Managementul comenzilor

Podaru Bogdan

Capitolul 1.Obiectivul temei

Obiectivul principal al lucrarii este realizarea unei modalitati de procesare a unei comenzi folosind paradigma de programare POO,tehnica de reflection pentru metodele care modeleaza operatiile pe obiectele comenzii si baze de date relationale pentru obiectele care intervin in constructia comenzii.

1.1.Obiectivele secundare sunt adresate in urmatoarele capitole :

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

Cadrul de cerinte functionale e realizat de adaugarea , stergerea editarea unui element dintr-un tabel al bazei de date ,precum si de realizarea unei comenzi (perechi de intrari din tabele , ex:client-produs). Se vor prezenta descrieri de use-case impreuna cu lista pasilor de executie a fiecarui use-case.

Cap.3 : Proiectarea diagramelor UML , a algoritmilor si intefetelor utilizator

Se vor prezenta decizii legate de proiectarea claselor , structurilor de date, interfetelor utilizate in descrierea problemei , precum si a relatiile dintre acestea , organizarea in pachete precum si diagramele de clase si pachete utile in modelarea OOP pentru prelucrarea produselor,clientilor si a comenzilor cu acestea.

Cap.4 : Implementarea propriu-zisa

Prezentarea campurilor , constructorilor , metodelor importante care realizeaza corespondenta OOP-actiuni specifice bazelor de date cu ajutorul SQL.Se detaliaza de ce s-a ales tehnica de reflection pentru implementare.Tot aici se va prezenta interfata cu utilizatorul si cateva exemple de utilizare.

Cap5. : Concluzii

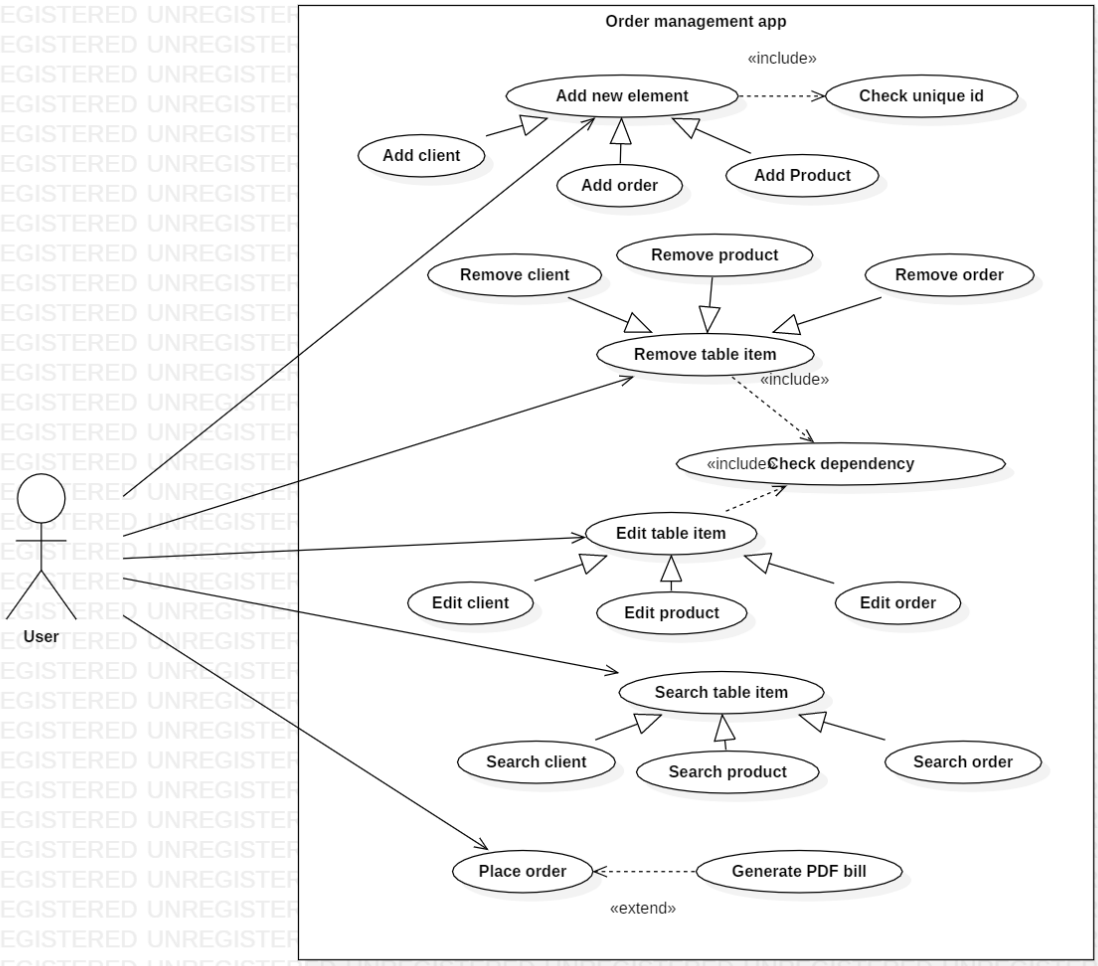
In acest capitol , se discuta concluzii , cunostiinte dobandite , precum si dezvoltari posibile ulterioare.

Cap6. : Bibliografie

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

Se prezinta in continuare functionalitatile oferite de aplicatie sub forma unei diagrame ce prezinta principalele cazuri de utilizare.

Diagrama contine 5 cazuri principale de utilizare si toate rezuma management-ul unei aplicatii de facut comenzi cu ajutorul unei baze de date . Cazurile principale sunt : adaugarea unui nou element in tabele, stergerea unui element, editarea unui element ,cautarea unui element si realizarea unei comenzi (corespondenta intre tabele). Fiecare caz principal presupune unele secundare care reprezinta particularizarea unui caz principal la o operatie concreta pe unul dintre tabele asa cum se observa din urmatoarea diagrama use case :



Se prezinta cazurile principale: adaugare si crearea unei comenzi , celelate 3 fiind similare cu cel de adaugare , cel de editare avand un grad de complexitate putin mai ridicat.

- **Use Case: Adauga un element nou**
- **Primary Actor: Utilizatorul aplicatiei**
- **Main Success Scenario:**

1. Utilizatorul alege un tabel.
2. Utilizatorul apasa butonul de adaugare a unui element nou.
3. Utilizatorul e ghidat spre introducerea datelor coresunzatoare elementului nou
4. Datele sunt verificare pentru a fi respectate constrangerile tabelelor (se verifica id unic).
5. Se proceseaza datele .
6. Se actualizeaza tabelul vizual.
7. Se actualizeaza baza de date.

- **Alternative Sequences:**

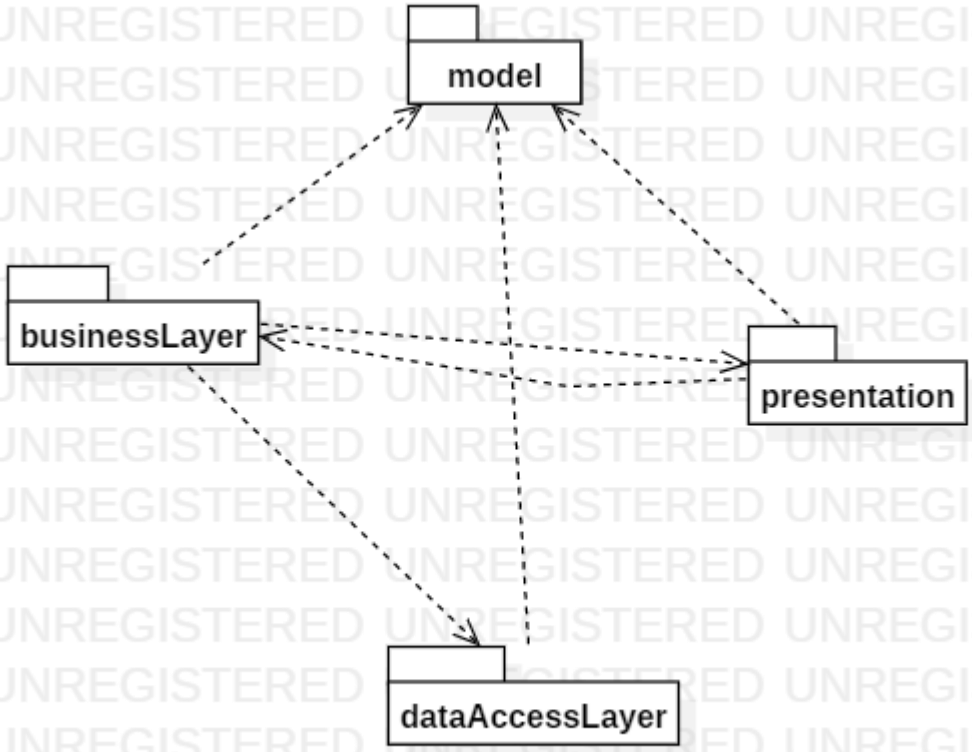
1. Id-ul elementului nou adaugat exista deja in tabel.
2. Se afiseaza un mesaj coresunzator (“Id existent”).
3. Utilizatorul e ghidat spre aplicatia principala.

- **Use Case : Crearea unei comenzi**
- **Primary Actor: Utilizatorul aplicatiei**
- **Main Success Scenario:**

1. Utilizatorul apasa butonul de creare de comanda nou din tabelul “orders”
2. Utilizatorul e ghidat spre introducerea datelor corespunzatoare comenzii noi (selecteaza un client , un produs si o cantitate pentru produs)
3. Este intrebat daca doreste sa genereze facture(sub forma pdf) sau nu

Capitolul 3. Proiectare (decizii de proiectare, diagrame UML, structuri de date, proiectare clase, interfete, relatii, packages, algoritmi, interfata utilizator)

Proiectul respecta urmatoarea arhiectura de proiectare fiind impartit in 4 pachete : businessLayer, dataAccessLayer, model, presentation pentru a facilita modularizarea si eventuala extindere a proiectului . Acest lucru e tradus in urmatoarea diagrama de pachete:



Se observa dependenta puternica (reciproca) intre logica aplicatiei si prezentare (materializarea logicii) , precum si utilizarea extensiva a pachetului model de care depend toate clasele , pentru ca acesta contine entitatile pe care se desfasoara toate operatiile aplicatiei

Pachetul model

Aceste contine clasele utile in modelarea elementelor cu care se fac operatiile : Client, Product si Order. Totodata acestea reprezinta concretizarea metodelor de reflection din API-ul de reflection al java, listele de obiecte care dau starea curenta a aplicatiei si corespondenta 1:1 cu elementele bazei de date.

Pachetul businessLayer contine urmatoarele clase care modeleaza logica aplicatiei :

Interfata Manipulation care este implementata de ClientManipulation, OrderManipulation si ProductManipulation . Fiecare dintre acestea contin metode de adaugare,stergere ,editare si cautare a unor elemente . La apelul acestor metode se actualizeaza baza de date si tabelele vizibile folosind logica de implementare.

Pachetul dataAccessLayer

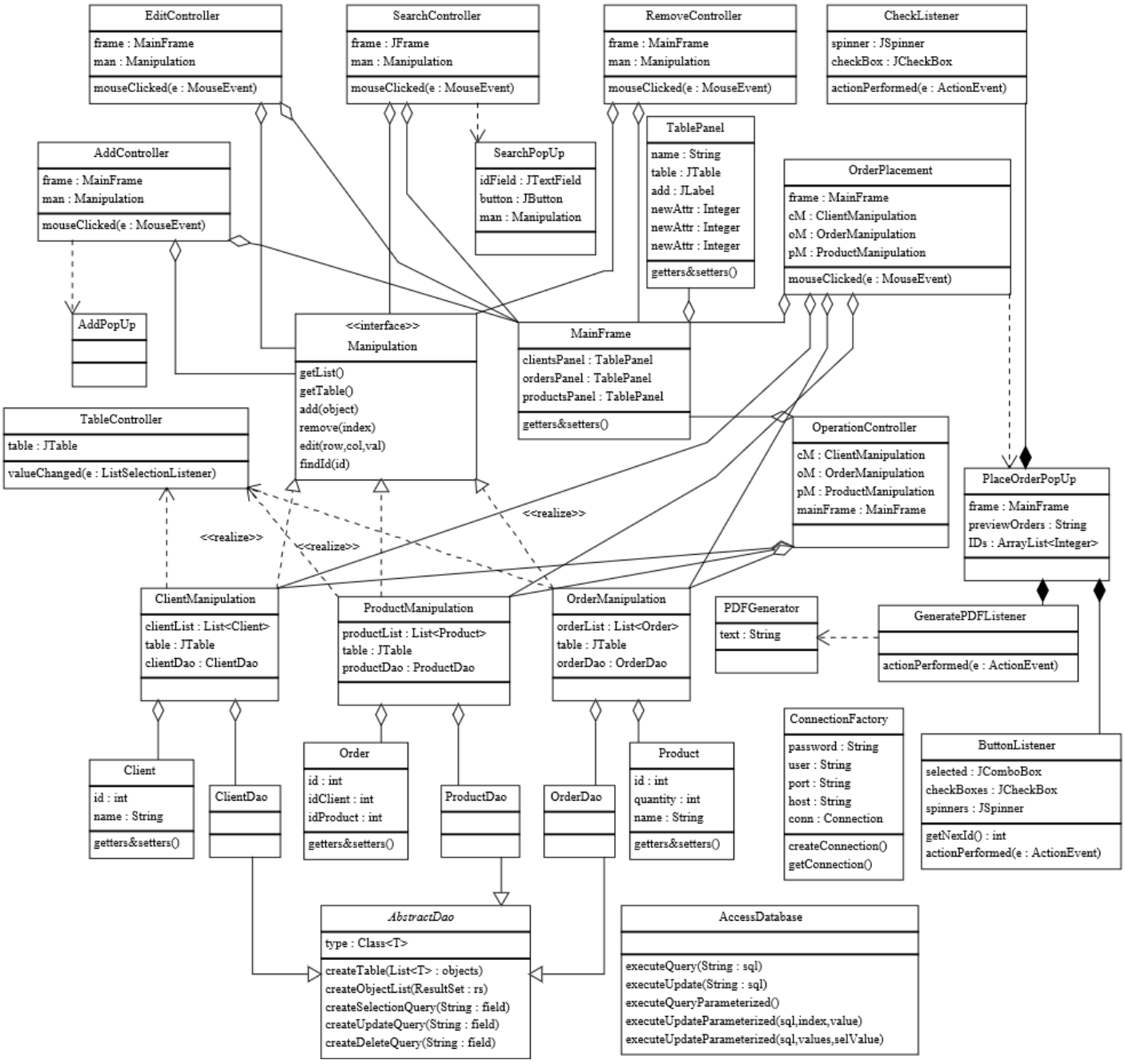
Pachet util pentru interactiunea cu baza de date si implementarea de logica generic a. Acesta contine clasa abstracta “AbstractDao”care contine metode generice realizate cu tehnica de reflection precum si 3 clase care extind pe cea de mai sus , realizandu-se astfel executia metodelor pe obiecte concrete : ProductDao, ClientDao si OrderDao. Tot aici intalnim si 2 clase utile In interactiunea cu baza de date: clasa ConnectionFactory care intretine conexiunea cu baza de date si AccessDatabase care contine metode pentru executarea de comenzi SQL.

Pachetul presentation

Aici intalnim perechi de clase de tipul urmator : fereastra-controller , unde fereastra reprezinta ceea ce utilizatorul vede (butoane,campuri pentru introducere date) si controller care specifica ce anume sa se intample la interactiunea cu elementele ferestrei. Clasa AddController deschide frame-ul AddPopUp pentru adaugarea de elemente noi. Similare sunt si urmatoarele: SearchController-SearchPopUp, EditController,RemoveController,PlaceOrderController-OrderPlacement,etc. Alte 2 clase importante sunt MainFrame care este prima fereastra care se deschide la pornirea aplicatiei care contine mai multe obiecte din clasa TablePanel si care sunt controlate de TableController.

Nota: In aplicatie se mai intalneste si pachetul main care contine metoda de start main care seteaza clasele de manipulare si primul frame al aplicatiei (primul contact pe care utilizatorul il are la deschiderea aplicatiei).

Toate aceste pachete si clasele continute prezinta relatiile descrise de urmatoarea diagrama de clase:



Cap.4 : Implementarea propriu-zisa

Urmeaza detalii legate de implementarea celor mai importante clase , deoarece cateva prezinta similaritati . Se ia pentru explicatie urmatoarele 2 cazuri de functionare : editarea unui produs si generarea unei comenzi noi , se explica fiecare pas prin care trece un utilizator pentru a ajunge la rezultatul dorit ,facandu-se legetura cu portiunile de cod care realizeaza acesti pasi.

1. **Utilizatorul doreste sa porneasca aplicatia** deci intra in discutie metoda main care pregateste obiectele necesare pentru operatiile cu tabele (ClientDao,OrderDao si ProductDao) fereastra principala a aplicatiei (mainframe) , obiectele responsabile pentru tinerea in evidenta a statutului aplicatiei si tabelelor(ClientManipulation,OrderManipulation si ProductManipulation) , si un controller pentru operatii (OperationController) :

```
ClientManipulation cM=new ClientManipulation(new ClientDao());
ProductManipulation pM=new ProductManipulation(new ProductDao());
OrderManipulation oM=new OrderManipulation(new OrderDao());
MainFrame mainFrame=new MainFrame(cM,oM,pM);
new OperationController(cM,oM,pM,mainFrame);
```

La instantierea lui “pM” avem parte de clasa care tine in evidenta starea aplicatiei :

```
this.productDao=productDao;
List<Product> products=productDao.projectionQuery();  ← se creeaza tabelele vizual
productList=products;
JTable table= productDao.createTable(products);
this.table=table;
```

Metoda projectionQuery() din clasa AbstractDao , apelata pe un obiect care o extinde (products) preia prin API-ul de reflection o lista de produse :

```
String sql=createProjectionQuery();
ResultSet resultSet=AccessDatabase.executeQuery(sql);
return createObjectList(resultSet);
```

Metoda createTable() apelata pe lista de produse obtinuta mai sus va returna un JTable folosit in interfata grafica de un obiect TablePanel instantiat ulterior:

```
DefaultTableModel tableModel = new DefaultTableModel(columnNames, 0);
JTable table = new JTable(tableModel);
for (T object : objects) {          ←parcurge lista de obiecte
```

```
ArrayList<Object> oneRowData = new ArrayList<Object>(); ← un rand in tabel
for (Field field : object.getClass().getDeclaredFields()) {
    …
    oneRowData.add(field.get(object)); ←valorile campurilor obiectului current
…
tableModel.addRow(oneRowDataArray);←adauga in structura tabelului
```

Metoda createObjectList(ResultSet rs) e o metoda generica care apelata pe obiectul productList va retura o lista de produse:

```
while (rs.next()) {
  Field[] fields = type.getDeclaredFields();  ← se preiau campurile din clasa Product
  Class[] argTypes = new Class[fields.length]; ←se preiau tipurile campurilor
  Object[] fieldsData = new Object[fields.length]; ← se preiau valorile din tabel
  int i = 0;
  for (Field field : fields) {      ← se parcurg campurile clasei Produs
    argTypes[i] = field.getType();//get field types
    Object value = rs.getObject(field.getName());  ← se preia valoarea coloanei cu
acelasi nume din tabel
    fieldsData[i] = value;
    i++;
  }
  Constructor desiredConstructor = type.getConstructor((Class<T>[]) argTypes);
  T instanceObject = (T) desiredConstructor.newInstance(fieldsData);← apel
constructor
  list.add(instanceObject);← se adauga produs nou in lista
```

La instantierea unui obiect de tip OperationController se adauga urmatorii controlleri:

```
mainFrame.getProductsPanel().getAdd().addMouseListener(new AddController(mainFrame,pM));
mainFrame.getProductsPanel().getRmv().addMouseListener(new RemoveController(mainFrame,pM));
mainFrame.getProductsPanel().getEdit().addMouseListener(new EditController(mainFrame,pM));
mainFrame.getProductsPanel().getSearch().addMouseListener(new SearchController(mainFrame,pM))
mainFrame.getOrdersPanel().getAdd().addMouseListener(new AddController(mainFrame,oM));
mainFrame.getOrdersPanel().getRmv().addMouseListener(new RemoveController(mainFrame,oM));
mainFrame.getOrdersPanel().getEdit().addMouseListener(new EditController(mainFrame,oM));
mainFrame.getOrdersPanel().getOrder().addMouseListener(new OrderPlacementController (
mainFrame,cM,oM,pM));
```

mainFrame.getOrdersPanel().getSearch().addMouseListener(new SearchController(mainFrame,oM));

Ne intereseaza ceea ce face EditController si OrderPlacementController (astfel s-a ales exemplul)care preiau obiectele de manipulare care tin evidenta interfetei cu utilizatorul si starea aplicatiei(listele de produse,clienti si comenzi)
MainFrame e alcatuit din 3 TablePanel : ordersPanel, clientsPanel si productsPanel care gestioneaza tabelele cu elemente corespunzatoare (clienti,comenzi,produse).Totodata mainframe contine metode de acces pentru aceste panouri (TablePanel) care la randul lor contin metode de access la JLabel-urilor care contin imagini si care se manifesta asemenea unor butoane . Pe aceste "butoane" se pun ascultatori (ex :EditController)). La apasarea unui astfel de buton , in cazul nostru editare sau adaugare comanda se deschide un nou pop-up PlaceOrderPopUp pentru comanda sau se trateaza un caz special pentru editare.

2. **Utilizatorul alege sa editeze**

La apasarea butonului de editare ascultatorul acestuia de tip MouseAdapter adauga la randul lui asculator structurii tabelului :

```
final int i=man.getTable().getSelectedRow();
    final int j=man.getTable().getSelectedColumn();←preia coloana selectata de click
    if(i==-1)
      return;
    JTable table=man.getTable();
    DefaultTableModel model = (DefaultTableModel) table.getModel();←structura tabel
    …
    table.setDefaultEditor(Object.class , new DefaultCellEditor(new JTextField()));
    …
    table.getDefaultEditor(String.class).addCellEditorListener(new CellEditorListener() {
      public void editingStopped(ChangeEvent e) {
        man.edit(i, j, oldId);←incepe editarea
        …
      }
```

Metoda edit a interfetei Manipulation si implementata in acest caz de ProductManipulation

Preia randul si coloana celulei in curs de editare , si stabileste ce camp trebuie actualizat si cu ce valoare, lucru vazut in urmatoarele linii de cod:

```
 DefaultTableModel model = (DefaultTableModel) table.getModel();
    Object[] values={model.getValueAt(rowIndex,columnIndex)};
    Object[]
columns={table.getTableHeader().getColumnModel().getColumn(columnIndex).getHeader
Value()};
    int result=productDao.update(values,columns,field,productToEdit.getId());
← actualizeaza si in baza de date
    Product productToEdit=productDao.getObjectById(oldId);
```

productList.set(productList.indexOf(aux),editedProduct);

Ceea ce nu s-a discutat este felul in care se face actualizarea in baza de date , si anume apeland metode diferite de actualizarea dupa caz. In aces caz se apeleaza:

```
public static int executeUpdateParameterized(String sql,Object[] values,Object selvalue){
    Connection dbConnection = ConnectionFactory.getConnection();←conexiunea la BD
    …
      stmt = dbConnection.prepareStatement(sql);
      for(i=0;i<values.length;i++)
      stmt.setObject(i+1,values[i]); ←inlocuieste ? cu campuri
      stmt.setObject(i+1,selvalue); ← stabileste valoarea de proiectie
      rs=stmt.executeUpdate(); ← executa update
      ConnectionFactory.close(stmt);
      ConnectionFactory.close(dbConnection);
```

3. **Utilizatorul alege sa faca o comanda**

Controllerul butonului de “comanda” deschide o noua fereastra :

new PlaceOrderPopUp(frame,cM,pM,oM);

In fereastra noua deschisa utilizatorul alege un client prin intermediul unui JComboBox si selecteaza dintr-o lista un produs printr-un JCheckBox . La selectarea unui JCheckBox devine disponibil un JSpinner care pune la dispozitie introducerea unui intreg ce reprezinta cantitatea din produsul selectat cu care se doreste sa se faca o comanda .

```
for(int i=0;i<checkBoxes.length;i++){ ← se cauta elementele selectate prin checkbox-uri
        if(checkBoxes[i].isSelected()) {
          int orderId=getNextId();←genereaza id unic
          int clientId=selectedClient.getId();←cauta clientul care face comanda
          Product selectedProduct=pM.getList().get(i);←produsul comandat
          int productId=selectedProduct.getId();
          Order newOrder=new Order(orderId,clientId,productId);←noua comanda
          oM.add(newOrder); ← actualizeaza lista de produse
          int quantity=Integer.parseInt(spinners[i].getValue().toString());
← preia cantitatea introdusa
          int newQ=pM.getList().get(i).getQuantity()-quantity;← actualizeaza cantitatea
          pM.update(pM.getList().get(i),"quantity",newQ);←actualizeaza lista si tabelul
de produse
          previewOrders+="      --- "+selectedProduct.getName()+" in quantity of
"+spinners[i].getValue()+"\n";←pregaeteste mesaj pentru factura
        }
      }
```

Tot aici utilizatorul poate sa decida daca doreste factura sau nu. Acest lucru se traduce tot printr-un buton “Generate PDF” care poate fi apasat din greseala , din acest motiv utilizatorului I se cere sa confirme acest lucru .

```
int op= JOptionPane.showConfirmDialog(null, "Open pdf now?", "Question",
JOptionPane.YES_NO_OPTION);
        if(op==0)
            Runtime.getRuntime().exec("rundll32 url.dll,FileProtocolHandler " + fileName);
```

Capitolul 6. Concluzii

Subiectul proiectului este complex , si prezinta multe posibilitati de implementare insa varianta aleasa a adus urmatoarele cunostiinte dobandite : realizarea de diagrame de pachete, diagrame use case , metode de implementare pentru interactiunea cu baze de date respectand paradigma OOP,realizarea unei conexiuni cu IDE-ul MySQL Workbench, intelegerea modului de lucru cu metodele de realizare a statement-urilor SQL , intelegerea si punerea in practica a tehnicii si API-ului de reflection si cel mai important : realizarea corespunzatoare a unui proiect Java de la proiectare pana la implementare ,testare si crearea unei documentatii.

Proiectul poate fi dezvoltat in multe directii datorita claselor cu caracter generic insa propunerea adusa ar fi adaugarea de noi tabele si crearea de noi relatii pentru o mai buna detaliere a comenzilor , adaugarea de noi operatii posibile pe elementele tabelelor sau chiar crearea unei aplicatii care sa gestioneze mai multe aplicatii de management al comenzilor in care aceasta aplicatie sa fie suboordonata.

Capitolul 7:Bibliografie

Tehnica de reflection :

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

Utilizarea JTabel :

https://docs.oracle.com/javase/tutorial/uiswing/components/table.html