DOCUMENTATIE

TEMA 3

NUME STUDENT: Ungureanu Iulia Iarina

GRUPA: 30225

# CUPRINS

[1. Obiectivul temei 3](#_Toc95297885)

[2. Analiza problemei, modelare, scenarii, cazuri de utilizare 3](#_Toc95297886)

[3. Proiectare 3](#_Toc95297887)

[4. Implementare 3](#_Toc95297888)

[5. Rezultate 3](#_Toc95297889)

[6. Concluzii 3](#_Toc95297890)

[7. Bibliografie 3](#_Toc95297891)

# Obiectivul temei

Proiectați și implementați o aplicație care are ca scop procesarea comenzilor unui depozit. Sa se folosească baze de date relaționale pentru depozitarea produselor, clienților si a comenzilor. Aplicația trebuie sa fie implementata conform modelului arhitectural stratificat și ar trebui să folosească

(minimum) următoarele clase:

• Clase de modele - reprezintă modelele de date ale aplicației

• Clasele de Business Logic - conțin logica aplicației

• Clase de prezentare – clase legate de GUI

• Clase de acces la date - clase care conțin accesul la bazele de date

Sub-obiective:

• Analizați problema și identificați cerințele ( 2 )

• Proiectați aplicația de simulare ( 3 )

• Implementarea aplicației de simulare ( 4 )

• Testați aplicația de simulare ( 5 )

# Analiza problemei, modelare, scenarii, cazuri de utilizare

Use Case: ștergere client

Primary Actor: user

Main Success Scenario:

1. Utilizatorul alege clientul dorit

2. Utilizatorul apasă butonul delete.

3. Aplicația actualizează tabela de clienți.

Alternative Sequence: Utilizatorul nu a selectat nici un client => mesaj de eroare

Use Case: adăugare client

Primary Actor: user

Main Success Scenario:

1. Utilizatorul introduce de la tastatura datele clientului : nume, prenume si vârsta.

2. Utilizatorul apasă butonul New Client.

3. Aplicația actualizează tabela de clienți.

Alternative Sequence: Utilizatorul nu a introdus toate informațiile, sau a introdus date incorecte (Ex: vârsta negativa sau un sir de caractere) => mesaj de eroare

Use Case: editare client

Primary Actor: user

Main Success Scenario:

1. Utilizatorul alege câmpul pe care își dorește sa îl editeze
2. Utilizatorul introduce noua data in câmpul respectiv.

3. Utilizatorul apasă butonul Edit Client.

4. Aplicația actualizează tabela de clienți.

Alternative Sequence:

* Utilizatorul nu a selectat nici un client => mesaj de eroare
* Utilizatorul nu a introdus toate informațiile, sau a introdus date incorecte (Ex: vârsta negativa sau un sir de caractere) => mesaj de eroare

Use Case: ștergere produs

Primary Actor: user

Main Success Scenario:

1. Utilizatorul alege produsul dorit

2. Utilizatorul apasă butonul delete.

3. Aplicația actualizează tabela de produse.

Alternative Sequence: Utilizatorul nu a selectat nici un produs => mesaj de eroare

Use Case: adăugare produs

Primary Actor: user

Main Success Scenario:

1. Utilizatorul introduce de la tastatura datele produsului : nume, preț si cantitate.

2. Utilizatorul apasă butonul New Produs.

3. Aplicația actualizează tabela de produse.

Alternative Sequence: Utilizatorul nu a introdus toate informațiile, sau a introdus date incorecte (Ex: preț sau cantitate negativa sau un sir de caractere) => mesaj de eroare

Use Case: editare produs

Primary Actor: user

Main Success Scenario:

1. Utilizatorul alege câmpul pe care își dorește sa îl editeze
2. Utilizatorul introduce noua data in câmpul respectiv.
3. Utilizatorul apasă butonul Edit Produs.
4. Aplicația actualizează tabela de produse.

Alternative Sequence:

* Utilizatorul nu a selectat nici un produs => mesaj de eroare
* Utilizatorul nu a introdus toate informațiile, sau a introdus date incorecte (Ex: preț sau cantitate negativa sau un sir de caractere) => mesaj de eroare

Use Case: adăugare comanda

Primary Actor: user

Main Success Scenario:

1. Utilizatorul alege clientul care face comanda, produsul comandat si introduce cantitatea in căsuța respectiva.
2. Utilizatorul apasă butonul Comanda.
3. Aplicația actualizează tabela de comenzi.

Alternative Sequence:

* Utilizatorul nu a selectat nici un produs sau client => mesaj de eroare
* Utilizatorul nu a introdus cantitatea sau a introdus date incorecte (Ex: cantitate negativa sau un sir de caractere) => mesaj de eroare

Cerințe funcționale:

Aplicația trebuie sa afișeze toate produsele, clienții si comenzile existente.

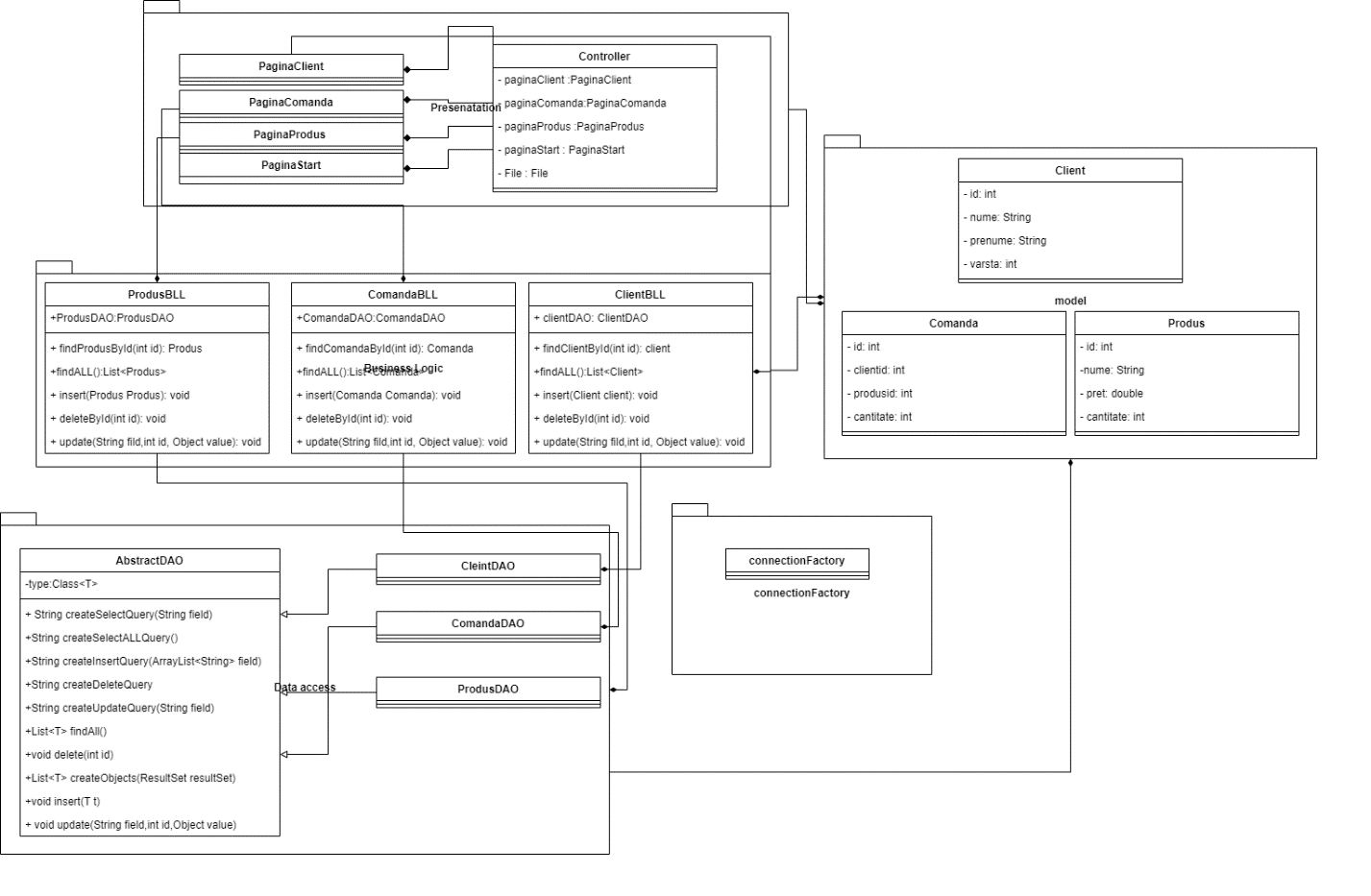
Aplicația trebuie sa lase utilizatorul sa adauge noi client, produse si comenzi;

Aplicația trebuie sa lase utilizatorul sa editeze produsele si clienții existenți.

Aplicația trebuie sa lase utilizatorul sa șteargă produsele si clienții existenți.

# Proiectare

* 1. Diagramele UML de clase si de pachete:



**DAO: Data access**

AbstractDAO implementează metode generale de inserare, ștergere, actualizare si selectare a tuturor claselor folosite:

Metoda findALL returnează o lista de obiecte din tabela aleasa conform clasei care a apelat funcția. Ea creează conexiunea cu baza de date, apoi apelează CreateSelectALLQuery care returnează un String cu un Querry care selectează toate tuplele tabelei T : „SELECT \* FROM ...tipul obiectului.. ”. Dupa obține rezultatul in resultSet si apelează pe el metoda createObjects().

Metoda findByID returnează obiectul cu id-ul trimis ca parametru din tabela corespunzătoare clasei care a apelat metoda. Ea creează conexiunea cu baza de date, apoi apelează CreateSelectQuery care returnează un Querry general de selecție a tuturor tuplelor care au field = paramField: „SELECT \* FROM .tipul obiectului. WHERE fild =?”.

Metoda delete șterge obiectul cu id-ul trimis ca parametru din tabela corespunzătoare clasei care a apelat metoda. Ea creează conexiunea cu baza de date, apoi apelează CreateDeleteQuery care returnează un String cu un Querry de ștergere a unei tuple conform câmpului trimis ca parametru: „DELETE FROM tipul obiectului WHERE field=?”. Adauga ca field valoarea „id” si executa update().

Metoda insert inserează in tabela obiectul trimis ca parametru. Ea creează conexiunea cu baza de date, apoi apelează CreateInsertQuery care returnează un String cu un Querry de inserare a unei tuple într-o tabela in funcție de lista de câmpuri a obiectului si tipul acestuia: „INSERT INTO clasa obiect (feild1, field2 , field3,.....) VALUES (?,?,?,...)”. Aceasta metoda parcurge toate câmpurile obiectului si le adaugă in field1,field2, ... si apoi adaugă același număr de „?”. Metoda aplica retriveField(t) si timite lista de câmpuri catre crateInsertQuerry , după executa addToStatement pentru a înlocui „?” cu valorile obiectului t si executa update.

Metoda update modifica tupla cu id-ul dat ca parametru astfel: valoarea din câmpul „field” este înlocuita cu valoarea data ca parametru „value”. Ea creează conexiunea cu baza de date, apoi apelează apelează CreateUpdateQuery, care returnează un String cu un Querry de modificare a unei tuple conform câmpului trimis ca parametru: „UPDATE nume\_clasa SET field = ? WHERE id=?”. Apoi verifica tipul câmpului „value” : String, Double si Integer si executa metoda potrivita: setString, setDouble, respectiv setInt, care o vor pune in statement pe poziția corecta „1”. Pe pozita „2” v-a fi adaugat id-ul obiectului si se v-a executa Update.

StringBuilder sb = new StringBuilder();”

Clasa Reflection conține doua metode:

* retrieveField care returnează o lista de String-uri cu fiecare câmp instanța al obiectului trimis ca parametru.
* addToStatement care adauga in statement-ul trimis ca parametru toate câmpurile din obiectul obiect.

Clasele ClientDAO, ComandaDAO, PordusDAO implementează toate metodele specifice fiecărei clase, iar operațiile CRUD v-or fi moștenite de la superclasa AbstractDAO.

**BLL: BUSINESS LOGIC**

Clasele ClientBLL, ComandaBLL, ProdusDAO încapsulează logica aplicației. Ele apelează metodele din DAO.

**Presentation:**

Controller-ul verifica corectitudinea datele de intrare, generează fiecare pagina si datele din acestea. Pentru fiecare pagina invoca funcții din Bll pentru a prelua datele din baza de date. Acesta efectuează actualizări , ștergeri si creări prin intermediul acestor funcții. Conține următorii ActionListener:

ActionListener pentru fiecare buton din pagina Client:

\* newClient alege valorile inserate de utilizator si adaugă clientul creat in tabela: i-a din interfata valorile celor 3 campuri: nume, prenume si varsta. Daca sunt corecte creeaza un obiect Client cu aceste valori si il insereaza in tabela, altfel arunca o exceptie si afizeaza un mesaj de eroare corespunzator. La final efectueaza un refresh pe pagina.

\* DeleteClient șterge clientul selectat de utilizator(acesta trebuie sa selecteze linia respectiva// nu contează coloana) . In functie de linia selectata de utilizator, putem prelua din JTable id-ul si il trimitem la metoda deleteByID care v-a sterge din tabela obiectul respectiv. La final efectueaza un refresh pe pagina.

\* EditClient editează câmpul selectat de utilizator care trebuie sa selecteze câmpul respectiv si sa insereze un număr in textFildul respectiv. Metoda declara variabilele row, column, id si introduce in ele datele preluate din interfata prin JTable. Daca nu a fost selectata nici o coloana sau linie atunci returneaza o exceptie si se afiseaza un mesaj corespunzator, altfel cu ajutorul liniei poate selecta id-ul care stim ca se afla pe coloana 3. Supa i-a din interfata valorea care trebuie modificata: nume, prenume sau varsta. Daca sunt corecte apeleza un Update pe valoarea respectiva cu aceste valori si il insereaza in tabela, altfel arunca o exceptie si afizeaza un mesaj de eroare corespunzator. La final efectueaza un refresh pe pagina.

* BackBackListener – iesire din pagina

ActionListener pentru fiecare buton din pagina Produs

\* newProdus alege valorile inserate de utilizator si adauga produsul creat in tabela. Metoda preia din interfata valorile celor 3 campuri: nume, pret si cantitate. Daca sunt corecte creeaza un obiect Produs cu aceste valori si il inserează in tabela, altfel arunca o excepție si afizeaza un mesaj de eroare corespunzator. La final efectueaza un refresh pe pagina.

\* DeleteProdus șterge Produsul selectat de utilizator(acesta trebuie sa selecteze linia respectiva// nu contează coloana). In funcție de linia selectata de utilizator, putem prelua din JTable id-ul si il trimitem la metoda deleteByID care v-a șterge din tabela obiectul respectiv. La final efectuează un refresh pe pagina.

\* EditProdus editează câmpul selectat de utilizator. Acesta trebuie sa selecteze câmpul respectiv si sa insereze un număr in textFildul respectiv.

* BackListener – iesire din pagina

ActionListener pentru fiecare buton din pagina Comanda:

* BackComandaListener – iesire din pagina

NewComandaListener – care declara doua variabile : row1 si row2 reprezentand cele doua linii selectate de utilizator din tabelele client, respectiv produs. Aceasta preia cele doua valori din interfata prin intermediul JTable si selecteaza id-ul clientului si al produsului in Cid si Pid. Verifica daca cantitatea intodus se potriveste cu cantitatea disponibila a produsului cu id-ul Pid, altfel returneaza o eroare si afiseaza un mesaj corespunzator. Creaza un obiect de tip Comanda cu campurile : Cid, Pid , cantitateaIntrodusa si efectueaza insert cu acest obiect. Tot aici genereaza si bonurile: creează un obiect de tip file cu titlul : comanda\_id si un mesaj de tiplul: ”„Clientul "+ nume\_client +" "+prenume\_client+" a comandat "+cantitateIntrodusa+" "+nume\_produs+" la pretul de "+pret\_produs+"."”. Aplicat metoda writeFile() cu text-ul creat ca parametru. . La final efectuează un refresh pe pagina.

PaginaStart este prima pagina a aplicației. In ea exista trei butoane: CLIENTI, PRODUSE, COMENZI Fiecare dintre acestea duc la paginile respective: PaginaClient, PaginaProdus, PaginaComenzi.

Interfața PaginaClient conține: trei butoane pentru : ADAUGARE, STRGERE si EDITARE de clienți si trei câmpuri pentru adăugarea de client : nume, prenume, vârsta. Afișare tabelei Client se face într-o JTabel. Atributele se pun pe prima linie apoi afișam variabilele fiecărei tuple din tabela in ordinea respectiva.

Interfața PaginaProdus conține: trei butoane pentru : ADAUGARE, STRGERE si EDITARE de produse si trei câmpuri pentru adăugarea de produse : nume, preț, cantitate. Afișare tabelei Produs se face într-o JTable. Atributele se pun pe prima linie apoi afișam variabilele fiecărei tuple din tabela in ordinea respectiva.

Pagina Comanda conține trei tabele : clienți, produse si comenzi. Utilizatorul poate crea o comanda prin selectarea unui client, a unui produs si introducerea unei cantități valide.

**Model:**

In Model avem trei clase pentru cele trei tabele din baza de date:

* Clientul conține 4 câmpuri : id, nume, prenume, vârsta
* Comanda conține 4 câmpuri: id, clientid, produsid si cantitate
* Produsul conține patru câmpuri: nume, preț, cantitate si id;

# 

Clasa Reflection avem doua metode: retriveField care primeste un obiect, creasa un ArrayList fields si parcurge cu for each fiecare camp al obectului respectiv. Obiectul value primeste numele campului si il adauga in lista numai daca este diferit de id.

AddToStatement primeste statement-ul si obectul ale carui valori trebuie introduse in el. Printr-un for each parcurgem toate campurile obectului si salvam in value valorile date. Daca acesta este diferit de campul „id” verificam tipul campului: String, Double, Integer (pentru cazul nostru) si aplicam metodele statement-ului: setString, setDouble , respectiv setInt adaugandu-le dupa contoulrul i pe pozititle corecte din statement.

# Concluzii

In acest proiect am invatat sa utilizaez baze de date cu ajutorul programarii orientate pe obiect, crearea de tabele si utilizarea lor in interfata si interfete cu mai multe frame-uri.

# Bibliografie

[1] Kishori Sharan, Peter Späth, More Java 17 - An In-Depth Exploration of the Java

Language and Its Features, Third Edition, Apress, 2021.

[2] B. Goetz et al., Java Concurrency in Practice, First Edition, Addison-Wesley

Professional, 2006.

[3] https://docs.oracle.com/javase/tutorial/essential/concurrency/index.html

[4] https://www.baeldung.com/java-synchronized-collections