MINISTERUL EDUCAȚIEI ȘI CERCETĂRII ȘTIINȚIFICE

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE

Managementul comenzilor

Documentație

Ciubotaru Andrei-Mihai

Grupa: 30227

Cuprins

1.Obiectivul temei

Obiectivul acestei teme este sa realizam o aplicatie care se ocupa cu managementul comenzilor.

Aplicatia va permite cautarea, adaugarea, editarea si stergerea clientilor sau produselor din baza de date. Pe langa acestea se poate plasa o noua comanda, odata cu aceasta generandu-se o factura.

Acesta va dispune de o interfata „User Friendly”, care poate fi utilizata de oricine.

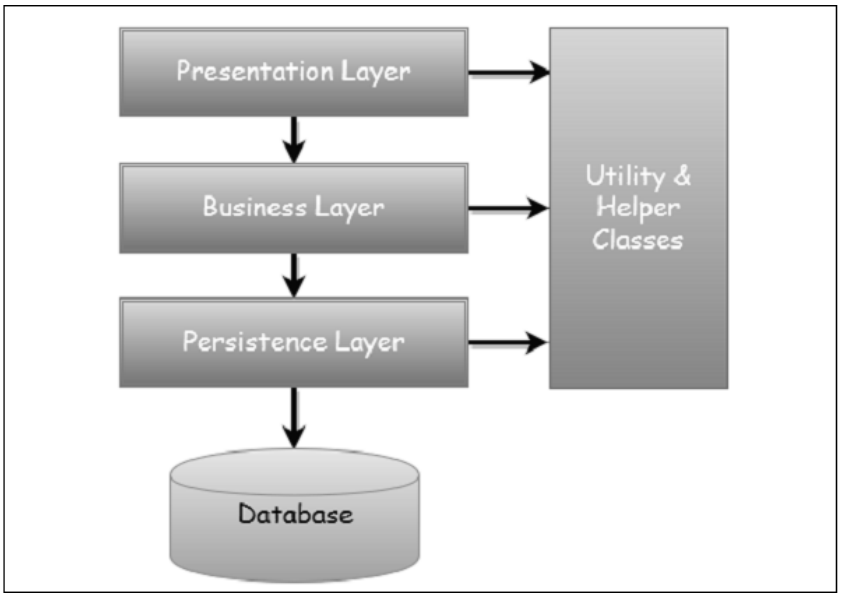
2.Analiza problemei

Pentru a putea face operatii asupra bazei da date este necesar sa cunoastem limbajul SQL.

SQL (Structured Query Language - limbaj de interogare structurat) este un limbaj de programare specific pentru manipularea datelor în sistemele de manipulare a bazelor de date relationale, iar la origine este un limbaj bazat pe algebra relațională. Acesta are ca scop inserarea datelor, interogatii, actualizare si stergere, modificarea si crearea schemelor, precum si controlul accesului la date.

3.Proiectare

Am folosit o baza de date alcatuita din trei tabele (Client, Produs, Comanda), iar aplicatia respecta arhitectura layered. Aceasta arhitectura imparte aplicatia in diferite straturi, fiecare avand o functie bine definita.



Presentation Layer contine clsele ce definesc interfata. Business Layer contine clasele care se ocupa de logica aplicatiei. Persistence Layer se ocupa de interogari si de conexiunea la baza de date, Modelul contine clasele ce mapeaza baza de date.

Diagrama UML presupune modelarea unui sistem prin lucrurile care sunt importante pentru acesta. Aceste lucruri sunt modelate folosind clase.

3.1.Diagrama de clase

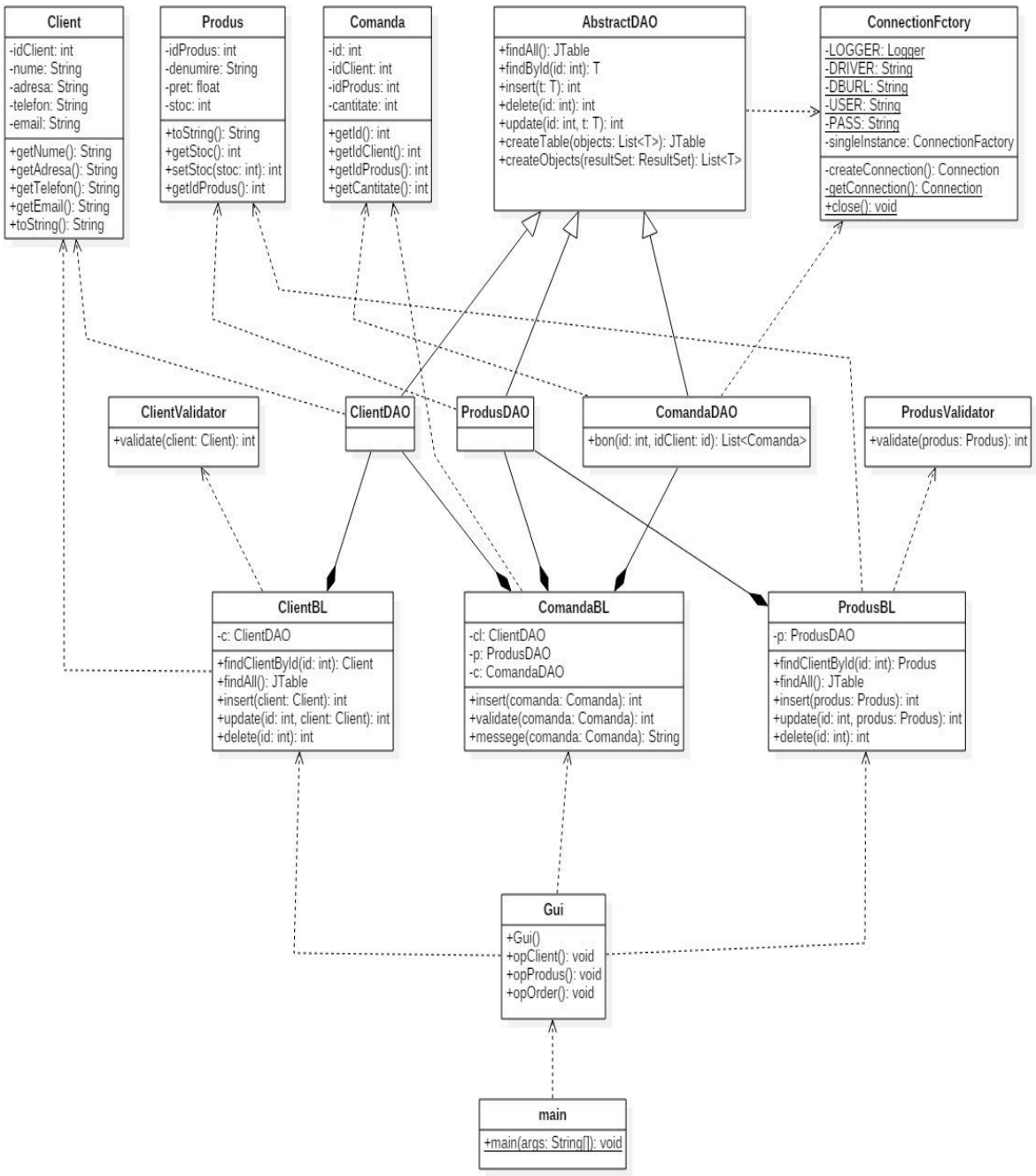


Diagrama de pachete

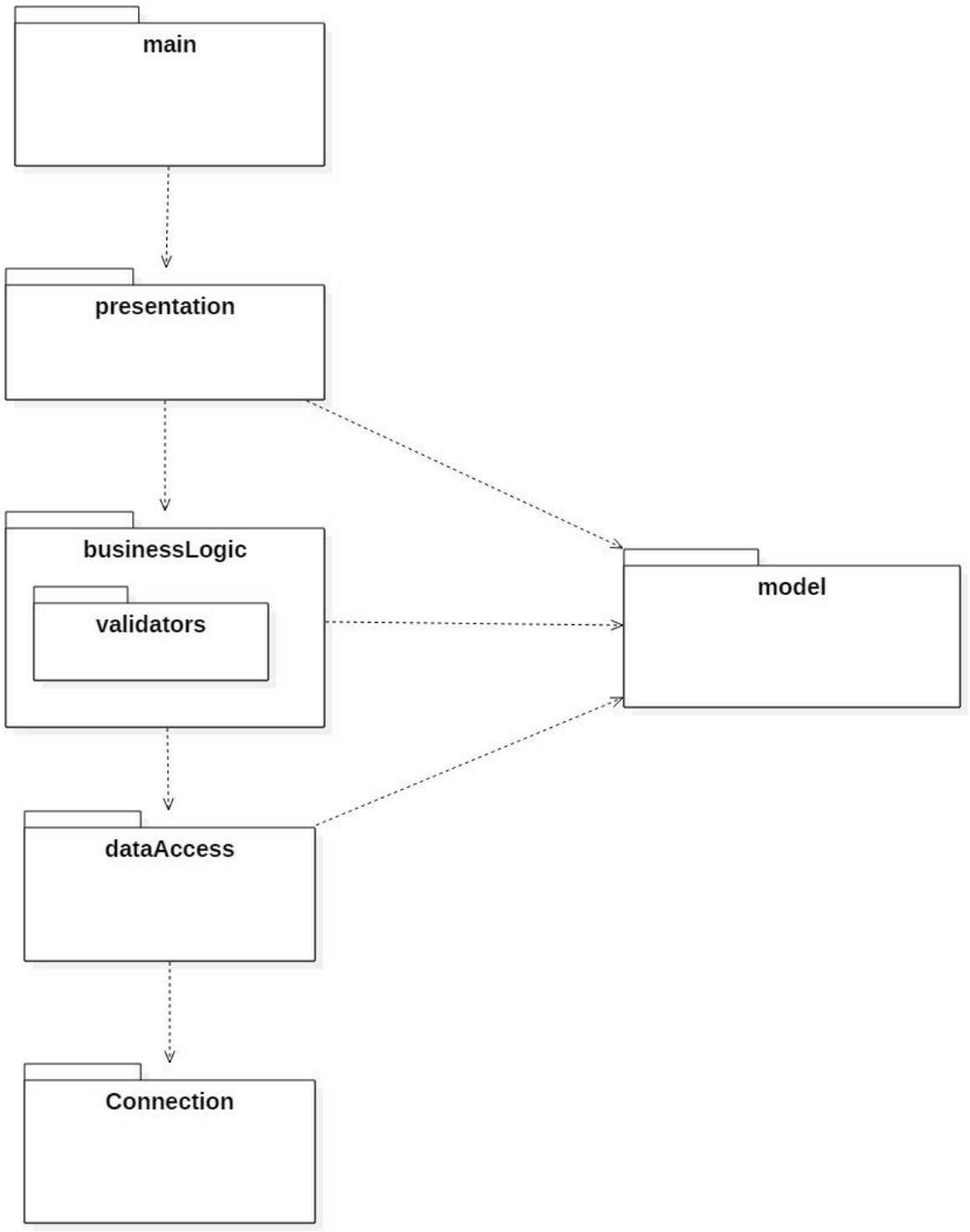
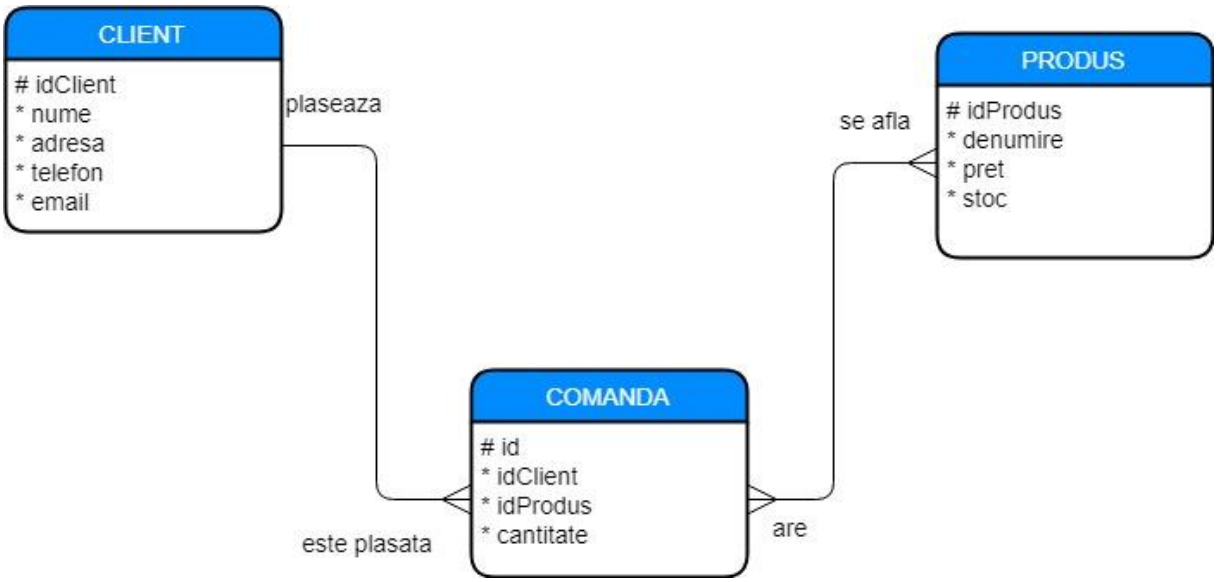


Diagrama ERD

Diagrama ERD (entity–relationship model) defineste baza de date si desrcie relatiile care se stabilesc intre entitati.



4.Implementare

4.1.Clase si pachete

Am organizat aplicatia in 7 pachete.

Pachetul Connection contine clasa connectionFactory care realizeaza conexiunea la baza de date. Pentru aceasta clasa am folosit Singelton design pattern pentru a ne asigura ca aceasta clasa are o singura insatnta, astfel nu se realizeaza mai multe conexiuni la baza de date.

```
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/my_db?autoReconnect=true&useSSL=false";
    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();
        }
    }

    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();
    }
}
```

In pachetul Model regasim clasele Client, Produs si Comanda clase care reprezinta tabelele ce se afla in baza de date. Variabilele instanta ale acestor clase sunt reprezentate de atributele entitatilor din digrama ERD / coloanele tabelelor din baza de date. Clasele au mai mutli constructori, astfel se pot initializa la instantiere diferite variabile, getter si setter pentru fiecare variabila si metoda toString.

Clasa Client are ca si variabile idClient, nume, adresa, telefon si email.

```
    public String toString() {
        return "Client " + "[idClient= " + idClient + " nume= " + nume + " adresa=
" + adresa + " telefon= " + telefon
                + " email= " + email + "]";
    }
```

Clasa Produs contine idProdus, denumire, pret si stoc.

```
public String toString() {
    return "Produs " + "[idProdus= " + idProdus + " denumire= " + denumire + "
pret= " + pret
            + " stoc= " + stoc + "]";
}
```

Clasa Comanda contine id, idClient, idProdus si cantitatea.

In pachetul DataAccess sunt clasele care se ocupa cu manipularea bazei de date si sunt implementate metodele pentru adaugare, cautare, editare si stergere. Acesta contine clasele ClientDAO, ProdusDAO, ComandaDAO care mostenesc clasa AbstractDAO.

Clasa AbstractDAO<T> are parametru generic T astfel vom putea apela pentru orice tip de obiect metodele din aceata clasa.

```
private String createSelectQuery(String field) {
    StringBuilder query = new StringBuilder();
    query.append("SELECT ");
    query.append(" * ");
    query.append(" FROM ");
    query.append(type.getSimpleName());
    query.append(" WHERE " + field + " =?");
    return query.toString();
}
```

Metoda createSelectQuery primeste ca parametru un string care reprezinta creiteriul dupa care se va face selectarea din baza de date. type.getSimpleName() preia numele parametrului generic, astfel vom stii din ce tabela selectam. (SELECT * FROM .... WHERE ....);

```
public JTable findAll() {
    Connection connection = null;
    PreparedStatement statement = null;
    ResultSet resultSet = null;
    String query = "SELECT * FROM " + type.getSimpleName();
    try {
        connection = ConnectionFactory.getConnection();
        statement = connection.prepareStatement(query);
        resultSet = statement.executeQuery();
        return createTable(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;
}
```

Metoda findAll returneaza un tabel cu toate intrarile dintr-o tabela. Se realizeaza conexiunea la baza de date, apoi se pregateste interogarea, iar rezultatul este trensmis spre metoda createObjects();

```
public List<T> createObjects(ResultSet resultSet) {
    List<T> list = new ArrayList<T>();
    try {
        while (resultSet.next()) {
            T instance = type.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 (InstantiationException e) {
        e.printStackTrace();
    } catch (IllegalAccessException e) {
        e.printStackTrace();
    } catch (SecurityException e) {
        e.printStackTrace();
    } catch (IllegalArgumentException e) {
        e.printStackTrace();
    } catch (InvocationTargetException e) {
        e.printStackTrace();
    } catch (SQLException e) {
        e.printStackTrace();
    } catch (IntrospectionException e) {
        e.printStackTrace();
    }
    return list;
}
```

Folosind tehnica de refection aceasta metoda primeste rezultaul interogarii SQL si creeaza o lista de tipul T, fiecare rand din rezultat reprezinta un obiect in lista.

```
public T findById(int id) {
    Connection connection = null;
    PreparedStatement statement = null;
    ResultSet resultSet = null;
    String query = createSelectQuery(getFirstField());
    try {
        connection = ConnectionFactory.getConnection();
        statement = connection.prepareStatement(query);
        statement.setInt(1, id);
        resultSet = statement.executeQuery();
        if(!resultSet.next())
            return null;
        resultSet.beforeFirst();
        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;
    }
```

Metoda findById returneaza un obiect de tipul T daca in tabela a fost gasita intrarea cu id-ul id sau null in caz contrar.

```
String query = "INSERT INTO " + type.getSimpleName() + " VALUES(";
int nr=0;
for(Field field : type.getDeclaredFields())
    nr++;
int i=0;
try {
    for (Field field : type.getDeclaredFields()) {
        i++;
        PropertyDescriptor propertyDescriptor = new
PropertyDescriptor(field.getName(), type);
        Method method = propertyDescriptor.getReadMethod();
        if(field.getType().getSimpleName().equals("String")){
            if(i==nr)
                query = query + "'" + method.invoke(t) + "');";
            else
                query = query + "'" +  method.invoke(t) + "',";
        }
        else
            if(i==nr)
                query = query + method.invoke(t) + ");";
            else
                query = query + method.invoke(t) + ",";

    }
```

Pentru a realiza inserarea intr-o tabela am utilizat tehnica de reflection pentru a crea interogarea pas cu pas. Astfel la inceput numar cate campuri are clasa T, pentru a stii cate valori trebuie sa inserez. Apoi preiau numele variabilelor instanta si cu ajutorul metodei invoke preiau valoarea acelei variabile si o adaug in interogare. Interogare este de forma INSERT INTO tabela VALUES(` `,` `,..);

La fel procedez si la update.

```
String query = "UPDATE " + type.getSimpleName() + " SET ";
int nr=0;
for(Field field : type.getDeclaredFields())
    nr++;
int i=0;
try {
    for (Field field : type.getDeclaredFields()) {
        i++;
        PropertyDescriptor propertyDescriptor = new
PropertyDescriptor(field.getName(), type);
        Method method = propertyDescriptor.getReadMethod();
            if (field.getType().getSimpleName().equals("String")){
                if(i==nr)
                    query = query + field.getName() + "='" + method.invoke(t) +
"'";
                else if(i>1)
                    query = query + field.getName() + "='" + method.invoke(t) +
"',";
            }
            else
                if(i==nr)
                    query = query + field.getName() + "=" + method.invoke(t);
                else if(i>1)
                    query = query + field.getName() + "=" + method.invoke(t) + ",";
```

```
    }
    query = query + " WHERE " + getFirstField() + "= " + id;
}
```

Metoda delete primeste ca parametru un id si returneaza 1 in caz de succes sau 0 daca in tabela nu exista acel id.

```
public int delete(int id) {
    Connection connection = null;
    PreparedStatement statement = null;
    String query = "DELETE FROM " + type.getSimpleName() + " WHERE " +
getFirstField()  + "=?";
    try {
        connection = ConnectionFactory.getConnection();
        statement = connection.prepareStatement(query);
        statement.setInt(1, id);
        System.out.println(statement);
        return statement.executeUpdate();
    } catch (SQLException e) {
        LOGGER.log(Level.WARNING, type.getName() + "DAO:delete " + e.getMessage());
    } finally {
        ConnectionFactory.close(statement);
        ConnectionFactory.close(connection);
    }
    return 0;
}
```

Pachetul BusinessLogic se ocupa de logica aplicatiei. Acesta contine un subpachet Validators care se ocupa cu validarea datelor.

```
public class ClientValidator {

    public int validate(Client client) {
        String nume_pattern = "^[a-zA-Z\\s]*$";
        String telefon_pattern = "([0-9]){10}";
        String email_pattern = "\\b[a-zA-Z0-9._%+-]+@[a-zA-Z]+\\.[a-zA-Z]{2,}\\b";
        int ok = 0;
        Pattern nume = Pattern.compile(nume_pattern);
        Pattern telefon = Pattern.compile(telefon_pattern);
        Pattern email = Pattern.compile(email_pattern);
        if (!nume.matcher(client.getNume()).matches() ||
client.getNume().equals(""))
            return 2; //nume gresit
        if (!nume.matcher(client.getAdresa()).matches() ||
client.getAdresa().equals(""))
            return 3; //adresa gresita
        if (!telefon.matcher(client.getTelefon()).matches())
            return 4; //telefon gresit
        if (!email.matcher(client.getEmail()).matches())
            return 5; //email gresit
        return ok;
    }
}
```

Metoda validate returneaza diferite valori corespunzatoare unor cazuri neprezaute. Utilizatorul introduce date invalide. Aceste coduri sunt tratate in clasa Gui atunci cand se face o operatie asupra bazei de date. Pentru validare folosesc diferite regex-uri.

In clasele ClinetBL, ProdusBL, ComandaBL se gasesc metodele de adaugare, stergere, cautare si editare care sunt verificate, iar daca sintaxa interogarilor sau datele introduse de utilizator nu sunt bune metodele returneaza un cod de eroare ce va fi transformat intr-un mesaj in clasa Gui.

In clasa ComandaBL se gaseste o metoda care returneaza mesajul ce va fi printat pe factura la efectuarea unei comenzi.

```
public String message(Comanda comanda) {
    String msg = "";
    float total = 0;
    Client client = cl.findById(comanda.getIdClient());
    msg = msg + "ID Comanda " + comanda.getId() + "\n";
    DateFormat dateFormat = new SimpleDateFormat("yyyy/MM/dd HH:mm:ss");
    Date date = new Date();
    msg = msg + dateFormat.format(date) + "\n";
    msg = msg + client.toString() + "\n";
    msg = msg + "Nr. Crt.    "  + "Produs     "  +  "          Pret" + "\n";
    List<Comanda> list;
    list = c.bon(comanda.getId(), comanda.getIdClient());
    int nr = 1;
    for(Comanda  i : list) {
        Produs produs;
        produs = p.findById(i.getIdProdus());
        float aux;
        aux = produs.getPret() * i.getCantitate();
        msg = msg + nr + "           " + produs.getDenumire() + "            "+
i.getCantitate() + " X " +
                produs.getPret() + " = " + aux + "\n";
        total+=aux;
        nr++;
    }
    msg = msg + "Total: " + total + "\nVa multumim!!!";
    return msg;
}
```

La factura sunt adugate pas cu pas, id-ul comenzii, data, clientul, nr produselor si valoare per produs si totalul.

```
public List<Comanda> bon (int id, int idClient) {
    Connection connection = null;
    PreparedStatement statement = null;
    ResultSet resultSet = null;
    String query = "SELECT * FROM comanda WHERE id= " + id + " and idClient= " +
idClient;
    try {
        connection = ConnectionFactory.getConnection();
        statement = connection.prepareStatement(query);
        System.out.println(statement);
        resultSet = statement.executeQuery();
        return createObjects(resultSet);
    } catch (SQLException e) {
        e.printStackTrace();
    } finally {
        ConnectionFactory.close(resultSet);
        ConnectionFactory.close(statement);
        ConnectionFactory.close(connection);
    }
    return null;
}
```

Metoda bon care se afla in clasa ComandaDAO realizeaza o interogare in tabela comanda dupa id-ul comenzii si al clientului si returneaza o lista cu

produsele comandate de acel client. Aceasta lista este folosita in metoda messege, pentru a putea stii ce produse se trec pe factura.

Aplicatia se porneste din clasa Main

4.2.GUI

In clasa Gui se realizeaza interfata.

In constructor este creat frame-ul principal cu 3 butoane, iar la apasarea unui buton se apeleaza metoda corespunzatoare.

Metodele opClient, opProdus, opOrder realizeaza alte ferestre cu elemente specifice fiecarei operatii.

```
public class Gui extends JPanel {

    public Gui() {
        JFrame frame = new JFrame("Operatii");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(400, 100);

        JPanel firstPanel = new JPanel();
        JButton clientOp = new JButton("Client");
        clientOp.setPreferredSize(new Dimension(100,50));
        JButton productOp = new JButton("Product");
        productOp.setPreferredSize(new Dimension(100,50));
        JButton orderOp = new JButton("Order");
        orderOp.setPreferredSize(new Dimension(100,50));
        firstPanel.add(clientOp,BorderLayout.LINE_START);
        firstPanel.add(productOp,BorderLayout.CENTER);
        firstPanel.add(orderOp,BorderLayout.LINE_END);

        clientOp.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                opClient();
            }
        });
        productOp.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                opProdus();
            }
        });
        orderOp.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                opOrder();
            }
        });
        frame.setContentPane(firstPanel);
        frame.setVisible(true);
    }

    public void opProdus() {
        final JFrame frame = new JFrame("Operatii Produs");
```

```
        frame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        frame.setSize(800, 400);
        final JPanel panel1 = new JPanel();
        JLabel lid = new JLabel("id");
        final JTextField id = new JTextField("",3);
        JLabel ldenumire = new JLabel("denumire");
        final JTextField denumire = new JTextField("",12);
        JLabel lpret = new JLabel("pret");
        final JTextField pret = new JTextField("",5);
        JLabel lstoc = new JLabel("stoc");
        final JTextField stoc = new JTextField("",5);
        panel1.add(lid);
        panel1.add(id);
        panel1.add(ldenumire);
        panel1.add(denumire);
        panel1.add(lpret);
        panel1.add(pret);
        panel1.add(lstoc);
        panel1.add(stoc);
        panel1.setLayout(new FlowLayout());

        JPanel panel2 = new JPanel();
        JButton find = new JButton("find by id");
        JButton add = new JButton("add");
        JButton edit = new JButton("edit");
        JButton delete = new JButton("delete");
        JButton view = new JButton("view");
        panel2.add(find);
        panel2.add(add);
        panel2.add(edit);
        panel2.add(delete);
        panel2.add(view);
        panel2.setLayout(new FlowLayout());
        final JPanel panel = new JPanel();
        find.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                int findId;
                findId = Integer.parseInt(id.getText());
                ProdusBL produsBL = new ProdusBL();
                Produs p =produsBL.findClientById(findId);
                if(p==null)
                    JOptionPane.showMessageDialog(panel,"Produs inexistent!!!");
                else
                    JOptionPane.showMessageDialog(panel,p.toString());

            }
        });
        final JPanel panel3 = new JPanel();
        view.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                ProdusBL produsBL = new ProdusBL();
                JTable table = produsBL.findAll();
                table.setCellSelectionEnabled(false);
                table.setEnabled(false);
                JScrollPane t = new JScrollPane(table);
                t.setPreferredSize(new Dimension(700,150));
                panel3.add(t);
                panel.add(panel3);
                frame.setContentPane(panel);
                frame.setVisible(true);

            }
        });

        add.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                String denumireAd;
                float pretAd;
```

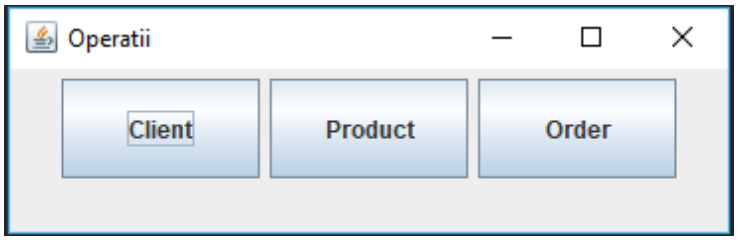
```
                int stocAd;
                denumireAd = denumire.getText();
                pretAd = Float.parseFloat(pret.getText());
                stocAd = Integer.parseInt(stoc.getText());

                Produs p = new Produs(denumireAd,pretAd,stocAd);
                ProdusBL produsBL = new ProdusBL();
                if(produsBL.insert(p)==1)
                    JOptionPane.showMessageDialog(panel,"Succes!!!");
                else if(produsBL.insert(p)==2)
                    JOptionPane.showMessageDialog(panel,"Date invalide: denumire
invalid!!!");
                else if(produsBL.insert(p)==3)
                    JOptionPane.showMessageDialog(panel,"Date invalide: pret
invalid!!!");
                else if(produsBL.insert(p)==4)
                    JOptionPane.showMessageDialog(panel,"Date invalide: stoc
invalid!!!");
            }
        });

        delete.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                int delId;
                delId = Integer.parseInt(id.getText());
                ProdusBL produsBL = new ProdusBL();
                if(produsBL.delete(delId)==1)
                    JOptionPane.showMessageDialog(panel,"Succes: S-a sters produsul
cu id = " + delId);
                else
                    JOptionPane.showMessageDialog(panel,"Produs inexistent");

            }
        });

        edit.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                String denumireUp;
                float pretUp;
                int idUp,stocUp;
                idUp = Integer.parseInt(id.getText());
                ProdusBL produsBL = new ProdusBL();
                Produs p = produsBL.findClientById(idUp);
                if(p==null)
                    JOptionPane.showMessageDialog(panel, "ID Invalid");
                if(!denumire.getText().equals("")) {
                    denumireUp = denumire.getText();
                    p.setDenumire(denumireUp);
                }
                if(!pret.getText().equals("")) {
                    pretUp = Float.parseFloat(pret.getText());
                    p.setPret(pretUp);
                }
                if(!stoc.getText().equals("")) {
                    stocUp = Integer.parseInt(stoc.getText());
                    p.setStoc(stocUp);
                }
                if(produsBL.update(p,idUp)==1)
                    JOptionPane.showMessageDialog(panel,"Succes!!!");
                else
                    JOptionPane.showMessageDialog(panel,"Date invalide!!!");
            }
        });

        panel.add(panel1);
        panel.add(panel2);
        panel.add(panel3);
        panel.setLayout(new BoxLayout(panel,BoxLayout.Y_AXIS));
```

```
        frame.setContentPane(panel);
        frame.setVisible(true);

    }

    public void opOrder() {
        JFrame frame = new JFrame("Comanda");
        frame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        frame.setSize(600, 200);
        final JPanel panel = new JPanel();
        JPanel panel1 = new JPanel();
        JLabel lid = new JLabel("idComanda");
        final JTextField id = new JTextField("",3);
        JLabel lidc = new JLabel("idClient");
        final JTextField idc = new JTextField("",3);
        JLabel lidp = new JLabel("idProdus");
        final JTextField idp = new JTextField("",3);
        JLabel lcant = new JLabel("cantitate");
        final JTextField cant = new JTextField("",3);
        panel1.add(lid);
        panel1.add(id);
        panel1.add(lidc);
        panel1.add(idc);
        panel1.add(lidp);
        panel1.add(idp);
        panel1.add(lcant);
        panel1.add(cant);

        JPanel panel2 = new JPanel();
        JButton add = new JButton("Adauga produs");
        JButton finish = new JButton("Plasare comanda");
        panel2.add(add);
        panel2.add(finish);

        add.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                int idAdd,idcAdd,idpAdd,cantAdd;
                idAdd = Integer.parseInt(id.getText());
                idcAdd = Integer.parseInt(idc.getText());
                idpAdd = Integer.parseInt(idp.getText());
                cantAdd = Integer.parseInt(cant.getText());
                Comanda com = new Comanda(idAdd,idcAdd,idpAdd,cantAdd);
                ComandaBL c = new ComandaBL();
                int ok = c.insert(com);
                if(ok==2)
                    JOptionPane.showMessageDialog(panel,"ID Client Invalid!!!");
                else if(ok==3)
                    JOptionPane.showMessageDialog(panel,"ID Produs Invalid!!!");
                else if(ok==4)
                    JOptionPane.showMessageDialog(panel,"Cantitate Invalida!!!");
                else if(ok==5)
                    JOptionPane.showMessageDialog(panel,"Stoc Insuficient!!!");
                else if(ok==1)
                    JOptionPane.showMessageDialog(panel,"Produs adaugat cu
succes!!!");
            }
        });

        finish.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                int idAdd,idcAdd,idpAdd,cantAdd;
                idAdd = Integer.parseInt(id.getText());
                idcAdd = Integer.parseInt(idc.getText());
                idpAdd = Integer.parseInt(idp.getText());
                cantAdd = Integer.parseInt(cant.getText());
                Comanda com = new Comanda(idAdd,idcAdd,idpAdd,cantAdd);
                ComandaBL c = new ComandaBL();
```

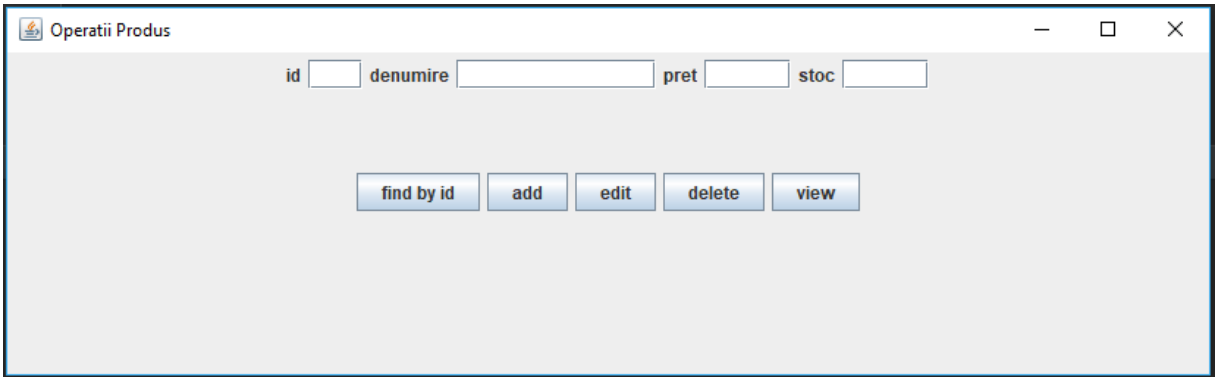
```
                try {
                    PrintWriter writer = new PrintWriter("factura.txt", "UTF-8");
                    writer.print(c.message(com));
                    writer.close();
                    JOptionPane.showMessageDialog(panel,"Comanda finalizata");
                } catch (FileNotFoundException a) {
                    a.printStackTrace();
                } catch (UnsupportedEncodingException a){
                    a.printStackTrace();
                }
            }
        });

        panel.add(panel1);
        panel.add(panel2);
        panel.setLayout(new BoxLayout(panel,BoxLayout.Y_AXIS));

        frame.setContentPane(panel);
        frame.setVisible(true);
    }

}
```

5.Testare

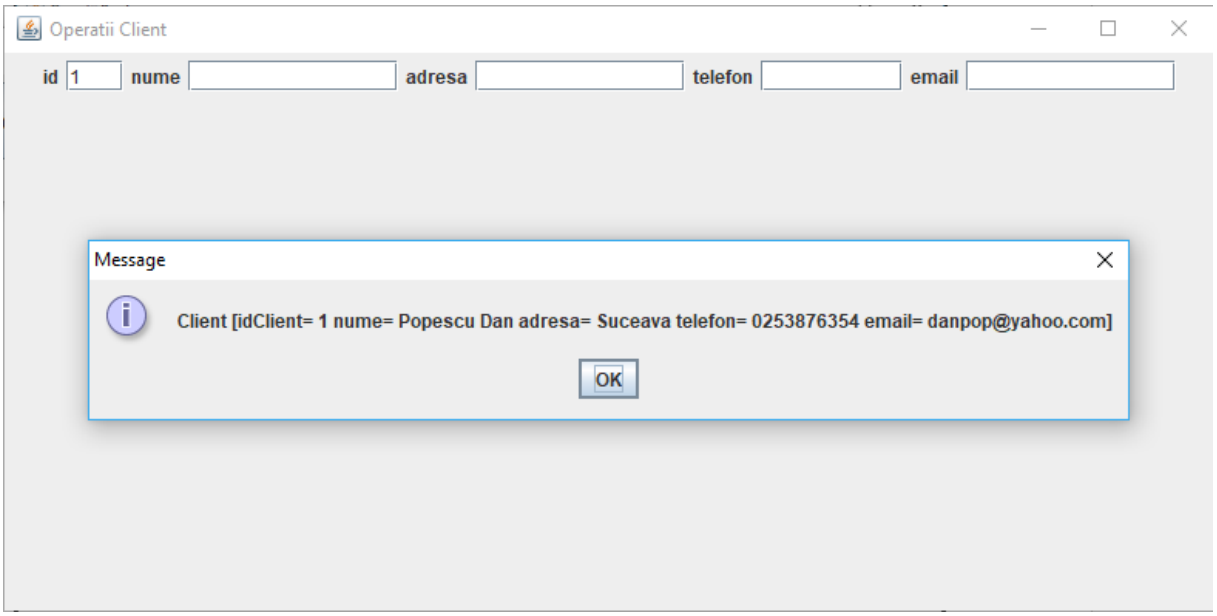
Din panoul principal se alege operatia dorita.

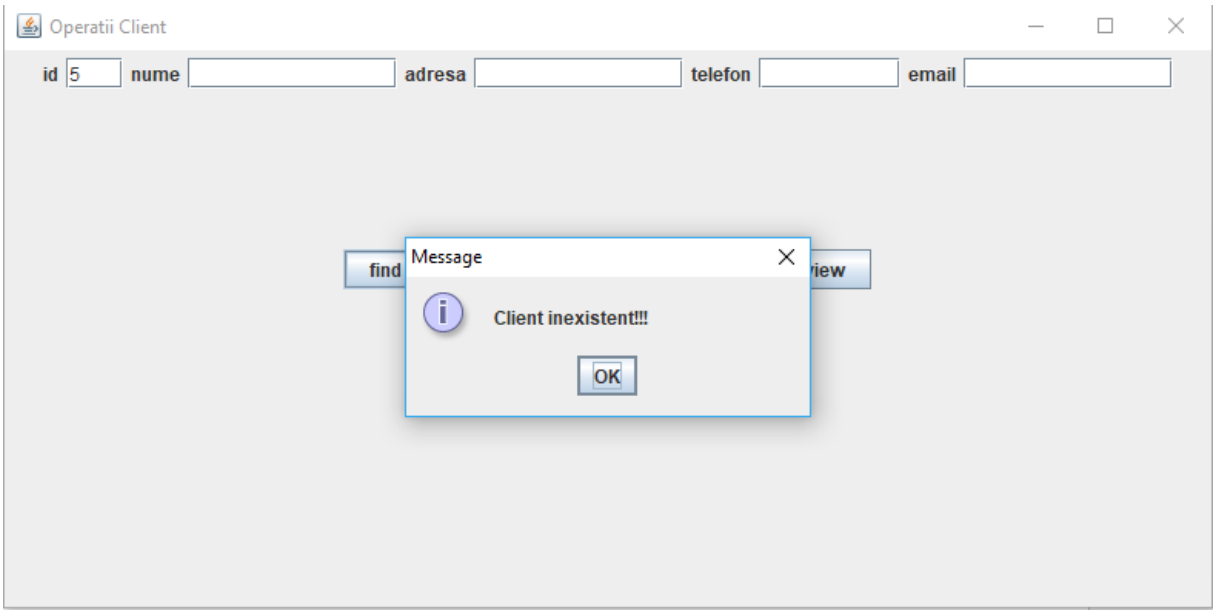


La apasarea butonului Client sau Product va aparea o noua fereastra ce va contine campurile specifice acestor tabele si butoane cu operatiile ce se pot efectua asupra lor.

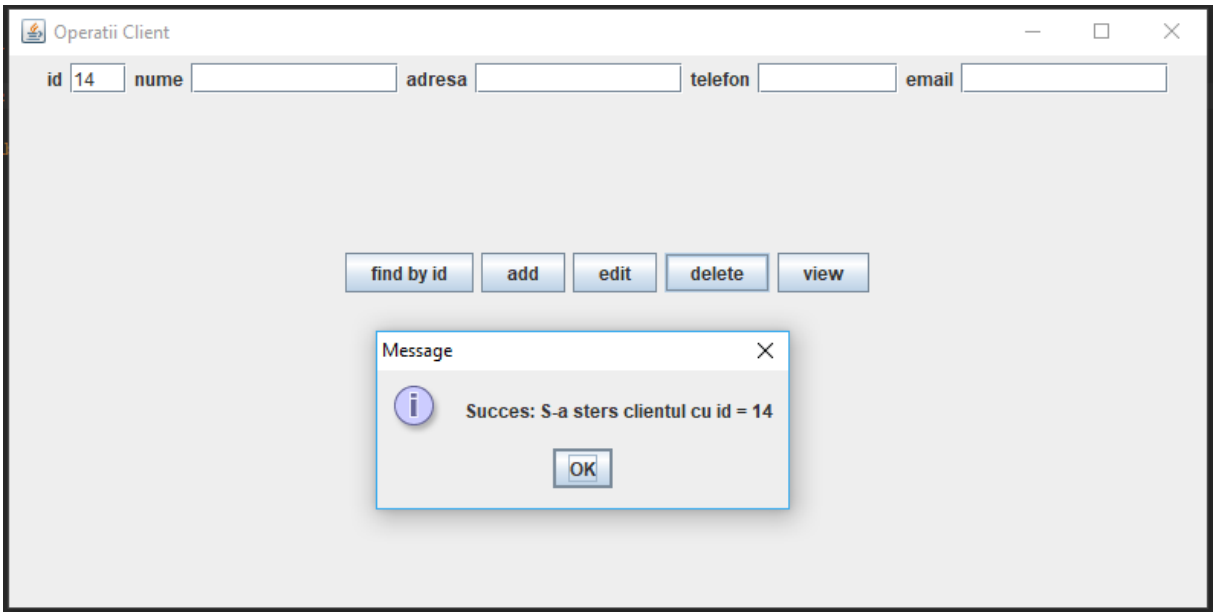


Pentru a afisa detaliile despre un anumit client sau produs se intoduce id-ul si se apasa butonul „find by id” si va fi afisat un mesaj de succes sau eroare.

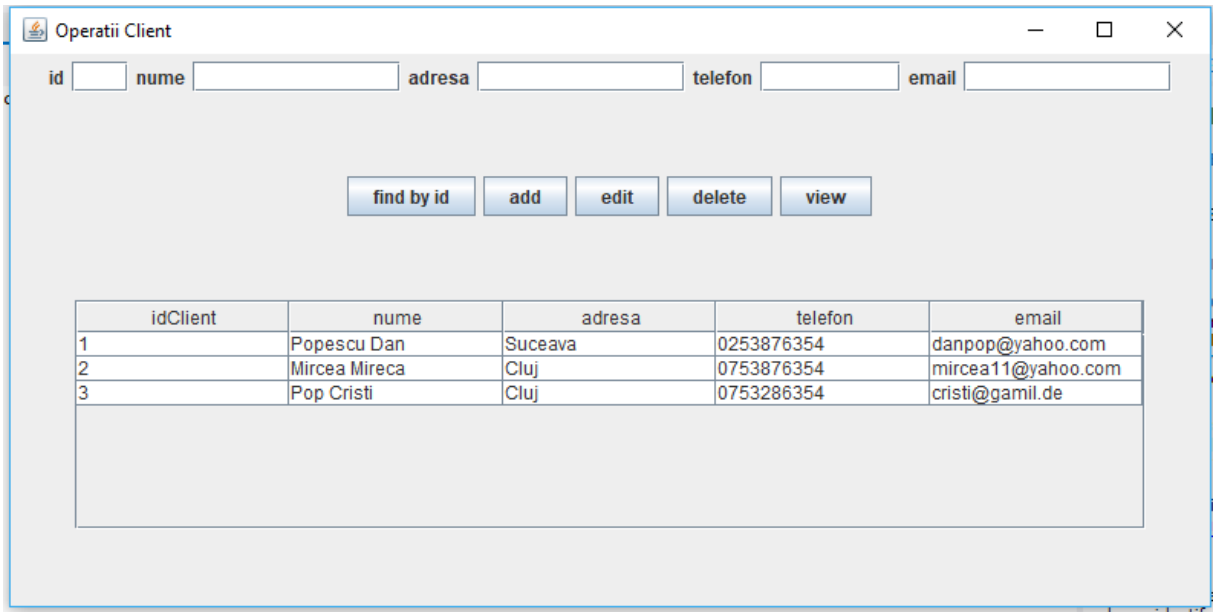




Pentru stergere se introduce id-il si se apasa butonul „delete”.

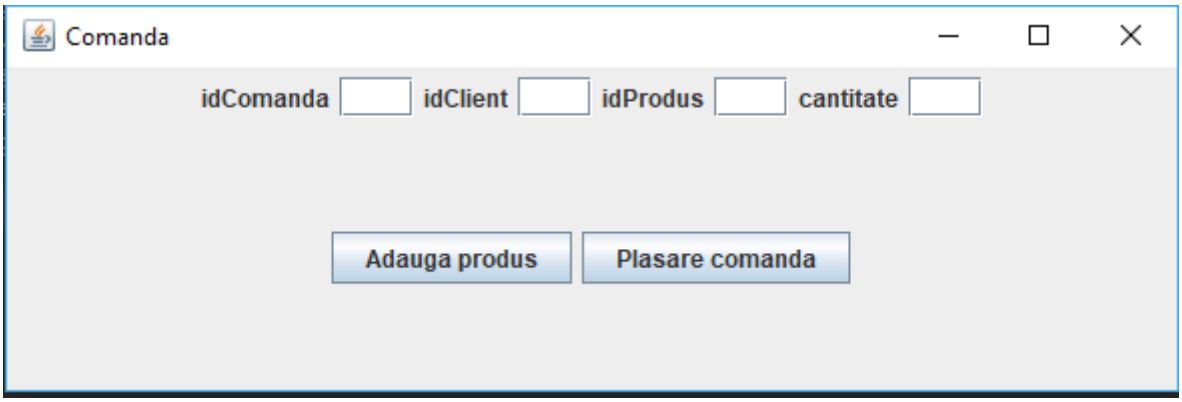


La apasarea butonului „view” se va afisa un tabel cu toti clientii.



Pentru adaugare se introduc date in toate campurile, mai putin id (acesta este generat automat) si se apasa butonul „add”, iar pentru editare se introduce id-ul clientului si valori in campurile care se doaresc a fi modificate.

La apasarea butonului Order se va deschide o fereastra noua in care se poate introduce o noua comanda.



Pentru a dauga produse in comanda se introduce idComanda, idClient si pt fiecare produs idProdus si cantitate si se apasa „Adauga produs” si se va afisa un measj de succes sau „Stoc insuficient”. Penrtru terminarea comenzii se apasa „Plasare comanda” si se va genera factura.

6.Rezultate

Aplicatia gestioneaza o baza de date alacatuita din 3 tabele (Clienti, Produs, Comanda) si se pot efectua diferite operatii asupra acesteia: cautare, adaugare, editare, stergere si adaugarea unei comenzi.

7.Concluzii

In aceasta tema am invatat sa lucrez cu o baza de date din Java.

Aplicatia se poate dezvolta ulterior adaugand tabele in baza de date (se poate rezolva relatia many-to-many dintre comanda si pordus). Se mai poate lucra si la interfata si validarea datelor.

8.Bibliografie

www.stackoverflow.com