DOCUMENTAȚIE

TEMA3

-Managementul comenzilor-

Nume student: Buda Andreea Rodica

Grupa: 30221

CUPRINS

1. OBIECTIVUL TEMEI

- **Obiectivul principal al temei:**

Implementarea unei aplicații de gestionare a comenzilor pentru un depozit, utilizând o bază de date relațională și arhitectura stratificată.

- **Obiective secundare:**

- Proiectarea și implementarea claselor de model (Capitolul 3-4)

- Proiectarea și implementarea claselor de logica de afaceri (Capitolul 3-4)

- Proiectarea și implementarea claselor de prezentare (Capitolul 3-4)

- Proiectarea și implementarea claselor de acces la date (Capitolul 3-4)

- Crearea și utilizarea bazelor de date relaționale pentru stocarea informațiilor despre clienți, produse și comenzi (Capitolul 4)

- Dezvoltarea funcționalității de adăugare, editare, ștergere și vizualizare a clienților (Capitolul 2)

- Dezvoltarea funcționalității de adăugare, editare, ștergere și vizualizare a produselor (Capitolul 2)

- Dezvoltarea funcționalității de creare a comenzilor de produse (Capitolul 2)

- Implementarea clasei Factura ca o clasă imutabilă utilizând Java records și stocarea acestora într-o tabelă Log (Capitolul 4)

- Utilizarea tehnicilor de reflexie pentru crearea unei clase generice care să conțină metodele pentru accesarea bazei de date (Capitolul 3)

- Documentarea proiectului utilizând Javadoc și generarea fișierelor JavaDoc corespunzătoare (Capitolul 5)

Aceste obiective secundare vor fi detaliate în capitolele corespunzătoare ale documentației proiectului.

2. ANALIZA PROBLEMEI, MODELARE, SCENARII, CAZURI DE UTILIZARE

a. Analiza problemei:

a) Cerințe funcționale:

1. Adăugarea unui nou client:

- Utilizatorul introduce datele noului client (nume, adresa, etc.).

- Aplicația validează și stochează datele în baza de date.

2. Adăugarea unui nou produs:

- Utilizatorul introduce datele noului produs (denumire, preț, stoc, etc.).

- Aplicația validează și stochează datele în baza de date.

3. Adăugarea unei comenzi:

- Utilizatorul selectează un client existent și un produs existent.

- Utilizatorul introduce cantitatea dorită pentru produs.

- Aplicația verifică disponibilitatea stocului și afișează un mesaj în cazul în care cantitatea solicitată nu este disponibilă.

- În caz contrar, aplicația înregistrează comanda, actualizează stocul și generează o factură.

4. Căutarea unui client după diferite criterii:

- Utilizatorul introduce criteriile de căutare (nume, adresă, etc.).

- Aplicația caută în baza de date și afișează rezultatele.

5. Căutarea unui produs după diferite criterii:

- Utilizatorul introduce criteriile de căutare (denumire, preț, etc.).

- Aplicația caută în baza de date și afișează rezultatele.

6. Modificarea unui client în funcție de un criteriu:

- Utilizatorul selectează clientul pe care dorește să-l modifice.

- Utilizatorul actualizează datele clientului.

- Aplicația validează și actualizează datele în baza de date.

7. Modificarea unui produs în funcție de un criteriu:

- Utilizatorul selectează produsul pe care dorește să-l modifice.

- Utilizatorul actualizează datele produsului.

- Aplicația validează și actualizează datele în baza de date.

8. Ștergerea unui client în funcție de un criteriu:

- Utilizatorul selectează clientul pe care dorește să-l șteargă.

- Aplicația șterge clientul din baza de date.

9. Ștergerea unui produs în funcție de un criteriu:

- Utilizatorul selectează produsul pe care dorește să-l șteargă.

- Aplicația șterge produsul din baza de date.

10. Afișarea unui tabel cu clienți:

- Aplicația extrage informațiile despre clienți din baza de date și le afișează într-un tabel.

11. Afișarea unui tabel cu produse:

-Aplicația extrage informațiile despre produse din baza de date și le afișează într-un tabel.

12. Afișarea unui tabel cu comenzi:

- Aplicația extrage informațiile despre comenzi din baza de date și le afișează într-un tabel.

b) Cerințe non-funcționale:

- Intuitivitate și ușurință în utilizare: Aplicația trebuie să fie ușor de înțeles și de utilizat de către utilizatori, având o interfață grafică intuitivă și eficientă.

b. Modelare:

Diagrama de pachete:

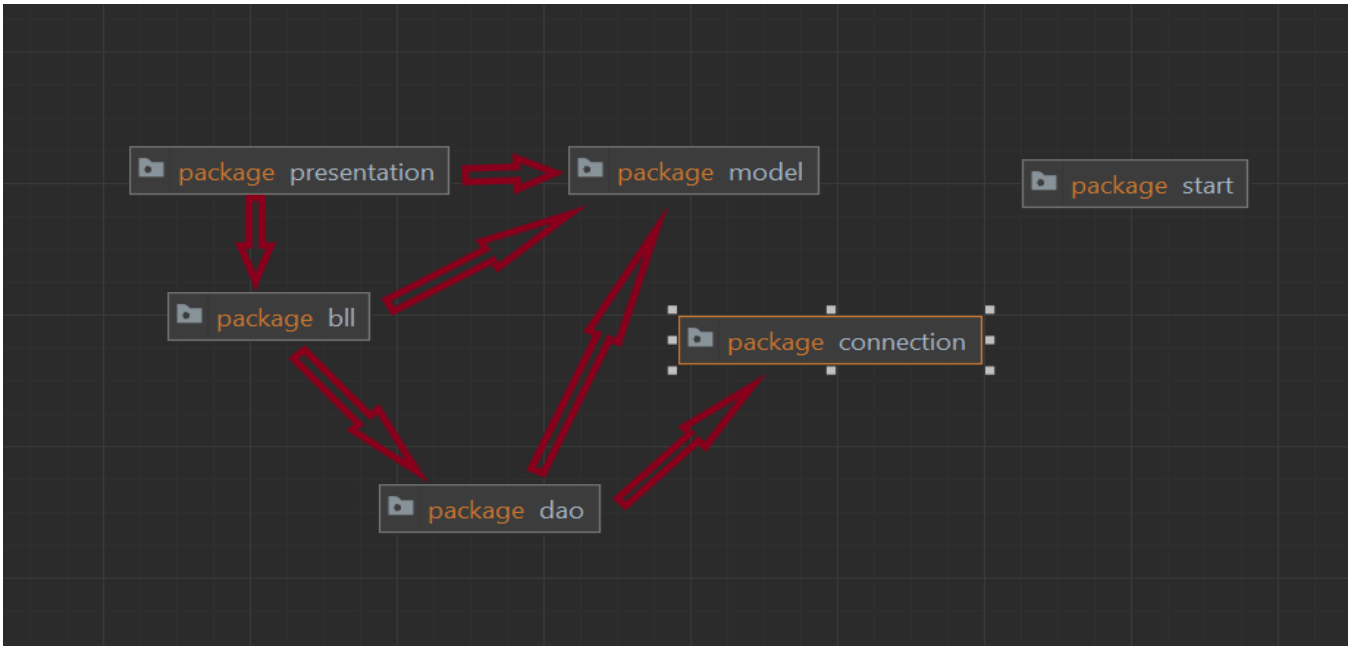
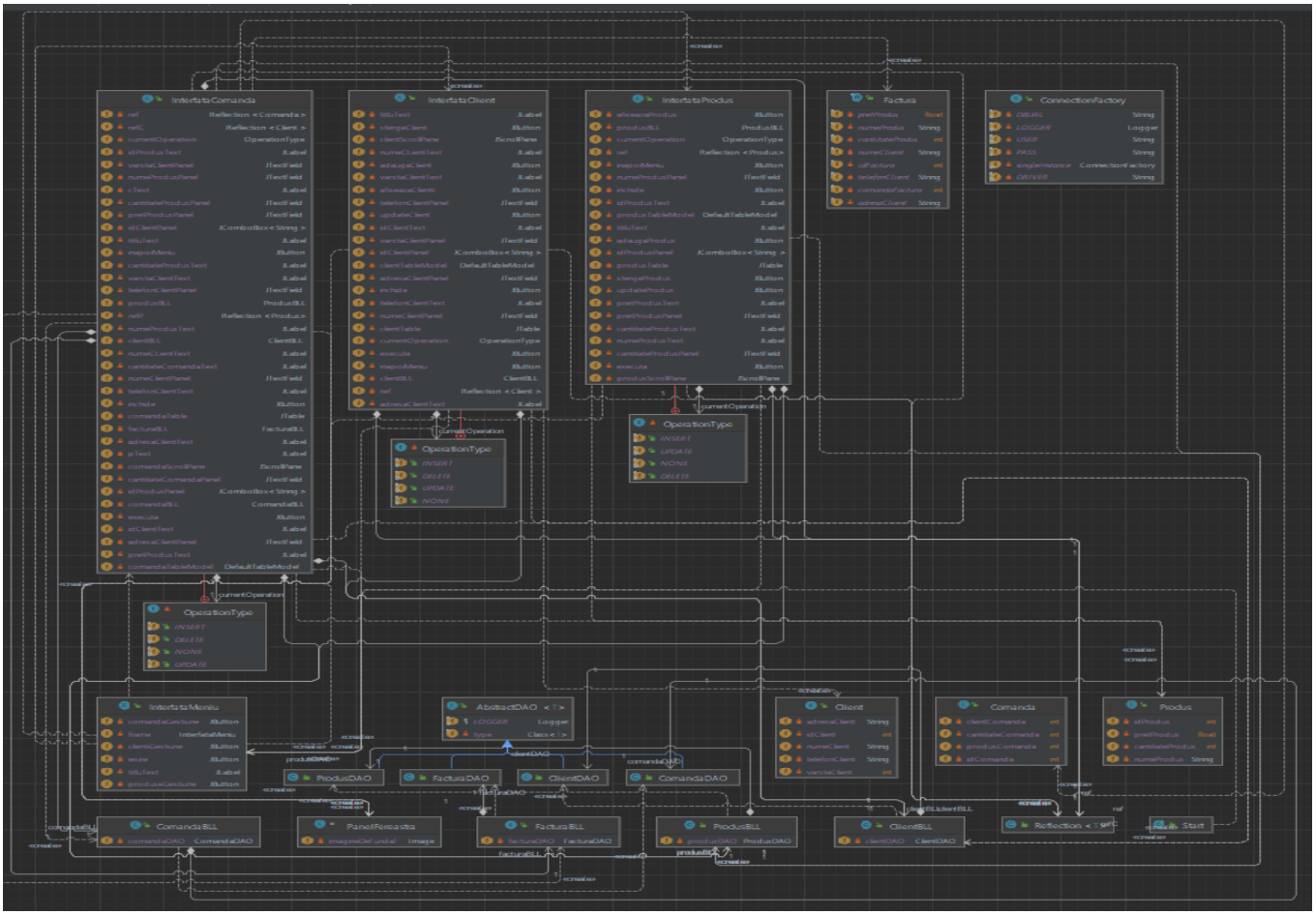


Diagrama de clase:



Unified Modeling Language (UML) este un limbaj standard de modelare utilizat în ingineria software pentru a reprezenta grafic și comunica modele conceptuale ale sistemelor software. UML oferă un set de diagrame grafice standardizate pentru a descrie diferite aspecte ale sistemelor software, inclusiv diagrame de clase, diagrame de secvențe, diagrame de activități și diagrame de stări. Aceste diagrame sunt concepute pentru a fi ușor de înțeles atât de către dezvoltatorii software cât și de către persoanele non-tehnice.

UML este foarte util pentru a detecta probleme și a identifica riscuri în timpul fazei de proiectare a sistemului. Prin vizualizarea modelelor UML, putem identifica posibile probleme în design sau interacțiuni defectuoase între componentele sistemului. Aceste probleme pot fi apoi corectate înainte ca sistemul să fie construit și implementat, reducând astfel costurile și timpul de dezvoltare.

c. Cazuri de utilizare și scenarii:

Cazuri de utilizare:

1. Utilizatorul adaugă un client nou:

- Actorul principal: Utilizator
- Date de intrare: Detaliile clientului (nume, adresa, număr de telefon, etc.)
- Date de ieșire: Mesaj de confirmare că clientul a fost adăugat cu succes sau mesaj de eroare în caz contrar.

2. Utilizatorul editează un client existent:

- Actorul principal: Utilizator
- Date de intrare: ID-ul clientului și noile detalii ale clientului
- Date de ieșire: Mesaj de confirmare că modificările au fost salvate cu succes sau mesaj de eroare în caz contrar.

3. Utilizatorul șterge un client existent:

- Actorul principal: Utilizator
- Date de intrare: ID-ul clientului
- Date de ieșire: Mesaj de confirmare că clientul a fost șters cu succes sau mesaj de eroare în caz contrar.

4. Utilizatorul vizualizează toți clienții:

- Actorul principal: Utilizator
- Date de intrare: -
- Date de ieșire: Tabel cu detalii despre toți clienții existenți.

5. Utilizatorul adaugă un produs nou:

- Actorul principal: Utilizator
- Date de intrare: Detaliile produsului (nume, preț, cantitate, etc.)
- Date de ieșire: Mesaj de confirmare că produsul a fost adăugat cu succes sau mesaj de eroare în caz contrar.

6. Utilizatorul editează un produs existent:

- Actorul principal: Utilizator
- Date de intrare: ID-ul produsului și noile detalii ale produsului
- Date de ieșire: Mesaj de confirmare că modificările au fost salvate cu succes sau mesaj de eroare în caz contrar.

7. Utilizatorul șterge un produs existent:

- Actorul principal: Utilizator
- Date de intrare: ID-ul produsului
- Date de ieșire: Mesaj de confirmare că produsul a fost șters cu succes sau mesaj de eroare în caz contrar.

8. Utilizatorul vizualizează toate produsele:

- Actorul principal: Utilizator
- Date de intrare: -
- Date de ieșire: Tabel cu detalii despre toate produsele existente.

9. Utilizatorul creează o comandă nouă:

- Actorul principal: Utilizator
- Date de intrare: Clientul și produsul selectate, cantitatea dorită
- Date de ieșire: Mesaj de confirmare că comanda a fost plasată cu succes și stocul a fost actualizat sau mesaj de eroare în caz contrar.

Scenariul principal de succes:

1. Utilizatorul deschide aplicația de management al comenzilor.

2. Utilizatorul adaugă un client nou furnizând detaliile necesare.

3. Utilizatorul adaugă un produs nou furnizând detaliile necesare.

4. Utilizatorul creează o comandă nouă selectând un client, un produs și specificând cantitatea dorită.

5. Sistemul verifică disponibilitatea produsului în stoc.

6. Dacă cantitatea dorită este disponibilă, comanda este plasată și stocul este actualizat.

7. Utilizatorul primește un mesaj de confirmare că comanda a fost plasată cu succes.

8. Utilizatorul poate continua să adauge, editeze sau șteargă clienți și produse, precum și să plaseze alte comenzi.

Scenariu alternativ:

1. Utilizatorul creează o comandă nouă selectând un client, un produs și specificând cantitatea dorită.

2. Sistemul verifică disponibilitatea produsului în stoc.

3. Dacă cantitatea dorită nu este disponibilă, se afișează un mesaj de eroare.

4. Utilizatorul poate alege să modifice cantitatea sau să selecteze un alt produs în funcție de disponibilitatea stocului.

3. PROIECTARE

Proiectarea este una dintre cele mai importante etape în dezvoltarea oricărui proiect software, inclusiv în dezvoltarea aplicației Java descrisă în tema dată. Aceasta implică stabilirea structurii generale a aplicației, a modului în care componentele sale interacționează între ele și a modului în care datele sunt stocate și procesate.

Am implementat această aplicație cu ajutorul a două zeci și una de clase și am folosit următoarea structură de date: List< > - liste in care se stochează clienții, produsele și comenzile.

Clasele sunt organizate în următoarele pachete:

- bll - conține clasele ClientBLL, FacturaBLL, ComandaBLL și ProdusBLL.
- connection - conține clasa ConnectionFactory.
- dao - conține clasele AbstractDAO, ClientDAO, FacturaDAO, ComandaDAO și ProdusDAO.
- model - conține clasele Client, Factura, Comanda și Produs.
- presentation - conține clasele InterfataClient, InterfataMeniu, InterfataComanda, InterfataProdus și PanelFereastra.
- start- conține clasele Start și Reflection.

Descrierea claselor:

- **Clasa “ClientBLL”:**

Clasa "ClientBLL" este responsabilă de gestionarea operațiunilor legate de clienți în cadrul aplicației de management al comenzilor. Aceasta realizează interacțiunea între nivelul de prezentare (GUI) și nivelul de acces la date (DAO), asigurând logica de business necesară pentru operațiuni precum căutarea, inserarea, actualizarea și ștergerea clienților din baza de date.

Prin intermediul metodelor sale, clasa "ClientBLL" permite:

- Găsirea unui client după ID-ul acestuia.

- Obținerea unei liste cu toți clienții existenți în baza de date.

- Inserarea unui client nou în baza de date.

- Actualizarea informațiilor unui client existent în baza de date.

- Ștergerea unui client din baza de date.

Clasa "ClientBLL" utilizează obiectul "ClientDAO" pentru a accesa și manipula datele din baza de date. Aceasta gestionează cazurile de excepție și asigură că operațiunile de CRUD (create, read, update, delete) asupra clienților sunt efectuate în mod corespunzător.

Prin respectarea principiilor designului orientat pe obiecte și a convențiilor de denumire din Java, clasa "ClientBLL" îndeplinește cerințele de implementare și logica funcțională asociate operațiunilor legate de clienți în aplicația de management al comenzilor.

- **Clasa “FacturaBLL”:**

Clasa "FacturaBLL" reprezintă componenta de logică a aplicației pentru gestionarea operațiunilor legate de facturi. Aceasta este responsabilă de interacțiunea între nivelul de prezentare (GUI) și nivelul de acces la date (DAO) pentru a permite inserarea unei noi facturi în baza de date.

Clasa "FacturaBLL" utilizează obiectul "FacturaDAO" pentru a accesa și manipula datele facturilor în baza de date. Metoda "insertFactura" permite inserarea unei facturi noi în baza de date și returnează factura inserată. În cazul în care inserarea nu poate fi realizată cu succes, se aruncă o excepție de tip "NoSuchElementException".

Prin intermediul clasei "FacturaBLL", se respectă principiile designului orientat pe obiecte și se asigură implementarea și funcționarea corespunzătoare a operațiunilor de inserare a facturilor în cadrul aplicației de gestionare a comenzilor.

- **Clasa “ComandaBLL”:**

Clasa "ComandaBLL" reprezintă componenta de logică a aplicației pentru gestionarea operațiunilor legate de comenzile clienților. Aceasta este responsabilă de interacțiunea între nivelul de prezentare (GUI) și nivelul de acces la date (DAO) pentru a permite găsirea comenzilor, inserarea unei noi comenzi și obținerea id-ului maxim al comenzii.

Clasa "ComandaBLL" utilizează obiectul "ComandaDAO" pentru a accesa și manipula datele comenzilor în baza de date. Metodele precum "findComandaById", "findAllOrders" și "insertComanda" permit găsirea comenzilor după ID, returnarea listei tuturor comenzilor și inserarea unei noi comenzi în baza de date. În cazul în care o operațiune nu poate fi realizată cu succes, se aruncă o excepție de tip "NoSuchElementException".

De asemenea, clasa "ComandaBLL" oferă și o metodă "maxIdComanda" care returnează id-ul maxim al comenzii, utilizat pentru generarea facturii.

Prin intermediul clasei "ComandaBLL", se respectă principiile designului orientat pe obiecte și se asigură implementarea și funcționarea corespunzătoare a operațiunilor de gestionare a comenzilor în cadrul aplicației de management al comenzilor pentru un depozit.

- **Clasa “ProdusBLL”:**

Clasa "ProdusBLL" este responsabilă de gestionarea operațiunilor legate de produse în cadrul aplicației. Aceasta servește ca o punte de legătură între nivelul de prezentare (interfața utilizatorului) și nivelul de acces la date (accesul la baza de date).

Prin intermediul clasei "ProdusBLL", se pot efectua diverse operațiuni cu produsele, cum ar fi găsirea unui produs după ID, returnarea unei liste de toate produsele, inserarea unui nou produs, actualizarea unui produs existent și ștergerea unui produs din baza de date.

Această clasă utilizează obiectul "ProdusDAO" pentru a accesa și manipula datele specifice produselor în baza de date. Metodele din clasa "ProdusBLL" sunt concepute pentru a apela funcționalitățile corespunzătoare din obiectul "ProdusDAO" și pentru a gestiona eventualele excepții care pot apărea în timpul operațiunilor.

Prin respectarea principiilor designului orientat pe obiecte și prin intermediul clasei "ProdusBLL", se asigură o separare eficientă a responsabilităților în cadrul aplicației și se permite o gestionare mai ușoară și modulară a operațiunilor legate de produse.

- **Clasa “ConnectionFactory”:**

Clasa "ConnectionFactory" reprezintă o componentă utilitară care facilitează crearea și gestionarea conexiunilor la baza de date. Aceasta oferă metode pentru crearea și închiderea conexiunilor, a obiectelor de instrucțiuni (Statement) și a obiectelor ResultSet.

Clasa utilizează driverul JDBC pentru conectarea la o bază de date specificată prin URL-ul de conexiune, numele utilizatorului și parola. Prin intermediul metodei "getConnection", este returnată o conexiune la baza de date curentă, iar apelurile metodelor "close" permit închiderea conexiunilor, a obiectelor Statement și a obiectelor ResultSet, asigurând astfel o eliberare corespunzătoare a resurselor și evitarea eventualelor erori sau scurgeri de memorie. Clasa "ConnectionFactory" implementează un design Singleton, ceea ce înseamnă că există o singură instanță a acestei clase în întreaga aplicație, garantând astfel o utilizare eficientă a resurselor de conexiune.

În general, clasa "ConnectionFactory" asigură crearea și gestionarea corectă a conexiunilor la baza de date, respectând principiile programării robuste și evitând potențialele erori asociate manipulării directe a conexiunilor în cadrul codului aplicației.

- **Clasa “AbstractDAO”:**

Clasa `AbstractDAO` este o implementare generică a unui obiect DAO (Data Access Object), care furnizează operații de bază CRUD (Create, Read, Update, Delete) pentru lucrul cu o bază de date. Această clasă servește drept bază pentru alte clase DAO specializate și oferă funcționalitatea comună necesară pentru a interacționa cu baza de date.

Clasa `AbstractDAO` are un parametru de tip `<T>` care reprezintă tipul obiectelor gestionate de către DAO. Această clasă folosește reflexia pentru a crea și executa interogări SQL generice, care sunt personalizate în funcție de tipul specificat.

Această clasă conține metode pentru a realiza operațiile de bază pe obiecte, cum ar fi: `findAll()` pentru a returna o listă cu toate obiectele din tabelul corespunzător, `findById()` pentru a găsi un obiect pe baza unui ID și a numelui coloanei, `insert()` pentru a insera un nou obiect în tabel, `update()` pentru a actualiza un obiect existent în tabel, și `delete()` pentru a șterge un obiect din tabel.

Clasa `AbstractDAO` utilizează conexiunea la baza de date prin intermediul clasei `ConnectionFactory`, care gestionează crearea și închiderea conexiunilor la baza de date. Această clasă abstractă oferă o abordare generică pentru manipularea datelor într-o bază de date și poate fi extinsă de alte clase DAO pentru a adăuga funcționalități specifice.

- **Clasa "ClientDAO":**

Clasa `ClientDAO` este o clasă specializată care se ocupă de accesul la date pentru entitatea `Client`. Această clasă extinde clasa `AbstractDAO` și implementează operațiile specifice pentru gestionarea datelor asociate clienților într-o bază de date.

Prin moștenirea clasei `AbstractDAO`, clasa `ClientDAO` beneficiază de funcționalitățile generice oferite de aceasta, precum operațiile CRUD (Create, Read, Update, Delete), care sunt adaptate pentru lucrul cu obiecte de tip `Client`. Astfel, prin intermediul clasei `ClientDAO`, se pot realiza operațiuni de citire, inserare, actualizare și ștergere a datelor despre clienți în baza de date.

Clasa `ClientDAO` abstractizează și ascunde detalii specifice legate de manipularea datelor în baza de date, permițând utilizatorului să se concentreze pe operațiile de gestionare a clienților în loc să se preocupe de detalii tehnice legate de interacțiunea cu baza de date.

Prin implementarea clasei `ClientDAO`, se promovează separarea responsabilităților în aplicație, urmând principiul de proiectare a arhitecturii software numit "Separation of Concerns" (Separarea preocupărilor). Astfel, funcționalitatea specifică clienților este encapsulată în această clasă, facilitând reutilizarea și menținerea codului.

- **Clasa "ComandaDAO":**

Clasa `ComandaDAO` este o clasă specializată care se ocupă de accesul la date pentru entitatea `Comanda`. Această clasă extinde clasa `AbstractDAO` și implementează operațiile specifice pentru gestionarea datelor asociate comenzilor într-o bază de date.

Prin moștenirea clasei `AbstractDAO`, clasa `ComandaDAO` beneficiază de funcționalitățile generice oferite de aceasta, precum operațiile CRUD (Create, Read, Update, Delete), care sunt adaptate pentru lucrul cu obiecte de tip `Comanda`. Astfel, prin intermediul clasei `ComandaDAO`, se pot realiza operațiuni de citire, inserare, actualizare și ștergere a datelor despre comenzile în baza de date.

Clasa `ComandaDAO` abstractizează și ascunde detalii specifice legate de manipularea datelor în baza de date, permițând utilizatorului să interacționeze cu entitatea `Comanda` fără a fi nevoie să cunoască detalii despre implementarea și manipularea efectivă a datelor în baza de date. Aceasta favorizează o separare clară între logica de acces la date și logica aplicației.

Prin intermediul clasei `ComandaDAO`, se poate accesa și manipula informațiile legate de comenzile din baza de date, cum ar fi obținerea tuturor comenzilor, găsirea unei comenzi după un ID specific sau actualizarea unei comenzi existente.

Clasa `ComandaDAO` oferă un nivel de abstractizare și reutilizare a codului, deoarece implementează operațiile generice definite în clasa `AbstractDAO`, permițând dezvoltatorilor să se concentreze pe logica specifică comenzilor, fără a fi nevoie să rescrie în mod repetitiv aceleași operații de bază pentru fiecare entitate.

În concluzie, clasa `ComandaDAO` reprezintă un instrument esențial pentru manipularea datelor asociate comenzilor într-o bază de date, oferind metode și funcționalități specifice acestei entități, într-un mod abstractizat și reutilizabil.

- **Clasa “FacturaDAO”:**

Clasa `FacturaDAO` este responsabilă de gestionarea accesului la date pentru entitatea `Factura`. Aceasta extinde clasa abstractă `AbstractDAO` și implementează operațiile specifice necesare pentru lucrul cu entitatea `Factura` în baza de date.

Prin intermediul clasei `FacturaDAO`, se pot efectua operații precum obținerea tuturor facturilor. Această clasă oferă o abstractizare a operațiilor de bază pentru manipularea datelor legate de facturi și asigură separarea logicii de acces la date de logica aplicației.

Utilizarea clasei `FacturaDAO` permite dezvoltatorilor să interacționeze cu entitatea `Factura` într-un mod simplu și eficient, fără a fi nevoie să se implice direct în detalii specifice bazei de date. Aceasta promovează modularitatea și reutilizabilitatea codului, deoarece operațiile generice sunt definite în clasa `AbstractDAO`, iar clasa `FacturaDAO` se concentrează pe operațiile specifice entității `Factura`.

În rezumat, clasa `FacturaDAO` reprezintă un instrument esențial pentru manipularea datelor asociate facturilor într-o bază de date, oferind funcționalități specifice și abstractizate, pentru o utilizare ușoară și eficientă.

- **Clasa “ProdusDAO”:**

Clasa `ProdusDAO` este responsabilă de gestionarea accesului la date pentru entitatea `Produs`. Aceasta extinde clasa abstractă `AbstractDAO` și implementează operațiile specifice necesare pentru lucrul cu entitatea `Produs` în baza de date.

Prin intermediul clasei `ProdusDAO`, se pot efectua operații precum obținerea tuturor produselor, găsirea unui produs după un ID specific, actualizarea sau ștergerea unui produs existent. Această clasă oferă o abstractizare a operațiilor de bază pentru manipularea datelor legate de produse și asigură separarea logicii de acces la date de logica aplicației.

Utilizarea clasei `ProdusDAO` permite dezvoltatorilor să interacționeze cu entitatea `Produs` într-un mod simplu și eficient, fără a fi nevoie să se implice direct în detalii specifice bazei de date. Aceasta promovează modularitatea și reutilizabilitatea codului, deoarece operațiile generice sunt definite în clasa `AbstractDAO`, iar clasa `ProdusDAO` se concentrează pe operațiile specifice entității `Produs`.

În rezumat, clasa `ProdusDAO` reprezintă un instrument esențial pentru manipularea datelor asociate produselor într-o bază de date, oferind funcționalități specifice și abstractizate, pentru o utilizare ușoară și eficientă.

- **Clasa “Client”:**

Clasa `Client` este o clasă model care reprezintă un client. Aceasta conține câmpuri pentru a stoca informații despre un client, cum ar fi ID-ul clientului, numele, vârsta, adresa și numărul de telefon.

Clasa `Client` dispune de mai mulți constructori, care permit crearea unui obiect `Client` cu diferite combinații de parametri. Constructorii cu parametri permit inițializarea obiectului `Client` cu valorile specifice furnizate, cum ar fi ID-ul, numele, vârsta, adresa și numărul de telefon. Constructorul fără parametri permite crearea unui obiect `Client` cu câmpurile neinițializate.

Clasa `Client` conține metode de acces (gettere și settere) pentru a accesa și modifica valorile câmpurilor. Aceste metode permit obținerea valorilor câmpurilor individuale ale unui obiect `Client`, precum și actualizarea acestora.

În rezumat, clasa `Client` reprezintă modelul unui client, oferind câmpuri pentru stocarea informațiilor și metode pentru a accesa și modifica aceste informații. Este utilizată în cadrul aplicației pentru a reprezenta și manipula datele despre clienți.

- **Clasa “Comanda”:**

Clasa `Comanda` este o clasă model care reprezintă o comandă. Aceasta conține câmpuri pentru a stoca informații despre o comandă, cum ar fi ID-ul comenzii, ID-ul clientului asociat comenzii, ID-ul produsului asociat comenzii și cantitatea comenzii.

Clasa `Comanda` dispune de mai mulți constructori, care permit crearea unui obiect `Comanda` cu diferite combinații de parametri. Constructorii cu parametri permit inițializarea obiectului `Comanda` cu valorile specifice furnizate, cum ar fi ID-ul comenzii, ID-ul clientului, ID-ul produsului și cantitatea comenzii. Constructorul fără parametri permite crearea unui obiect `Comanda` cu câmpurile neinițializate.

Clasa `Comanda` conține metode de acces (getteri și setteri) pentru a accesa și modifica valorile câmpurilor. Aceste metode permit obținerea valorilor câmpurilor individuale ale unui obiect `Comanda`, precum și actualizarea acestora.

În rezumat, clasa `Comanda` reprezintă modelul unei comenzi, oferind câmpuri pentru stocarea informațiilor și metode pentru a accesa și modifica aceste informații. Este utilizată în cadrul aplicației pentru a reprezenta și manipula datele despre comenzile efectuate.

- **Clasa "Factura":**

Clasa `Factura` este o clasă model specifică limbajului de programare Java, denumită "record". Aceasta reprezintă o înregistrare care stochează informații despre o factură.

Clasa `Factura` definește automat o serie de funcționalități, cum ar fi definirea și inițializarea câmpurilor, generarea automată a getterilor pentru toate câmpurile și suprascrierea metodei `toString()` pentru a afișa informațiile despre factură.

Clasa `Factura` conține câmpuri pentru a stoca informații despre o factură, cum ar fi ID-ul facturii, ID-ul comenzii asociate facturii, numele clientului, numărul de telefon al clientului, adresa clientului, numele produsului, prețul produsului și cantitatea produsului.

Câmpurile sunt definite în lista de parametri a constructorului implicit al înregistrării `Factura`. Acestea pot fi inițializate în momentul creării obiectului `Factura` utilizând valorile furnizate în aceeași ordine ca și parametrii constructorului.

Clasa `Factura` nu definește alte metode în afara celor deja menționate, deoarece acestea sunt generate automat de către compilatorul Java într-un mod convenabil pentru gestionarea înregistrărilor.

În rezumat, clasa `Factura` este o înregistrare care stochează informații despre o factură. Aceasta folosește funcționalitățile specifice înregistrărilor din Java pentru a defini câmpurile și metodele necesare pentru a accesa și afișa aceste informații. Este utilizată în cadrul aplicației pentru a reprezenta și manipula datele despre facturile emise.

- **Clasa "Produs":**

Clasa `Produs` este o clasă model care reprezintă un produs în cadrul unei aplicații. Aceasta conține informații specifice despre un produs, cum ar fi ID-ul produsului, numele produsului, prețul produsului și cantitatea disponibilă a acestuia.

Clasa `Produs` definește un constructor cu parametri care permite inițializarea obiectelor `Produs` cu valorile corespunzătoare pentru ID, nume, preț și cantitate. Există și un constructor suplimentar care permite inițializarea obiectelor `Produs` fără a furniza ID-ul produsului, acesta putând fi setat ulterior prin intermediul metodei `setIdProdus()`. Clasa mai conține și un constructor fără parametri, care poate fi utilizat pentru crearea unui obiect `Produs` fără inițializarea inițială a câmpurilor.

Clasa `Produs` oferă metode de acces (gettere și settere) pentru toate câmpurile sale, permițând astfel obținerea și modificarea valorilor acestora.

În rezumat, clasa `Produs` reprezintă un produs în cadrul aplicației, stocând informații precum ID-ul, numele, prețul și cantitatea produsului. Aceasta oferă constructori pentru inițializarea obiectelor `Produs` cu sau fără furnizarea ID-ului produsului, precum și metode de acces și modificare pentru toate câmpurile.

- **Clasa "InterfataClient":**

Această clasă, denumită "InterfataClient", este o fereastră de interfață grafică (GUI) utilizată pentru gestionarea clienților. Este o clasă derivată din clasa "JFrame" și conține o serie de componente și acțiuni asociate acestora.

Componentele prezente în această clasă includ:

- Etichete (JLabel) pentru afișarea textului asociat fiecărui câmp al unui client (id, nume, telefon, adresa, varsta);

- Câmpuri de text (JTextField) și o casetă combinată (JComboBox) pentru introducerea și afișarea valorilor câmpurilor unui client;

- Butoane (JButton) pentru diverse acțiuni, precum inserarea, ștergerea, actualizarea și afișarea tuturor clienților;

- Un tabel (JTable) și un model de tabel (DefaultTableModel) pentru afișarea datelor despre clienți;

- Un panou de derulare (JScrollPane) pentru gestionarea afișării tabelului.

Clasa "InterfataClient" conține și o serie de metode pentru configurarea și gestionarea interfeței, precum și pentru tratarea acțiunilor utilizatorului. Aceste metode includ:

- "aspectInterfata": setează aspectul interfeței utilizator, configurând dimensiunile, pozițiile și fonturile componentelor;

- "adaugaComponente": adaugă toate componentele în fereastra interfeței;

- "ascundeElemente": ascunde elementele care nu sunt necesare în funcție de operația curentă selectată;

- "afiseazaPtInsert", "afiseazaPtUpdate", "afiseazaPtDelete": afișează elementele necesare pentru inserarea, actualizarea și ștergerea unui client;

- "InterfataClient": constructorul clasei, inițializează fereastra interfeței, adaugă componentele și setează acțiunile pentru butoane și alte elemente.

Această clasă face parte dintr-un pachet numit "presentation" și folosește alte clase din pachetele "bll" și "model" pentru gestionarea clienților și a datelor acestora. De asemenea, utilizează o clasă numită "Reflection" din pachetul "start" pentru a realiza anumite operații cu reflexie.

- **Clasa “InterfataComanda”:**

Clasa `InterfataComanda` reprezintă o interfață grafică pentru gestionarea comenzilor. Această clasă extinde clasa `JFrame` din librăria `javax.swing` și conține diverse componente și metode pentru manipularea datelor și interacțiunea cu utilizatorul.

Principalele componente ale clasei includ etichete (`JLabel`), câmpuri de text (`JTextField`), liste derulante (`JComboBox`), butoane (`JButton`), o tabelă (`JTable`) și un panou de derulare (`JScrollPane`). Aceste componente sunt utilizate pentru a afișa și manipula informațiile referitoare la clienți, produse, și comenzile efectuate.

Clasa `InterfataComanda` conține și câteva obiecte de tipul claselor BLL (`ClientBLL`, `ProdusBLL`, `ComandaBLL`, `FacturaBLL`) și de tipul claselor `Reflection`. Aceste obiecte sunt utilizate pentru a accesa și manipula datele din baza de date.

Metoda `aspectInterfata()` este responsabilă de configurarea aspectului interfeței grafice, setând poziționarea și fonturile componentelor.

Metoda `adaugaComponente()` adaugă componentele în interfața grafică.

Constructorul clasei `InterfataComanda` inițializează fereastra, configurează aspectul interfeței grafice, adaugă componentele, și setează acțiunile pentru butoane și evenimentele de selectare din liste derulante. De asemenea, se realizează și inițializarea datelor din tabele și se gestionează acțiunile utilizatorului.

Această clasă este folosită pentru crearea și afișarea unei ferestre de interfață grafică cu scopul de a gestiona comenzile, inclusiv adăugarea, actualizarea și ștergerea acestora, afișarea informațiilor despre clienți și produse, precum și generarea facturilor corespunzătoare comenzilor.

- **Clasa “InterfataMeniu”:**

Clasa "InterfataMeniu" reprezintă o interfață grafică pentru gestionarea comenzilor într-o aplicație de management. Aceasta este o clasă care extinde clasa "JFrame" din biblioteca Swing și conține componentele grafice necesare pentru interfața utilizator.

Principalele componente ale clasei includ:

- Eticheta "titluText": reprezintă un text vizibil în interfață și indică titlul ferestrei.

- Butonul "clientiGestiune": este un buton pentru gestionarea clienților.

- Butonul "produseGestiune": este un buton pentru gestionarea produselor.

- Butonul "comandaGestiune": este un buton pentru crearea comenzilor.

- Butonul "iesire": este un buton pentru a închide aplicația.

Clasa conține, de asemenea, metode pentru configurarea aspectului interfeței grafice și adăugarea componentelor în interfață.

Pe lângă componentele menționate, clasa conține și un constructor care primește un titlu pentru fereastra și inițializează frame-ul. Constructorul setează opțiuni precum închiderea aplicației la închiderea ferestrei, setarea ferestrei ca neredimensionabilă și afișarea ferestrei în centru.

De asemenea, clasa conține și metode pentru gestionarea acțiunilor butoanelor. Atunci când se apasă butonul "iesire", aplicația se închide. Pentru celelalte butoane, se creează și se afișează alte interfețe grafice specifice pentru gestionarea clienților, produselor sau a comenzilor.

Aceasta este o clasă centrală în aplicația de management, care permite utilizatorului să acceseze și să gestioneze diverse aspecte ale comenzilor prin interacțiunea cu componentele grafice afișate.

- **Clasa “InterfataProdus”:**

Clasa `InterfataProdus` reprezintă o interfață grafică pentru gestionarea produselor într-o aplicație. Această clasă extinde clasa `JFrame` și implementează funcționalități specifice pentru adăugarea, actualizarea, ștergerea și afișarea produselor.

Caracteristicile clasei:

- Clasa `InterfataProdus` conține o instanță a clasei `ProdusBLL`, care este responsabilă pentru gestionarea operațiilor cu produse în baza de date.

- Clasa folosește o instanță a clasei `Reflection` pentru a realiza anumite operații de introspecție.

- Clasa conține mai multe componente grafice, cum ar fi etichete (`JLabel`), câmpuri de text (`JTextField`), butoane (`JButton`), o casetă de selecție (`JComboBox`) și o tabelă (`JTable`), pentru a permite utilizatorului să interacționeze cu aplicația.

- Metoda `aspectInterfata()` este responsabilă de setarea aspectului interfeței utilizator și poziționarea componentelor în fereastră.

- Metoda `adaugaComponente()` adaugă componentele în fereastra de gestionare a produselor.

- Clasa conține și mai multe metode pentru gestionarea afișării și ascunderii elementelor în funcție de operația curentă.

- În constructorul clasei, se initializează fereastra de interfață utilizator pentru gestionarea produselor. Sunt adăugate componente, setate proprietăți pentru fereastră și este afișată fereastra pe ecran.

- Clasa conține și mai mulți ascultători de evenimente pentru butoanele și caseta de selecție, care sunt responsabili de execuția operațiilor specifice și de actualizarea interfeței grafice în consecință.

Deoarece nu ai solicitat descrierea metodelor individuale, aceasta este o prezentare generală a clasei `InterfataProdus` și a funcționalităților sale.

- **Clasa “PanelFereastra”:**

Clasa `PanelFereastra` face parte din pachetul `presentation` și extinde clasa `JPanel`. Aceasta reprezintă o componentă grafică utilizată în interfața utilizatorului și are rolul de a afișa o imagine de fundal pe o fereastră sau panou.

Clasa are un membru privat `imagineDeFundal` de tip `Image`, care stochează imaginea de fundal ce va fi afișată pe componentă.

Constructorul clasei `PanelFereastra` initializează imaginea de fundal folosind o cale către fișierul "Purple.jpg". Imaginea este încărcată într-un obiect `ImageIcon`, iar apoi este extrată și atribuită membrului `imagineDeFundal`.

Metoda `paintComponent(Graphics g)` este suprascrisă din clasa `JPanel` și are rolul de a desena imaginea de fundal pe componenta curentă. Este utilizat obiectul `Graphics` pentru a realiza desenarea, iar imaginea de fundal este redimensionată pentru a se potrivi cu dimensiunile componentei curente utilizând metoda `drawImage()` a obiectului `Graphics`.

- **Clasa "Reflection":**

Clasa `Reflection` face parte din pachetul `start` și este o clasă generică, parametrizată cu tipul `T`. Această clasă are metode pentru a facilita lucrul cu reflexia în Java.

Metoda `completeazaTabel(JTable tabela, List<T> lista)` este responsabilă pentru completarea unei tabele `JTable` cu datele dintr-o listă de obiecte. Metoda utilizează reflexia pentru a obține informații despre structura obiectelor din listă. Creează un model de tabel `DefaultTableModel` și adaugă coloane în tabel corespunzătoare câmpurilor obiectelor din lista. Apoi, pentru fiecare obiect din listă, extrage valorile câmpurilor utilizând reflexia și le adaugă într-un vector, care este apoi adăugat ca rând în modelul de tabel. La final, modelul de tabel este setat ca model al tabelei.

Metoda `idTabele(JComboBox<String> comboBox, List<T> lista)` este responsabilă pentru popularea unui combobox `JComboBox<String>` cu identificatorii obiectelor dintr-o listă. Metoda utilizează reflexia pentru a obține informații despre câmpurile obiectelor și adaugă identificatorii în combobox. Aceștia sunt obținuți utilizând reflexia și convertiți la șir de caractere. La final, combobox-ul este setat să afișeze primul element din listă ca element selectat implicit.

- **Clasa "Start":**

Clasa `Start` face parte din pachetul `start` și reprezintă clasa de start a aplicației. Această clasă conține metoda `main`, care este punctul de intrare în aplicație. În metoda `main`, sunt realizate anumite acțiuni comentate, cum ar fi crearea unui obiect `Client` și inserarea acestuia în baza de date prin intermediul clasei `ClientBLL`. Aceste acțiuni pot fi utilizate pentru testarea funcționalităților. De asemenea, se creează o instanță a clasei `InterfataMeniu`, care reprezintă interfața grafică a meniului principal al aplicației. Alte interfețe, cum ar fi `InterfataClient`, `InterfataProdus` și `InterfataComanda`, sunt și ele create, dar sunt comentate în prezent în metoda `main`. Aceasta indică faptul că în aplicație se pot utiliza diverse interfețe pentru diferite funcționalități.

* Utilizarea excepțiilor personalizate ajută la creșterea robusteții și fiabilității codului, permițând gestionarea și semnalarea erorilor specifice în mod corespunzător, în loc de a trata toate erorile într-un mod general.

4. IMPLEMENTARE

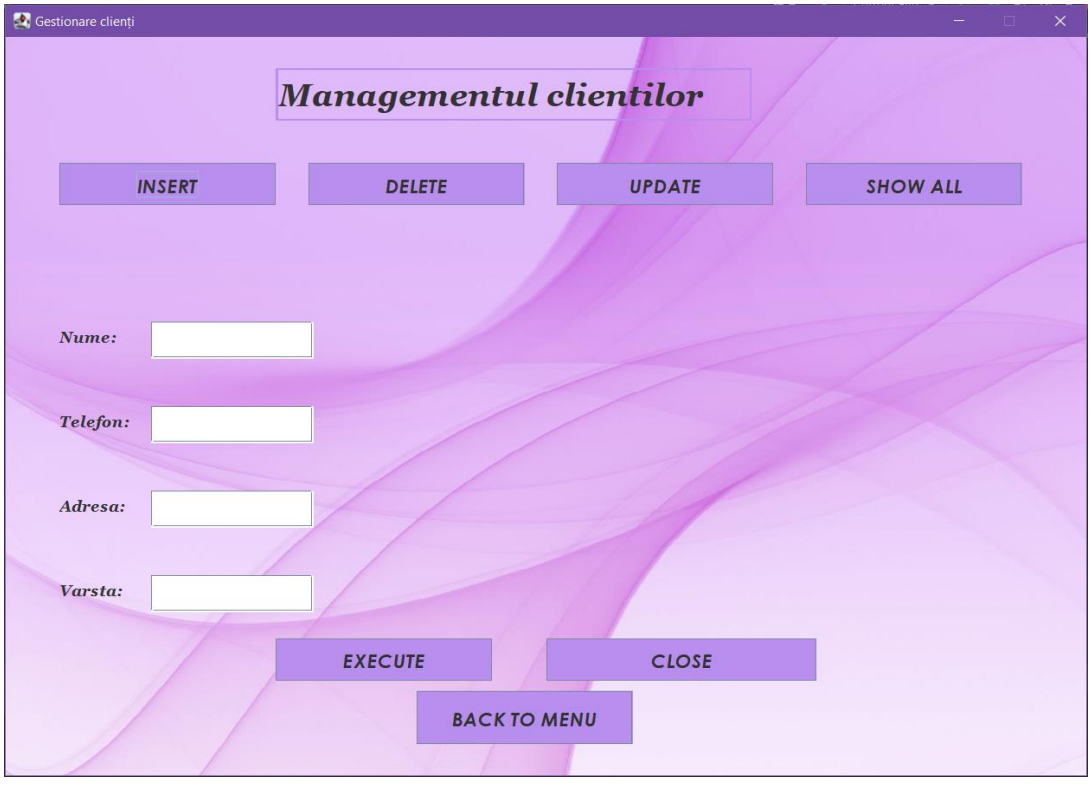
a. Aspect interfață:

Clasa InterfataClient oferă o interfață grafică utilizator pentru gestionarea clienților, cu funcționalități precum adăugarea, actualizarea, ștergerea și afișarea datelor despre clienți. Componentele grafice și metodele asociate facilitează interacțiunea cu utilizatorul și interogarea/modificarea datelor din sistemul de gestionare a clienților.

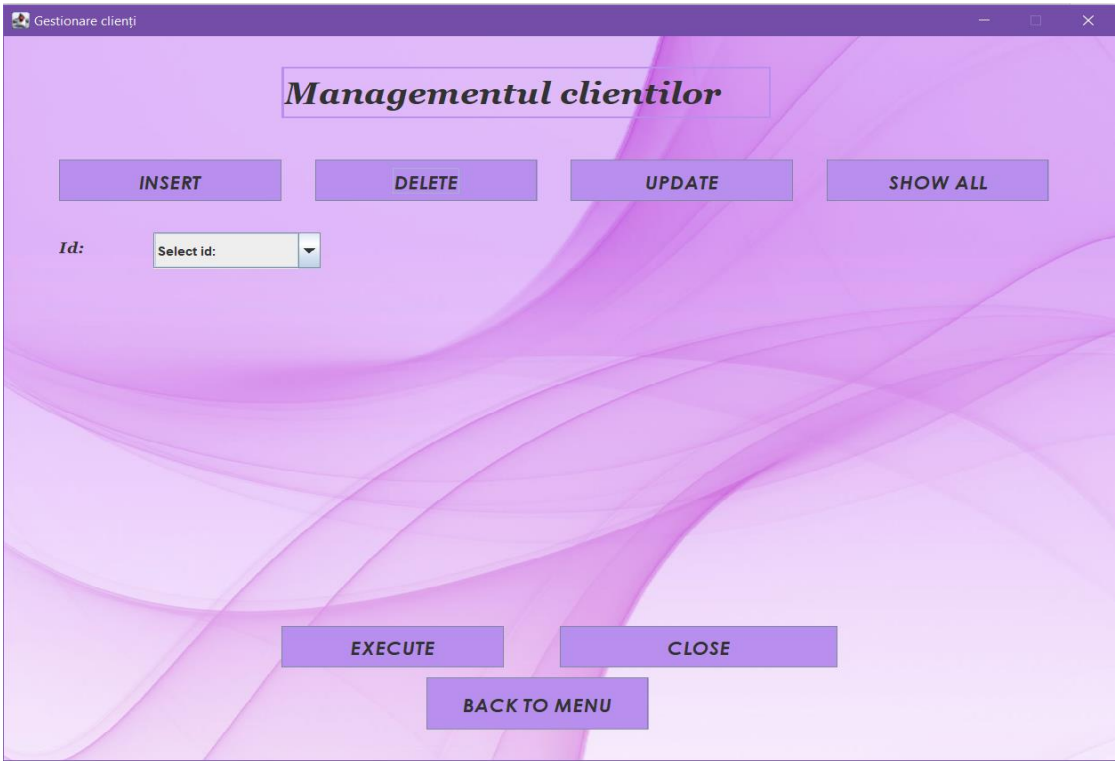
Am împărțit generalitatea de "interfață" în ami multe părți.

- InterfataMeniu - din care se poate alege în care interfață următoare a programului să trecem.

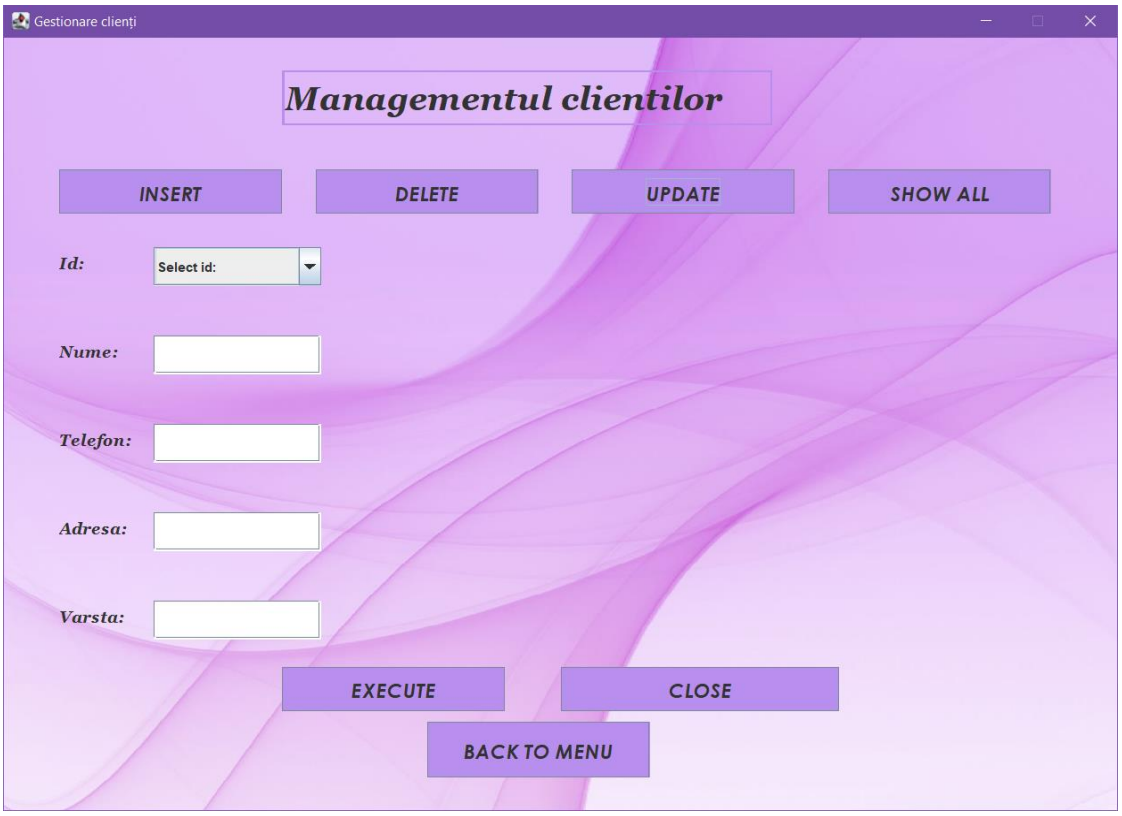
- InterfataClient - din care se poate alege ce instrucțiune să se execute asupra unui client (inserare, actualizare, stergere), afișarea tabelei de clienți sau întoarcerea la meniul principal. În funcție de ce buton se apasă, vor apărea pe ecram diferite componente de care este nevoie pentru executarea acțiunii butonului.
  - butonul “INSERT” a fost apăsat



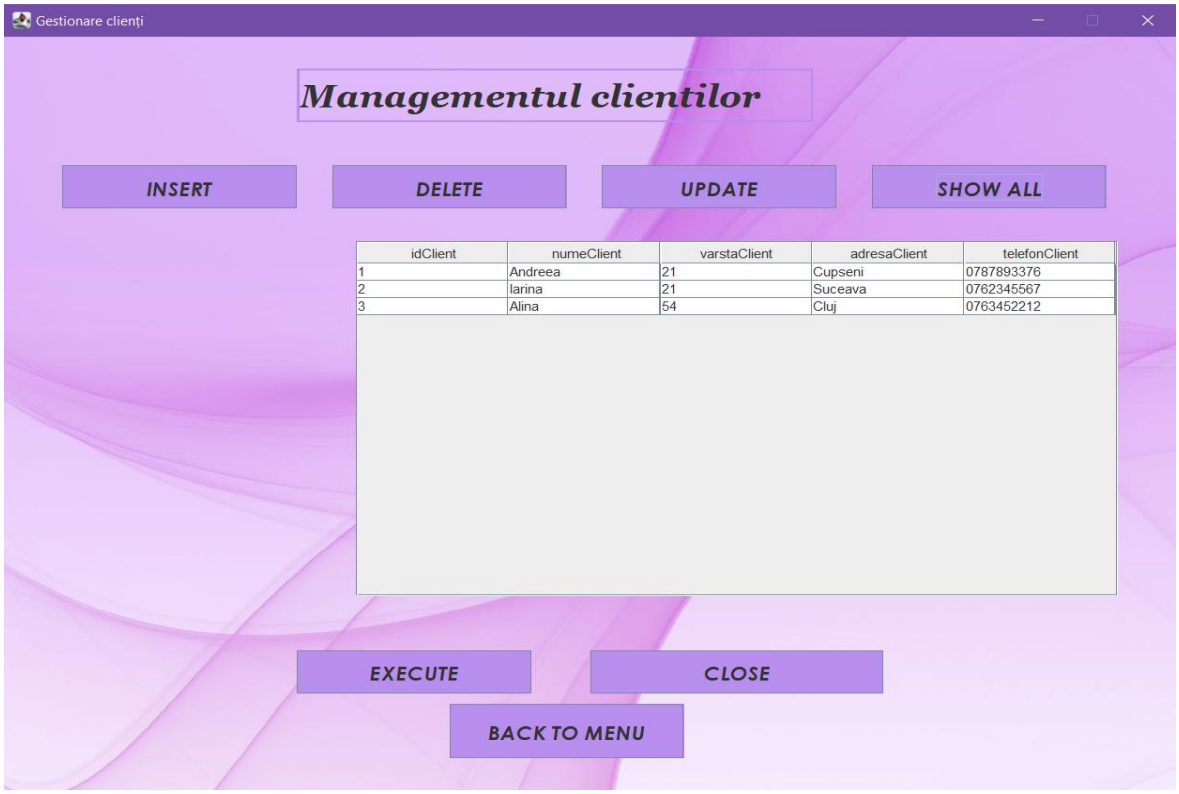
-butonul “DELETE” a fost apasat



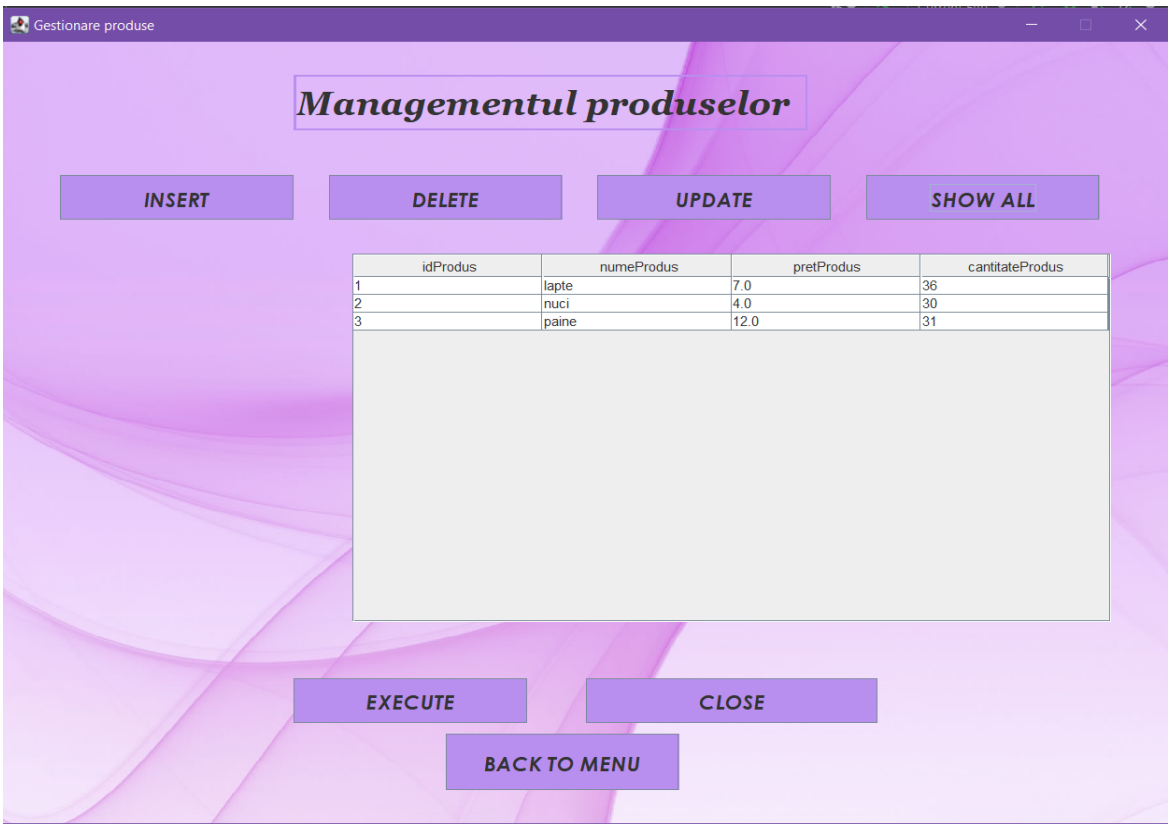
-butonul “UPDATE” a fost apasat



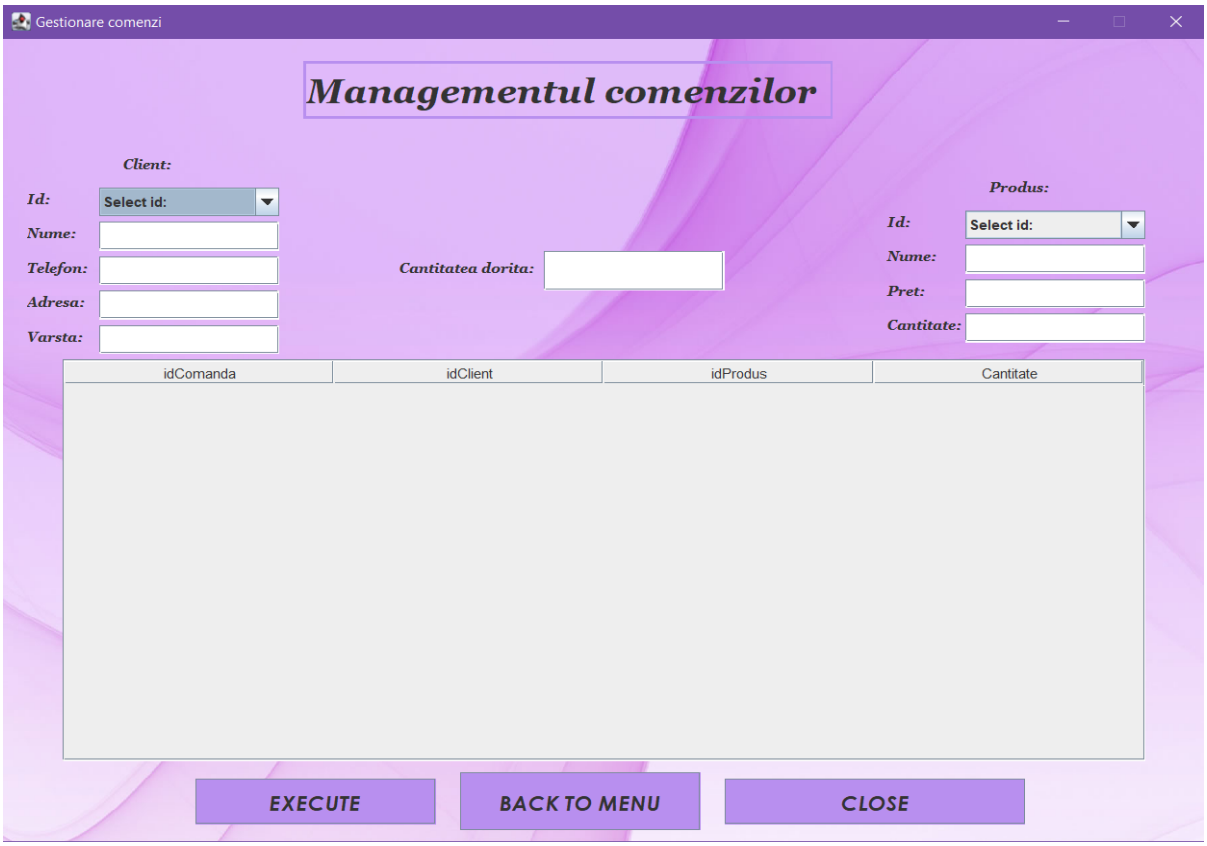
-butonul “SHOW ALL” a fost apasat



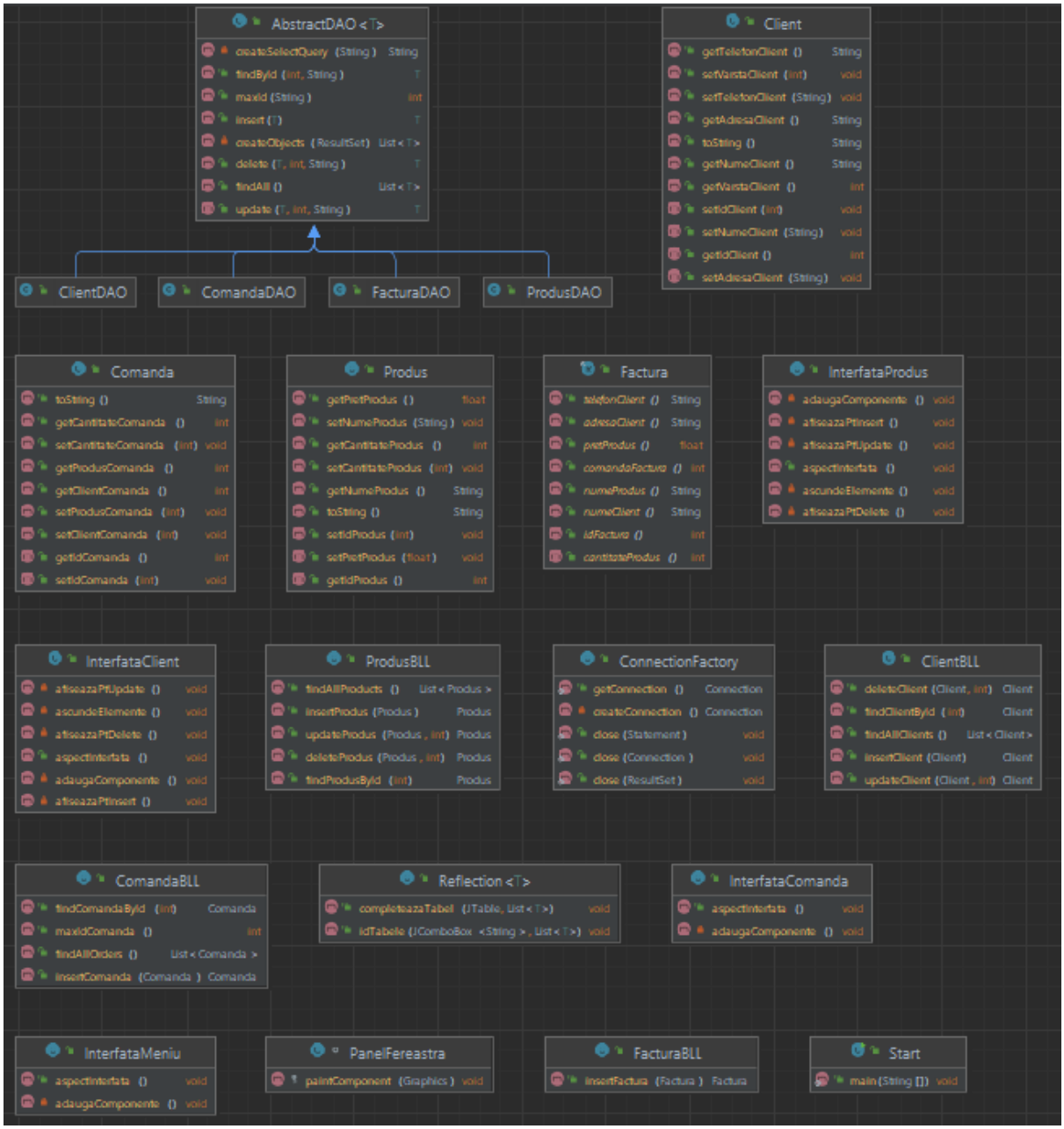
- InterfataProdus - din care se poate alege ce instrucțiune să se execute asupra unui produs (inserare, actualizare, stergere), afișarea tabelei de produse sau întoarcerea la meniul principal. În funcție de ce buton se apasă, vor apărea pe ecram diferite componente de care este nevoie pentru executarea acțiunii butonului.



- InterfataComanda - din care se poate alege un client, un produs și o cantitate dorită din acel produs. La apăsarea butonului de execuție se vor afișa comenzile în tabelă și se va genera o factură.



b. Implementare metodelor claselor:



- Metodele din “ClientBLL”

1. `ClientBLL()`: Constructorul clasei inițializează obiectul `ClientDAO`, creând astfel o instanță a acestuia pentru a putea utiliza metodele de acces la date pentru entitatea `Client`.

2. `findClientById(int id)`: Această metodă primește un ID de client și utilizează obiectul `ClientDAO` pentru a căuta un client în baza de date pe baza acestui ID. Dacă clientul nu este găsit, se aruncă o excepție `NoSuchElementException`. În caz contrar, clientul găsit este returnat.

3. `findAllClients()`: Această metodă returnează o listă cu toți clienții existenți în baza de date. Utilizează metoda `findAll()` a obiectului `ClientDAO` pentru a obține lista de clienți. Dacă lista este goală, se aruncă o excepție `NoSuchElementException`. Altfel, lista de clienți este returnată.

4. `insertClient(Client c)`: Această metodă primește un obiect `Client` și îl inserează în baza de date utilizând metoda `insert()` a obiectului `ClientDAO`. Dacă inserarea nu reușește, se aruncă o excepție `NoSuchElementException`. Altfel, clientul inserat este returnat.

5. `updateClient(Client c, int id)`: Această metodă primește un obiect `Client` actualizat și un ID de client și actualizează în baza de date clientul cu acel ID, utilizând metoda `update()` a obiectului `ClientDAO`. Dacă actualizarea nu reușește, se aruncă o excepție `NoSuchElementException`. Altfel, clientul actualizat este returnat.

6. `deleteClient(Client c, int id)`: Această metodă primește un obiect `Client` și un ID de client și șterge clientul cu acel ID din baza de date, utilizând metoda `delete()` a obiectului `ClientDAO`. Dacă ștergerea nu reușește, se aruncă o excepție `NoSuchElementException`. Altfel, clientul șters este returnat.

- Metodele din "ProdusBLL"

1. `ProdusBLL()`: Constructorul clasei inițializează obiectul `ProdusDAO`, creând astfel o instanță a acestuia pentru a putea utiliza metodele de acces la date pentru entitatea `Produs`.

2. `findProdusById(int id)`: Această metodă primește un ID de produs și utilizează obiectul `ProdusDAO` pentru a căuta un produs în baza de date pe baza acestui ID. Dacă produsul nu este găsit, se aruncă o excepție `NoSuchElementException`. În caz contrar, produsul găsit este returnat.

3. `findAllProducts()`: Această metodă returnează o listă cu toate produsele existente în baza de date. Utilizează metoda `findAll()` a obiectului `ProdusDAO` pentru a obține lista de produse. Dacă lista este goală, se aruncă o excepție `NoSuchElementException`. Altfel, lista de produse este returnată.

4. `insertProdus(Produs p)`: Această metodă primește un obiect `Produs` și îl inserează în baza de date utilizând metoda `insert()` a obiectului `ProdusDAO`. Dacă inserarea nu reușește, se aruncă o excepție `NoSuchElementException`. Altfel, produsul inserat este returnat.

5. `updateProdus(Produs p, int id)`: Această metodă primește un obiect `Produs` actualizat și un ID de produs și actualizează în baza de date produsul cu acel ID, utilizând metoda `update()` a obiectului `ProdusDAO`. Dacă actualizarea nu reușește, se aruncă o excepție `NoSuchElementException`. Altfel, produsul actualizat este returnat.

6. `deleteProdus(Produs p, int id)`: Această metodă primește un obiect `Produs` și un ID de produs și șterge produsul cu acel ID din baza de date, utilizând metoda `delete()` a obiectului `ProdusDAO`. Dacă ștergerea nu reușește, se aruncă o excepție `NoSuchElementException`. Altfel, produsul șters este returnat.

- Metodele din "ComandaBLL"

1. `ComandaBLL()`: Constructorul clasei inițializează obiectul `ComandaDAO`, creând astfel o instanță a acestuia pentru a putea utiliza metodele de acces la date pentru entitatea `Comanda`.

2. `findComandaById(int id)`: Această metodă primește un ID de comandă și utilizează obiectul `ComandaDAO` pentru a căuta o comandă în baza de date pe baza acestui ID. Dacă comanda nu este găsită, se aruncă o excepție `NoSuchElementException`. În caz contrar, comanda găsită este returnată.

3. `findAllOrders()`: Această metodă returnează o listă cu toate comenzile existente în baza de date. Utilizează metoda `findAll()` a obiectului `ComandaDAO` pentru a obține lista de comenzi. Dacă lista este goală, se aruncă o excepție `NoSuchElementException`. Altfel, lista de comenzi este returnată.

4. `insertComanda(Comanda c)`: Această metodă primește un obiect `Comanda` și îl inserează în baza de date utilizând metoda `insert()` a obiectului `ComandaDAO`. Dacă inserarea nu reușește, se aruncă o excepție `NoSuchElementException`. Altfel, comanda inserată este returnată.

5. `maxIdComanda()`: Această metodă returnează ID-ul maxim al comenzii din baza de date, care reprezintă ultima comandă inserată. Utilizează metoda `maxId()` a obiectului `ComandaDAO` pentru a obține acest ID

- Metodele din “FacturaBLL”

1. `FacturaBLL()`: Constructorul clasei inițializează obiectul `FacturaDAO`, creând astfel o instanță a acestuia pentru a putea utiliza metodele de acces la date pentru entitatea `Factura`.

2. `insertFactura(Factura f)`: Această metodă primește un obiect `Factura` și îl inserează în baza de date utilizând metoda `insert()` a obiectului `FacturaDAO`. Dacă inserarea nu reușește, se aruncă o excepție `NoSuchElementException`. Altfel, factura inserată este returnată.

- Metodele din “AbstractDAO”

1. `LOGGER`: Un obiect `Logger` utilizat pentru înregistrarea mesajelor de avertizare.

2. `type`: Un obiect `Class` care reprezintă tipul obiectelor gestionate de DAO. Este inițializat în constructorul clasei prin utilizarea reflecției pentru a obține tipul generic al clasei concrete care extinde `AbstractDAO`.

3. Constructorul clasei: Inițializează atributul `type` cu tipul generic al clasei concrete care extinde `AbstractDAO`.

4. `createSelectQuery(String field)`: Metodă privată care construiește și returnează o interogare SELECT bazată pe numele unui câmp. Această metodă este utilizată în metodele `findById()` și `delete()`.

5. `findAll()`: Metodă publică care returnează o listă cu toate obiectele din tabelul corespunzător clasei `AbstractDAO`. Metoda creează interogarea SELECT și utilizează conexiunea la baza de date pentru a obține rezultatele. Apoi, utilizează metoda `createObjects()` pentru a crea obiectele corespunzătoare din rezultatele interogării.

6. `findById(int id, String nameC)`: Metodă publică care returnează un obiect din tabelul corespunzător clasei `AbstractDAO` pe baza unui ID și a numelui coloanei. Metoda creează interogarea SELECT utilizând numele coloanei și utilizează conexiunea la baza de date pentru a obține rezultatul. Apoi, utilizează metoda `createObjects()` pentru a crea obiectul corespunzător din rezultat.

7. `createObjects(ResultSet resultSet)`: Metodă privată care primește un obiect `ResultSet` (rezultatul unei interogări) și creează o listă de obiecte pe baza acestuia. Metoda utilizează reflecția pentru a instanția obiectele și pentru a seta valorile câmpurilor pe baza datelor din rezultat.

8. `insert(T t)`: Metodă publică care inserează un obiect în tabelul corespunzător clasei `AbstractDAO`. Metoda construiește și executează o interogare INSERT pentru a adăuga obiectul în baza de date. Returnează obiectul inserat.

9. `update(T t, int id, String nameColumn)`: Metodă publică care actualizează un obiect în tabelul corespunzător clasei `AbstractDAO` pe baza unui ID și a numelui coloanei. Metoda construiește și executează o interogare UPDATE pentru a actualiza obiectul în baza de date. Returnează obiectul actualizat.

5. REZULTATE

În timpul dezvoltării proiectului, am implementat și executat mai multe scenarii de testare pentru a verifica funcționalitatea corectă a sistemului. Aceste scenarii acoperă diferite aspecte ale aplicației și ne asigură că totul funcționează conform așteptărilor. De exemplu tebela creata pentru facturi ne arata ca totul functoneaza corect.

Am utilizat în mod corespunzător JavaDoc pentru a documenta clasele și metodele. Acest lucru este esențial pentru a face codul mai ușor de înțeles și de întreținut. De asemenea, am generat fișierele JavaDoc, care reprezintă o resursă valoroasă pentru a înțelege mai bine structura și funcționalitatea proiectului.

Am generat un fișier SQL dump care implică crearea și organizarea scripturilor SQL necesare pentru a crea și popula tabelele din baza de date. Fișierul SQL dump conține instrucțiunile SQL pentru crearea structurii tabelelor și inserarea datelor corespunzătoare.

Acest fișier este esențial pentru a asigura portabilitatea și replicabilitatea bazei de date. Prin generarea unui fișier SQL dump, se poate distribui și restaura baza de date cu ușurință, fără a fi nevoie să creați manual tabelele și să inserați datele în mod individual.

6. CONCLUZII

Pe baza proiectului de gestionare a clienților, produselor și comenzilor într-un depozit diversificat, există oportunitatea de a dezvolta funcționalități suplimentare, precum implementarea unui sistem de raportare avansat. Acest sistem ar permite generarea de rapoarte personalizate în aplicație, furnizând informații relevante despre vânzări, stocuri, performanța clienților și alte aspecte importante ale afacerii. Aceasta ar îmbunătăți perspectiva utilizatorilor asupra activității și le-ar facilita luarea deciziilor informate.

7. BIBLIOGRAFIE

https://dsrl.eu/courses/pt/materials/PT2023_A3.pdf

https://dsrl.eu/courses/pt/materials/PT2023_A3_S1.pdf

https://dsrl.eu/courses/pt/materials/PT2023_A3_S2.pdf

https://www.baeldung.com/java-jdbc

https://mkyong.com/jdbc/how-to-connect-to-mysql-with-jdbc-driver-java/

https://dzone.com/articles/layers-standard-enterprise

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

https://www.baeldung.com/java-pdf-creation

https://www.baeldung.com/javadoc

https://dev.mysql.com/doc/workbench/en/wb-admin-export-import-management.html