DOCUMENTATIE

TEMA *NUMARUL_3*

NUME STUDENT: Catruc Alexandru-Dan
GRUPA: 30228

CUPRINS

1. Obiectivul temei

Obiectivul principal al proiectului este de a dezvolta o aplicație de gestionare a comenzilor pentru procesarea comenzilor clienților pentru un depozit. Aplicația trebuie să urmeze un model de arhitectură stratificat și să utilizeze baze de date relaționale pentru stocarea datelor. Aplicația ar trebui să includă componente GUI și să implementeze diverse funcționalități, cum ar fi adăugarea de clienți, editarea clienților, ștergerea clienților, vizualizarea informațiilor despre clienți într-un tabel, adăugarea de produse, editarea produselor, ștergerea produselor, vizualizarea informațiilor despre produs într-un tabel și crearea comenzilor de produse.

Obiective secundare:

- Realizarea unei aplicatii simple si usor de utilizat
- Validarea datelor pentru baza de date
- Scrierea de cod lizibil si bine documentat pentru viitoare implementari

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

Cerinte functionale:

Adaugarea, stergerea si editarea unui client.

Adaugarea, stergerea si editarea unui produs.

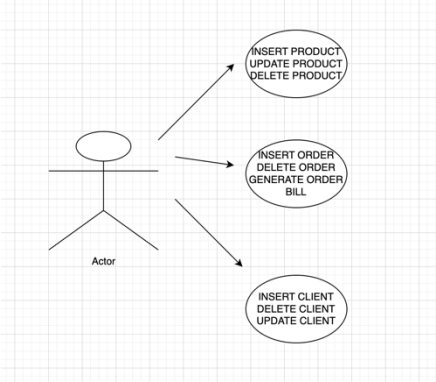
Adaugarea, stergerea si crearea de facturi pentru comenzi.

Cerinte non-functionale:

Aplicatie usor de folosit, intuitiva.

Crearea unei baze de date logica, usor de utilizat.

Validarea datelor astfel incat sa nu fie introduse date gresite.



3. Proiectare

Diagrama de clase:

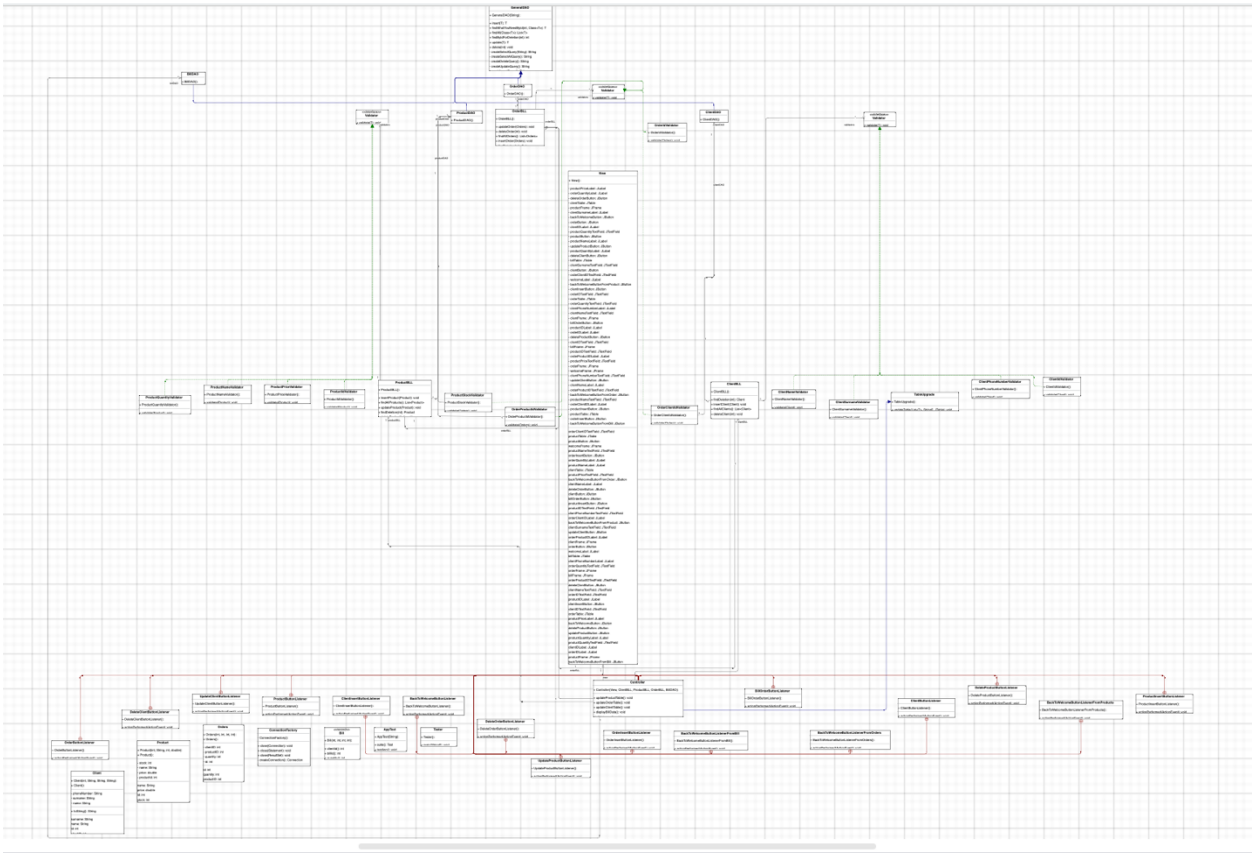
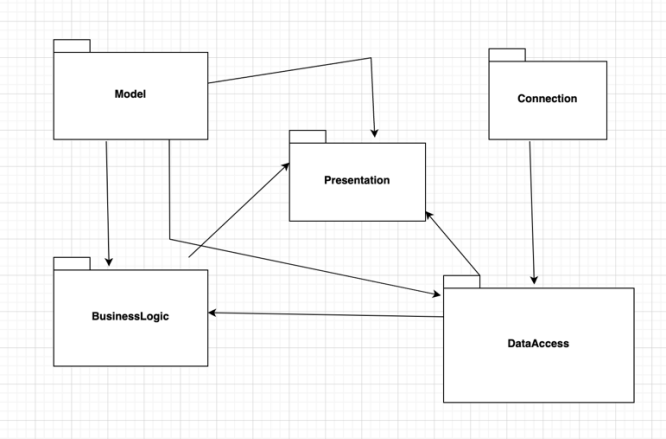


Diagrama de pachete:



4. Implementare

Pachetul Connection conține clasa ConnectionFactory, care este responsabilă pentru gestionarea conexiunii cu baza de date.

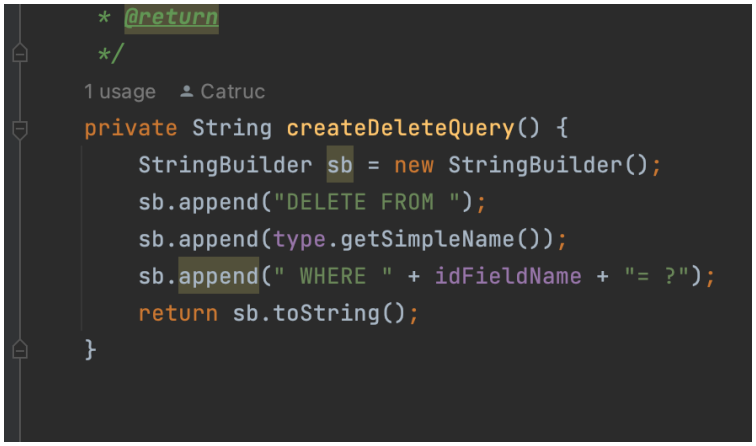
Câteva detalii despre această clasă:

1. **Atributele ConnectionFactory**: Clasa are câteva atribute private statice, inclusiv LOGGER, DRIVER, DBURL, USER și PASS. Aceste atribute rețin detalii precum driver-ul JDBC utilizat (DRIVER), URL-ul bazei de date (DBURL), numele de utilizator (USER) și parola (PASS) pentru autentificarea în baza de date. De asemenea, există o instanță statică a clasei ConnectionFactory (singleInstance) care este folosită pentru a implementa pattern-ul Singleton.
2. **Constructorul**: Constructorul privat al clasei încarcă driver-ul JDBC. Acest lucru este necesar pentru a stabili conexiunea cu baza de date.
3. **Metoda createConnection()**: Această metodă private creează o conexiune cu baza de date utilizând DriverManager. Dacă apar erori în timpul creării conexiunii, acestea sunt capturate și înregistrate în LOGGER.
4. **Metoda getConnection()**: Aceasta este metoda publică prin care alte clase obțin conexiunea la baza de date. Ea returnează rezultatul metodei createConnection() apelată pe singleInstance.
5. **Metodele close()**: Există trei metode close() care sunt utilizate pentru a închide conexiunea, statement-ul și resultSet-ul respectiv. Aceste metode sunt importante pentru gestionarea resurselor și prevenirea scurgerilor de memorie.

Pachetul DataAccess include clasa GeneralDAO, care este o clasă abstractă generică ce implementează funcționalități de bază pentru a opera asupra unei baze de date.

Aici sunt câteva detalii despre această clasă:

1. **Atributele GeneralDAO**: Clasa conține un Logger, o clasă de tip T și un șir de caractere ce reprezintă numele câmpului ID din baza de date.
2. **Constructorul GeneralDAO**: Constructorul primește un șir de caractere ce reprezintă numele câmpului ID și inițializează tipul T prin analiza tipurilor generice ale clasei.
3. **Metodele de creare a interogărilor SQL**: Clasa conține mai multe metode private ce creează interogări SQL pentru selectare, inserare, actualizare și ștergere de înregistrări în baza de date.



```
     * @return
     */
    1 usage   Catruc
    private String createDeleteQuery() {
        StringBuilder sb = new StringBuilder();
        sb.append("DELETE FROM ");
        sb.append(type.getSimpleName());
        sb.append(" WHERE " + idFieldName + "= ?");
        return sb.toString();
    }
```

4. **Metodele de executare a interogărilor SQL**: Clasa conține metode publice ce execută interogările SQL pentru inserare (insert), actualizare (update) și ștergere (delete) de înregistrări.

```
Catruc
public T insert(T object) {
    Connection connection = null;
    PreparedStatement statement = null;
    ResultSet generatedKeys = null;
    String query = createInsertQuery();

    try {
        connection = ConnectionFactory.getConnection();
        statement = connection.prepareStatement(query, Statement.RETURN_GENERATED_KEYS

        int i = 1;
        for (Field field : object.getClass().getDeclaredFields()) {
            field.setAccessible(true);
            Object value = field.get(object);
            statement.setObject(i, value);
            i++;
        }
        int affectedRows = statement.executeUpdate();

        if (affectedRows == 0) {
            throw new SQLException("Creating object failed, no rows affected.");
        }
```

5. **Metoda findAll**: Aceasta este o metodă ce returnează o listă cu toate înregistrările dintr-un anumit tabel din baza de date. Ea creează o instanță a tipului T pentru fiecare înregistrare și adaugă instanța în listă.
6. **Metodele de căutare**: Clasa conține două metode de căutare, findByIdForDeletion și findWhatYouNeedById. Prima returnează un ID dacă există o înregistrare cu acel ID, iar cea de-a doua returnează o instanță a tipului T ce conține datele înregistrării cu acel ID.

Din aceasta clasa se etind si clasele pentru Tabelele pentru care trebuie facute interogarile: ClientDAO, ProductDAO, BillDAO, OrderDAO.

Pachetul DataAccess include clasa BillDAO, care reprezintă un Data Access Object (DAO) specific pentru manipularea facturilor (bills) din baza de date.

Aici sunt câteva detalii despre această clasă:

1. **Atributele BillDAO**: Clasa conține un Logger, precum și două șiruri de caractere ce reprezintă interogările SQL pentru inserarea și selectarea tuturor facturilor.
2. **Metoda insertBill**: Aceasta este o metodă ce inserează o factură în baza de date. Ea primește ca parametri un obiect de tip Bill și un obiect de tip Orders și utilizează datele din acestea pentru a executa interogarea SQL de inserare.
3. **Metoda findAllBills**: Aceasta este o metodă ce returnează o listă cu toate facturile din baza de date. Ea execută interogarea SQL de selectare a tuturor facturilor, creează un obiect de tip Bill pentru fiecare înregistrare și adaugă obiectul în listă.

Pachetul `BusinessLogic` include clasa abstractă `TableUpgrade` care are o metodă pentru actualizarea unui tabel GUI cu date noi.

Această clasă conține metoda `updateTable` care primește o listă

de obiecte, numele coloanelor pentru tabel și tabelul care urmează a fi actualizat. Metoda este generică, ceea ce înseamnă că poate fi utilizată cu orice tip de obiect (<T>).

Iată ce face metoda pas cu pas:

1. Crează o matrice de obiecte pentru a stoca datele care urmează a fi introduse în tabel. Numărul de rânduri este egal cu dimensiunea listei de obiecte, iar numărul de coloane este egal cu numărul de nume de coloane.
2. Parcurge fiecare obiect din listă și fiecare nume de coloană.
3. Pentru fiecare pereche obiect-coloană, încearcă să obțină valoarea corespunzătoare în obiect prin apelarea metodei getter a acelei coloane. Acest lucru se face prin reflexie: construiește numele metodei getter din numele coloanei ("get" + `columnName`), obține `Method` corespunzător din clasa obiectului și apoi o invocă pe obiect.
4. Pune valoarea obținută în matricea de date la poziția corespunzătoare.
5. La sfârșit, creează un nou model de tabel folosind matricea de date și numele coloanelor și setează acest model ca model pentru tabelul primit ca parametru.

Această metodă abstractizează procesul de actualizare a unui tabel GUI cu date noi, făcându-l aplicabil oricărui tip de obiect, atâta timp cât acesta are metode getter corespunzătoare pentru fiecare coloană a tabelului.

```
3 usages  1 Catruc
public <T> void updateTable(List<T> items, String[] columnNames, JTable table) {
    Object[][] data = new Object[items.size()][columnNames.length];
    for (int i = 0; i < items.size(); i++) {
        T item = items.get(i);
        for (int j = 0; j < columnNames.length; j++) {
            String columnName = columnNames[j];
            Object value = null;
            try {
                String getterMethodName = "get" + columnName;
                Method getter = item.getClass().getMethod(getterMethodName);
                value = getter.invoke(item);
            } catch (NoSuchMethodException | IllegalAccessException | InvocationTargetException e) {
                e.printStackTrace();
            }
            data[i][j] = value;
        }
    }

    DefaultTableModel model = new DefaultTableModel(data, columnNames);
    table.setModel(model);
}
```

Clasa `ClientBLL` (Business Logic Layer) reprezintă o componentă a aplicației care se ocupă cu logica de afaceri a obiectelor de tip `Client`. Aceasta interacționează cu baza de date prin intermediul clasei `ClientDAO` (Data Access Object) și validează datele prin intermediul unei liste de validatori.

Constructorul clasei `ClientBLL` inițializează lista de validatori și obiectul `ClientDAO`.

Metodele acestei clase sunt:

1. `insertClient(Client client)`: Această metodă primește un obiect `Client` ca argument și îl inserează în baza de date. Înainte de inserare, se

verifică dacă există deja un client cu același ID în baza de date. Dacă există, se afișează un mesaj de eroare și se aruncă o excepție. Dacă nu, datele clientului sunt validate și clientul este inserat în baza de date.

2. `deleteClient(int id)`: Această metodă șterge un client din baza de date pe baza ID-ului său. Înainte de ștergere, se verifică dacă există un client cu ID-ul dat în baza de date. Dacă nu, se afișează un mesaj de eroare și se aruncă o excepție. Dacă există, se verifică dacă există comenzi active asociate clientului. Dacă există, se afișează un mesaj de eroare și nu se șterge clientul. Dacă nu, clientul este șters din baza de date.
3. `updateClient(Client client)`: Această metodă actualizează datele unui client în baza de date. Înainte de actualizare, se verifică dacă există un client cu același ID în baza de date. Dacă nu, se afișează un mesaj de eroare și se aruncă o excepție. Dacă există, datele clientului sunt validate și clientul este actualizat în baza de date.
4. `findAllClients()`: Această metodă returnează o listă cu toți clienții din baza de date.
5. `findDeletion(int id)`: Această metodă returnează un client cu un anumit ID din baza de date.

Această clasă permite gestionarea eficientă și sigură a operațiunilor de bază de date asociate cu obiectele de tip `Client`.

Clasa OrderBLL (Business Logic Layer) reprezintă o componentă a aplicației care se ocupă de logica de afaceri a obiectelor Comenzi. Interacționează cu baza de date prin clasa OrderDAO (Obiect de acces de date) și validează datele folosind o listă de validatori.

Constructorul clasei OrderBLL inițializează lista validatorilor și obiectul OrderDAO.

Metodele acestei clase sunt:

1.insertOrder (Comenzi de comenzi): Această metodă ia un obiect de comenzi ca argument și îl introduce în baza de date. Înainte de introducere, verifică dacă există deja o comandă cu același ID în baza de date. Dacă există, este afișat un mesaj de eroare și se aruncă o excepție. Dacă nu există, datele comenzii sunt validate și comanda este introdusă în baza de date.

2.deleteOrder (int id): Această metodă șterge o comandă din baza de date pe baza ID-ului său. Înainte de ștergere, verifică dacă există o comandă cu ID-ul dat în baza de date. Dacă nu există, este afișat un mesaj de eroare și se aruncă o excepție. Dacă există, comanda este ștersă din baza de date.

3.updateOrder (Comenzi de comenzi): Această metodă actualizează datele unei comenzi din baza de date. Înainte de actualizare, verifică dacă există o comandă cu același ID în baza de date. Dacă nu există, este afișat un mesaj de eroare și se aruncă o excepție. Dacă există, datele comenzii sunt validate și comanda este actualizată în baza de date.

4.findAllOrders ( ): Această metodă returnează o listă cu toate comenzile din baza de date.

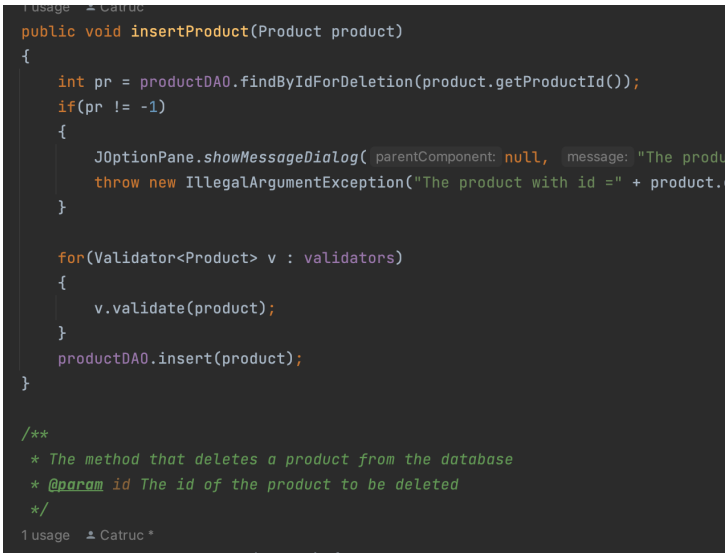
5.findDeletion (int id): Această metodă returnează o comandă cu un ID specific din baza de date.

Clasa ProductBLL (Business Logic Layer) este o componentă a aplicației care gestionează logica de afaceri pentru obiectele Produsului. Interacționează cu baza de date prin clasa ProductDAO (Obiect de acces de date) și validează datele folosind o listă de validatori.

Constructorul clasei ProductBLL inițializează lista validatorilor și obiectul ProductDAO. Exista un BLL si pentru restul obiectelor.

Iată metodele acestei clase:

1.insertProduct (Produs produs): Această metodă ia un obiect Produs ca argument și îl introduce în baza de date. Înainte de introducere, verifică dacă există deja un produs cu același ID în baza de date. Dacă există, este afișat un mesaj de eroare și se aruncă o excepție. Dacă nu există, datele produsului sunt validate și produsul este introdus în baza de date.



```
public void insertProduct(Product product)
{
    int pr = productDAO.findByIdForDeletion(product.getProductId());
    if(pr != -1)
    {
        JOptionPane.showMessageDialog( parentComponent: null,  message: "The produ
        throw new IllegalArgumentException("The product with id =" + product.
    }

    for(Validator<Product> v : validators)
    {
        v.validate(product);
    }
    productDAO.insert(product);
}

/**
 * The method that deletes a product from the database
 * @param id The id of the product to be deleted
 */
1 usage  Catruc *
```

2.deleteProduct (int id): Această metodă șterge un produs din baza de date pe baza de date pe baza ID-ului său. Înainte de ștergere, verifică dacă există un produs cu ID-ul dat în baza de date. Dacă nu există, este afișat un mesaj de eroare și se aruncă o excepție. Dacă există, metoda verifică și dacă există comenzi active asociate cu produsul. Dacă există, produsul nu poate fi șters și este afișat un mesaj de eroare.

3.updateProdus (Produs produs): Această metodă actualizează datele unui produs din baza de date. Înainte de actualizare, verifică dacă există un produs cu același ID în baza de date. Dacă nu există, este afișat un mesaj de eroare și se aruncă o excepție. Dacă există, datele produsului sunt validate și produsul este actualizat în baza de date.

4.findAllProducts ( ): Această metodă returnează o listă cu toate produsele din baza de date.

5.findDeletion (int id): Această metodă returnează un produs cu un ID specific din baza de date.
Interfața Validator este o interfață simplă, generică, care reprezintă o operație de validare pe un obiect de tip T. Definește o singură metodă de validare (T t).

Această interfață vă permite să aplicați anumite condiții sau reguli privind obiectele de orice tip.

Iată cum funcționează:

Un obiect de tip T este trecut la metoda validată (T t).
În cadrul acestei metode, puteți implementa orice verificări de validare necesare pentru cazul dvs. de utilizare specifică.
Dacă obiectul nu reușește oricare dintre aceste verificări, puteți arunca o excepție sau puteți gestiona defecțiunea în orice mod adecvat pentru aplicația dvs.

Făcând Validator o interfață, puteți furniza diferite implementări ale metodei validate (T t) pentru diferite tipuri de obiecte sau reguli de validare diferite. Acest lucru face ca codul dvs. să fie mai flexibil și reutilizabil.

Când această interfață este implementata într-o clasă, veți oferi logica de validare reală care asigură că obiectul trecut îndeplinește anumite criterii.

Interfața `ProductIdValidator` implementează interfața generică `Validator` pentru tipul de obiect `Product`. Aceasta definește o singură metodă, `validate(Product product)`, care verifică dacă ID-ul produsului este un număr pozitiv.

Iată cum funcționează:

7. Un obiect de tip `Product` este transmis metodei `validate(Product product)`.
8. În interiorul acestei metode, verificăm dacă ID-ul produsului este mai mic decât zero.
9. Dacă ID-ul produsului este negativ, se afișează un mesaj de eroare și se aruncă o excepție de tip `IllegalArgumentException`.

Această clasă este utilă pentru a asigura că toate produsele dintr-o aplicație sau bază de date au un ID valid, care este un număr pozitiv. Aceasta este o bună practică pentru a asigura integritatea datelor și pentru a preveni erorile.

Clasa `Controller` este partea de control din modelul MVC (Model-View-Controller). Ea găzduiește toată logica de aplicație și este responsabilă pentru coordonarea modelelor și a vizualizărilor.

Iată o descriere generală a câteva dintre metodele și clasa interioară `ActionListener`:

1. Constructorul `Controller` inițializează toate componentele necesare, inclusiv vizualizarea și diferitele obiecte de tip business logic layer (BLL), precum `ClientBLL`, `ProductBLL`, `OrderBLL` și `BillDAO`. De asemenea, este responsabil pentru inițializarea ascultătorilor de evenimente.
2. Metodele `updateClientTable`, `updateProductTable` și `updateOrderTable` sunt responsabile pentru actualizarea datelor din tabelele corespunzătoare.
3. Metoda `displayBillData` este responsabilă pentru afișarea datelor facturilor în tabelul corespunzător.
4. Metoda `initListeners` inițializează toți ascultătorii de evenimente pentru butoanele de interfață.
5. Clasele interioare care implementează interfața `ActionListener`, cum ar fi `ClientButtonListener`, `ProductButtonListener`, `OrderButtonListener` și alții, sunt ascultători pentru evenimente de acțiune. Acestea sunt declanșate atunci când un utilizator interacționează cu un element de interfață, cum ar fi un buton. Fiecare dintre aceste clase are o metodă `actionPerformed` care definește acțiunea care trebuie realizată atunci când evenimentul este declanșat.

Clasa View: reprezintă stratul de vizualizare al aplicației. Creează și gestionează componentele GUI.

Rame: Instanțele clasei JFrame reprezintă cadrele principale ale aplicației, inclusiv cadrul de bun venit, cadrul clientului, cadrul produsului, cadrul comenzii și cadrul facturii.

Etichete: Instanțele clasei JLabel reprezintă diverse etichete text utilizate în cadre pentru afișarea informațiilor.

Butoane: Instanțele clasei JButton reprezintă butoane clickabile utilizate pentru diferite acțiuni, cum ar fi introducerea, ștergerea, actualizarea sau navigarea între cadre.

Câmpuri de text: Instanțele clasei JTextField reprezintă câmpuri de intrare în care utilizatorii pot introduce informații, cum ar fi detalii despre client, detalii despre produs sau detalii despre comandă.

Tabele: Instanțele clasei JTable reprezintă date tabulare, cum ar fi informații despre clienți, informații despre produs sau informații despre comandă. Aceste tabele sunt afișate folosind un model implicitTableModel.

Codul stabilește aspectul inițial, dimensiunea și aspectul cadrelor și componentelor. De asemenea, definește metodele de a prelua referințe la aceste componente, permițând altor părți ale aplicației să interacționeze cu acestea.

5. Rezultate

Rezultatele au fost cele asteptate din punctul de vedere al inserarilor, stergerilor si update-urilor.

6. Concluzii

Din aceasta tema am invatat sa gestionez mai bine lucrul cu baze de date, sa lucrez mai bine cu reflection si am descoperit tipul record si lucruri interesante despre el.

7. Bibliografie

o http://tutorials.jenkov.com/java-reflection/index.html
https://dzone.com/articles/layers-standard-enterprise
o https://www.baeldung.com/java-jdbc
o http://www.mkyong.com/jdbc/how-to-connect-to-mysql-with-jdbc-driver-java/
o https://www.baeldung.com/javadoc
https://www.youtube.com/watch?v=cbmaA0hpDbY&ab_channel=QVisible