Aplicatie de management comenzi

Universitatea Tehnică Cluj-Napoca

Facultatea de Calculatoare și Automatică

Departamenul de Calculatoare și Tehnologia informației

Nume: Camară Aida

Grupa: 30228

Materie: Tehnici de programare

Contents

[1. Cerinta 3](#_Toc480414474)

[2. Obiectivul temei 3](#_Toc480414475)

[3. Analiza problemei 4](#_Toc480414476)

[4.Proiectare și implementare 5](#_Toc480414477)

[5.Testare 10](#_Toc480414478)

[6. Concluzie și dezvoltări ulterioare 14](#_Toc480414479)

[7. Bibliografie 15](#_Toc480414480)

# 1. Cerinta

Consider an order management application for processing customer orders for a warehouse. Relational databases are used to store the products, the clients and the orders. Furthermore, the application uses (minimally) the following classes:

1. Domain specific classes: Order, Customer and Product

2. Business Logic (warehouse-specific processing) classes: OrderProcessing, WarehouseAdministration, ClientAdministration

3. Presentation classes: GUI related classes

4. Data access classes: Database access related classes

Other classes and packages can be added to implement the full functionality of the application.

Requirements

a. Analyze the proposed application, determine the structure and behavior of its classes and draw an extended UML class diagram.

b. Design, implement and test the application classes. Use javadoc for documenting classes.

c. Define, design and implement a system of utility programs (examples: reports for under-stock, totals, filters, etc.).

d. Design and implement a comprehensive demo driver for the order management application.

# 2. Obiectivul temei

Aplicatia are ca obiectiv familiarizarea cu lucrul cu baze de date, combinand limbajul MySQL si Java. In present bazele de date se regasesc in spatele oricarui program, de aceea este important sa stim sa dezvoltam un program care realizeaza o conexiune cu o baza de date si ofera posibilitatea sa se faca operatii pe aceasta, de exemplu: cautarea dupa ID, inserarea, stergerea si actualizarea si vizualizarea tuturor inregistrarilor dintr-un tabel.

# 3. Analiza problemei

În partea de analiză a problemei trebuie depistate principalele clase, respectiv caracteristicile și funcționlitățile acestora și realizate legăturile dintre ele. Programarea orientata pe obiecte oferă avantajul de a putea începe dezvoltarea unui proiect folosind doar informațiile de la suprafață, fără a fi nevoie de implementarea efectivă a funcționalităților. Această strategie este cunoscută sub numele “Top-Down”. Ea este foarte avantajoasă din punctul de vedere al găsirii componentelor constituente, deoarece pot fi găsite, relativ ușor, structuri cu o legătură directă în lumea reală( obiecte, acțiuni etc.). Din păcate această versatilitate vine cu prețul complexității, ea crescând spre măsură ce se avansează pe nivelele inferioare.

De cele mai multe ori se pornește de la specificația proiectului, căutându-se:

• Substantive, care devin eventuale clase candidat

• Verbe ce ar putea juca rolul metodelor din clasă.

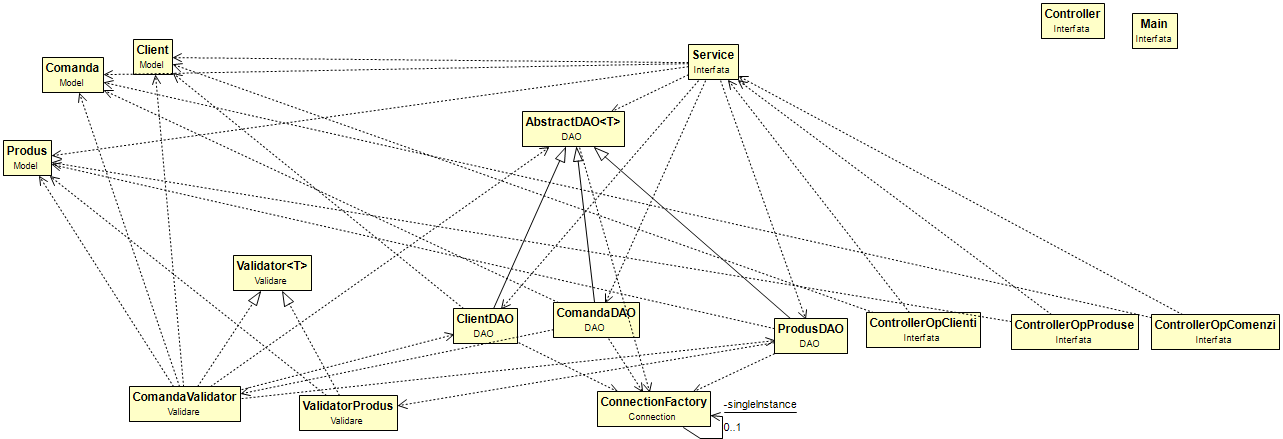
Astfel se formează o vedere în ansamblu despre cum va arăta proiectul final la suprafață, ulterior urmând să se realizeze implementarea detaliată a fiecărei metode.

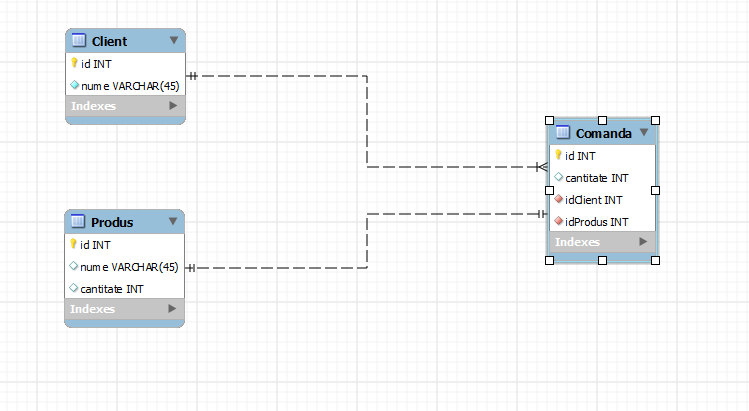
Scopul principal al aplicatiei este acela de a realiza comenzi de produse de catre clienti. Utilizatorul aplicatiei trebuie sa aleaga un client din baza de date, un produs si sa introduca numarul de bucati pe care vrea sa le comande. Odata facuta o comanda, se genereaza o factura sub forma unui fisier text, iar stocul de produse scade.

Am folosit o arhitectura pe nivele, organizand proiectul pe mai multe pachete care comunica intre ele. Astfel, aplicatia are un pachet care se ocupa de interfata grafica cu utilizatorul, pachetul DAO care contine operatiile efective pe baza de date, pachetul validare care asigura faptul ca datele introduse sau actualizate sunt corecte, pachetul connection care realizeaza efectiv conexiunea la baza de date si pachetul model unde se gasesc obiectele pe care se lucreaza, tebelele bazei de date.

In cazul in care se introduc valori gresite, se aplicatia arunca exceptii ce afiseaza un mesaj sugestiv.

# 4.Proiectare și implementare

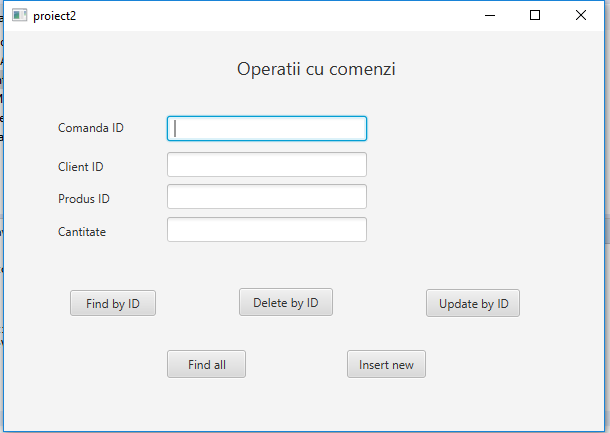
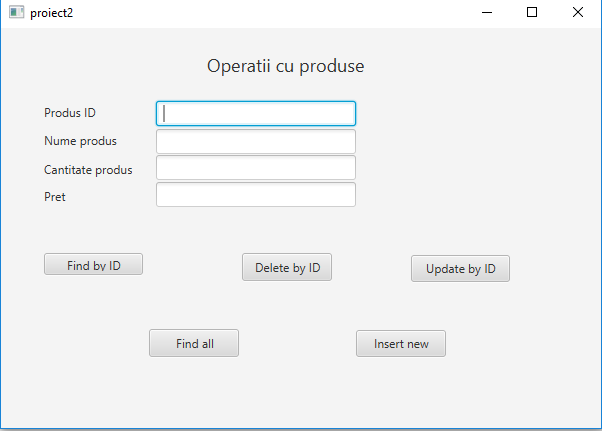
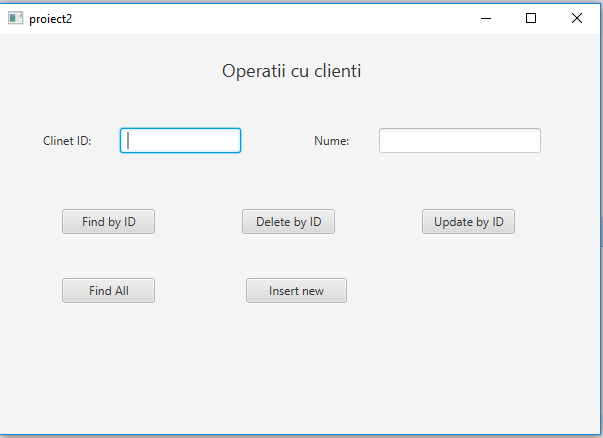
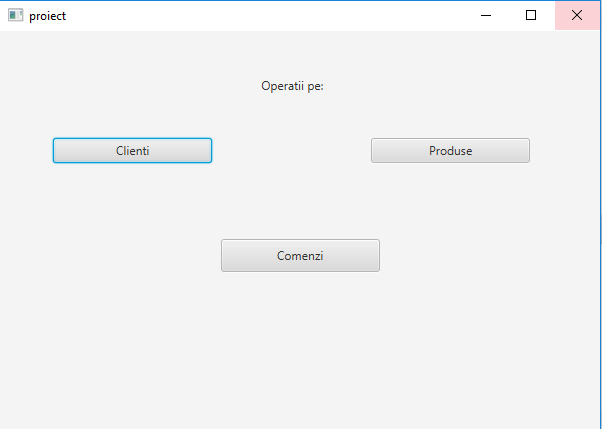




**Interfața grafică** este realizată în programul JavaFX Scene Builder 1.1. Acesta generează un fișier .fxml care este introdus în proiect prin oferirea ca parametru a căii către fișier în constructorul unui obiect de tip FileInputStream. ActionEvent-urile ale fiecărui buton sunt setate în JavaFX Scene Builder prin introducerea în câmpul “On Action” numele metodei din clasa Controller care trebuie apelată. La fel, cotroller-ul anchor panel-ului trebuie setat în câmpul “Controller Class” și id-ul Text field-urilor, în câmul “fx:id”.

Aplicatia are mai multe clase de tip Controller deoarece contine o fereastra principala de unde se mai pot deschide inca trei ferestre care corespund tipului de operatii care se vor face: operatii pe client, pe produse sau pe comenzi.

In fereastrele de operatii se gasesc campuri, care sunt de fapt coloanele tabelelor si cinci butoane reprezentand operatiile implementate in proiect. Fiecare buton are asociata o metoda, care se apeleaza atunci cand butonul este apasat. In metodele de gasire (findAll si findById) se creeaza o fereastra noua in care se afiseaza datele gasite in baza de date.



Pachetul Model contine trei clase: Client, Produs si Comanda. Acestea sunt obiecte de tip POJO, contin doar variabile de instanta si settere si gettere. De asemenea, se folosesc metode de get pentru variabilele de instanta, dar care returneaza obiecte de tip SimpleStringProperty. Acestea se folosesc pentru afisarea in tabel a valorilor din baza de date.

Pachetul DAO contine patru clase: AbstractDAO<T>, ClientDAO, ProdusDAO si ComandaDAO. Clasa AbstractDAO<T> este generic si contine metodele de findById, findAll si deleteById. Operatiile implementate aici se bazeaza pe reflection pentru a extrage campurile modelelor, de aceea pot fi apelate pentru toate trei modelele. Principiul acestor metode este urmatorul: se face conexiunea la baza de date, se incarca interogarea portivita, se inlocuiesc semnele “?” cu parametrii functiei si se executa interogarea.

Sunt implementate trei metode pentru a crea interogarile MySQL necesare pentru operatiile descrise:

**private** String createSelectQuery(String field){

StringBuilder sb=**new** StringBuilder();

sb.append("SELECT ");

sb.append(" \* ");

sb.append(" FROM ");

sb.append(type.getSimpleName());

sb.append(" WHERE "+field+" =?");

**return** sb.toString();

}

**private** String createSelectAllQuery(){

StringBuilder sb=**new** StringBuilder();

sb.append("SELECT ");

sb.append(" \* ");

sb.append(" FROM ");

sb.append(type.getSimpleName());

**return** sb.toString();

}

**private** String createDeleteQuery(String field){

StringBuilder sb=**new** StringBuilder();

sb.append("DELETE ");

sb.append(" FROM ");

sb.append(type.getSimpleName());

sb.append(" WHERE "+field+" =?");

**return** sb.toString();

}

Obiectele care rezulta in urma interogarii bazei de date sunt create cu metoda List<T> createObjects(ResultSet resultSet) . Astfel, se realizeaza o noua instant de tipul T, se acceseaza fiecare camp al clasei si prin invocarea metodelor de get si set se creeaza un obiect echivalent cu cel din rezultat. Dupa ce se creeaza astfel obiectul, el este adaugat la o lista, care va fi returnata.

Clasele ClientDAO, ProdusDAO si ComandaDAO extind clasa AbstractDAO si contin celelalte doua operatii posibile pe baza de date: inserarea unui rand nou si actualizarea unui rand din tabel. Implementarea operatiilor in fiecare clasa este asemanatoare, fiind diferite coloanele tabelelor si pentru ca nu s-a mai folosit strategia reflection a fost nevoie sa se faca aceste metode pentru fiecare model in parte. Se foloseste un String pentru interogarea in MySQL, cu observatia ca valorile efective care trebuie introduse in tabel sunt inlocuie cu “?”. In metode, se face conexiunea la baza de date, se incarca interogarea portivita, se inlocuiesc semnele “?” cu parametrii functiei si se executa interogarea. In final se inchide conexiunea.

In pachetul Validare, se afla o interfata care contine doar metoda validate. Clasele care implementeaza interfata sunt ComandaValidator si ValidatorProdus. Validarea unui produs consta in verificarea ca pretul si cantitatea intrduse sa fie strict positive. Ca o comanda sa fie valida este necesar ca atat clientul care face comanda cat si produsul comandat sa existe. De asemenea cantitatea de produse comandate trebuie sa fie strict pozitiva. Daca aceste conditii sunt indeplinite, urmatoarea verificare care se face este ca stocul de produse comandate sa fie indeajuns de mare. In caz contrar se afiseaza un mesaj de avertizare.

Pachetul Connection are o singura clasa ConnectionFactory care prin metoda createConnection realizeaza conexiunea la baza de date face legatura dintre baza de date facuta in programul MySQL Workbench si aplicatia Java implementata.

**private** Connection createConnection(){

Connection connection=**null**;

**try** {

connection=DriverManager.*getConnection*(***DBURL***,***USER***,***PASS***);

} **catch** (SQLException e) {

***LOGGER***.log(Level.***WARNING***, "Eroare la conectare la baza de date");

e.printStackTrace();

}

**return** connection;

}

In aceasta clasa mai sunt implementate metode de inchidere a conexiunii, a rezultatului si a Statement-ului, fiind folosit principiul de supraincarcare.

**public** **static** **void** close(Connection connection){

**if**(connection!=**null**){

**try**{

connection.close();

}**catch**(SQLException e){

***LOGGER***.log(Level.***WARNING***, "Eroare la inchiderea conexiunii");

}

}

}

**public** **static** **void** close(Statement statement){

**try** {

statement.close();

} **catch** (SQLException e) {

***LOGGER***.log(Level.***WARNING***, "Eroare la inchiderea statement ului");

}

}

**public** **static** **void** close(ResultSet resultSet){

**try** {

resultSet.close();

} **catch** (SQLException e) {

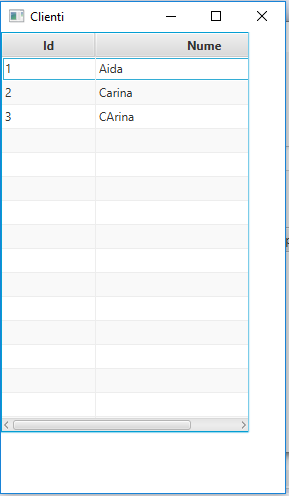
***LOGGER***.log(Level.***WARNING***, "Eroare la inchiederea rezultatului");

}

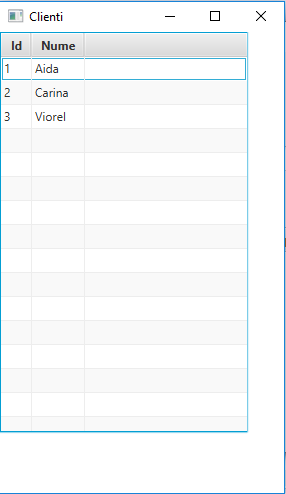
}

# 5.Testare

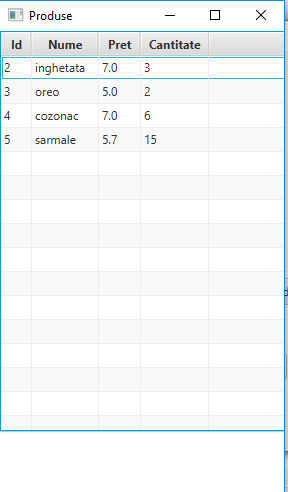
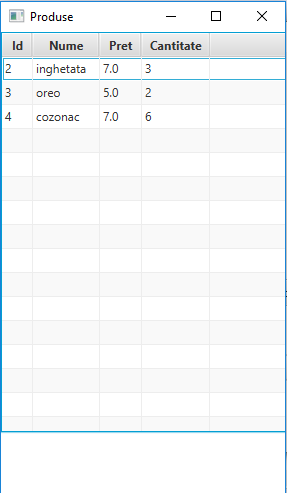
Dupa apasarea butonului Find All pentru clienti



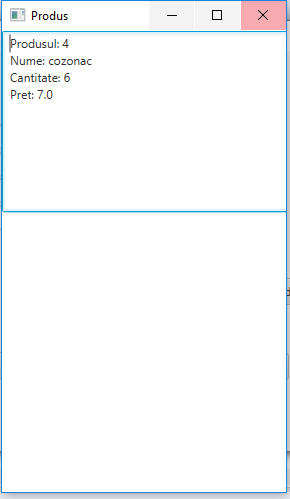
Dupa ce am realizat o actualizare a clientului cu id-ul 3



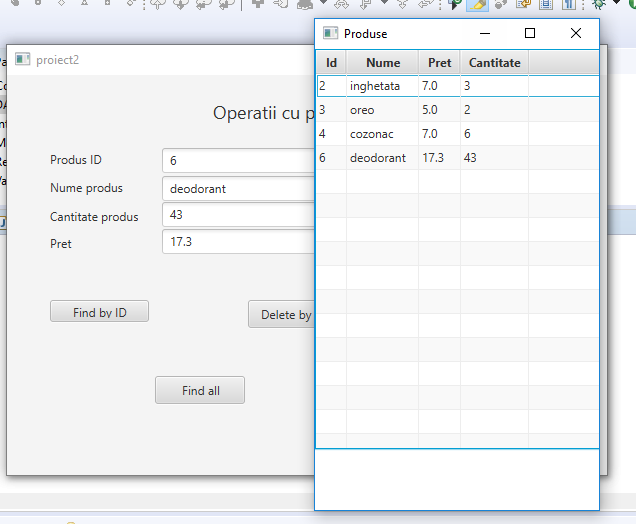
Afisarea tutror produselor si stergerea produsului cu id-ul 5

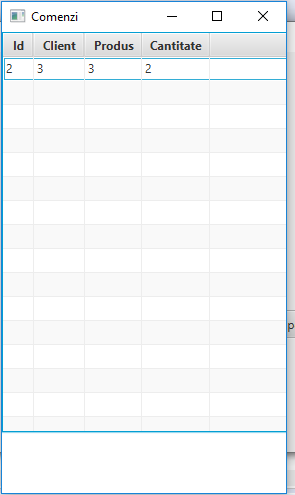
Gasirea produsului cu id-ul 4



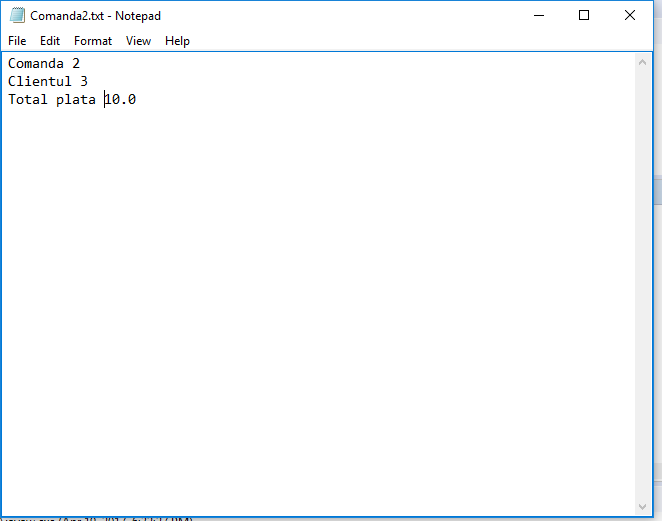
Introducerea unui produs nou



Afisarea tuturor comenzilor



Fisierul text generat de introducerea unei comenzi noi



# 6. Concluzie și dezvoltări ulterioare

Datorită stilului de implementare și a ogranizarii pe clase și interfețe a programului, dezvoltările ulterioare sunt ușor de implementat și de introdus în aplicatie. Există foarte multe posibile îmbunătățiri ale programului, de exemplu: introducerea mai multor operatii pe baza de date (gasirea datelor dupa fiecare camp si nu numai dupa id, posibiliatea de a actualiza toate datele din tabel preintr-o singura operatie, actualizarea numai unor anumite campuri etc), la introducerea unei comenzi, clientii si produsele disponibile sa fie afisate intr-un combo box pentru a asigura ca nu se face o comanda invalida, realizarea unei baze de date mai complexe care contine mai multe tabele, inregistrarile din tabele sa aiba mai multe campuri care sa ofere informatii mai detaliate.

Interfața grafică poate fi dezvoltată în scopuri educaționale, adăugând mai multe posibilități de interacțiune.

Aplicatia are drept scop lucrul cu o baza de date, se pot introduce clienti, produse si comenzi noi, cauta date in functie de id-ul lor, afisa toate datele din tabele, sterge date si acutaliza date . Datele se preiau din interfata grafica si prin apasarea butoanelor se efectueaza operatia dorita.

# 7. Bibliografie

www.wikipedia.com

www.stackoverflow.com

http://www.java2s.com/Tutorials/Java/JavaFX/0650\_\_JavaFX\_TableView.htm

http://docs.oracle.com/javafx/2/collections/jfxpub-collections.htm

https://bitbucket.org/utcn\_dsrl/pt-layered-architecture

http://coned.utcluj.ro/~salomie/PT\_Lic/3\_Lab/HW3\_Tema3/HW3\_Indications.pdf