**FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE**

**CATEDRA CALCULATOARE**

Orders Management

Documentație

Ilies Oana-Elena

Grupa 30222 | An 2 semestrul 2

Cuprins

1. Obiectiv

2. Studiul probelemei

3.Implementare

3.1 Diagrame UML

3.2 Clase

3.3 GUI

3.4 Metode

4.Concluzii

5.Dezvoltari ulterioare

6.Bibliografie

1. Obiectiv

Acest proiect se concentrează pe dezvoltarea unei aplicații sofisticate de gestionare a clienților, produselor și comenzilor într-un depozit diversificat. Scopul principal al inițiativei constă în crearea și implementarea unei soluții profesionale și bine definită.

Prin intermediul acestei aplicații, utilizatorul beneficiază de o interfață grafică interactivă, care îi permite să efectueze operații de inserare, actualizare și ștergere a clienților și produselor, precum și să plaseze comenzi prin selectarea unui client și a unui produs.

Aplicația se remarcă prin sistemul său eficient de notificare în timp real, care alertează utilizatorul cu privire la stocul excedentar sau insuficient al unui produs. Astfel, gestionarea și controlul depozitului devin mai precise și mai eficiente.

Interfața intuitivă a aplicației este împărțită în trei module distincte, asigurând un acces rapid și ușor la funcționalitățile dorite. Utilizatorul poate efectua operații asupra produselor din depozit, gestiona și actualiza informațiile referitoare la clienți și administra comenzile într-un mod simplu și eficient. Meniul din interfața Java facilitează navigarea între module și permite utilizatorului să selecteze opțiunea dorită cu doar câteva clicuri.

Fiecare panel al interfeței afișează un tabel actualizat în timp real, oferind informații despre produsele, clienții și comenzile existente în baza de date. Astfel, utilizatorul are mereu acces la datele actualizate și poate lua decizii în cunoștință de cauză.

Proiectul a fost abordat cu o metodologie riguroasă, includând etapele de analiză a problemei, modelare, definire a scenariilor și cazurilor de utilizare. O atenție deosebită a fost acordată proiectării aplicației, astfel încât să ofere o funcționalitate superioară și o experiență plăcută utilizatorului. Implementarea propriu-zisă a fost realizată cu meticulozitate și profesionalism, respectând standardele și practicile moderne de programare.

Pe lângă obiectivul principal, proiectul a urmărit și atingerea unor obiective secundare, care sprijină realizarea cu succes a aplicației de gestionare a clienților, produselor și comenzilor unui depozit. Aceste obiective includ analiza problemei și modelarea, definirea scenariilor și cazurilor de utilizare, proiectarea aplicației, implementarea propriu-zisă și testarea exhaustivă.

2.Studiul Problemei

Pentru a îndeplini cerințele temei într-un mod eficient, primul pas esențial este să avem o înțelegere solidă a bazei de date MySQL. Aceasta este de o importanță crucială, deoarece toate operațiunile, precum vizualizarea, ștergerea, actualizarea și plasarea comenzilor, vor fi efectuate direct pe baza de date.

Scopul nostru este să implementăm o aplicație de gestionare a comenzilor efectuate de diferiți clienți, care să fie utilă și accesibilă oricărui utilizator. Aplicația utilizează o bază de date externă creată în MySql WorkBench, în care sunt stocate informațiile despre clienți, produse și comenzile efectuate. Pe lângă gestionarea comenzilor, utilizatorii pot efectua și alte operațiuni, cum ar fi adăugarea, ștergerea și actualizarea informațiilor despre clienți, produse și furnizori.

Aplicația verifică și validează datele introduse pentru a asigura integritatea și corectitudinea acestora. În cazul în care se introduce o valoare invalidă sau se neglijează completarea unor câmpuri obligatorii, utilizatorul va primi o atenționare și va fi solicitată introducerea unor date valide și corecte. Orice modificare realizată în aplicație se va reflecta în timp real în baza de date, astfel încât să se asigure o sincronizare corespunzătoare.

Aplicația oferă, de asemenea, posibilitatea de a vizualiza conținutul bazei de date, cum ar fi lista clienților sau a produselor, prin intermediul unor tabele care respectă structura bazei de date. Aceasta permite utilizatorului să aibă o imagine actualizată și completă a informațiilor din depozit.

• Cerințe funcționale:

-Utilizatorul trebuie să poată selecta operațiunea dorită: operații pe tabela de clienți, operații pe tabela de produse și operații pe tabela de comenzi.

-Utilizatorul trebuie să poată adăuga un client nou.

-Utilizatorul trebuie să poată edita un client existent.

-Utilizatorul trebuie să poată șterge un client existent.

-Utilizatorul trebuie să poată adăuga un produs nou.

-Utilizatorul trebuie să poată edita un produs existent.

-Utilizatorul trebuie să poată șterge un produs existent.

Cerințe non-funcționale

Aplicația trebuie să fie intuitivă și ușor de utilizat.

Aplicația trebuie să aibă o interfață grafică interactivă care să faciliteze utilizarea

Descriere: A picture containing diagram, text, line, screenshot

Description automatically generatedCazuri de utilizare

Utilizatorul trebuie să poată crea o comandă prin selectarea unui client și a unui produs dintr-un tabel de clienți și un tabel de produse, și introducerea unei cantități valide, iar această comandă trebuie să fie apoi inserată în tabela de comenzi.

Utilizatorul accesează fereastra principală a aplicației și selectează operația dorită: Operații pe tabela de clienți.

**Client:**

Utilizatorul introduce datele noului client în câmpurile relevante din interfață.

Pentru a valida numărul de telefon, se verifică dacă numărul introdus respectă un format specific.

Utilizatorul apasă butonul "Add" pentru a adăuga noul client în baza de date.

Editare client:

Utilizatorul vizualizează lista clienților afișată în interfață și selectează clientul pe care dorește să-l editeze.

Utilizatorul introduce noile date sau modificările dorite în câmpurile corespunzătoare ale clientului selectat.

Pentru a valida numărul de telefon, se verifică dacă numărul introdus respectă un format specific.

Utilizatorul apasă butonul "Edit" pentru a actualiza informațiile clientului în baza de date conform modificărilor introduse.

Ștergere client:

Utilizatorul vizualizează lista clienților afișată în interfață și selectează clientul pe care dorește să-l șteargă.

Utilizatorul apasă butonul "Remove" pentru a elimina clientul selectat din baza de date.

Vizualizare clienți:

Utilizatorul accesează operația de vizualizare clienți din fereastra principală.

Lista clienților din baza de date este afișată în interfață.

Pentru operațiile pe tabela de produse:

Utilizatorul accesează fereastra principală a aplicației și selectează operația dorită: Operații pe tabela de produse.

**Produs:**

Utilizatorul introduce datele noului produs în câmpurile relevante din interfață.

Utilizatorul apasă butonul "Add" pentru a adăuga noul produs în baza de date.

Editare produs:

Utilizatorul vizualizează lista produselor afișată în interfață și selectează produsul pe care dorește să-l editeze.

Utilizatorul introduce noile date sau modificările dorite în câmpurile corespunzătoare ale produsului selectat.

Utilizatorul apasă butonul "Edit" pentru a actualiza informațiile produsului în baza de date conform modificărilor introduse.

Ștergere produs:

Utilizatorul vizualizează lista produselor afișată în interfață și selectează produsul pe care dorește să-l șteargă.

Utilizatorul apasă butonul "Remove" pentru a elimina produsul selectat din baza de date.

**Comenzi:**

Utilizatorul selectează un client din lista disponibilă de clienți afișată în interfață.

Utilizatorul selectează un produs din lista disponibilă de produse afișată în interfață.

Utilizatorul introduce o cantitate validă pentru produsul selectat în câmpul corespunzător.

Utilizatorul apasă butonul "Add" pentru a genera o nouă comandă.

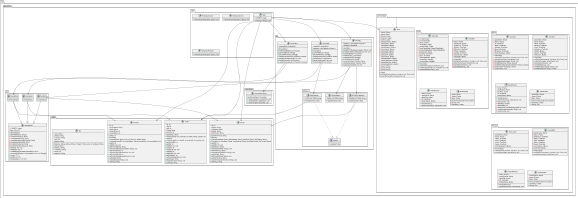
Informațiile despre comanda nouă, inclusiv clientul, produsul și cantitatea, sunt inserate în tabela de comenzi din baza de date.

Baza de date corespunzătoare comenzilor se actualizează în timp real, reflectând comanda nou creată.

Aceasta permite utilizatorului să creeze și să înregistreze noi comenzi în sistemul de gestionare a comenzilor. Pe măsură ce comenzile sunt adăugate, baza de date se actualizează pentru a reflecta starea curentă a comenzilor existente.

**3.Implementare**

3.1 Diagrame UML



3.2 Clase

**A.Business Logic Layer**

**Validators : -Date Validator:** responsabilă de validarea datelor în cadrul unei aplicații. În particular, clasa se ocupă de validarea câmpului de dată al unei comenzi.

**-Email Validator:**responsabilă de validarea emails în cadrul unei aplicații. În particular, clasa se ocupă de validarea câmpului de email al unei client.

**-NrTelefonValidators:** responsabilă de validarea numerelor de telefon în cadrul unei aplicații. În particular, clasa se ocupă de validarea câmpului de numar de telefon al unei client.

**ClientBLL:** se ocupă de operațiuni legate de clienți, cum ar fi adăugarea, actualizarea și ștergerea informațiilor despre clienți.

**OrderBLL:** se ocupă de gestionarea comenzilor, inclusiv crearea, procesarea și finalizarea lor.

**ProductBLL:** gestionează operațiunile legate de produse, cum ar fi adăugarea, actualizarea și ștergerea informațiilor despre produse.

**B.Connection:**

**ConnectionFactory:** realizeaza conexiunea intre baza de date si Java code.

**C.** **Data Access Object**

-Clasa **AbstractDAO** oferă operații generice de bază pentru accesul la date pentru un anumit tip de entitate. Această clasă este folosită ca bază pentru clasele DAO specifice care manipulează datele entităților specifice.

Clasele **ClientDAO**, **OrdersDAO**, **ProductDAO** sunt goale si fac extindere la AbstractDAO<Clients/Orders/Products>;

**D.** **Model**

Clasa **Bill** reprezintă o factură cu informații despre achiziții sau tranzacții. Este o clasă record imutabilă, cu atributele: name (nume), prName (nume produs), nrTelefon (număr de telefon), price (preț) și address (adresă).

Această clasă oferă o modalitate convenabilă de a stoca și accesa informațiile despre factură. Atributele sunt accesibile prin intermediul getterilor.

Clasa Bill este imutabilă pentru a asigura integritatea datelor și siguranța în cadrul thread-urilor.

Constructorul clasei Bill este generat implicit de către Java, deoarece este o clasă record. Acesta primește ca argumente valorile atributelor și le atribuie automat.

Clasa Bill oferă următoarele metode:

Metodele getter (name(), prName(), nrTelefon(), price(), address()) returnează valorile atributelor respective.

De exemplu, pentru a accesa numele asociat unei facturi create ca instanță a clasei Bill, puteți utiliza billInstance.name(), unde billInstance este o instanță a clasei Bill.

Clasa Bill este concepută pentru a fi folosită pentru stocarea informațiilor despre o factură și pentru a facilita accesul la aceste informații.

Clasa **Clients** reprezintă un client cu informații de bază. Aceasta oferă constructori, getteri și setteri pentru manipularea datelor despre client. Câmpul ID este generat automat. Metoda toString() returnează o reprezentare sub formă de șir a obiectului client.

Clasa Clients are următoarele atribute private:

ID - reprezintă ID-ul clientului (un număr întreg)

CName - reprezintă numele clientului (un șir de caractere)

CAddress - reprezintă adresa clientului (un șir de caractere)

CEmail - reprezintă adresa de email a clientului (un șir de caractere)

CNrTelefon - reprezintă numărul de telefon al clientului (un șir de caractere)

Clasa **Orders** reprezintă o comandă cu informații precum ID-ul comenzii, ID-ul clientului, ID-ul produsului, data comenzii și cantitatea.

Clasa Orders are următoarele atribute private:

ID - reprezintă ID-ul comenzii (un număr întreg)

cID - reprezintă ID-ul clientului asociat comenzii (un număr întreg)

pID - reprezintă ID-ul produsului asociat comenzii (un număr întreg)

Odate - reprezintă data comenzii (un șir de caractere)

quantity - reprezintă cantitatea produsului comandat (un număr întreg)

Clasa **Products** reprezintă un produs cu informații precum ID-ul, numele, prețul, stocul și detalii.

Clasa Products are următoarele atribute private:

ID - reprezintă ID-ul produsului (un număr întreg)

productName - reprezintă numele produsului (un șir de caractere)

detalii - reprezintă detalii despre produs (un șir de caractere)

productPrice - reprezintă prețul produsului (un număr întreg)

productStock - reprezintă stocul disponibil pentru produs (un număr întreg)

**E.** **Prezentation**

Cuprinde toate clasele si pachetele responsabile pentru GUI.

**F.** **Start**

Cuprinde clasele Refraction pentru clasele principale: Client, Order, Product si o clasa „Start” ce cuprinde metoda „Main” care da drumul aplicatiei.

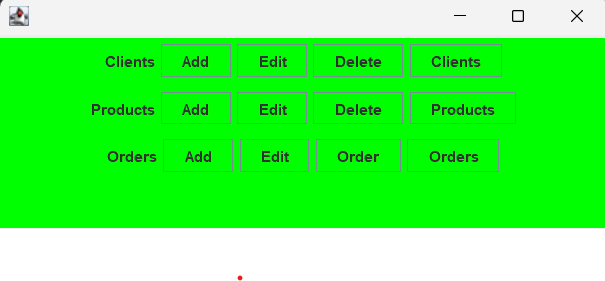
Astfel, o clasa Refraction permite inspectarea și afișarea proprietăților (câmpurilor) unui obiect dat utilizând reflexia

Exemplu: package org.tpIliesOana.start;  
  
import java.lang.reflect.Field;  
public class RefractionClients {  
  
 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();  
 }  
 }  
 }  
}

3.3 GUI

Pachetul „**Presentation”** cuprinde pachetele cu clasele specifice interfetei GUI

Prima fereastra **View** este o clasa in care se hotareaste ce actiune se va utiliza. In functie de denumirea specificata de buton in dreptul clasei corespunzatoare

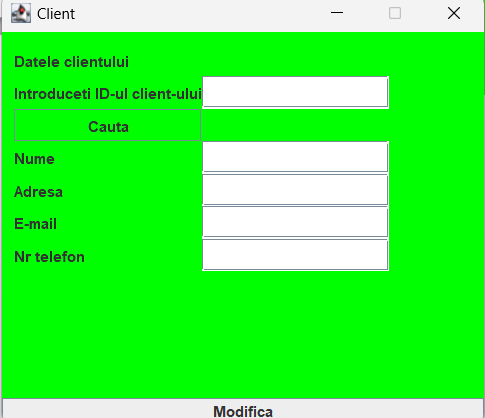
****

**Campuri Client:**

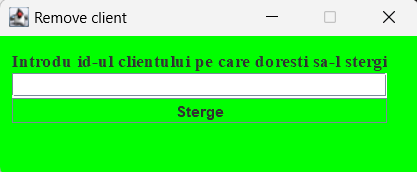
**ADD:**Aici se adauga datele in tabela Client

****

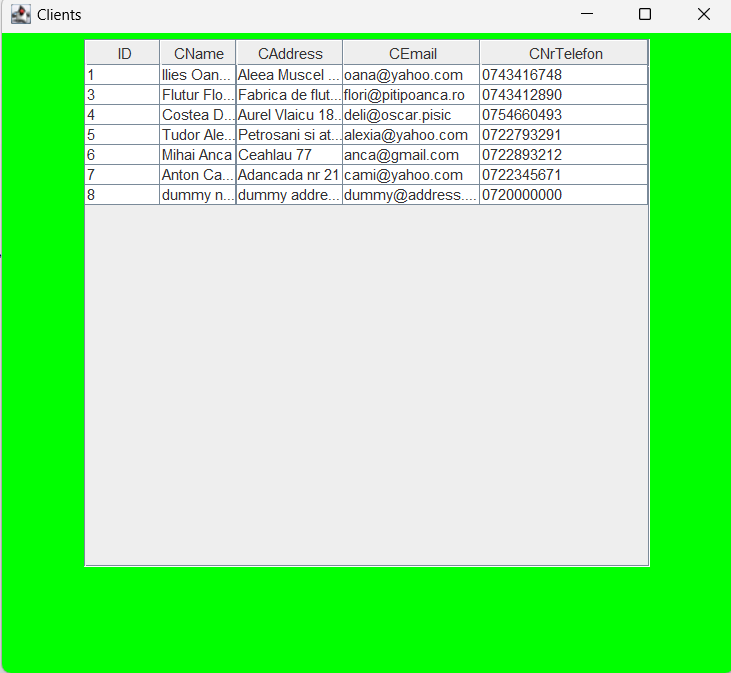
**Edit:** Se face update la datele din tabel, butonul de cauta, cauta clientul in functie de id.



**Delete**: Sterge un client in functie de ID

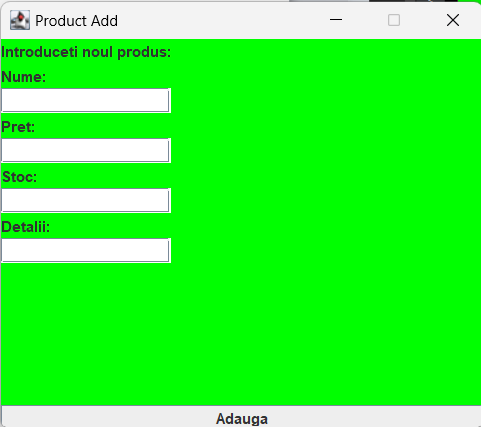


**Clients**: Aici vizualizezi tabela de client

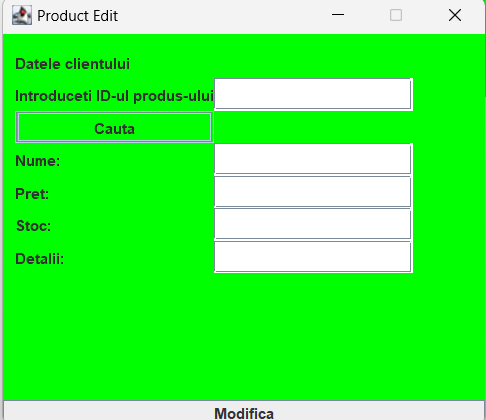


**Campuri Product:**

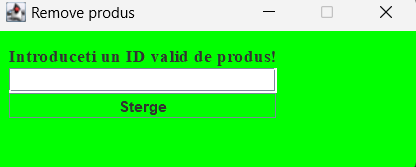
**ADD** aici sunt adauga produse in tabela din baza de date

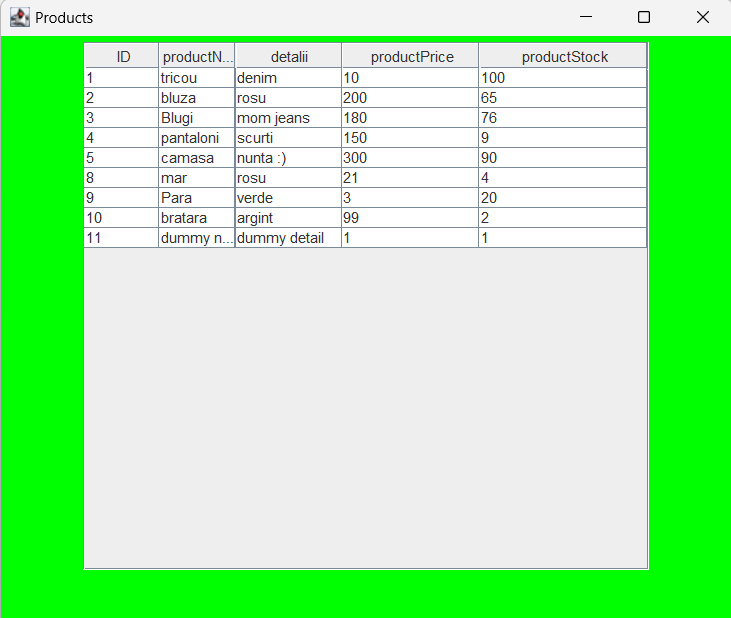


**Edit** se ideateaza datele din tabela, butonul de cautare, cauta in functie de id un produs



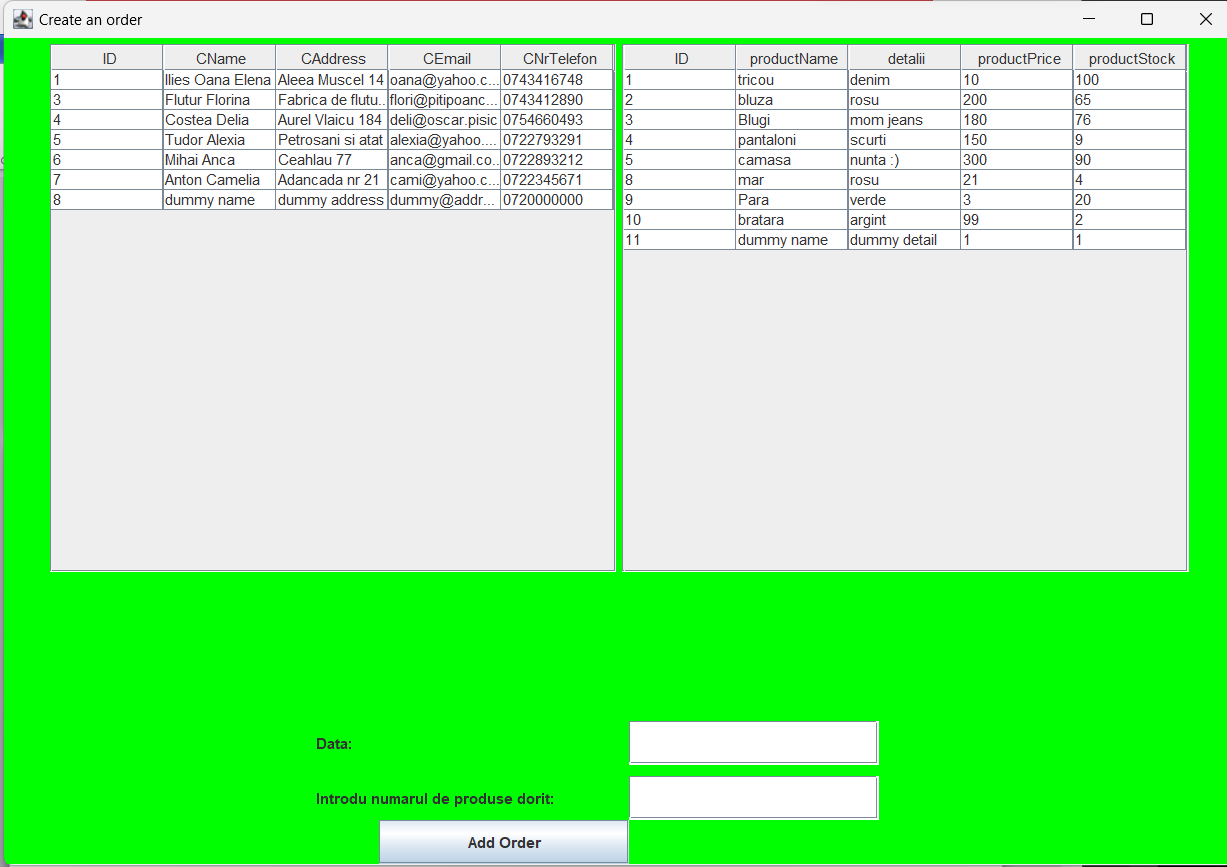
**Delelte** sterge un produs in functie de id-ul sau



**Products:** Vizualizezi tabela de produse din baza de date: 

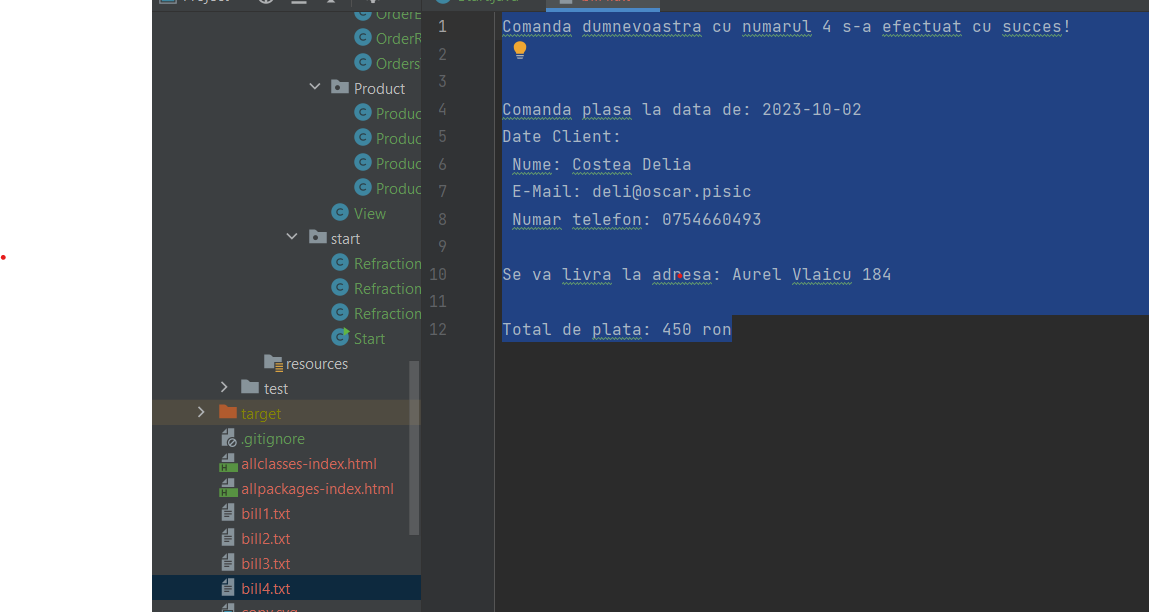
**Campul Orders:**

**ADD:** creeza o comanda in functie de un client si un produs si selecteaza in functie de o linie din tabele.

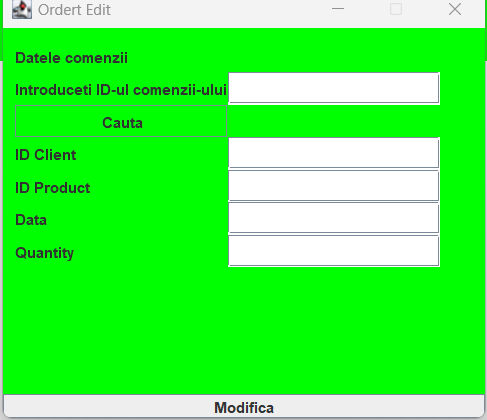


In tabele sunt prezentate datele existeste, in campul **Data** se introduce data in care se face comanda, iar al doilea camp se introduce numarul de pruse dorit, in urma acestei actiuni se realiaza o decrementare la nivelul tabelului de produs. In momentul in care se creeaza comanda, prin butonul ”Add Order” pe langa decrementarea cantitatii produsului ce a fost selectat in comanda se realizeaza si un fiser bill care cuprinde factura comenzii respective:

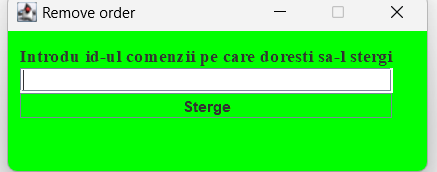
Exemplu:



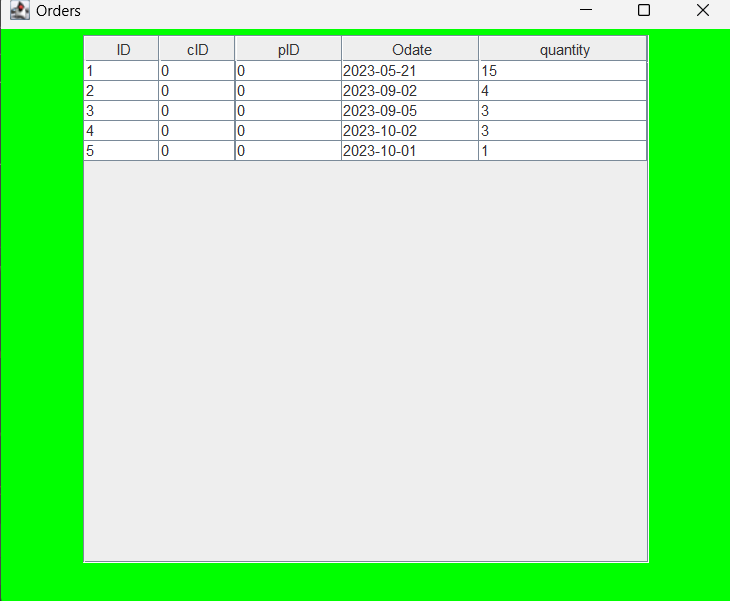
**Edit:** Face editarea tabelului „Order” dar doar la nivelul cantitatii si am detaliilor(id urile raman neschimbate)



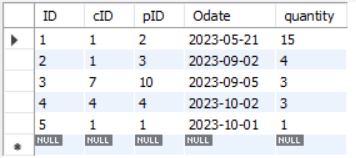
**Delete:** Sterge o comanda in functie de id



**Orders:** Se vizualizeaza tabelul Orders din baza de date:



Aici am intampinat o greseala, in afisarea tabelului in interfata id ul clientului si al produslui se va afisa cu valoarea 0. Dar in baza de date SQL tabelul curpinde datele corespunzatoare, iar in momentul printarii in consola la fel cele doua campuri din tabela sunt afisate corect.



3.4 Metode

Obținerea numelor de câmpuri ale clasei entității: Metoda getFieldNames() returnează un array de nume de câmpuri ale clasei entității.

Crearea de interogări SQL pentru operațiile de inserare, actualizare, selectare și ștergere: Metodele precum createInsertQuery(), createUpdateQuery(), createSelectQuery(), createSelectQueryAll(), createDeleteQuery() și createDeleteOrdersQuery() construiesc interogări SQL pentru a efectua operațiuni specifice asupra tabelei corespunzătoare entității.

Exemplu: private String createInsertQuery() {  
 StringBuilder sb = new StringBuilder();  
 sb.append("INSERT INTO ");  
 sb.append(type.getSimpleName());  
 sb.append(" (");  
 for (Field f : type.getDeclaredFields()) {  
 if (!f.equals(type.getDeclaredFields()[0]))  
 sb.append(f.getName() + ",");  
 }  
 sb.deleteCharAt(sb.length() - 1);  
 sb.append(") VALUES (?, ?, ?, ?)");  
 return sb.toString();  
}

Manipularea și transformarea rezultatelor obținute dintr-un ResultSet: Metodele precum createObjects() și convertListOfObjectsToArray() preiau rezultatele obținute dintr-un ResultSet și le transformă în obiecte corespunzătoare sau într-un array de string-uri.

Operații de bază CRUD (Create, Read, Update, Delete): Metodele precum insert(), findById(), findAll() și update() permit inserarea, căutarea, obținerea tuturor obiectelor și actualizarea entităților în baza de date.

Exemplu: public T findById(int id) {  
 Connection connection = null;  
 PreparedStatement statement = null;  
 ResultSet resultSet = null;  
 String query = createSelectQuery("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 FIND BY ID! " + e.getMessage());  
 } finally {  
 ConnectionFactory.*close*(resultSet);  
 ConnectionFactory.*close*(statement);  
 ConnectionFactory.*close*(connection);  
 }  
 return null;  
}

**retrieveProperties**-această metodă primește un obiect ca argument și utilizează reflexia pentru a itera prin câmpurile obiectului și a afișa valorile acestora.

private void createOrder(int clientID, int productID)

Se creează obiecte BLL (Business Logic Layer) pentru comenzile, produsele și clienții, utilizate pentru interacțiunea cu baza de date sau alte resurse necesare gestionării comenzilor.

Se construiește un obiect Orders pe baza ID-urilor clientului și produsului, precum și a datelor de introducere și cantității furnizate.

Comanda este inserată în baza de date prin intermediul obiectului ordersBLL și se obține comanda inserată.

Se caută produsul și clientul corespunzător ID-urilor furnizate utilizând metodele findProductById și findClientById.

Se construiește un obiect Bill pe baza informațiilor despre client și produs.

Se creează un fișier de tip text cu numele format din directorul specificat și ID-ul comenzii, în care se scriu informații relevante despre comandă.

Fișierul de tip text este închis.

Se afișează un mesaj de informare utilizatorului prin intermediul unui dialog JOptionPane pentru a indica salvarea comenzii.

Se actualizează stocul produsului prin scăderea cantității comenzii din stocul existent al produsului.

Se închide fereastra curentă.

4.Concluzii

Dezvoltarea unei aplicații sofisticate de gestionare a clienților, produselor și comenzilor într-un depozit diversificat: Scopul principal al proiectului este de a crea și implementa o soluție profesională și bine definită pentru gestionarea eficientă a clienților, produselor și comenzilor într-un mediu de depozit diversificat.

Interfață grafică interactivă: Utilizatorii beneficiază de o interfață grafică interactivă care facilitează efectuarea operațiilor de inserare, actualizare și ștergere a clienților și produselor, precum și plasarea comenzilor prin selectarea unui client și a unui produs.

Sistem eficient de notificare în timp real: Aplicația dispune de un sistem eficient de notificare în timp real care alertează utilizatorii cu privire la stocul excedentar sau insuficient al unui produs. Această funcționalitate contribuie la o gestionare mai precisă și eficientă a depozitului.

Interfață intuitivă și modulară: Interfața aplicației este intuitivă și împărțită în module distincte, asigurând un acces rapid și ușor la funcționalitățile dorite. Utilizatorii pot gestiona produsele din depozit, actualiza informațiile despre clienți și administra comenzile într-un mod simplu și eficient.

Actualizare în timp real a informațiilor: Prin intermediul panelurilor interfeței, utilizatorii au acces la tabele actualizate în timp real, oferindu-le informații despre produsele, clienții și comenzile existente în baza de date. Astfel, utilizatorii pot lua decizii în cunoștință de cauză având acces la datele actualizate.

Metodologie riguroasă de dezvoltare: Proiectul a fost abordat cu o metodologie riguroasă, includând etapele de analiză a problemei, modelare, definire a scenariilor și cazurilor de utilizare. S-a acordat o atenție deosebită proiectării aplicației și implementării propriu-zise, respectând standardele și practicile moderne de programare.

În cadrul problemei noastre, am utilizat o varietate de structuri de date, inclusiv tipuri de date primitive și colecții de date. Pentru a afișa și manipula datele, am utilizat componenta JTable din biblioteca Java Swing, oferind o interfață grafică intuitivă.

Am aplicat tehnici de reflexie pentru a accesa și manipula dinamic clase și obiecte în timpul execuției, oferind flexibilitate și extensibilitate.

Pentru accesul și interacțiunea cu baza de date, am utilizat conexiunea JDBC, permițând gestionarea conexiunii, executarea interogărilor SQL și obținerea rezultatelor în mod eficient și sigur.

Am implementat clasele DAO utilizând un design modular, oferind metode standardizate pentru operațiile CRUD în baza de date.

Am utilizat mecanisme de validare pentru asigurarea integrității datelor și validarea intrărilor utilizatorului.

Pentru gestionarea detaliilor facturilor, am utilizat public record, asigurându-ne că datele sunt imutabile și ușor de utilizat.

Pentru construirea și manipularea șirurilor de caractere, am utilizat variabile de tip StringBuilder, asigurându-ne o performanță eficientă.

În final, am dezvoltat o aplicație complexă și robustă de gestionare a clienților, produselor și comenzilor într-un mediu de depozit, oferind o interfață utilizator intuitivă și performantă.

5.Dezvoltari ulterioare

Pe baza proiectului de gestionare a clienților, produselor și comenzilor într-un depozit diversificat, există câteva dezvoltări ulterioare pe care le puteți lua în considerare: Implementarea unui sistem de raportare avansat: Puteți adăuga funcționalități suplimentare pentru generarea de rapoarte personalizate în aplicație. Aceste rapoarte pot include informații despre vânzări, stocuri, performanța clienților, etc. Aceasta va oferi utilizatorilor o perspectivă mai detaliată asupra afacerii și le va permite să ia decizii informate.

6.Bibliografie

1.Sierra, K., & Bates, B. (2020). OCP Oracle Certified Professional Java SE 11 Programmer I

2.Pratt, P., & Adamski, J. J. (2018). SQL Programming (6th Edition). Cengage Learning.

Oracle Java Documentation: https://docs.oracle.com/en/java/

Java Tutorials - Oracle: https://docs.oracle.com/javase/tutorial/index.html

Baeldung: https://www.baeldung.com/

Java Code Geeks: https://www.javacodegeeks.com/