Universitatea Tehnica din Cluj-Napoca

Facultatea de Automatica si Calculatoare

Calculatoare

Tehnici de Programare



Order Management

Student Balog Helga

Grupa 30227

Cuprins

[1.Cerințe Funcționale 2](#_Toc1103078120)

[2.Obiective 2](#_Toc761739151)

[2.1 Obiective principale 2](#_Toc1500434350)

[2.2 Obiective secundare 2](#_Toc2075516221)

[3. Analiza Problemei 2](#_Toc1645510810)

[4. Proiectare și Implementare 3](#_Toc489900725)

[4.1 Poiectare 3](#_Toc885360588)

[4.2 Pachete și Clase 4](#_Toc546761405)

[5. Funcționare 9](#_Toc1233470829)

[6. MySQL Workbench 11](#_Toc255672874)

[7. Concluzii și Dezvoltări ulterioare 11](#_Toc2068696857)

[8. Bibliografie 12](#_Toc2058879665)

# 1.Cerințe Funcționale

Proiectați și implementați o aplicație care se ocupă de managementul unui depozit. Aplicația trebuie să lucreze cu baza de date care conține tabele pentru produse, clienți și comenzi și trebuie să ofere posibilitatea de a adăuga, a edita, a șterge și de a vizualiza entitățile de mai sus. Toate aceste acțiuni trebuie implementate pe o interfață grafică. De asemenea, la o comanda cu o cantitate care depășește stocul, utilizatorul trebuie anunțat că nu sunt suficienta produse.

În aplicație, trebuie folosită metoda de Layered Architecture, adică să aibă trei straturi: de vizualizare, de logică și de comunicare cu baza de date, și, de asemenea, să folosească metoda de reflexie.

# 2.Obiective

## 2.1 Obiective principale

Obiectivul principal este de a avea o interfață prin care utilizatorul poate alege cu ce entitate să lucreze ( client, produs, comenzi) și ce operație să aplice pe aceste entități: de adăugare, de editare, de ștergere sau de vizualizare.

## 2.2 Obiective secundare

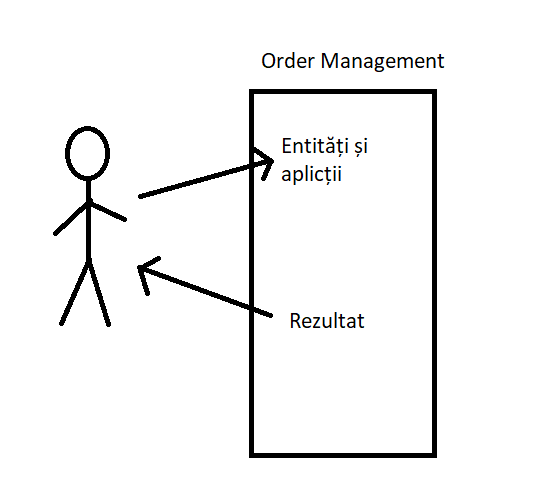
Pentru a atinge obiectivul principal propus, este necesar să satisfacem și anumite obiective secundare, cum ar fi: scrierea corecte a interogărilor pentru a adăuga, a edita , a șterge, respectiv, de a vizualiza datele dintr-un tabel, implementarea interfețelor cu care va lucra utilizatorul, imprimarea unei comenzi în momentul în care se face, definirea corectă a câmpurilor la fiecare entitate, astfel încât să corespundă cu coloanele definite în baza de date, folosirea corectă a reflexiei,...etc.

# 3. Analiza Problemei

Acest proiect susține următoarea scenă de utilizare: persoana care folosește aplicația va alege una dintre entități (client, product, order). Mai departe va alege ce acțiuni să aplice pe acea entitate. Dacă dorește să introducă o entitate nouă în tabel, va trebui să completeze câmpurile ce apar pe interfață.

Dacă dorește să editeze, este necesar să introducă și id-ul clientului pa care vrea să aplice schimbările. La ștergere este de ajuns să introducă un id, iar la vizualizare este suficient să apese pe buton. Pentru a lucra și cu o altă entitate, poate să se întoarcă la interfața principală și să selecteze din nou ceea ce dorește.

În imaginea de mai jos se poate observa diagrama de utilizare.



# 4. Proiectare și Implementare

## 4.1 Poiectare

Aplicația ‘Order Management’, descris în limbajul java, are o arhitectură de tip Layered, adică are trei straturi semnificative. Primul strat, cel mai e sus, este cel de vizualizare, care se ocupă de realizarea interfețelor grafice și de comunicarea dintre interfețe cu restul proiectului.

Al doilea strat este cel de logică, adică Controllerul va transmite datele citite la acest strat unde se va decide ce se face mai departe și cum. De exemplu: la primirea unei comenzi clasa care se ocupa de logica pentru comenzi va verifica dacă sunt destule produse în stoc, dacă da atunci va introduce comanda, altfel va avertiza utilizatorul.

Al treilea strat, cel mai de jos, se ocupă cu accesarea datelor din baza de date. În pachetul respectiv se vor crea interogările, prin Reflection, pentru acțiunea primită. De exemplu, dacă această clasă trebuie să insereze în tabelul Client, va completa câmpurile cu cele de la entitatea client.

## 4.2 Pachete și Clase

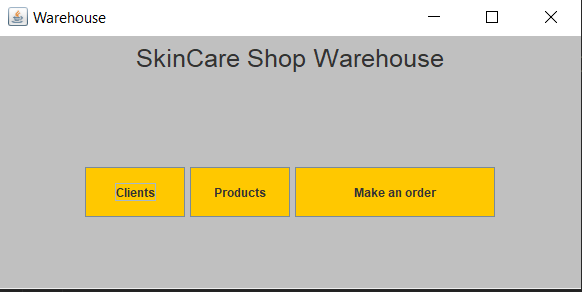
Pachetul Shop

Pachetul Shop are două clase, clasa Start și clasa Reflection. Clasa Start, conține metoda Main, din care pornește aplicația propriu-zisă, iar clasa Reflection se ocupă de generarea tabelului, în caz că butonul de vizualizare este apăsat. Tabelele se realizează prin metoda de reflexie, adică, printr-o listă de obiecte, de tip Client, Produs sau Order, primită, se analizează câmpurile declarate pentru acel obiect și se iau aceste câmpuri cu valorile respective. De exemplu, pentru o listă de obiecte de tip client, câmpurile declarate sunt id, nume, email și address. Se iau aceste câmpuri, care vor fi numele coloanelor din tabel și se populează cu datele luate tot din acea listă de obiecte.

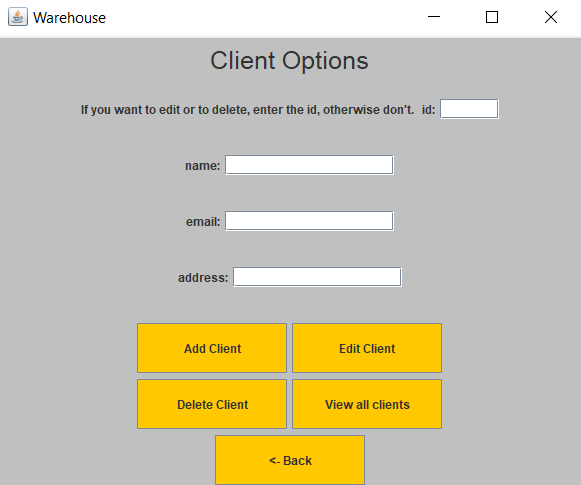
Pachetul Presentation

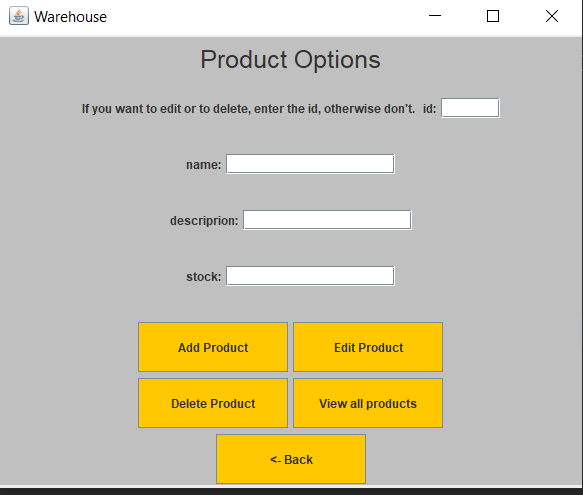
Acest pachet reprezintă primul strat din arhitectura de tip Layered al aplicației, astfel , cum s-a precizat și mai sus , el se ocupă de interfețele grafice ale programului și de comunicarea dintre interfețe si program. Pachetul are două clase, View și Controller. Prima clasă se ocupă de construirea interfețelor, în timp ce a doua de comunicarea dintre interfață și program.

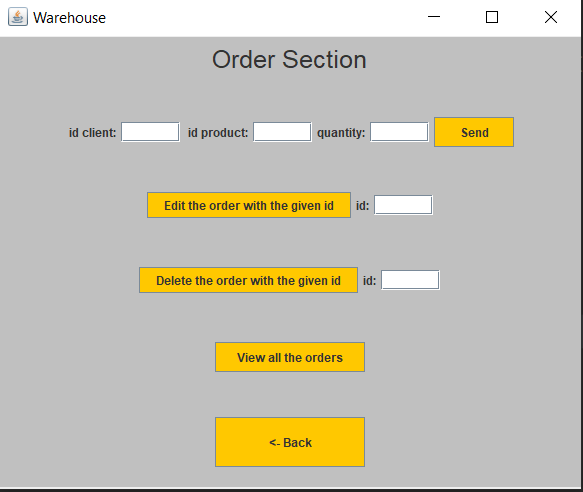
Mai jos se pot observa interfețele grafice ale programului.



Programul are patru interfețe: interfața principală, de unde se alege entitatea cu care se lucrează, interfața care se ocupă de manipularea clienților din tabel, interfața care se ocupă de manipularea produselor și interfața care se ocupă de comenzi. Din ultimele trei interfețe putem reveni la cea principală apăsând butonul intitulat ‘ <- Back ‘.







Pachetul Model

Pachetul Model, deține trei clase: Client, Product și Order. Aceste clase reprezintă entitățile din baza de date cu care lucrăm. Mai precis, în baza de date o să avem trei tabele, un tabel pentru clienți, un tabel pentru produse și un tabel pentru comenzi. Coloanele acestor tabele o să corespundă cu variabilele declarate în clasele din proiect, adică tabelul Client din baza de date are 4 coloane: id, name, email și address, astfel clasa Client din proiect, la fel va fi definită cu aceste câmpuri.

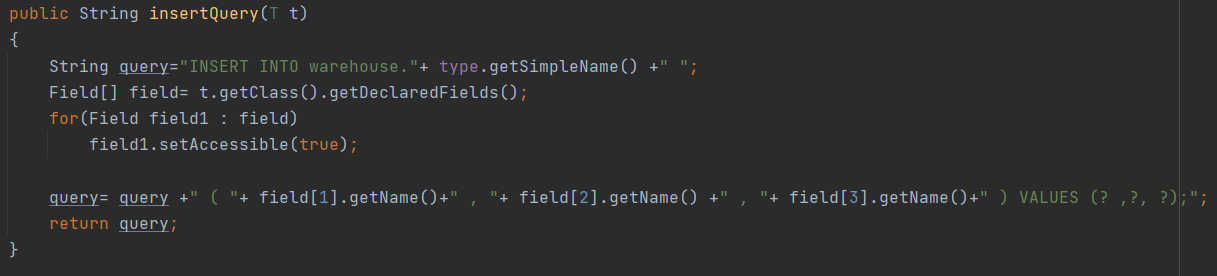
Pachetul DataAccess

Acest pachet corespunde cu cel mai de jos strat din arhitectura folosită. Astfel acesta se ocupă de realizarea conexiunii cu baza de date, de formarea interogărilor și de aplicarea acțiunilor propuse.

Pachetul are cinci clase importante. Prima clasa, ConnectionFactory, face legătura cu baza de date.

Aceasta primește numele utilizatorului (root), parola, numele bazei de date și alte informații necesare pentru conectare.Clasa, are mai multe metode printre care și craeteConnection, pentru a crea conexiune, getConnection, pentru a obține conexiunea și close, pentru a închide conexiunea realizată.

Clasa AbstractDAO<T> se ocupă cu cele menționate la introducerea pachetului din care face parte. Aceasta este generică, adică metodele se pot apela pe mai multe entități diferite. Metodele din clasă efectuează inserarea, editarea, ștergerea, căutarea după un id dat și obținerea tuturor datelor dintr-un anumit tabel. Faptul că toate aceste aplicații pot efectuate pe cele trei entități din proiect se datorează utilizării metodei de Reflection ( metoda generala, nu cea din pachetul shop). Mai departe se va exemplifica metoda de inserare, pentru o înțelegere mai bună.





Metoda insertQuery construiește șirul de caractere pentru comanda de inserare în baza de date și o returnează. Aceasta pornește cu un șir deja construit pe jumătate: ‘ INSERT INTO warehouse.’ care se concatenează cu ce se returnează după ce se apelează type.getName(). Metoda menționată anterior returnează tipul pe care s-a apelat inserarea (Client, Product, Order), în baza de date având tabele cu același nume și aceleași câmpuri. Mai departe se iau câmpurile declarate pentru acel tip, de exemplu pentru Order se vor lua câmpurile orderID, clientID, productID și quantity. În următoarea etapă completăm șirul de caractere cu câmpurile obținute. Duă VALUES avem trei semne de întrebare. Acestea vor fi înlocuite cu valorile corespunzătoare din metoda insert. La final se returnează șirul de caraceter obținut.

Metoda insert, în primul rând, obține conexiunea la baza de date, după care, înlocuiește semnele de întrebare din șirul returnat de insertQuery. Acest fapt se realizează tot prin metoda de Reflection, dar acuma se iau valorile câmpurilor, nu numele. De exemplu, dacă inserăm un client , field[1].get(t) va fi valoarea introdusă în interfață în câmpul reprezentativ pentru nume. Mai departe, se execută interogarea și se închide legătura la baza de date.

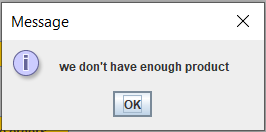
La fel se procedează și la celelalte acțiuni și entități.

Clasele ClientDAO, ProductDAO și OrderDAO, extind clasa generică AbstractDAO. Astfel la inserare, editare, ștergere și vizualizare se vor apela metodele din AbtrsactDAO, cu toate că folosim clasele care o extind pe aceasta. Variabila type va lua tipul pe care am apelat metoda, de exemplu la clientDAO.isnert(...) , type va reprezenta clasa Client, fiind astfel asigurată corectitudinea interogărilor.

Pachetul businessLogic

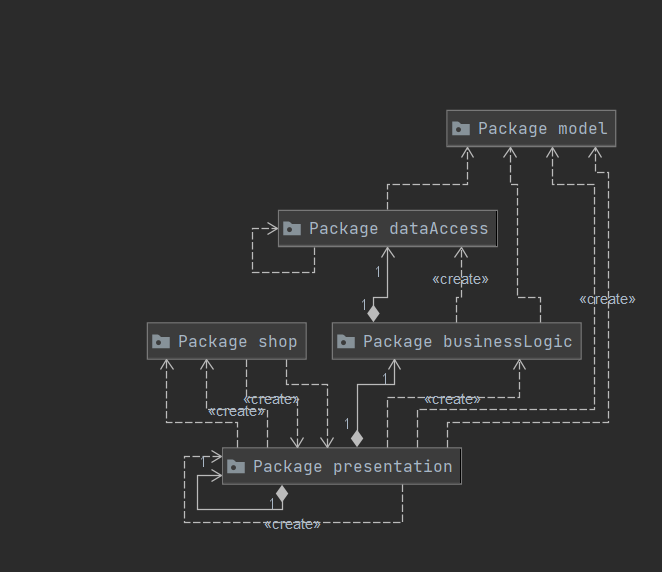
Acest pachet reprezintă stratul din mijloc din arhitectura folosită și se ocupă de logica din spatele acțiunilor. Avem trei clase în pachet: clientBll, productBll și OrderBll. În aceste clase se apelează pe clasele DAO corespunzătoarele acțiunile cerute și se efectuează anumite teste.

De exemplu, la orderBll, când dorim să efectuăm o comandă se testează prima dată dacă avem destule produse în stoc. În caz negativ, utilizatorul va fi informat de această situație, cum se vede în imaginea de mai jos.



De asemenea, dacă sunt suficiente produse, se va tipării o factură cu numele clientului, numele produsului și cantitatea de produs cerută.

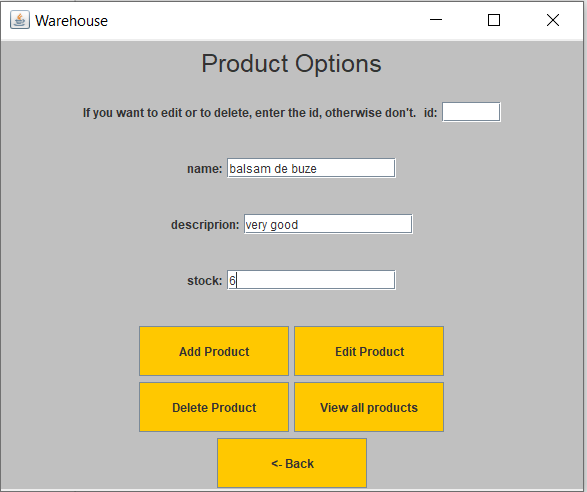
În imaginea de mai jos, se poate observa o diagramă UML pentru pachete:

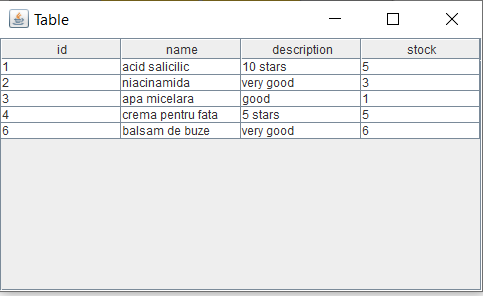


# 5. Funcționare

Pentru a înțelege funcționarea vom lua ca și exemplu un produs. Pe interfața corespunzătoare vom introduce datele dorite, de exemplu: nume: balsam de buze, description: very good, stock: 6, după care vom apăsa pe butonul de Add Prodcut. După ce am apăsat butonul de inserare, ne putem verifica apăsând butonul de View All Products.

În prima imagine de mai jos, se ilustrează datele introduse corect și butoanele pe care trebuie să apăsăm.



Tabelul care se generează după apăsarea butonul de vizualizare este următorul.  


După cum se observă, id-ul este incrementat automat, opțiune care trebuie aleasă în momentul în care creăm tabelele în MySql Workench.

Pentru a edita un produs, trebuie să introducem și id-ul produsului pe care dorim să îl edităm. De exemplu, putem introduce id-ul 6 și să schimbăm stocul. Pentru aceasta, trebuie să păstrăm câmpurile pe care nu dorim să le schimbăm la fel. Astfel, vom completa din nou toate câmpurile cu aceleași date ca și mai sus, dar vom schimba valoarea introdusă la câmpul reprezentativ pentru stoc.

Pentru a șterge un produs, este suficient să introducem id-ul acestuia. De exemplu, dacă introducem id-ul 2, se va șterge produsul cu numele niacinamidă. Chiar dacă se vor introduce anumite informații în câmpuri, acestea nu vor fi luate în considerare. De asemenea, când se șterge un id, care nu este ultimul din tabel, de exemplu dacă se șterge din tabelul de mai sus rândul cu id-ul 3, restul id-urilor nu se vor actualiza, la introducere aplicația va incrementa ultimul id din tabel.

Pentru o funcționare cât mai corectă , este important să introducem datele corect în câmpuri.

# 6. MySQL Workbench

În această aplicație, se lucrează cu baze de date, de aceea aplicația aleasă ca și suport este MySQL Workbench. În momentul în care rulăm programul nostru, aplicația trebuie să fie pornită și utilizabilă, adică să trecem de introducerea parolei. Numele bazei de date trebuie să fie la fel ca și cel introdus în clasa ConnectionFactory, la fel și numele tabelelor cu câmpurile trebuie să corespundă cu entitățile din program. Numele bazei de date folosită de această aplicație este warehouse, cu următoarele tabele: Client, Product și Order. În momentul în care dorim să verificăm daca într-adevăr s-au introdus corect datele, rulăm o interogare în MySQL Workbench. Pentru aceasta avem nevoie de minime cunoștințe în limbajul de programare SQL.

# 7. Concluzii și Dezvoltări ulterioare

În final, aplicația de Order Management îsî atinge atât obiectivele secundare cât și pe cel principal. Programul îndeplinește toate condițiile și implementează toate cerințele puse.

Ca și dezvoltări ulterioare, s-ar putea adăuga anumite teste pentru a verifica corectitudinea datelor introduse. De exemplu, la client la email, am putea impune ca acesta să conțină cel puțin caracterul @. La fel când edităm un client/ produs/ comandă, am putea verifica dacă id-ul introdus apare în baza de date sau nu. De asemenea la editare, am putea implementa metoda astfel încât, să schimbăm doar anumite câmpuri, deoarece, în momentul de față , trebuie completate toate câmpurile pentru a edita în mod corect. De exemplu dacă nu se introduce nimica la câmpul de nume, aplicația ori ne va afișa o eroare, ori ne va pune în baza de date un șir de caractere gol.

Tot la testări, am putea verifica, dacă, informațiile introduse nu apar deja în baza de date. Dacă se introduc aceleași date de două ori, aplicația va insera cu un alt id, ceea ce nu este destul de eficient.

# 8. Bibliografie

1. <https://www.w3schools.com/sql/default.asp> (20.04.2021)
2. <https://www.oracle.com/technical-resources/articles/java/javareflection.html> (20. 04.2021)
3. <https://docs.oracle.com/javase/tutorial/uiswing/components/table.html> (25.04.2021)
4. <https://www.jetbrains.com/help/idea/working-with-code-documentation.html> (25.04.2021)
5. <https://en.wikipedia.org/wiki/Javadoc> (25.04.2021)
6. <https://www.oreilly.com/library/view/software-architecture-patterns/9781491971437/ch01.html> (21.04.2021)