DOCUMENTATIE

TEMA *3*

**Nume prenume: Hruban Andrada-Bianca**

**Grupa: 30226**

Cuprins

[1. Obiectivul temei 3](#_Toc481576928)

[2. Analiza problemei, modelare, scenarii, cazuri de utilizare 3](#_Toc481576929)

2.1 [Diagramă USE CASE 4](#_Toc481576933)

[3. Proiectare 4](#_Toc481576934)

[3.1. Stabilirea unei conexiuni la baza de date 4](#_Toc481576935)

[3.2. Diagrama de clase 5](#_Toc481576936)

[4. Implementare 6](#_Toc481576937)

5. Rezultate…………………………………………………………………………………………………………….8

[6. Concluzii 11](#_Toc481576943)

[7. Bibliografie 12](#_Toc481576944)

# Obiectivul temei

Obiectivul principal al acestei teme este să proiecteze și să implementeze o aplicație pentru gestionarea comenzilor de clienți pentru un depozit. Acest obiectiv principal poate fi împărțit în mai multe sub-obiective:

* Analiza problemei și identificarea cerințelor
* Proiectarea aplicației de gestionare a comenzilor
* Implementarea aplicației de gestionare a comenzilor
* Testarea aplicației de gestionare a comenzilor

Aplicația ar trebui să faciliteze un proces mai eficient și mai organizat de gestionare a comenzilor, reducând astfel timpul și efortul necesar pentru înregistrarea manuală.

# Analiza problemei, modelare, scenarii, cazuri de utilizare

### **Analiza problemei**: Gestionarea produselor, a clienților și a comenzilor pentru un depozit folosind registre scrise manual este dificilă și consumatoare de timp. Prin urmare, este nevoie de o aplicație care să automatizeze și să eficientizeze acest proces.

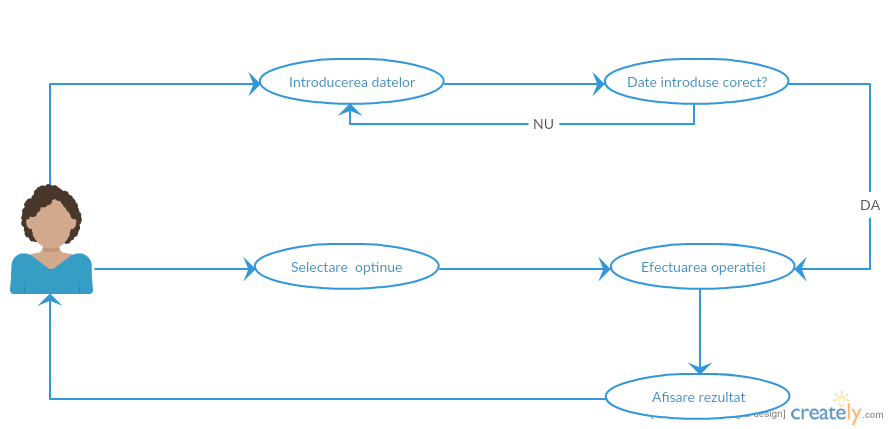
### **Modelare**: Aplicația va fi proiectată folosind modelul de arhitectură stratificată, care va include cel puțin patru pachete: dataAccessLayer, businessLayer, model și presentation. Va fi folosită o bază de date relațională pentru stocarea datelor.

### **Scenarii și cazuri de utilizare**: Una dintre utilizările principale ale aplicației ar fi adăugarea unui produs nou. În acest scenariu, actorul principal este angajatul. Acesta selectează opțiunea de a adăuga un produs nou, aplicația afișează un formular în care trebuie să fie introduse detaliile produsului, angajatul introduce numele produsului, prețul și stocul curent, apasă butonul "Adaugă", aplicația stochează datele produsului în baza de date și afișează un mesaj de confirmare. Dacă angajatul introduce o valoare negativă pentru stocul produsului, aplicația va afișa un mesaj de eroare și îi va cere să introducă un stoc valid.

### Alte cerințe funcționale pot include: permisiunea de a adăuga un client nou, editarea detaliilor unui produs sau a unui client existent, ștergerea unui produs sau a unui client, vizualizarea tuturor produselor și clienților într-un tabel, crearea de comenzi de produse etc.

### Cerințele non-funcționale pot include: intuitivitatea și ușurința de utilizare a aplicației, limitarea claselor la maximum 300 de linii și a metodelor la maximum 30 de linii, utilizarea convențiilor de denumire Java, generarea automată a fișierelor JavaDoc pentru documentarea claselor etc.

### Diagramă USE CASE



# Proiectare

Aplicația este structurată folosind arhitectura stratificată în șase pachete: start, presentation, dao, connection, bll și model.

În pachetul "start", avem clasa App în care creăm fereastra principală și stabilim controlerul interfeței utilizator.

În pachetul "presentation", avem toate clasele legate de interfața aplicației: MainWindow, ClientPanel, ProductPanel, OrderPanel și Controller.

Pachetul "dao" conține clasa AbstractDAO, în care avem metodele de adăugare, actualizare, ștergere și căutare implementate cu ajutorul tehnicii de reflecție. Avem, de asemenea, clasele ClientDAO, ProductDAO și OrderDAO.

În pachetul Business Logic Layer (bll), avem clasele ClientBLL, ProductBLL și OrderBLL.

În pachetul "connection" avem clasa ConnectionFactory, care este folosită pentru a crea și a închide conexiunea cu baza de date.

Pachetul "model" stochează toate clasele necesare pentru a modela datele stocate în baza de date. Avem clasa Client, clasa Product, clasa Order și clasa OrderDetail.

## Diagrama de clase

A picture containing text, diagram, plan, screenshot

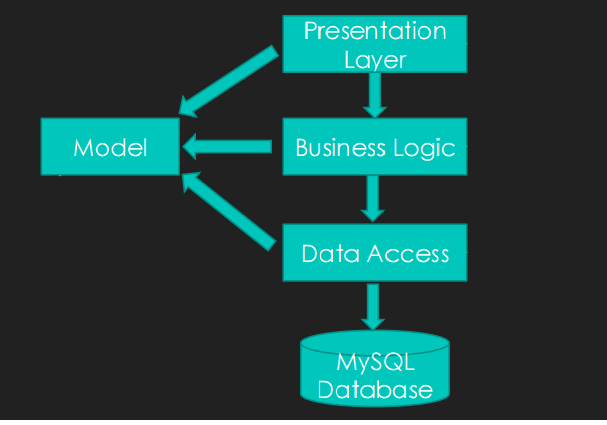
Description automatically generated A diagram of a model

Description automatically generated with low confidence

A picture containing diagram, plan, technical drawing, line

Description automatically generated

# Implementare



Toate datele despre clienți, produse și comenzi sunt stocate într-o bază de date numită "order\_management" în trei tabele: client, product și order.

În tabela "client", stocăm id-ul clientului și vârsta ca numere întregi, iar numele, e-mailul și adresa ca VARCHAR(100).

În tabela "product", stocăm id-ul produsului și stocul ca numere întregi și numele și producătorul ca VARCHAR(100).

În tabela "order", stocăm id-ul comenzii, id-ul clientului, id-ul produsului și cantitatea ca numere întregi.

Clasa Client stochează id-ul, numele, e-mailul, adresa și vârsta clientului. Avem getteri și setteri pentru toate câmpurile clasei. Avem de asemenea patru constructori, unul cu toate câmpurile, unul cu numele, e-mailul, adresa și vârsta, unul cu doar id-ul și un constructor gol.

Clasa Product stochează id-ul, numele, producătorul unui produs și stocul. Clasa oferă, de asemenea, getteri și setteri pentru toate câmpurile și patru constructori: unul care acceptă ca parametri toate câmpurile, unul care acceptă ca parametri numele, producătorul și stocurile, unul care acceptă ca parametru doar id-ul produsului și un constructor gol.

Clasa Order stochează id-ul comenzii, id-ul clientului care a făcut comanda, id-ul produsului și cantitatea acelui produs. Clasa oferă setteri și getteri pentru toate câmpurile. De asemenea, oferă patru constructori: unul care acceptă ca parametri toate câmpurile, unul care accept ca parametri id-ul clientului, id-ul produsului și cantitatea, unul care acceptă ca parametru doar id-ul produsului și un constructor gol.

În ConnectionFactory, stocăm toate câmpurile și metodele folosite pentru a crea conexiunea cu baza de date și pentru a închide conexiunea.

În pachetul dao, avem clasa AbstractDAO care generează, folosind reflecția, interogările specifice pentru accesarea bazei de date în vederea adăugării, ștergerii, actualizării, găsirii unui obiect dat ca parametru și pentru a găsi toate obiectele.

Pentru a găsi un obiect în baza de date, mai întâi cream interogarea SQL prin concatenarea șirului "SELECT \* FROM order\_management." cu numele clasei obiectului furnizat ca parametru și cu șirul "where id = ?". După aceea, cream conexiunea la baza de date și căutăm câmpul cu numele "id" în obiect, luăm valoarea câmpului și înlocuim caracterul de întrebare cu id-ul găsit. Executăm interogarea și apoi verificăm dacă avem un element cu id-ul furnizat. Dacă avem un rezultat, atunci apelăm fiecare setter al obiectului cu rezultatele returnate de la baza de date.

Metoda de inserare primește un obiect ca parametru și creează șirul de inserare prin parcurgerea tuturor numelor de câmpuri ale obiectului. După ce avem șirul de inserare, înlocuim toate caracterele de întrebare cu valorile câmpurilor. Id-ul elementului nou inserat este stocat în variabila insertedId și apoi returnat.

Pentru a actualiza un element în baza de date, mai întâi generăm șirul de actualizare bazat pe numărul și numele câmpurilor. După ce am generat șirul, înlocuim toate caracterele de întrebare cu valoarea concretă a fiecărui câmp și apoi executăm interogarea.

În metoda de ștergere, generăm mai întâi șirul de ștergere luând numele simplu al clasei obiectului și apoi înlocuind caracterul de întrebare din acesta cu valoarea câmpului id al obiectului. La final, executăm interogarea.

Pentru a obține toate elementele din tabel, mai întâi cream șirul de interogare, apoi generăm un arrayList cu toate elementele returnate de la execuția interogării.

Avem de asemenea DAO-uri specifice pentru client, produs și comandă. Aceste clase extind AbstractDAO cu clasa lor specifică.

În OrderDAO avem metoda getOrderDetails care primește o comandă ca parametru și generează un obiect OrderDetails care conține id-ul comenzii, numele clientului care a făcut comanda, numele produsului și cantitatea comenzii.

În pachetul Business Logic Layer (bll), avem clasele ClientBLL, ProductBLL și OrderBLL. Aceste clase sunt folosite pentru a comunica cu toate cele trei clase DAO.

Pentru interfață, folosim toolkit-ul Java Swing. Avem clasa MainWindow în care avem trei panouri, unul pentru fereastra clientului, unul pentru fereastra produsului și unul pentru fereastra comenzii. Avem de asemenea trei butoane “Clients window”, “Products window” și “Order window” pentru a naviga către fereastra dorită.

În toate cele trei clase ClientPanel, ProductPanel și OrderPanel stocăm JPanel, JTable în care afișăm datele din baza de date și elementele de interfață necesare pentru a adăuga, șterge și actualiza clienții și produsele și pentru a crea comenzi.

În panoul clientului, avem tabelul în care sunt afișați clienții din baza de date și câmpuri de text pentru datele pentru un nou client, pentru actualizarea unui client existent sau un câmp de text pentru id-ul clientului pe care dorim să-l ștergem.

În panoul de produse, avem un tabel cu toate produsele stocate și câmpuri de text și etichete pentru datele necesare pentru a crea, actualiza și șterge un produs.

În panoul de comenzi, avem un tabel pentru afișarea comenzilor din baza de date și câmpuri de text pentru id-ul clientului, id-ul produsului și numărul de produse.

În clasa Controller, stocăm toate metodele folosite pentru a interacționa cu interfața utilizatorului. Pentru a afișa datele pe fiecare tabel, avem metoda generateTableData care primește un ArrayList de obiecte și un DefaultTableModel ca parametri și adaugă la DefaultTableModel antetul coloanelor și rândurile folosind reflecția. Mai întâi setăm numărul de coloane și rânduri la 0 pentru a reseta datele modelului. Apoi obținem numele antetului coloanei din primul element al matricei folosind metoda getFieldsName. În această metodă, folosim reflecția pentru a genera un ArrayList de Strings reprezentând numele câmpurilor obiectului dat ca parametru. După ce avem toate numele câmpurilor, adăugăm coloanele la model și apoi folosim metoda addObjectToTable care primește ca parametri un obiect și un DefaultTableModel și extrage valorile câmpurilor și adaugă un nou rând în tabel.

La nivelul interfeței, toate acțiunile sunt tratate prin metode specifice în clasa Controller. De exemplu, pentru a adăuga un client nou, utilizatorul trebuie să completeze toate câmpurile necesare și apoi să apese butonul "Adaugă client". În momentul în care butonul este apăsat, metoda addClient din Controller este apelată. Această metodă creează un nou obiect Client folosind datele introduse de utilizator și apoi îl trimite la metoda insert a ClientDAO pentru a fi adăugat în baza de date. La final, tabelul este actualizat pentru a afișa noile date.

Similar, pentru a șterge un client, utilizatorul trebuie să introducă id-ul clientului pe care dorește să îl șteargă și apoi să apese butonul "Șterge client". Metoda deleteClient din Controller este apelată, care la rândul ei apelează metoda delete a ClientDAO cu id-ul introdus de utilizator. Dacă operația este reușită, clientul este șters din baza de date și tabelul este actualizat.

Actualizarea datelor unui client existent se face într-un mod similar. Utilizatorul trebuie să introducă toate datele noi în câmpurile corespunzătoare și apoi să apese butonul "Actualizează client". Metoda updateClient din Controller este apelată, care la rândul său apelează metoda update a ClientDAO cu datele introduse de utilizator.

Procesul de creare, ștergere și actualizare a produselor și comenzilor este similar cu cel descris mai sus pentru clienți.

Clasa **BillDAO** este o clasă de tip DAO (Data Access Object) specifică pentru gestionarea operațiilor asupra facturilor în baza de date. Aceasta extinde clasa **AbstractDAO** și oferă funcționalitatea necesară pentru a manipula datele facturilor în baza de date. Clasa **AbstractDAO** conține metode generice pentru adăugarea, actualizarea, ștergerea și găsirea obiectelor în baza de date, iar **BillDAO** le moștenește pentru a le folosi în contextul facturilor.

Clasa **BillBLL** face parte din stratul de Business Logic Layer (BLL). Aceasta conține o instanță a **BillDAO** și oferă metode pentru a procesa datele înainte de a fi transmise în baza de date. În exemplul dat, metoda **insertBill** primește patru argumente: id-ul facturii, id-ul comenzii, totalul facturii și data facturii. Aceasta creează un nou obiect de tip **Bill** cu aceste valori și apelează metoda **insert** a **BillDAO** pentru a adăuga noua factură în baza de date.

Clasa **Bill** este o clasă de model care reprezintă o factură în sistemul nostru. Aceasta conține patru câmpuri: id-ul facturii, id-ul comenzii, totalul facturii și data facturii. Observăm că aceasta este o clasă **record**, o caracteristică introdusă în Java 14, care simplifică crearea claselor de date. Clasele **record** sunt imutabile și au metodele **equals()**, **hashCode()** și **toString()** generate automat, ceea ce le face potrivite pentru reprezentarea unor obiecte cu valori simple de date, cum ar fi facturile din acest context.

1. **Rezultate**

A screenshot of a computer screen

Description automatically generated with medium confidence

A screenshot of a computer screen

Description automatically generated with low confidence

A screenshot of a computer screen

Description automatically generated with low confidenceA screenshot of a computer

Description automatically generated

1. **Concluzii**

După construirea acestei aplicații, am învățat cum să conectez o aplicație Java la o bază de date, cum să creez o clasă generică, folosind tehnici de reflexie, pentru a adăuga, șterge, actualiza și găsi un element într-un tabel din baza de date. Am aprofundat, de asemenea, arhitectura stratificată a unei aplicații și am înțeles mai bine rolul fiecărui strat: prezentare, logică de afaceri și acces la date. În final, am acumulat experiență în lucrul cu Java Swing pentru a dezvolta o interfață grafică a utilizatorului.

1. **Biografie**
   1. Fundamental Programming Techniques Course
   2. Fundamental Programming Techniques Laboratory
   3. https://docs.oracle.com