Universitatea Tehnica din Cluj-Napoca

Aprilie 2020

Tehnici de Programare

Assignment no.3

Chiras Valentin-Fanica

Grupa 30223

Cuprins

Obiectivul temei

Consideră o aplicaţie de organizare a comenzilor şi procesarea comenzilor clienţilor pentru un depozit. O bază de date relaţionala este utilizată pentru stocarea produselor, a clienţilor şi a comenzilor efectuate de aceştia.

Obiective Principale

În plus, aplicaţia va folosi minim următoarele pachete şi clase:

Pachete:

- dataAccessLayer
- businessLayer
- model
- presentation
- start

Clase:

- Model Classes - reprezintă modelele de date ale aplicaţiei
- Bussiness Logic Classes - conţin logica aplicaţiei
- Presentation Classes - clase care conţin partea de interfaţă şi afişare a aplicaţiei
- Data Access Classes - clase care conţin acces la baza de date

Obiective secundare

- Documentaţie

- Prezentarea diagramei de clase UML a aplicaţiei

- Fişierele sursa

- Fişiere de tip JavaDoc

- Fişiere jar de executare al codului din linia de comandă

- Generarea de fişiere .pdf pentru raportarea rezultatelor

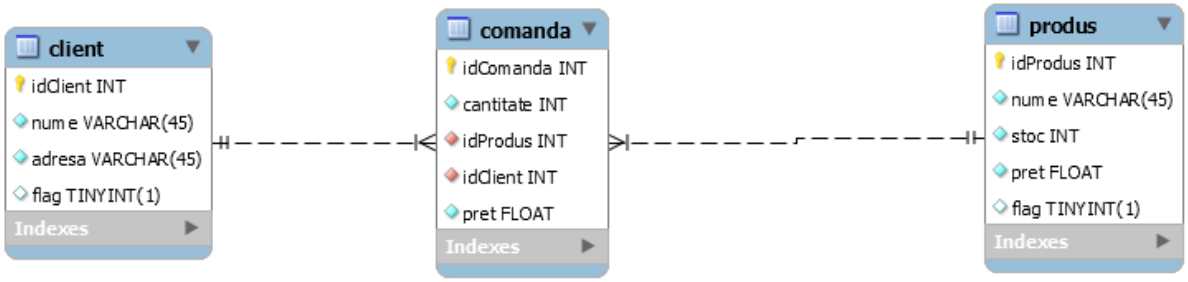
- Fişierul SQL folosit pentru crearea bazei de date şi a tabelelor

Analiza problemei

Pentru ca aveam o aplicaţie funcţională este nevoie de un sistem ce conţine o bază de date. Asta înseamnă că utilizatorul trebuie să fie capabil să manipuleze şi să modifice tabelele bazei de date şi să mai apoi să poate obţine datele care sunt stocate în aceste tabele. Restul aplicaţiei se bazează doar pe introducere comenzilor pe care utilizatorul doreşte să le aplice asupra bazei de date.

O bază de date relaţionala implica tabele interconectate prin diferite tipuri de relaţii. Fiecare tabel care reprezintă un model al aplicaţiei este caracterizat în mod unic printr-un ID care se incrementează automat la adăugarea unui nou element în tabelul respectiv. Acest ID este cel mai uşor mod de a face diferenţa între obiectele tabelelor. Celelalte câmpuri ale tabelelor nu sunt caracteristici sau intrări unice. În tabel pot exista doi clienţi care să aibă acelaşi nume sau două produse atât timp cât aceştia sunt diferenţiaţi prin ID-ul lor.

Pentru a simplifica problema avem nevoie de două tabele fundamentale, modelate după corespondenţa din lumea reală: '**product'** şi '**client'**. Un client poate să comande unul sau mai multe produse atâta timp cât există în stocul depozitului. Tabelul 'order' conţine ID-ul clientului,ID-ul produsului, preţul total si cantitatea dorita. Acest table face totodata legautra dintre client si produs cu ajutorul celor doua key secundare 'idProdus' si 'idClient'.



Operaţiile necesare pentru o comandă sunt:

- Adăugarea produselor în comandă
- Validarea comenzii
- Finalizarea comenzii şi actualizarea valorilor din stocul produsului

Implementarea acestei aplicaţii se bazează pe două unelte din programare şi anume: Java şi Mysql, care colaborează pentru atingerea obiectivului final. Pentru modelarea clară a proiectului, trebuie să existe o echivalentă între clasele model a proiectului şi modelul bazei de date. Fiecare model din aplicaţie are corespondent un tabel din baza de date. Fiecare coloană din tabel are un câmp care corespunde în clasa de atribute a fiecărui model, cu acelaşi tip.

Toate răspunsurile pe care la returnează programul corespund unui query din baza de date. Un query este o cerere pentru anumite date ce se găsesc în baza de date şi poate fi exprimat prin limbaj SQL.

Modelarea

Pentru modelarea şi implementarea acestui proiect sunt necesare mai multe elemente de luat în considerare, inclusiv specificaţiile detaliate ale implementării în MySQL. Aplicaţia Java şi MySQL trebuie să comunice, aşadar în Java trebuie să identificăm baza de date într-o manieră unică. Cu acest scop, proprietăţile bazei de date sunt începute în clasa **ConnectionFactory**, cum ar fi serverul, URL-ul bazei de date, 'username'-ul bazei de date şi parola.

Aplicaţia noastră trebuie să fie obligatoriu împărţită în mai multe 'componente', prin care separăm clasele în funcţie de capabilităţile şi proprietăţile lor. Aşadar se obţin două pachete care sunt legate de baza de date: pachetul **connection** şi pachetul **dataAccessLayer**. Pachetul connection implementează conexiunea actuală dorită între aplicaţia Java şi baza de date, iar dataAccessLayer referă la operaţiile: create, read, update, delete, care sunt funcţiile de bază pentru crearea, citirea, actualizarea şi ştergerea datelor.

Pachetul de bază al aplicaţiei este **model**, care conţine modelele de obiecte şi implementează obiectele care corespund cu entităţile din baza de date. Proprietăţile pachetului ne permit acces-ul la obiecte şi ne permit să modificăm caracteriticile lor.

Pachetul businessLayer modelează partea de aplicaţie. Acest pachet primeşte şi prelucrează datele care le primeşte din baza de date, le validează şi obţine rezultate pentru cererile utilizatorului.

Cazuri de utilizare

Utilizatorul are posibilitatea de a aplica 3 operaţii fundamentale asupra clientiilor şi produselor: **adăugare**, **editare**, **ştergere**. Un alt caz de utilizare, de o complexitate mai înaltă, implica crearea unei comenzi. Utilizatorul trebuie să urmeze câţiva paşi înainte de a face o efectua corect o comandă.

Pentru rularea aplicaţiei se creează o un fişier text cu numele commands.txt unde se adăugă comenzile pe care vrem să le execute baza de date. În cazul în care comenzile introduse nu sunt valide sau apar eventuale erori programul va arunca excepţii sau va afisa diferite mesaje care anunta problema în funcţie de situaţie. Un exemplu de date de intrare valide acceptate de program este dat mai jos.

Exemple de comenzi acceptate de program:

- Insert client: Ion Popescu, Bucuresti
- Delete client: Ion Popescu
- Insert product: apple, 20, 1
- Delete product: apple
- Order: Ion Popescu, apple, 5
- Report client
- Report order
- Report product

Proiectarea

Aplicaţia utilizează o împărţire a arhitecturii în pachete după cum urmează: dataAccessLayer, businessLayer, model, presentation, businessLayer, start. Pachetul presentation foloseşte pachetul bussinessLayer pentru operaţii, pachetul businessLayer foloseşte pachetul accessLayer pentru conexiunea cu baza de date şi accessLayer foloseşte pachetul connection care este format doar din clasa Connection care execută conexiunea programului cu baza de date. Fiecare dintre pachetele enumerate foloseşte şi pachetul model.

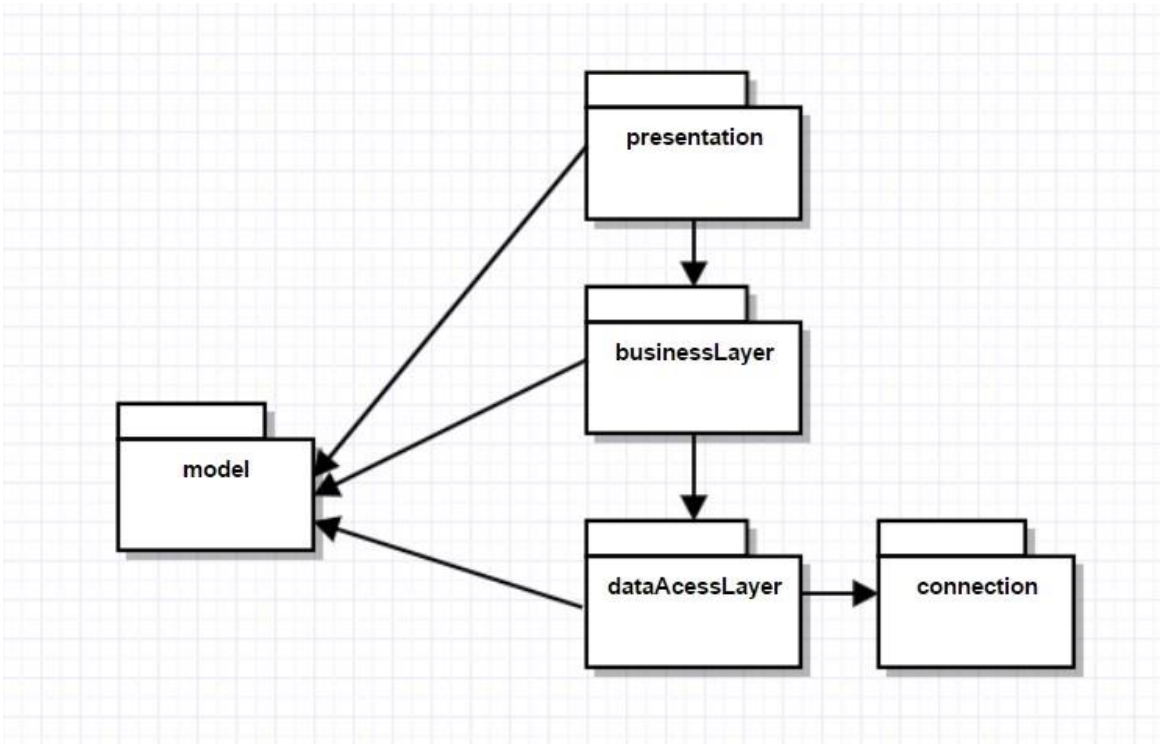
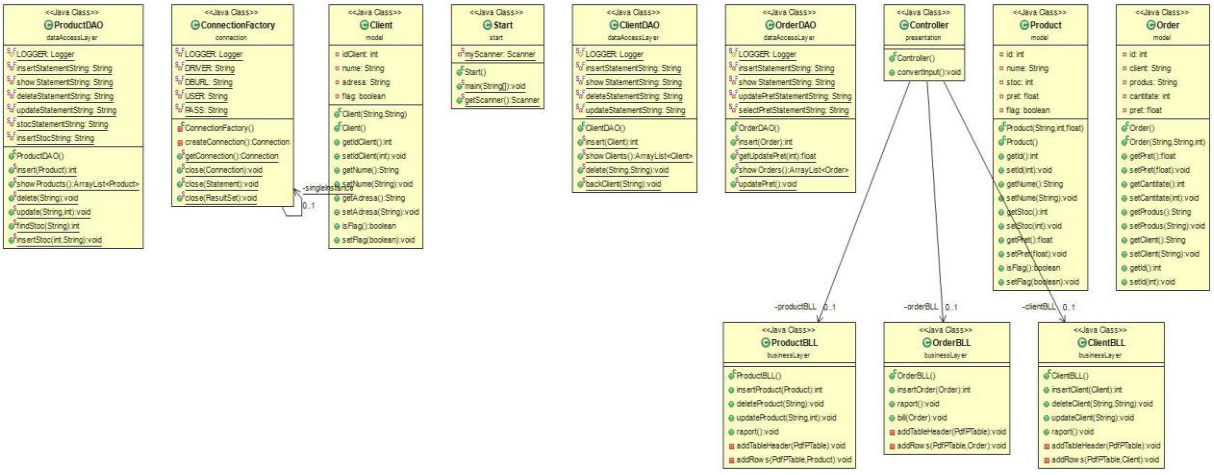


Diagrama de clase

Diagrama de clase este una mai degrabă vastă, însă nu extrem de complicată, acesta urmează împărţirea proiectului în pachete şi clasele corespunzătoare acestora.



Decizii de proiectare.

Model

Clasele care aparţin acestui pachet sunt echivalente tabelelor create în baza de date. Fiecare clasă are un tabel corespunzător cu toate elementele acestuia ca şi atribute de tip private. Pentru obţinerea datelor din clase am generat getters şi setters pentru toate atributele acestora.

Clasa Client modelează un exemplu real de obicet de tip persoană cu atribute utile în lumea reală pentru executarea unei comenzi de către această. Clasa conţine getters şi setters pentru toate atributele acesteia. Aceasta clasa precum si clasa Produs contin un camp additional numit ‘flag’ acesta este folosit pentru a decide daca clientul/produsul respectiv este ‘sters’ sau nu. Mai exact comanda de Delete nu va sterge fizic clientul sau produsul ci va actualiza campul flag=true in cazul in care se doreste ca produsul sau clientul sa fie sters

Clasa Product descrie un obicet de tip produs din lumea reala cu atributele: id-ul produsului, nume, cantitate in stoc si pret si deasemenea flag-ul mentionat mai sus.

Clasa Order corespunde unei comenzi efectuate de un client, cu toate informaţiile necesare executării acesteia: id-ul comenzii, id-ul clientului, numele clientului, numele produsului, cantitatea şi preţul total. Aceasta clasa contine campuri diferite fata de cea din baza de date. Acest lucru l-am facut datorita formatului comenzilor.

Connection

Clasa ConnectionFactory din acest pachet face conexiunea intra aplicatia noastra si baza de date. Clasa primeste atributele pentru efectuarea conexiunii:

```
private static final String DRIVER = "com.mysql.cj.jdbc.Driver";
private static final String DBURL =
"jdbc:mysql://localhost:3306/order_management";
private static final String USER = "root";
private static final String PASS = "root";
private static ConnectionFactory singleInstance = new ConnectionFactory();
```

Clasa oferă metode pentru crearea, deshiderea şi închiderea conexiunii. Clasa suprascrie metoda close, primind pe rând ca argument Statement, ResultSet şi Connection.

Data Acces Layer

Clasele din acest pachet au obiectivul de executa operaţii primite de la utilizator în fişierul text commands.txt asupra bazei de date. Aceste operaţii sunt Create, Read, Update şi Delete. Aceste clase introduc datele în baza de date din interiorul aplicaţiei. Pentru a obţine rezultatele, sunt aplicate interogări cu ajutorul limbajului SQL aplicat pe tabelele bazei de date. Ficare clasa din acest pachet conţine metode pentru înserarea, ştergere, actualizarea a bazei de date, selectarea unui obicet din tabel sa a tuturor.

În continuare este dat un exemplu pentru clasa Client a descrierii făcute mai sus.

```
public static int insert(Client client) {
		Connection connection = ConnectionFactory.getConnection();
		int id = 0;
		PreparedStatement insertStatement = null;
		try {
			insertStatement = connection.prepareStatement(insertStatementString,
Statement.RETURN_GENERATED_KEYS);
			insertStatement.setString(1, client.getNume());
			insertStatement.setString(2, client.getAdresa());
			insertStatement.executeUpdate();
			ResultSet rs = insertStatement.getGeneratedKeys();
			if (rs.next()) {
				id = rs.getInt(1);
			}
		} catch (SQLException e) {
			LOGGER.log(Level.WARNING, "ClientDAO insert : " + e.getMessage());
		} finally {
			ConnectionFactory.close(insertStatement);
			ConnectionFactory.close(connection);
		}
		return id;
	}
```

Algoritmul folosit este suficient de simplu. În functie de operaţia pe care dorim să o efectuăm alegem o metodă care va primi atributul corespunzător acesteia cu instrucţiunea în SQL care trebuie dorim să o aplicăm pe tabel. În exemplu de mai sus se obţin atributele unui client din clasa obiectului şi se setează consecutiv tuplelor din tabel, după care se adăugă în tabel şi astfel se obţine id-ul clientului care se incrementează automat.

Pentru interogările SQL am creat constante pentru fiecare clasă pe care le-am adaptat în funcţie de numărul şi tipul tuplelelor.

```
private static final String insertStatementString = "INSERT INTO
client(nume,adresa)" + "VALUES(?,?)";
      private static final String showStatementString = "SELECT * from client
where client.flag=false";
      private static final String deleteStatementString = "UPDATE client SET
client.flag=true where nume=? and adresa=?";
      private static final String updateStatementString = "UPDATE client SET
client.flag = false where nume = ?";
```

Exista totusi cateva inetrogari diferite in celelalte clase OrderDAO si ProductDAO.

```
private static final String insertStatementString = "INSERT INTO
comanda(idClient,idProdus,cantitate)"
                  + "VALUES((SELECT client.idClient from client where
client.nume = ?),(SELECT produs.idProdus from produs where produs.nume = ?
),?)";
       private static final String showStatementString = "select
comanda.idComanda,client.nume as client,produs.nume as
produs,comanda.cantitate,comanda.pret"
                  + " from comanda,client,produs where
comanda.idClient=client.idClient and comanda.idProdus=produs.idProdus";
      private static final String updatePretStatementString = "UPDATE
comanda,produs SET comanda.pret=comanda.cantitate*produs.pret where
comanda.idProdus=produs.idProdus";
      private static final String selectPretStatementString = "Select
comanda.pret from comanda where idComanda=?";
```

Interogarile din clasa OrderDAO.

Interogarea de insert este scrisa asa incat sa putem introduce id-ul clientului si al produsului folosind numele acestora.
Deasemenea interogarea updatePretStatementString este folosita pentru a seta pret-ul comenzii dupa formula pret=cantitate*pretul produsului. Iar cu ajutorul interogarii selectPretStatementString putem seta pretu-ul produsului cu valoarea actualizata a pretului din baza de date.

```
      private static final String insertStatementString = "INSERT INTO
produs(nume,stoc,pret)" + "VALUES(?,?,?)";
      private static final String showStatementString = "SELECT * from produs
where produs.flag=false";
      private static final String deleteStatementString = "UPDATE produs SET
produs.flag=true where nume=?";
      private static final String updateStatementString = "UPDATE
comanda,produs SET produs.stoc=produs.stoc-comanda.cantitate WHERE
produs.nume=? and comanda.idComanda=?";
      private static final String stocStatementString = "Select produs.stoc
from produs where produs.nume=?";
      private static final String insertStocString = "UPDATE produs SET stoc =
stoc + ? where produs.nume=?";
```

Interogarile din ProdusDAO.
Interogarea updateStatementString va fi filosita pentru a actualiza la fieacare comanda stoc-ul produsului. Deasemenea interogarea insertStocString este folosita pentru a eficientiza programul in care se introduce de doua ori acelasi produs, cu ajutorul acestei interogari atunci cand doreste sa se introduca de doua ori acelasi produs acesta va apela interogarea respectiva si va actualiza doar stoc-ul. Acest lucur se face doar cand produsul nu este sters. In caz contrar se va adauga produsul chiar daca el exista deja in baza de date cu flagul=false.
BusinessLayer

Acest pachet se ocupă cu logica aplicaţiei. Fiecare din clasele ClientBLL, OrderBLL, OrderItemBLL, ProductBLL este legată de clasele ClientDAO, OrderDAO, OrderItemDAO, ProductDAO şi conţine toate operaţiile implementate de acestea pe care le apelează pentru un obiect de orice tip din clasa model.

În fiecare clasa atunci cand se doreste raportul claselor respective, se apeleaza o metoda numita raport care exista in fiecare clasa, care va crea pdf-urile cerute. OrderBLL mai contine o metoda numita bill care va genera facutri pentru fiecare comanda creata.

Rezultate

Pentru a vizualiza rezultatele aplucatie se va face cu ajutorul fisierului text commands.txt. Acest fisier contine comenziile ce vor fi verificate. Rapoartele si facturile se vor crea in fisierul assignment3.demo.

Programul a fost testat generând bills şi reports pentru datele din exemplu dat în cerinţa assignment-ului. Rezultatele obţinute în urma testării sunt date mai jos.

Raport clienti

id	Nume	Adresa
2	Luca George	Bucuresti
3	Sandu Vasile	Cluj-Napoca

Raport Produse

id	Nume	Stoc	Pret
1	apple	15	1.0
3	orange	40	1.5
4	lemon	65	2.0

Raport Orders

id	Client	Produs	Cantitate	pret
1	Luca George	apple	5	5.0
2	Luca George	lemon	5	10.0

Bill: 1

id	Client	Produs	Cantitate	pret
1	Luca George	apple	5	5.0

Bill: 2

id	Client	Produs	Cantitate	pret
2	Luca George	lemon	5	10.0

Dupa cum se poate observa ultima comanda din commands.txt nu s-a efectuat deoarece cantitatea comandata este mai mare decat stoc-ul produsului. Acest lucru a dus la neefectuarea comenzii si deasemenea neinregistrarea ei in baza de date.

Concluzie

In urma aceste aplicatii am invatat cum sa creez pdf-uri si cum sa lucrez cu o baza de date. Totusi sunt mai mult ca sigur ca exista o cale si mai simpla de lucru decat cea pe care am abordat-o eu.

Dezvoltări pe viitor

Acesta aplicatie a mea va arunca o exceptie atunci cand se introduce o comanda in care exista 2 clienti cu acelasi nume.

Totusi poate fi dezvoltata modul de asociere a tabelelor prin adaugarea unor tabele care sa ne ajuta in legaturile dintre acestea. Si deasemnea atunci cand se introduce o comanda care nu poate fi efectuata aceasta sa creeze un pdf in care alerteze acest lucru.

Bilbliografie

https://www.w3schools.com/sql/

https://dzone.com/articles/layers-standard-enterprise

https://www.baeldung.com/java-pdf-creation

https://www.baeldung.com/javadoc

https://mkyong.com/jdbc/how-to-connect-to-mysql-with-jdbc-driver-java/