

Order management

Documentație

Nume: Moș Robert

Grupa: 30228

Cuprins

1. Obiectivul problemei
2. Analiza problemei
   1. Diagrama Use Case
3. Proiectarea problemei
   1. Diagrama de clase UML
4. Implementarea problemei
   1. Clase
   2. Metode
5. Rezultatele problemei
6. Concluzii
7. Bibliografie
8. Obiectivul problemei

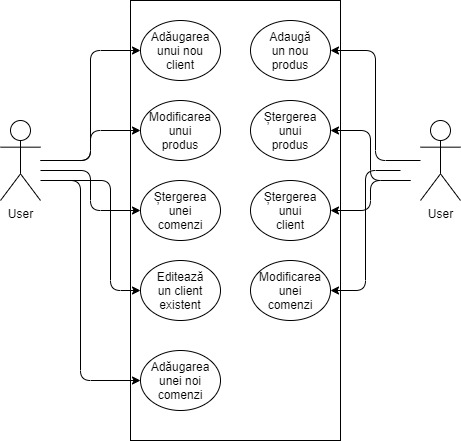
Obiectivul acestei teme este realizarea unui program in Java care simulează comenzi de produse date de către anumiți clienți. Cerința programului era ca acesta să conțină minim 3 tabele Client, Produs si Comandă în care se vor stoca datele clientului (id, nume, adresă), datele produsului (id, numele produsului, cantitatea, prețul), datele comenzii (id-ul comenzii, id-ul clientului, id-ul produsului, cantitatea produsului). Programul va trebui să afișeze clienții, produsele și comenzile din baza de date. Se vor implementa operații de inserare în tabel a unui client, a unui produs sau a unei comenzi, ștergere a acestora sau modificarea lor. După inserarea datelor despre client, produs si comanda se va genera si un bon unde va fi trecută suma comenzii.

1. Analiza problemei

Problema a fost rezolvată în mare parte după modelul pe care l-am primit în materialul de suport. Cu ajutorul programului se pot insera noi clienți, se pot modifica datele clienților, se pot șterge clienții, se pot adăuga noi produse, se pot modifica informațiile acestora, se pot șterge, iar pentru comenzi, la fel se pot insera noi comenzi, se pot modifica si se pot șterge.

Programul este împărțit în mai multe pachete: DataModel, DAO, BLL, Connection, Presentation pentru vizualizarea mai clară a programului.

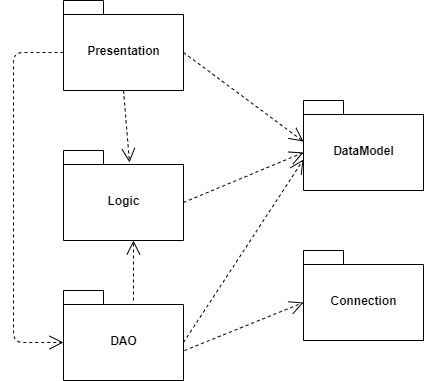
* 1. Diagrama Use Case

În acastă diagramă sunt prezentate toate operațiile care se pot face în program, adăugare client, stergere client, modificare client, adăugare produs, ștergere produs, editare produs si adăugare comandă, editare comandă și ștergere comandă.

1. Proiectarea problemei

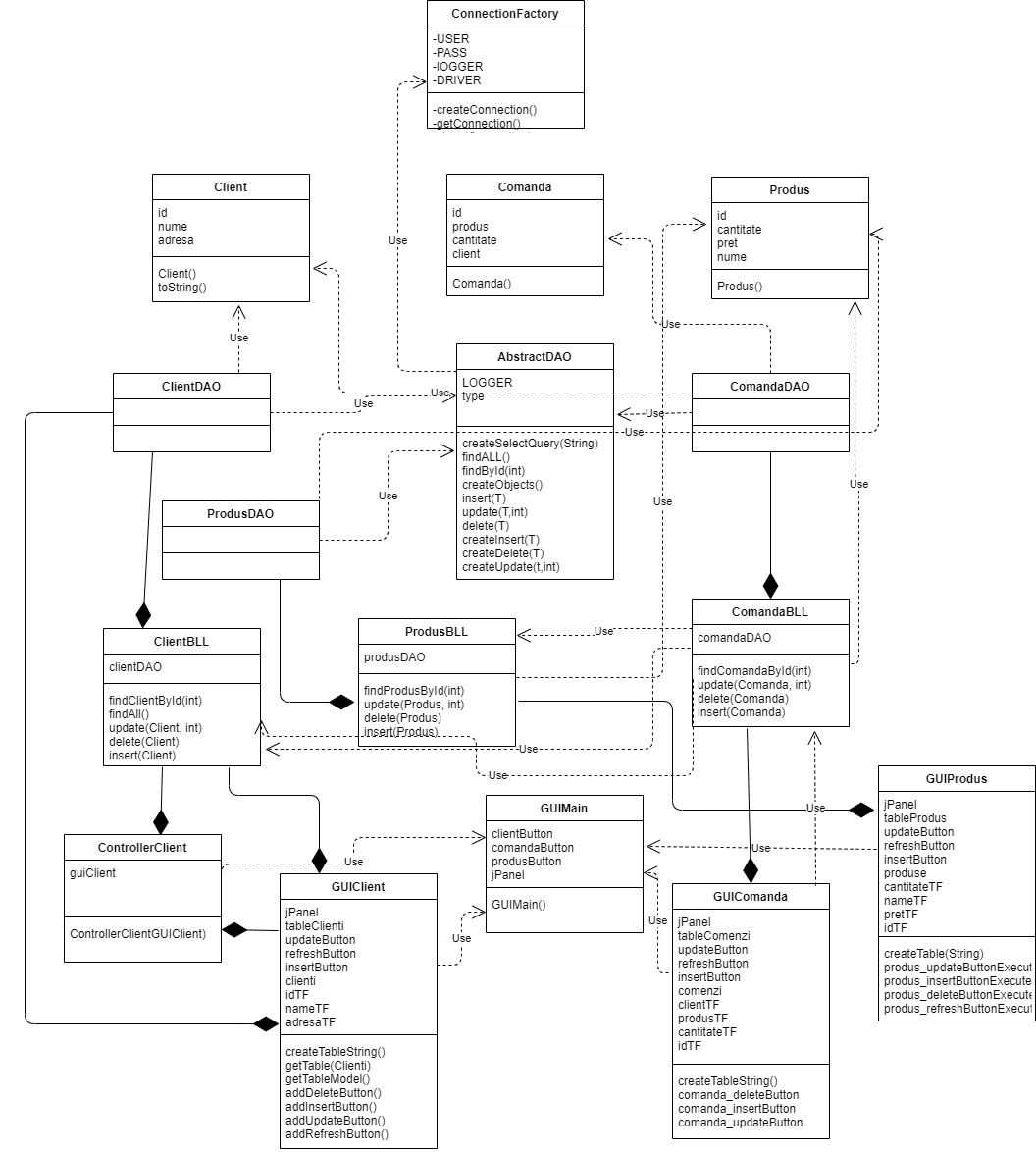
Pachetele Java ajută la organizarea mai multor module și grupează clase și interfețe. Eu pentru această problemă fiind mai complexă am folosit mai multe pachete, unul numit Logic in care am introdus clasele ClientBLL, ComandaBLL, ProdusBLL, unul numit Connection în care am introdus clasa Connection Factory, unul numit DAO în care am introdus clasele AbstractDAO, ClientDAO, ComandaDAO, ProdusDAO, unul DataModels în care am introdus clasele Client, Comandă, Produs iar unul numit Presentation în care am introdus Controller, View, GUI. Organizarea claselor în package-uri este foarte utilă pentru că oferă un aspect mai ordonat și ajută să înțelegem mai bine despre ce este vorba în programul respectiv.

Diagrama de pachete



* 1. Diagrama de clase UML

Unified Modeling Language sau UML pe scurt este un limbaj standard pentru descrierea de modele si specificatii pentru software. UML a fost la bază dezvoltat pentru reprezentarea complexității programelor orientate pe obiect, al căror fundament este structurarea programelor pe clase, și instanțele acestora ( numite și obiecte ). Diagrama UML ajută foarte mult la implementarea unui program pentru că iti poți face o idee mai clară la cum va arăta programul tău si va fi mai ușor de implementat.



1. Implementarea problemei
   1. Clase

Pentru rezolvarea temei propuse a fost nevoie de folosirea mai multor clase fiind o problema mai complexă care face și o conexiune cu o bază de date pentru a stoca datele despre client, produs și comandă. Construcția claselor a fost făcută după modelul prezentat în laborator.

Clasa Client: Această clasă conține informațiile despre client, cum ar fi id, nume si adresă. Id-ul este un număr unic pentru client de tipul int, numele este de tipul string și adresa care este tot de tipul string. De asemenea în această clasă se mai găsesc și settere si gettere pentru atributele clientului.

Clasa Comanda: Această clasă conține informațiile despre comanda, cum ar fi id, cantitate, client si produs. Id-ul este un număr unic pentru comandă de tipul int, cantitatea este un numar intreg care se referă la câte produse se vor comanda, client este de tipul int si arată id-ul clientului, iar produs este tot de tipul int care arată id-ul produsului care se dorește comandat.

Clasa Produs: Această clasă conține informațiile despre produsul care se dorește a fi cumpărat, cum ar fi id, cantitate, pret, nume. Id-ul este un număr unic de tip întreg pentru fiecare produs, cantitate este un număr de tipul int care ne arată câte produse sunt în stoc. Pret este un număr de tipul double care ne arată prețul pe care îl are un produs anume, iar nume este o valoare de tipul String care stochează numele fiecărui produs.

Clasa AbstractDAO: În această clasă se face legătura dintre baza de date și program, sunt definite funcțiile de creare a stringului pentru accesarea tabelei, functia de findall, findById, createObjects, insert care inserează un client, un produs sau o comanda in tabelul corespunzator fiecaruia, update care face update la un client, o comanda sau un produs si delete care șterge un client, un produs sau o comandă.

Clasa ClientDAO: Este clasa prin care se face legătura cu baza de date și conține funcțiile de insert, update și delete

Clasa ProdusDAO: Este clasa prin care se face legătura cu baza de date și conține funcțiile de insert, update și delete

Clasa ComandaDAO: Este clasa prin care se face legătura cu baza de date și conține funcțiile de insert, update și delete

Clasa ConnectionFactory: Cu ajutorul acestei clase se face conexiunea la baza de date pentru a stoca datele despre client, produs și comandă. Această clasă are un LOGGER, DRIVER, DBURL, USER, PASS și metode care ajută la conexiunea cu baza de date.

Clasa ClientBLL: În această clasă se află toate operațiile din ClientDAO, cum ar fi findClientById, findALL, delete, update și insert pentru a putea accesa datele din baza de date.

Clasa ProdusBLL: În această clasă se află toate operațiile din ProdusDAO, cum ar fi findProdustById, findALL, delete, update și insert pentru a putea accesa datele din baza de date.

Clasa ComandaBLL : În această clasă se află toate operațiile din ComandaDAO, cum ar fi findComandaById, findALL, delete, update și insert pentru a putea accesa datele din baza de date.

Clasa GUIMain: În această clasă este implementată fereastra principală care conține butoane pentru Client, Produs și Comandă, prin apăsarea lor se va deschide o nouă fereastră la fiecare dintre acestea unde se vor vedea datele stocate în baza de date despre fiecare din acestea în parte.

Clasa GUIClient: În această clasă se implementează interfața pentru fereastra clientului. Această clasă va conține crearea unui tabel specific pentru client unde se vor afișa datele corespunzătoare clientului din baza de date, din acest tabel se vor șterge dacă se dorește datele despre un client, se va insera un nou client sau se vor modifica datele unui client.

Clasa GUIProdus: În această clasă se implementează interfața pentru fereastra produsului. Această clasă va conține crearea unui tabel specific pentru produs unde se vor afișa datele corespunzătoare produsului din baza de date, din acest tabel se vor șterge dacă se dorește datele despre un produs, se va insera un nou produs sau se vor modifica datele unui produs

Clasa GUIComanda: În această clasă se implementează interfața pentru fereastra comenzii. Această clasă va conține crearea unui tabel specific pentru comandă unde se vor afișa datele corespunzătoare comenzii din baza de date, din acest tabel se vor șterge dacă se dorește datele despre o comandă, se va insera o nouă comandă sau se vor modifica datele unei comenzi.

* 1. Metode

Metode din clasa ClientBLL:

public Client findClientById(int id) {  
 Client cl = clientDAO.findById(id);  
 if (cl == null) {  
 throw new NoSuchElementException("The client with id =" + id + " was not found!");  
 }  
 return cl;  
}

Această metodă este metoda implementată în clasa AbstractDAO folosită pentru căutarea unui client după id în baza de date.

public Client update(Client c,int id) throws ClassNotFoundException, IllegalAccessException {  
 Client c2=clientDAO.update(c,id);  
 return c2;  
}

Această metodă este implementată în clasa AbstractDAO care modifică informațiile unui client existent în baza de date.

public void delete(Client c){  
 clientDAO.delete(c);  
}

Această metodă este implementată în clasa AbstractDAO care șterge informațiile unui client existent în baza de date.

public Client insert(Client c){  
 Client c2=clientDAO.insert(c);  
 return c2;  
}

Această metodă este implementată în clasa AbstractDAO pentru inserarea unui nou client în baza de date.

Metode din clasa ComandaBLL:

public List<Comanda> findAll(){  
 return comandaDAO.findAll();  
}

Această metodă afișează toate câmpurile unei comenzi din baza de date.

public void factura(Comanda com){  
 String factura="Factura\_nr\_"+com.getId()+".txt";  
 File fileOutput = new File(factura);  
 FileWriter write = null;  
 try {  
 write = new FileWriter(fileOutput);  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 PrintWriter pw = new PrintWriter(write);  
  
 ClientBLL cbll=new ClientBLL();  
 Client client=cbll.findClientById(com.getClient());  
 ProdusBLL pbll=new ProdusBLL();  
 Produs produs=pbll.findProdusById(com.getProdus());  
  
 String continut="Factura "+com.getId()+"\n"+  
 "Cumparator: "+client.getName()+"\n"+  
 "Produs: "+produs.getNume()+"\n"+  
 "Pret/buc: "+produs.getPret()+" Cantitate: "+com.getCantitate()+"\n"+  
 "Pret total: "+com.getPretTotal();  
 pw.println(continut);  
 pw.close();  
}

Această metodă este folosită pentru crearea unei facturi a unei comenzi, se creează un fișier .txt în care se vor afișa detaliile despre comandă cum ar fi id-ul comenzii, numele cumpărătorului, numele produsului, prețul pe bucată și cantitatea produsului și prețul total al comenzii.

Pentru clasa ProdusBLL avem aceleași metode de căutare, de editare, inserare, ștergere ca și în clasele prezentate anterior.

Clasa ConnectionFactory:

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;  
}

Această metodă face legătura cu baza de date folosită pentru stocarea informațiilor.

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

Această clasă returnează conexiunea la baza de date.

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

Această metodă este folosită pentru a întrerupe conexiunea la baza de date.

În clasa AbstractDAO:

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

Această metodă returnează un string în care este scrisă interogarea pentru afișarea din baza de date.

public T insert(T t) {  
 // *TODO:* Connection connection = null;  
 PreparedStatement statement = null;  
 ResultSet resultSet = null;  
 String query = createInsert(t);  
  
  
 try {  
 connection = ConnectionFactory.*getConnection*();  
 statement = connection.prepareStatement(query);  
 statement.execute();  
  
 } catch (SQLException e) {  
 *LOGGER*.log(Level.*WARNING*, type.getName() + "DAO:findById " + e.getMessage());  
 } finally {  
 ConnectionFactory.*close*(resultSet);  
 ConnectionFactory.*close*(statement);  
 ConnectionFactory.*close*(connection);  
 }  
  
 return t;  
}

În această metodă se creează un string în care vom avea interogarea pentru insert, apoi se face conexiunea, se creează interogarea apoi se execută.

private String createInsert(T t) {  
 String s = "INSERT INTO `ordermanagementdb`.`" + type.getSimpleName().toLowerCase() + "` (";  
 try {  
 Class cls = Class.*forName*("ro.tuc.tp.tema3.DataModels." + type.getSimpleName());  
 Field[] fields = cls.getDeclaredFields();  
 for (int i = 0; i < fields.length; i++) {  
 fields[i].setAccessible(true);  
 s = s + "`" + fields[i].getName() + "`";  
 if (i == fields.length - 1) {  
 s = s + ") ";  
  
 } else {  
 s = s + ", ";  
 }  
 }  
 s = s + "VALUES(";  
 for (int i = 0; i < fields.length; i++) {  
 fields[i].setAccessible(true);  
 try {  
 s = s + "\'" + fields[i].get(t) + "\'";  
 } catch (IllegalAccessException e) {  
 e.printStackTrace();  
 }  
 if (i == fields.length - 1) {  
 s = s + ") ;";  
  
 } else {  
 s = s + ", ";  
 }  
 }  
 System.*out*.println(s);  
 return s;  
 } catch (ClassNotFoundException e) {  
 e.printStackTrace();  
  
 }  
 return null;  
  
}

În această metodă se scrie interogarea specifică inserării in tabel a unui client, produs sau comandă.

private String createUpdate(T t, int id) throws ClassNotFoundException, IllegalAccessException {  
 //Update `ordermanagementdb`.`+type.getSimpleName().toLowerCase()+` WHERE `id`=  
 //UPDATE `ordermanagementdb`.`client` SET `id` = '5', `name` = 'bvc', `adresa` = 'asd' WHERE (`id` = '4');  
 String s="Update `ordermanagementdb`.`"+type.getSimpleName().toLowerCase()+"` SET ";  
 Class cls2 = Class.*forName*("ro.tuc.tp.tema3.DataModels." + type.getSimpleName());  
 Field[] fields = cls2.getDeclaredFields();  
 for (int i = 0; i < fields.length; i++) {  
 fields[i].setAccessible(true);  
 if(i<fields.length-1)  
 s = s + "`" + fields[i].getName() + "` = \'"+fields[i].get(t)+"\', ";  
 else  
 s = s + "`" + fields[i].getName() + "` = \'"+fields[i].get(t)+"\'";  
 }  
 s=s+" WHERE (`id`=\'"+id+"\');";  
 return s;  
  
}

Această metodă creează interogarea necesară pentru a face un update la un client, comandă sau produs în baza de date.

private String createDelete(T t){  
 //DELETE FROM `ordermanagementdb`.`+type.getSimpleName().toLowerCase()+` WHERE `id`=  
 String s="DELETE FROM `ordermanagementdb`.`"+type.getSimpleName().toLowerCase()+"` WHERE `id`=";  
 try {  
 Field field = Class.*forName*("ro.tuc.tp.tema3.DataModels." + type.getSimpleName()).getDeclaredField("id");  
 field.setAccessible(true);  
 s = s +"\'" + field.get(t)+"\'";  
 return s;  
 } catch (NoSuchFieldException e) {  
 e.printStackTrace();  
 } catch (ClassNotFoundException e) {  
 e.printStackTrace();  
 } catch (IllegalAccessException e) {  
 e.printStackTrace();  
 }  
 return null;  
}

Această metodă creează interogarea pentru ștergere din baza de date a clientului, a produsului, a comenzii, în funcție de ce se dorește a sterge.

În clasa Client: Avem o metodă numită Client care conține un id, un nume și o adresă. Avem settere si gettere pentru fiecare din acestea.

În clasa Comandă avem o metodă în care se stochează cantitatea produsului dorit, id-ul comenzii, id-ul clientului, id-ul produsului și prețul total al comenzii. De asemenea pentru fiecare din acestea avem câte un setter si getter.

În clasa Produs avem la fel un id care reprezintă id-ul produsului, un String nume în care se stochează numele produsului, un int numit cantitate în care se stochează numărul de produse aflate pe stoc și un double folosit pentru preț care va afișa prețul pe bucată a produsului. De asemenea avem si gettere si settere pentru fiecare din acestea.

În clasa GUIComandă avem o metodă care creează tabelul pentru comandă, o metodă care șterge o comandă selectată prin apăsarea unui buton. Mai avem o metodă care creează butonul pentru comanda de inserare, datele pentru comandă se scriu intr-un text field apoi se adaugă în tabel după apăsarea butonului de insert. De asemenea, mai avem o metodă care creează un buton pentru update și un buton pentru refresh care afișează informațiile stocate în tabel.

In clasele GUIClient și GUIProdus avem aceleași butoane care fac același lucru pentru tabelul specific.

În clasa GUIMain avem 3 metode care creează butoanele de la fereastra de început.

1. Rezultatele problemei

În ceea ce privește testarea problemei, programul meu funcționează fără nicio problemă. Pentru testarea problemei la fiecare comandă trebuie să se execute intr-o anumită ordine, pentru a insera un client sau un produs nu este nevoie de o constrângere, dar pentru a insera o comandă trebuie să existe clientul specificat si produsul specificat. Pentru a testa operația de ștergere trebuie să existe ceva valori în tabel să avem ce să ștergem. De asemenea pentru testarea inserării unei comenzi, unui produs sau unui client este nevoie să avem date în tabel pentru a le putea edita.

1. Concluzii

Având în vedere tema propusă, a fost una destul de interesantă, foarte complexă care mi-a dat destul de mult timp de gândit. De asemenea a fost o temă foarte practică, un mini model de aplicație pentru comandat produse cum sunt și aplicațiile din viața reală. Prin facerea aceste teme mi-am mai aprofundat cunoștiințele ce privesc bazele de date și programarea în Java.

1. Bibliografie

Ca surse de informare am folosit Youtube și Wikipedia