**FUNDAMENTAL PROGRAMMING TECHNIQUES**

**ASSIGNMENT 3**

**ORDERS MANAGEMENT**

**Moldovan Claudia**

**Grupa 30224**

**An universitar 2021-2022**

Cuprins

[1.Cerinta 3](#_Toc100596779)

[2.Diagrama UML 3](#_Toc100596780)

[3. Teorie 4](#_Toc100596781)

[4.Functionarea aplicatiei 5](#_Toc100596782)

[5. Implementare 5](#_Toc100596783)

[6.Rezultate 12](#_Toc100596784)

## 1.Cerinta

Luați în considerare o aplicație Managementul comenzilor pentru procesarea comenzilor clienților pentru un depozit. Bazele de date relaționale ar trebui folosite pentru a stoca produsele, clienții și comenzile. Aplicația ar trebui să fie proiectată conform modelului de arhitectură stratificată și ar trebui să utilizeze (minimum) următoarele clase:

• Clase de modele - reprezintă modelele de date ale aplicaţiei

• Clasele de Business Logic - conțin logica aplicației

• Clasuri de prezentare – clase legate de GUI

• Clase de acces la date - clase care conțin accesul la baza de date.

## 2.Diagrama UML

Text

Description automatically generated with low confidence

## 3. Teorie

Pentru a putea implementa aplicatia noastra, mai intai trebuie sa intelegem conceptul de baze de date relationale si modul prin care facem conexiunea intre aplicatia realizata si baza de date relationala.

O *bază de date relațională* este un tip de [bază de date](https://www.oracle.com/ro/database/what-is-database.html) care stochează și oferă acces la punctele de date care sunt legate între ele. Bazele de date relaționale se bazează pe modelul relațional, o modalitate intuitivă și simplă de reprezentare a datelor în tabele. Într-o bază de date relațională, fiecare rând din tabel este o înregistrare cu un cod unic numit *cheie*. Coloanele din tabel au atribute ale datelor și fiecare înregistrare are, de obicei, o valoare pentru fiecare atribut, facilitând stabilirea relațiilor dintre punctele de date.

## 4.Functionarea aplicatiei

Pentru a putea rula si obtine rezultatele in urma rularii aplicatiei trebuie mai intai sa ne alegem modulul in care dorim sa efectuam operatii: Client – adaugarea, editarea, stergerea unui client sau afisarea tabelului cu clienti; Produs - adaugarea, editarea, stergerea unui produs sau afisarea tabelului cu produse; Comanda – adaugarea unui produs in comanda si realizarea comenzii finale. Dupa ce trimitem o comanda, se va crea un fiiser text cu bonul respectiv si de asemenea, comanda va fi adaugata in fisierul text cu istoricul comenzilor.

## 5. Implementare

* Clasa Client: Aceasta clasa contine datele care mapeaza tabelul „client”. Contine ca field-uri toate coloanele din table, numele acestora fiind identice cu cu numele coloanelor din tabel pentru a se putea folosi thenica reflectiei in crearea obiectelor de tipul respectiv in urma unei interogari. Ca si metode existente in aceasta clasa sunt doar metoda de tipul setter si getter pentru a accesa variabilele de clasa. De asemenea, contine si metoda toString pentru a putea afisa informatiile despre un client.
* Clasa produs: Analog clasei Client, contine datele care mapeaza tabelul „produs”. Contine ca field-uri toate coloanele din table, numele acestora fiind identice cu cu numele coloanelor din tabel pentru a se putea folosi thenica reflectiei in crearea obiectelor de tipul respectiv in urma unei interogari. Ca si metode existente in aceasta clasa sunt doar metoda de tipul setter si getter pentru a accesa variabilele de clasa.
* Clasa comanda: Analog clasei Client, contine datele care mapeaza tabelul „comanda”. Contine ca field-uri toate coloanele din table, numele acestora fiind identice cu cu numele coloanelor din tabel pentru a se putea folosi thenica reflectiei in crearea obiectelor de tipul respectiv in urma unei interogari. Ca si metode existente in aceasta clasa sunt doar metoda de tipul setter si getter pentru a accesa variabilele de clasa.
* Clasa ComandaProdus: Analog clasei Client, contine datele care mapeaza tabelul „client”. Contine ca field-uri toate coloanele din table, numele acestora fiind identice cu cu numele coloanelor din tabel pentru a se putea folosi thenica reflectiei in crearea obiectelor de tipul respectiv in urma unei interogari. Ca si metode existente in aceasta clasa sunt doar metoda de tipul setter si getter pentru a accesa variabilele de clasa. De asemenea, contine si metoda toString pentru a putea afisa informatiile despre un un produs introdus intr-o anumita comanda.
* Clasa ConnectionFactory: Aceasta clasa are scopul de a returna o conexiune la baza de date si de a inchide unele componente folosite pentru a realiza o interogare. Clasa ConnectionFactory contine urmatoarele metode: createConnection (aceasta metoda este un privata, creaza o conexiune care mai apoi este returnata), getConnection (returneaza o conexiune la baza de date obtinuta prin apelarea metodei prezentate precedent). Aceasta clasa contine si un constructor care verifica existenta driver-ului in conectorul mysql. De asemenea, contine si metodele: close(Connection), close(Statement) si close(ResultSet), 3 metode supraincarcate care au rolul de a inchide obiectul primit ca argument.
* Clasa AbstractDAO: Aceasta clasa contine urmatoarele metode: creareInserare(Metoda care creeaza prin tehnica reflectiei query-ul necesar pentru a introduce date in tabela data de numele clasei obiectului care este primit ca argument, returneaza un StringBuilder care reprezinta query-ul), inserareTabel (Aceasta metoda insereaza in baza de date obiectul primit ca si argument care poate sa fie de tipul unuia din cele 3 clase care mapeaza baza de date. Query-ul este format prin apelarea creareInserare prezentate mai sus. Se returneaza un int care reprezinta un cod de eroare ce va fi folosit pentru a verifica tipul erorii pentru cazul in care se insereaza un produs deja existent, in cazul acesta se va face update la cantitatea produsului), creareStergere (Metoda care creeaza prin tehnica reflectiei query-ul necesar pentru a sterge date in tabela data de numele clasei obiectului care este primit ca argument, returneaza un StringBuilder care reprezinta query-ul), stergereTabel(Aceasta metoda sterge in baza de date obiectul primit ca si argument care poate sa fie de tipul unuia din cele 3 clase care mapeaza baza de date. Query-ul este format prin apelarea creareInserare prezentate mai sus. Se returneaza un int care reprezinta un cod de eroare ce va fi folosit pentru a verifica tipul erorii pentru cazul in care se insereaza un produs deja existent, in cazul acesta se va face update la cantitatea produsului), creareUpdate (Metoda care creeaza prin tehnica reflectiei query-ul necesar pentru a edita date in tabela data de numele clasei obiectului care este primit ca argument, returneaza un StringBuilder care reprezinta query-ul), update (Aceasta metoda editeaza in baza de date obiectul primit ca si argument care poate sa fie de tipul unuia din cele 3 clase care mapeaza baza de date. Query-ul este format prin apelarea creareInserare prezentate mai sus. Se returneaza un int care reprezinta un cod de eroare ce va fi folosit pentru a verifica tipul erorii pentru cazul in care se insereaza un produs deja existent, in cazul acesta se va face update la cantitatea produsului), creareaSelectare (Metoda care creeaza prin tehnica reflectiei query-ul necesar pentru a selecta date din tabela data de numele clasei obiectului care este primit ca argument si care are field-ul egal cu o valoare ceruta ulterior, returneaza un StringBuilder care reprezinta query-ul), gasireID (Aceasta metoda un obiect de tipul unuia din cele 3 clase care mapeaza baza de date . Query-ul este format prin apelarea creareSelectare prezentate mai sus.), afisareTabel (Va executa query-ul realizat in interiorul metodei, in urma caruia se va returna un JTable care ulterior va fi afisat in interfata in momentul apasarii unui anumit buton), afisareTabel2(Va executa query-ul realizat in interiorul metodei, in urma caruia se va returna un Jtable ce contine doar inregistrarile in care campul are valoarea ceruta drept al treilea argument si ulterior va fi afisat in interfata in momentul apasarii unui anumit buton), gasireIDplus1(returneaza id-ul maxim plus unu care va servi drept id pentru urmatorul obiect).
* Clasa ClientDAO: Aceasta clasa contine metodele: gasireID (Primeste ca argument o valoare int si returneaza Clientul ce are acel id), adaugare (Primeste ca argument un Client si il insereaza in baza de date, in tabelul client), stergere (Primeste ca argument un Client si id-ul sau si il sterge din baza de date, din tabelul client), update(Primeste ca argument un Client si id-ul sau si il editeaza in baza de date, in tabelul client). Toate aceste metode folosesc un String declarat la inceputul clasei.
* Clasa ProdusDAO: Aceasta clasa contine metodele: gasireID (Primeste ca argument o valoare int si returneaza Produs ce are acel id din tabelul produs), adaugare (Primeste ca argument un Produs si il insereaza in baza de date, in tabelul produs), stergere (Primeste ca argument un Produs si id-ul sau si il sterge din baza de date, din tabelul produs), update(Primeste ca argument un Produs si id-ul sau si il editeaza in baza de date, in tabelul produs). Toate aceste metode folosesc un String declarat la inceputul clasei.
* Clasa ComandaDAO: Aceasta clasa contine metodele: gasireIDplus1 (returneaza id-ul maxim plus unu pentru a fi folosit la inserarea unei noi comenzi din tabelul comanda), adaugare (Primeste ca argument un Comanda si il insereaza in baza de date, in tabelul comanda). Toate aceste metode folosesc un String declarat la inceputul clasei.
* Clasa ComandaProdusDAO: Aceasta clasa contine metodele: gasireID(Primeste ca argument o valoare int si returneaza Clientul ce are acel id), iadaugare(Primeste ca argument un Client si il insereaza in baza de date, in tabelul client), stergere (Primeste ca argument un Client si id-ul sau si il sterge din baza de date, din tabelul client), update(Primeste ca argument un Client si id-ul sau si il editeaza in baza de date, in tabelul client). Toate aceste metode folosesc un String declarat la inceputul clasei.
* Clasele din validators: Aceste clase implementeaza metoda validate din interfata Validator in care se verifca daca sunt bune informatile referitoare la un client, produs sau o comanda. Acestea vor fi apelate in restul claselor din pachetul bll.
* Clasa ClientBLL: Aceasta clasa implementeaza urmatoarele metode: adaugareClient(Primeste ca un Client, creeaza un obiect de tipul Client, mai apoi apeleaza metoda inserareTabel din ClientDAO cu clientul respectiv ca argument), gasireID(Primeste ca argument un int, creeaza un obiect de tipul Client, mai apoi apleaza metoda gasireID cu valoare respectiva si String-ul „client” ca argumente din clasa ClientDAO) , stergereClient (Identic cu adaugareClient, creaza un obiect de tipul Client din argumentele primite si apeleaza metoda de stergere din ClientDAO), updateClient (Identic cu adaugareClient, creaza un obiect de tipul Client din argumentele primite si apeleaza metoda de update din ClientDAO), gasireIDplus1 (Creeaza un obiect de tipul Client, mai apoi apeleaza metoda findMaxIdPlus din clasa ClientDAO) si afisareClient(Primeste ca argument un table de tipul Jtable in care introduce informatiile de afisat, returnand astfel tot un Jtable, apeland metoda afisareTabel cu tabelul dat ca argument).
* Clasa ProdusBLL: Aceasta clasa implementeaza urmatoarele metode: adaugareProdus (Primeste ca un Produs, creeaza un obiect de tipul Produs, mai apoi apeleaza metoda adaugareTable din ProdusDAO cu Produsul respectiv ca argument), gasireID(Primeste ca argument un int, creeaza un obiect de tipul Produs, mai apoi apleaza metoda gasireID cu valoare respectiva si String-ul „Produs” ca argumente din clasa ProdusDAO) , stergereProdus (Identic cu adaugareProdus, creaza un obiect de tipul Produs din argumentele primite si apeleaza metoda de stergere din ProdusDAO), updateProdus (Identic cu adaugareProdus, creaza un obiect de tipul Produs din argumentele primite si apeleaza metoda de update din ProdusDAO), gasireIDplus1 (Creeaza un obiect de tipul Produs, mai apoi apeleaza metoda findMaxIdPlus din clasa ProdusDAO) si afisareProduse (Primeste ca argument un table de tipul Jtable in care introduce informatiile de afisat, returnand astfel tot un Jtable, apeland metoda afisareTabel cu tabelul dat ca argument).
* Clasa ComandaBLL: Aceasta clasa implementeaza urmatoarele metode: adaugareComanda (Primeste ca un Comanda, creeaza un obiect de tipul Comanda, mai apoi apeleaza metoda adaugareTable din ComandaDAO cu comanda respectiv ca argument) si gasireID (Creeaza un obiect de tipul Comanda, mai apoi apeleaza metoda gasireIDplus1 din clasa ComandaDAO) .
* Clasa ComandaProdusBLL: Aceasta clasa implementeaza urmatoarele metode: adaugareComanda (Primeste ca un ComandaProdus, creeaza un obiect de tipul ComandaProdus, mai apoi apeleaza metoda adaugareTable din ComandaProdusDAO cu obiectul ComandaProdus respectiv ca argument), gasireID (Primeste ca argument un int, creeaza un obiect de tipul ComandaProdus, mai apoi apleaza metoda gasireID cu valoare respectiva si String-ul „ComandaProdus” ca argumente din clasa ComandaProdusDAO) si afisareIdComanda(Primeste ca argument un table de tipul Jtable in care introduce informatiile de afisat si o valoare int, returnand astfel tot un JTable, apeland metoda afisareTabel2 cu tabelul dat ca argument).
* Clasa ClientPres

In TextField-ul corespunzator id-ului, se va afisa automat urmatorul id pentru un client nou de inserat ( acesta fiind auto-increment ). La apasarea butonului „Afisare”, vom putea vedea toti clientii din baza noastra de date. Mai mult, cand vom selecta un client din tabel, informatiile acestuia se vor gasi detaliate in TextField-urile corespunzatoare pentru a fi mai usor sa il stergem sau sa il editam. Dupa o astfel de operatie, tabelul din partea dreapta a ecranului se reincarca actualizat.

Graphical user interface, application

Description automatically generated

* Clasa ProdusePres

In TextField-ul corespunzator id-ului, se va afisa automat urmatorul id pentru un produs nou de inserat ( acesta fiind auto-increment ). La apasarea butonului „Afisare”, vom putea vedea toate produsele din baza noastra de date. Mai mult, cand vom selecta un produs din tabel, informatiile acestuia se vor gasi detaliate in TextField-urile corespunzatoare pentru a fi mai usor sa il stergem sau sa il editam. Dupa o astfel de operatie, tabelul din partea dreapta a ecranului se reincarca actualizat.

Graphical user interface, application

Description automatically generated

* Clasa ComenziPres

Din cele 2 comboBox-uri vom alege produsele pe care vrem sa le introducem in comanda ( impreuna cu cantitatea specificata mai jos in TextField) si clientul caruia ii atribuim comanda, identificat prin id, nume, adresa si mail pentru a evita cazurile in care ar exista mai multi clienti cu acelasi nume/adresa.

In momentul selectarii butonului „Adauga produs”, produsul specificat impreuna cu cantitatea vor fi afisate in tabelul de sub buton. Pentru a incheia comanda, va trebui sa apasam pe „Finalizare”, moment in care tabelul se va goli, fiind gata pentru urmatoarea comanda. Totodata, se realizeaza/actualizeaza si cele 2 fisiere text, si anume: bonul corespunzator acesti comenzi si istoricul comenzilor.

Graphical user interface, application

Description automatically generated

* Clasa View

Graphical user interface, application, Word

Description automatically generated

* Clasa Afisare

Aceasta clasa ne ajuta sa printam rezultatele obtinute in fisierele text dorite, dat ca parametru in functia writeToFile(String fisier). Astfel, in momentul in care apasam pe butonul „Finalizare”, in fisierul „Istoric bonuri.txt” va afarea la sfarsit ultima comanda. Totodata, se va crea si fisierul „Bon comanda (idComenzii) (numele Clientului).txt” in care va fi printat bonul comenzii date de catre un anumit client.

## 6.Rezultate

Graphical user interface, application

Description automatically generatedGraphical user interface, text, application

Description automatically generated