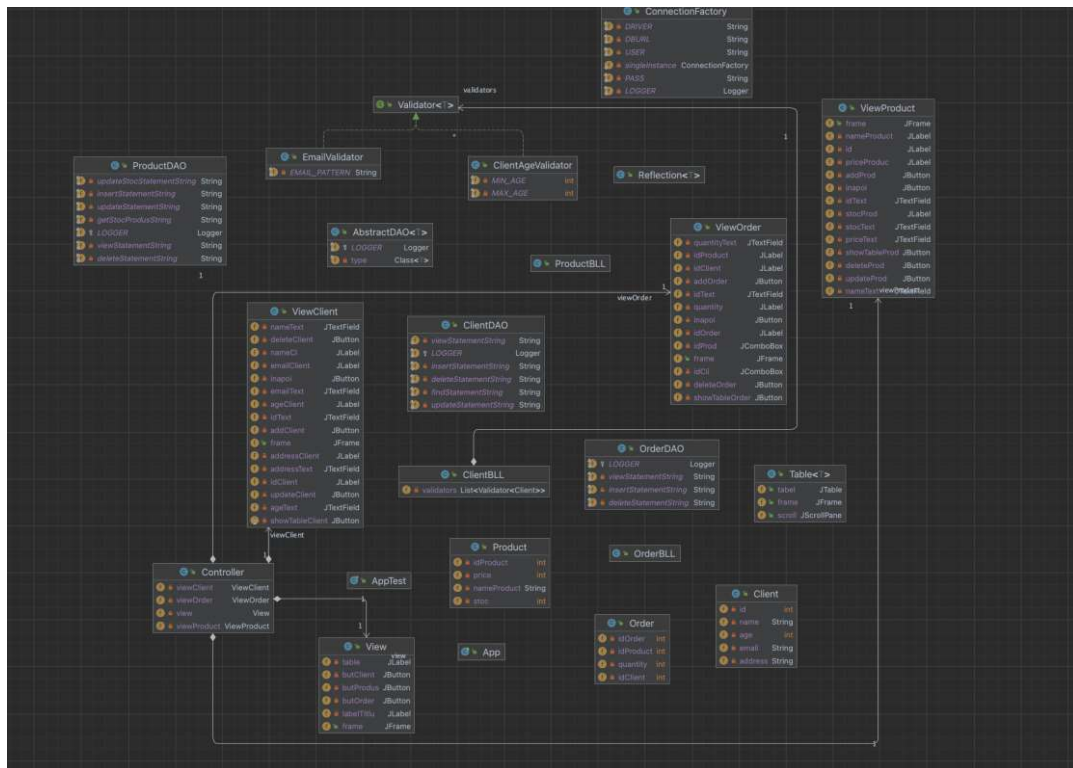
1. Pachetul BusinessLayer contine clasele ClientBLL, OrderBLL, ProductBLL si un alt pachet cu validators. Acest pachet este responsabil cu definirea actiunilor pentru fiecare clasa.
2. Pachetul Connection contine o singura clasa, ConnectionFactory, in care se face legatura dintre baza de date si programul implementat.
3. Pachetul DataAccessLayer contine clasele ClientDAO, OrderDAO, ProductDAO, AbstractDAO. In primele trei clase avem implementate interogările pentru a adauga un client/ un produs/ o comanda, a sterge un client/ un produs/ o comanda, a actualiza un client/ un produs si pentru a returna valorile din tabel pentru un client/ un produs/ o comanda. In clasa AbstractDAO se definesc operatiunile comune pentru accesarea unui tabel.
4. Pachetul Model contine clasele Client, Product, Order si este asociat cu tabelele din baza de date.
5. Pachetul org.example contine clasa App, unde avem metoda principala main.
6. Pachetul Presentation contine clasele Controller, View, ViewClient, ViewOrder, ViewProduct. Acest pachet se ocupa cu interfata si interactiunea utilizatorului cu aceasta.
7. Pachetul ShowData contine clasele Reflection, care utilizeaza reflectia pentru a extrage campurile din lista de obiecte date ca parametru si clasa Table unde este implementata afisarea unui tabel cu toate inregistrarile din baza de date.

Relatia dintre cele doua pachete este surprinsa in figura de mai jos:



Aplicatia este proiectata conform conceptelor de baza OOP. Abstractizarea presupune ca fiecare obiect are un rol bine definit, putand sa comunice cu celelalte obiecte, sa primeasca si sa furnizeze informatii, fara a da detalii despre implemetarea facilitatiilor. Astfel, sistemul este compus din 19 clase, fiecare un rol foarte bine stabilit. Incapsularea asigura ca obiectele nu pot schimba starea interna a altor obiecte decat prin metode de tip *setter*, toate attributele clasei avand modifierul de access private. Mostenirea se regaseste in cazul claselor View, ViewClient, ViewOrder, ViewProduct, care extinde *JFrame*, din Java Swing, realizand astfel interfata grafica a aplicatiei.

Relatiile dintre clasele definite in cadrul sistemului este surprinsa in diagrama UML de mai jos:



4. Implementare

4.1. Pachetul BusinessLayer

a. In clasa ClientBLL se gasesc urmatoarele metode: insertClient(Client client), updateClient(String Name, int age), deleteClient(String Name) si metoda pentru afisarea tabelului showAll() care returneaza o lista de clienti.

```

public ClientBLL() {
    validators = new ArrayList<Validator<Client>>();
    validators.add(new EmailValidator());
    validators.add(new ClientAgeValidator());
}

1 usage
public void insertClient(Client client) { ClientDAO.insert(client); }

1 usage
public void updateClient(String Name, int age) { ClientDAO.update(Name,age); }

1 usage
public void deleteClient(String name) { ClientDAO.delete(name); }

1 usage
public List<Client> showAll() throws SQLException {
    return ClientDAO.viewAll();
}

```

b. In clasa ProductBLL se gasesc urmatoarele metode: insertProduct(Product product), updateProduct(String Name, int stoc), deleteProduct(String nameProduct), metoda pentru afisarea tabelului showAll() care returneaza o lista de produse, metoda updateStocProduct(int cantitate, int idProduct) care actualizeaza stocul dupa o comanda si metoda getStocProduct(int id) pentru a verificare disponibilitatea produsului.

```

public class ProductBLL {
    1 usage
    public void insertProduct(Product produs) { ProductDAO.insert(produs); }

    1 usage
    public void deleteProduct(String nameProduct) { ProductDAO.delete(nameProduct); }

    1 usage
    public void updateProduct(String Name, int stoc) { ProductDAO.update(Name,stoc); }

    1 usage
    public void updateStocProduct(int cantitate, int idProduct){ProductDAO.updateStoc(cantitate, idProduct);}

    1 usage
    public List<Product> showAll() throws SQLException
    {
        return ProductDAO.viewAll();
    }

    1 usage
    public int getStocCurent(int id) { return ProductDAO.getStoc(id); }
}

```

c. In clasa OrderBLL se gasesc urmatoarele metode: insertOrder(Order order), deleteOrder(int idOrder), si metoda pentru afisarea tabelului showAll() care returneaza o lista de comenzi.

```

public class OrderBLL {
    1 usage
    public void insertOrder(Order order) { OrderDAO.insert(order); }

    1 usage
    public void deleteOrder(int idOrder) { OrderDAO.delete(idOrder); }

    1 usage
    public List<Order> showAll() throws SQLException {
        return OrderDAO.viewAll();
    }
}

```

In acest pachet mai avem un pachet numit validators, care contine o interfata Validator si doua clase ClientAgeValidator care verifica varsta introdusa si clasa EmailValidator care verifica email-ul introdus.

4.2. Pachetul Connection

a. In clasa ConnectionFactory se realizeaza conexiunea cu baza de date. Are drept attribute LOGGER, DRIVER, DBURL, USER si PASS. Ca metode avem: constructorul, createConnection(), getConnection(), closeConnection() si closeStatement().

```

public class ConnectionFactory {

    4 usages
    private static final Logger LOGGER = Logger.getLogger(ConnectionFactory.class.getName());
    1 usage
    private static final String DRIVER = "com.mysql.cj.jdbc.Driver";
    1 usage
    private static final String DBURL = "jdbc:mysql://localhost:3306/schooldb";
    1 usage
    private static final String USER = "root";
    1 usage
    private static final String PASS = "root";
}

```

4.3. Pachetul DataAccessLayer

Acest pachet contine 3 clase ClientDAO, ProductDAO, OrderDAO. In toate cele 3 clase se implementeaza aceleasi metode, doar cu argumente si attribute diferite, cu exceptie clasei OrderDA care nu are si metoda de update. In aceste clase se face legatura cu baza de date prin apelarea metodei de getConnection din clasa ConnectionFactory si prin implementarea interogarilor de inserare, stergere, actualizare si afisarea tuturor inregistrarilor din tabele.

a. Clase ClientDAO

```
public static void insert(Client student) {
    Connection dbConnection = ConnectionFactory.getConnection();

    PreparedStatement insertStatement = null;
    try {
        insertStatement = dbConnection.prepareStatement(insertStatementString, Statement.RETURN_GENERATED_KEYS);
        insertStatement.setInt( parameterIndex: 1, student.getId());
        insertStatement.setString( parameterIndex: 2, student.getName());
        insertStatement.setString( parameterIndex: 3, student.getAddress());
        insertStatement.setString( parameterIndex: 4, student.getEmail());
        insertStatement.setInt( parameterIndex: 5, student.getAge());
        insertStatement.executeUpdate();

        ResultSet rs = insertStatement.getGeneratedKeys();
    } catch (SQLException e) {
        LOGGER.log(Level.WARNING, msg: "ClientDAO:insert " + e.getMessage());
    } finally {
        ConnectionFactory.close(insertStatement);
        ConnectionFactory.close(dbConnection);
    }
}
```

b. ProductDAO

```
public static void delete(String nameProduct) {
    Connection dbConnection = ConnectionFactory.getConnection();
    PreparedStatement deleteStatement = null;
    try {
        deleteStatement = dbConnection.prepareStatement(deleteStatementString, Statement.RETURN_GENERATED_KEYS);
        deleteStatement.setString( parameterIndex: 1, nameProduct);
        deleteStatement.executeUpdate();
    } catch (SQLException e) {
        LOGGER.log(Level.WARNING, msg: "ProductDAO: delete " + e.getMessage());
    } finally {
        ConnectionFactory.close(deleteStatement);
        ConnectionFactory.close(dbConnection);
    }
}
```

c. OrderDAO

```
public static List<Order> viewAll() throws SQLException {
    Connection connection=ConnectionFactory.getConnection();
    PreparedStatement viewStatement=null;
    viewStatement=connection.prepareStatement(viewStatementString, Statement.RETURN_GENERATED_KEYS);
    ResultSet result=null;
    result=viewStatement.executeQuery(viewStatementString);
    List<Order> listOrders=new ArrayList<>();
    while(result.next())
    {
        Order o=new Order(result.getInt( columnLabel: "idOrder"), result.getInt( columnLabel: "idClient"),result.getInt(
        listOrders.add(o);
    }
    return listOrders;
}
```

d. In clasa AbstractDAO se definesc operatiunile comune pentru accesarea unui tabel

```
public class AbstractDAO<T> {  
    1 usage  
    protected static final Logger LOGGER = Logger.getLogger(AbstractDAO.class.getName());  
  
    6 usages  
    private final Class<T> type;  
  
    no usages  
    public AbstractDAO() {  
        this.type = (Class<T>) ((ParameterizedType) getClass().getGenericSuperclass()).getActualTypeArguments()[0];  
    }  
  
    1 usage  
    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 + " =?");  
        return sb.toString();  
    }  
}
```

4.4. Pachetul Model

a. In clasa Client avem attributele unui client: id, nume, adresa, email si varsta. In baza de date attributele sunt numele coloanelor, iar id-ul este cheie primara in tabel. Ca metode avem constructorul care are ca parametrii attributele clientului, metodele de get si set si metoda toString().

```

public class Client {
    4 usages
    private int id;
    4 usages
    private String name;
    4 usages
    private String address;
    4 usages
    private String email;
    4 usages
    private int age;
    /**
     * Constructorul initializeaza obiecte noi.
     * @param id
     * @param name
     * @param address
     * @param email
     * @param age
     */
    public Client(int id, String name, String address, String email, int age) {
        super();
        this.id = id;
        this.name = name;
        this.address = address;
        this.email = email;
        this.age = age;
    }
}

```

b. In clasa Product avem attributele unui produs: id, nume, stocul si pretul produsului. In baza de date attributele sunt numele coloanelor si id-ul este cheie primara in tabel. Ca metode avem constructorul care are ca parametrii attributele clasei, metode de get si set si toString().

```

public class Product {
    4 usages
    private int idProduct;
    4 usages
    private String nameProduct;
    4 usages
    private int stoc;
    4 usages
    private int price;

    /**\
     * Constructorul initializeaza obiecte noi.
     * @param idProduct
     * @param nameProduct
     * @param stoc
     * @param price
     */
    2 usages
    public Product(int idProduct, String nameProduct, int stoc, int price) {
        this.idProduct = idProduct;
        this.nameProduct = nameProduct;
        this.stoc = stoc;
        this.price = price;
    }
}

```

c. In clasa Order se simuleaza o comanda a unui client pentru un produs ales. Atributele clasei sunt idOrder, idClient, idProduct si quantity. In baza de date atributele sunt numele coloanelor si id-ul comenzii este cheie primara in tabel.


```

public class Order {
    4 usages
    private int idOrder;
    4 usages
    private int idClient;
    4 usages
    private int idProduct;
    4 usages
    private int quantity;

    /**
     * Constructorul initializeaza obiecte noi.
     * @param idOrder
     * @param idClient
     * @param idProduct
     * @param quantity
     */
    2 usages
    public Order(int idOrder, int idClient, int idProduct, int quantity) {
        this.idOrder = idOrder;
        this.idClient = idClient;
        this.idProduct = idProduct;
        this.quantity = quantity;
    }
}

```

4.5. Pachetul org.example

a. Clasa App contine metoda main in care avem ca si atribute View si Controller.

```

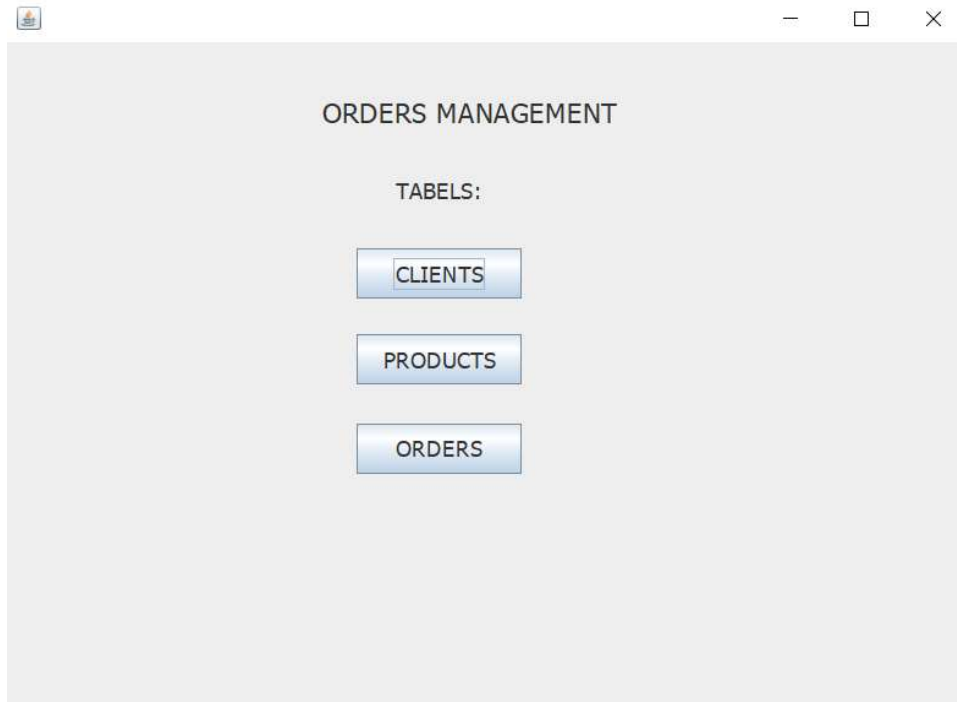
public class App
{
    no usages
    public static void main( String[] args ) throws SQLException {
        View view = new View();
        view.setVisible(false);
        Controller controller = new Controller(view);
    }
}

```

4.6. Pachetul Presentation

a. In clasa View am implementat interfata principala care contine 3 butoane butonul Clients, care atunci cand va fi apasat va deschide o fereastră noua in care se pot face diferite operatii in tabelul Client, butonul Product, care atunci cand va fi apasat va deschide o fereastră noua in

care se pot face diferite operatii in tabelul Product si butonul Order, care atunci cand va fi apasat va deschide o fereastră noua in care se pot face diferite operatii in tabelul Order.



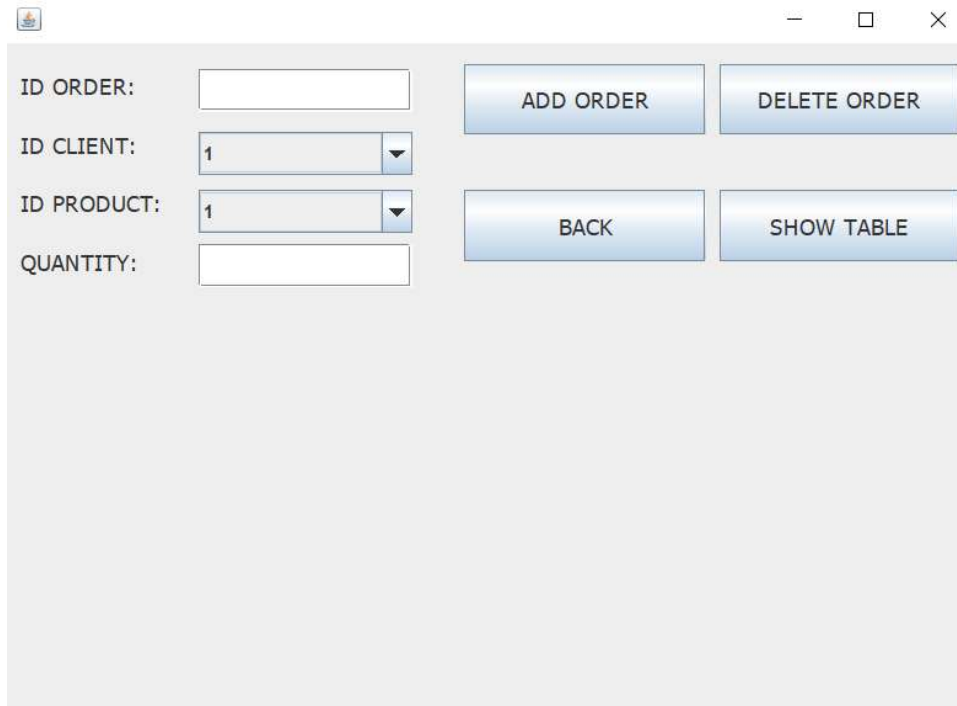
b. Clasa ViewClient contine: 5 textField – uri in care se introduce: ID , numele clientului , adresa, email-ul si varsta si 5 butoane . Primul buton este pentru adaugarea clientului in baza de date , al doilea buton este pentru actualizarea varstei clientului prin introducerea numelui, al treilea buton se foloseste pentru stergerea clientului prin introducerea numelui, al patrulea buton este pentru afisarea tabelului din baza de date, iar ultimul buton este pentru a reveni la meniul principal.

A Java Swing window titled "Client Management" with a standard title bar (minimize, maximize, close buttons). The window contains five text input fields on the left, each with a label: "ID CLIENT:", "NAME:", "ADDRESS:", "EMAIL:", and "AGE:". To the right of these fields are five buttons: "ADD CLIENT" and "EDIT CLIENT" are positioned above "DELETE CLIENT" and "SHOW TABLE", which are above a "BACK" button.

c. Clasa ViewProduct contine: 4 textField – uri in care se introduce: ID Product , numele produsului , stocul si pretul si 5 butoane . Primul buton este pentru adaugarea produsului in baza de date , al doilea buton este pentru actualizarea stocului produsului prin introducerea numelui , al treilea buton se foloseste pentru stergerea produsului prin introducerea numelui , al patrulea buton este pentru afisarea tabelului din baza de date iar ultimul buton este pentru revenirea la meniul principal .

A Java Swing window titled "Product Management" with a standard title bar (minimize, maximize, close buttons). The window contains four text input fields on the left, each with a label: "ID PRODUCT:", "NAME:", "STOC:", and "PRICE:". To the right of these fields are five buttons: "ADD PRODUCT" and "EDIT PRODUCT" are positioned above "DELETE PRODUCT" and "SHOW TABLE", which are above a "BACK" button.

d. Clasa ViewOrder contine : 2 textField – uri in care se introduce : ID Order si cantitatea care doreste sa fie comandata , 2 comboBox – uri pentru a alege din baza de date clientul si produsul care doreste a fi comandat si 4 butoane . Primul buton este pentru adaugarea comenzii in baza de date , al doilea buton este pentru stergerea comenzii prin introducerea id – lui , al treilea buton se foloseste pentru revenirea la meniul principal iar ultimul buton este pentru afisarea tabelului din baza de date.



The screenshot shows a Java Swing window titled "ViewOrder". It contains four text input fields on the left, each with a label: "ID ORDER:", "ID CLIENT:", "ID PRODUCT:", and "QUANTITY:". The "ID CLIENT:" and "ID PRODUCT:" fields are dropdown menus, both currently showing the value "1". To the right of these fields are four buttons arranged in a 2x2 grid: "ADD ORDER" (top-left), "DELETE ORDER" (top-right), "BACK" (bottom-left), and "SHOW TABLE" (bottom-right). The window has standard OS window controls (minimize, maximize, close) in the top right corner.

d. Clasa Controller face legatura intre interfata grafica si programul propriu – zis . In aceasta clasa am implementat raspunsul programului la apasarea fiecarui buton .

4.7. Pachetul ShowData

a. In clasa Reflection avem metoda retrieveProperties care extrage proprietatile obiectele folosind reflectia si metoda retrieveHeader care extrage numele coloanelor si returneaza o lista tot folosind reflectia .

```

public class Reflection<T> {

    /**
     * Aceasta metoda extrage proprietatile obiectelor folosind reflectie
     * @param object
     */
    no usages
    public static void retrieveProperties(Object object)
    {
        for(Field field: object.getClass().getDeclaredFields())
        {
            field.setAccessible(true);
            Object value;
            try{
                value=field.get(object);
                System.out.println(field.getName()+ "=" +value);
            } catch (IllegalArgumentException e) {
                e.printStackTrace();
            } catch (IllegalAccessException e){
                e.printStackTrace();
            }
        }
    }
}

```

b. Clasa Table se foloseste pentru a crea un tabel si pentru a afisa in interfata. In aceasta clasa avem o singura metoda , constructorul care primeste ca parametru o lista. In aceasta metoda se creeaza coloanele , capul de table folosind o structura for, in care, pentru fiecare coloana se extrage numele coloanei din lista dat ca si parametru, iar in alta structura for se pun valorile din baza de date in celulele tabelului.

```

public Table(List<T> lista) throws IllegalAccessException, NoSuchFieldException {
    setTitle("TABEL" + lista.get(0).getClass().getName());
    frame = new JFrame();
    Reflection<T> ref=new Reflection<T>();

    int nrColoane=ref.retrieveheader(lista).size();
    String[] coloana=new String[nrColoane];
    for(int i=0; i<nrColoane;i++)
    {
        coloana[i]=ref.retrieveheader(lista).get(i).toString();
    }

    int nrLinii=lista.size();
    String[][] celulaTabel=new String[nrLinii][nrColoane];

    int linie=0, col;
    for(T t: lista)
    {
        col=0;
        for(Field field : t.getClass().getDeclaredFields())
        {
            field.setAccessible(true);
            Object value=field.get(t);
            celulaTabel[linie][col]=value.toString();
            col++;
        }
        linie++;
    }
}

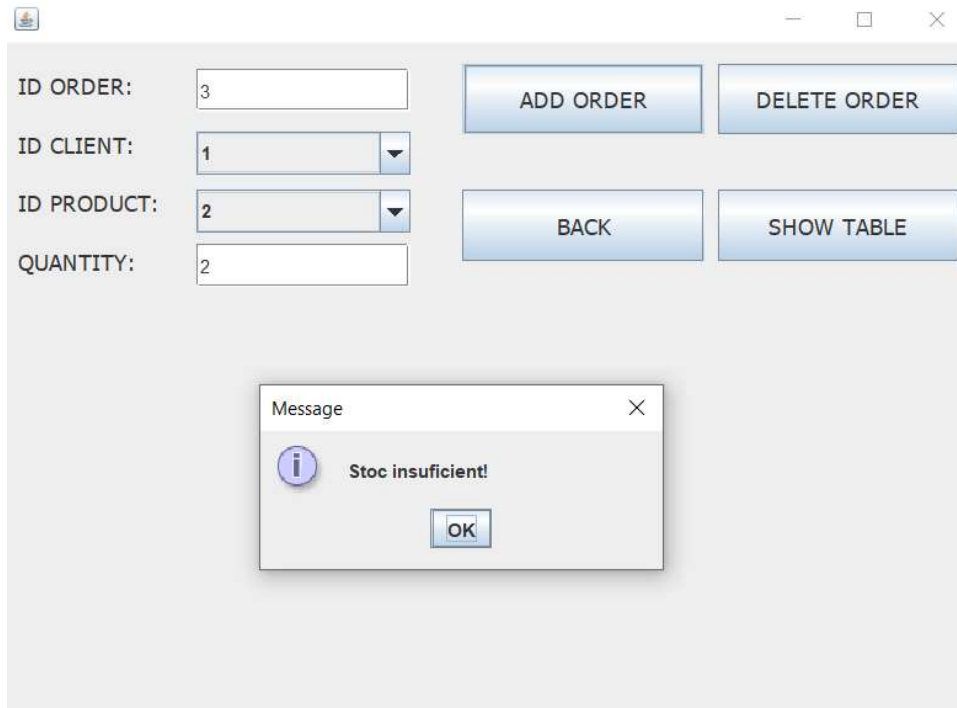
```

5. Rezultate

Ruland aplicatia, am observat ca datele pe care le introduce utilizatorul sunt adaugate in baza de date.

	id	name	address	email	age
▶	1	Adina	Sancel	ap@yahoo.com	21
	2	Cipri	Blaj	cipri@yahoo.com	22
	3	Andreea	Lunca	avelicea@yahoo.com	20
★	NULL	NULL	NULL	NULL	NULL

Daca utilizatorul introduce date gresite sau invalide, vor fi afisate mesaje:



6. Concluzii

Aplicatia a fost implementata utilizand accesul la baze de date si tehnica JavaReflection. Aceasta prezinta facilitati de baza, privind lucrul cu bazele de date: afisare tabele. O dezvoltare ulterioara ar putea fi adaugarea unor operatii mai complexe, precum autentificarea clientilor la lansarea aplicatiei, precum si dezvoltarea unei interfete grafice mai prietenoase.

7. Bibliografie

https://dsrl.eu/courses/pt/materials/PT2023_A3_S1.pdf

https://dsrl.eu/courses/pt/materials/PT2023_A3_S2.pdf

