Documentație TEMA 3

ORDER MANAGEMENT

Nume: Câmpean Adelina Ioana

Grupa: 30227

Profesor Laborator: Pop Cristina

Cuprins

1. Cerințe Funcționale
2. Obiectivul temei
3. Informații despre Baza de date și conexiunea cu JAVA
4. Analiza Problemei modelare, scenarii, cazuri de utilizare
   1. Diagrama USE CASE
5. Proiectare
   1. Diagrama de clase UML
   2. Diagrama de pachete
   3. Implementare
6. Rezultate
7. Concluzii
8. Bibliografie
9. Cerințe funcționale

Luați în considerare o cerere OrderManagement pentru procesarea comenzilor clienților pentru un depozit. Bazele de date relaționale sunt utilizate pentru a stoca produsele, clienții și comenzile.

a. Analizați domeniul aplicației, determinați structura și comportamentul clasei sale și desenați o extensie a diagramei de clase UML.

b. Implementați clasele de aplicații. Utilizați javadoc pentru documentația claselor.

c. Utilizați tehnici de reflection pentru a crea o metodă createTable care primește o listă de obiecte și generează antetul tabelului, prin extragerea prin reflection a proprietăților obiectului și apoi popularea tabelului cu valorile elementelor din listă:

JTable createTable (obiecte listate <Object>);

d. Implementați un sistem de programe de utilitate pentru raportare, cum ar fi: stocuri, totaluri, filtre etc.

1. Obiectivul temei

Această temă are ca scop simularea unui magazin, care preia comenzi, pe baza unui client și a unui produs ales și cantitatea dorită din produsul respectiv. Aplicația este realizată printr-o conexiune la o bază de date, creată de noi. Aplicația permite adăugarea de noi clienți, noi produse, actualizarea acestora și preluarea de comenzi. Pentru fiecare client se cunosc id-ul, care este o cheie unică, care identifică fiecare client, numele acestuia și vârsta. Pentru fiecare produs se cunosc id-ul care este la fel ca la client, o cheie unică, folosită pentru identificarea produsului, numele acestuia și cantitatea de produse care se afla în stocul magazinului. O comandă constă în alegerea clientului care comandă și alegerea produsului pe care îl dorește, în cantitatea preferată. În cazul în care nu sunt suficiente produse pe stoc, se va afișa un mesaj de eroare, prin care se anunță acest aspect.

1. Informații despre Baze de Date

Modalitate de stocare a unor informații (date) pe un suport extern, cu posibilitatea regăsirii acestora.

O aplicație JDBC utilizează unul sau mai multe drivere din pachetul java.sql care sunt utilizate de către clasa DriverManager. Driverele sunt specifice bazelor de date, deci pentru fiecare tip de baza de date se utilizează un driver special. In aceeași aplicație putem lucra cu baze de date diferite, deci implicit si cu mai multe drivere. Putem avea nevoie de un driver care comunica cu o baza de date Oracle aflata la distanta si de un altul care reprezintă legătura spre driverul ODBC local și comunica cu un server SQL.

Comunicarea unei baze de date cu o aplicație java, se realizează utilizând mysql-connector-java.jar, iar conexiunea se realizează printr-un obiect de tip java.sql.Connection.

Tehnologia JDBC este realizata prin utilizarea unei biblioteci de clase (java.sql) care permite executarea unei cereri de tip SQL pentru baze de date relaționale, din aplicații scrise in Java. Avantajul utilizării acestui pachet este ca aplicațiile Java sunt aceleași, indiferent de baza de date.

Cele mai importante clase in biblioteca java.sql sunt:

·       DriverManager, folosita pentru a facilita utilizarea mai multor drivere de baze de date într-o singura aplicație, fiecare dintre drivere putând fi utilizat pentru conectarea la o sursa diferita de date;

·       Connection, obiect care specifica cu ce baza de date se face comunicația;

·       Statement, o instrucțiune SQL aplicata bazei de date.

1. Analiza Problemei, modelare, scenarii, cazuri de utilizare
   1. Diagrama USE CASE

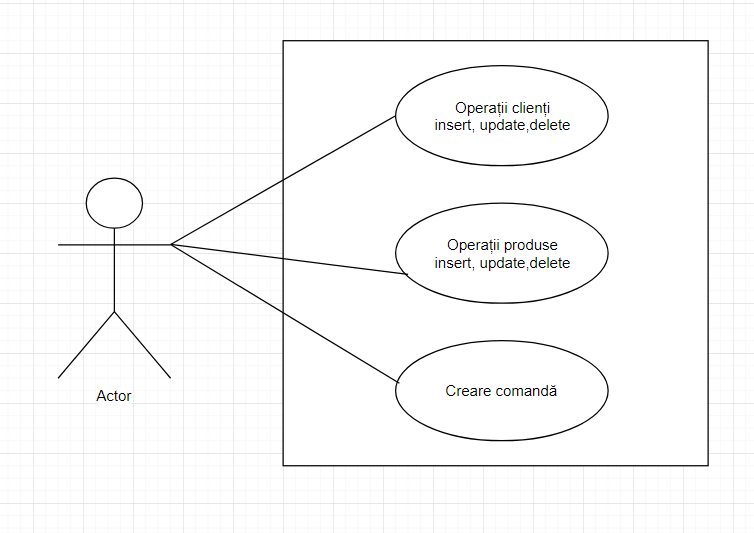


Diagrama USE – CASE este formată pentru a fi mai ușor de înțeles care este rolul aplicației.

Actorul este utilizatorul aplicației, care poate să execute o comandă prin alegerea unui client, alegerea unui produs și plasarea comenzii. De asemenea poate să introducă un nou client, să actualizeze unul deja prezent în baza de date, sau să șteargă un client. Asemănător poate executa și pentru produse. Actorul poate să și vizualizeze fiecare tabelă creată, pentru a vedea toți clienții și toate produsele, dar și comenzile executate anterior.

Scenariu de utilizare:

* Actorul introduce un nou client, și anume pe el însuși
* Vizualizează tabela de produse, dacă nu îi este necesar nimic de acolo, își introduce propriul produs
* Execută o comandă, apoi în fișierul text orderInfo, își poate vizualiza chitanța sau factura comenzii realizate.

1. Proiectare
   1. Diagrama de clase UML

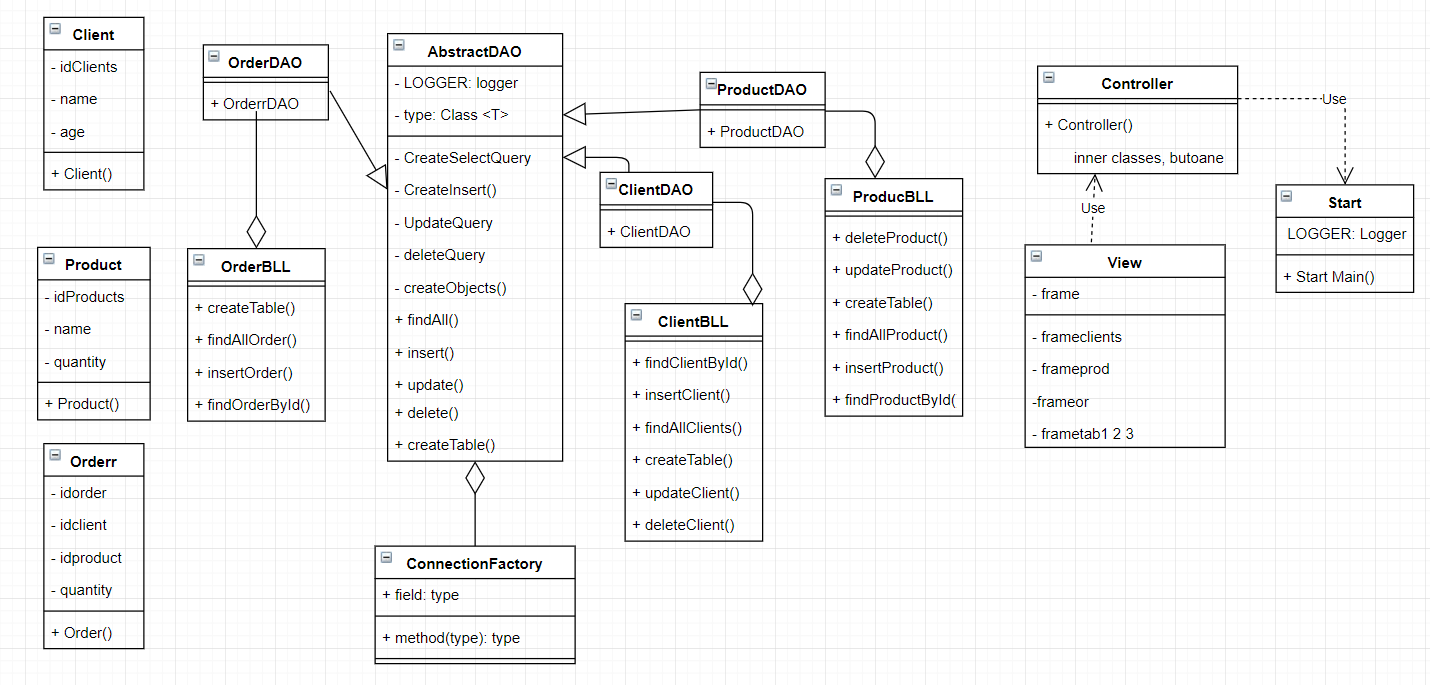
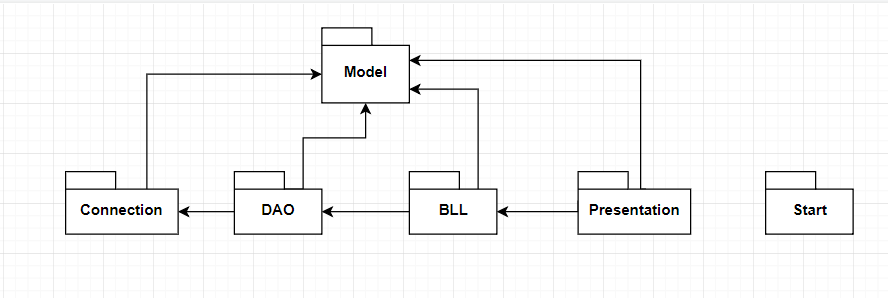


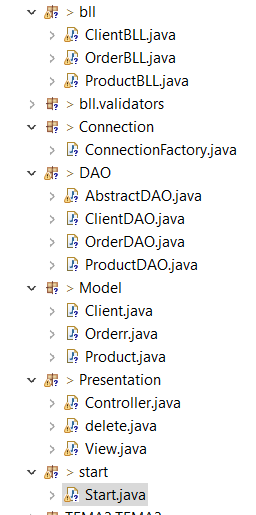
Diagrama UML este alcătuită din 14 clase, împărțite pe peachete. Aplicația utilizează tehnica Layered Architecture, în care se pot instanția obiecte doar de la clasa se deasupra, nu se poate sări peste alte clase (etaje).

Clasele Client, Product și Order sunt implementate în pachetul Model, clasele AbstractDAO, ClientDAO, ProductDAO și OrderDAO fac parte din pachetul DAO, clasele ClientBLL, ProductBLL și OrderBLL fac parte din pachetul BLL, clasele View si Controller fac parte din pachetul Presentation iar clasa start este din pachetul cu același nume.

* 1. Diagrama de pachete



Aplicația a fost realizată în pachete și anume **Presentation** conține clasele care definesc interfața grafică, **bll**, sau Bussiness, conține clasele care încapsulează logica aplicației, **Connection** face conexiunea bazelor de date cu limbajul java, **dao** conține clasele care generează interogările in bazele de date, **model** conține clasele care au corespondent câte o tabelă iar **start** este un pachet care rulează programul.



Pachetul Model

Clasele Client, Product și Order, sunt clasele create după tabelele din baza de date, care au aceleași nume cu acestea, iar fiecare coloană din tabelă este reprezentată ca variabilă de instanță în aceste clase, cum ar fi idclients, name, age din clasa client, respectiv tabela, isproduct, name, quantity, din clasa Product și idorder, idclient, idproduct, quantity din clasa order.

Pachetul Presentation

Clasa View conține datele despre interfața grafică. Au fost realizate 7 frame-uri, pentru fiecare clasă din pachetul model, Client, Product, Order, un frame inițial, și încă câte un frame pentru a vizualiza tabelele cu valori pentru fiecare clasă. În aceasta clasă, au fost scrise metode de ActionListener pentru toate butoanele din frame-uri.

Clasa Controller, are rolul de a prelua informațiile din interfața grafică, de a seta fiecare buton, în momentul apăsării să se execute diferite metode din cod, îm funcție de rolul butonului.

Pachetul Connection

Clasa ConnectionFactory face legătura dintre bazele de date si limbajul java. Clasa conține numele driver-ului, inițializat prin metoda reflection, path-ul către baza de date ți user-ul si parola pentru a putea accesa Serverul MySQL. Conexiunea cu baza de date va fi realizată prin intermediul unui obiect de tip Singleton.

Clasa conține metode de creare a conexiunii, de obținere a unei conexiuni active și de închidere a conexiunii, de închidere a unui Statement sau a unui ResultSet.

Pachetul DAO

Clasa AbstractDAO este o clasă generică, care lucrează cu orice tip de obiecte. Obiecte de tip <T>. În această clasă, au fost implementate metode care utilizează interogările din bazele de date. Și anume, createSelectQuery, create Insert, updateQuery, deleteQuery folosesc un StringBuilder in interiorul căruia se salvează interogarea corespunzătoare fiecărei operații ( SELECT \* FROM table\_name; - createSelectQuery, INSERT INTO table\_name (column1, column2, column3, …) VALUES (value1, value2, value3, ...); – create insert, UPDATE table\_name SET column1 = value1, column2 = value2, WHERE condition; - updateQuery , DELETE FROM table\_name WHERE condition; - deleteQuery).

Metoda findAll are rolul de a afișa toate elementele din tabele, metodă folosită la apăsarea butonului ”Table View”, din fiecare frame, pentru vizualizarea tuturor valorilor din tabelele Client, Product sau Order.

Metoda findById găsește un anumit client sau produs în funcție de id. Metoda este utilizată la realizarea facturii din comandă. În momentul în care se execută o comandă, se introduc id-urile clientului, respectiv al produsului comandat și pentru a realiza factura avem nevoie sa aflăm toate informațiile atât despre client cât și despre produs.

Metoda insert,update și delete, apelează create insert, updateQuery și deleteQuery. Se realizează conexiunea cu baza de date, se generează statement-ul și se execută.

Metoda createTable primește ca și parametru un obiect, pe baza căruia se realizează o listă de obiecte, apelând metoda findAll(). Am creat o matrice de obiecte, prin intermediul căreia se va popula tabela și un String în care se va salva antetul tabelei (numele coloanelor). Utilizând tehnica de reflection, extrage field-urile si valorile corespunzătoare, printr-un for each de field-uri, obțin clasa obiectului și câmpurile declarate în clasa respectivă. Permit accesarea variabilelor de instanță private și salvez numele câmpurilor în string-ul header. Apoi Pentru fiecare obiect din lista obținută anterior, se parcurg field-urile, obținând valorile și apoi sunt introduse în matricea care populează tabela. Metoda returnează un JTable.

**public** JTable createTable(Object object) {

**int** i=0;

**int** j=0;

List<T> obj = findAll();

Object[][] populate = **new** Object[obj.size()][object.getClass().getDeclaredFields().length];

String[] header=**new** String[object.getClass().getDeclaredFields().length];

**for** (Field field : object.getClass().getDeclaredFields()) {

field.setAccessible(**true**);

header[j]=field.getName();

j++;

}

j=0;

**for**(Object o:obj)

{

**for** (Field field : object.getClass().getDeclaredFields()) {

field.setAccessible(**true**);

**try** {

populate[i][j]=field.get(o);

j++;

}

**catch** (IllegalArgumentException e) {

e.printStackTrace();

} **catch** (IllegalAccessException e) {

e.printStackTrace();

}

}

i++;

j=0;

}

**return** **new** JTable(populate,header);

}

Clasele ClientDAO, ProductDAO, OrderDAO extind AbstractDAO, pentru a putea face legatura cu fiecare tip de obiect în parte. Astfel prin instanțierea unui obiect de tip ClientDAO, ProductDAO sau OrderDAO se pot implementa metodele generice din AbstractDAO.

Pachetul BLL

Conține clasele ClientBLL, ProductBLL, OrderBLL care instanțiază câte un obiect de tipul ClientDAO, ProductDAO sau OrderDAO. Sunt implementate metode în care se apelează metodele din AbstractDAO. În insertClient, din ClientBLL, se formează un String cu numele variabilelor de instanță, și se instanțiază un obiect de tip client, ca si parametru. Apoi se apelează prin intermediul obiectului clientDAO, metoda insert, căreia i se trimit ca și parametri, clientul si stringul cu nume. Asemănător fiind și pentru restul metodelor atât din ClientBLL cât și din ProductBLL.

În clasa OrderBLL, metoda insertOrder formează de asemenea un String cu numele variabilelor de instanță, dar totodată, aici se realizează și factura comenzii. Se instanțiază un obiect de tipul Product și Client, în care se salvează ceea ce returnează metoda findClientById, respectiv findProductbyId, apoi în câte 2 sttring-uri se salvează toString-ul claselor Client si Product. Apoi se apelează metoda insert din AbstractDAO. Pentru realizarea facturii, am creat un fisier text OrderInfo, în care am introdus informațiile preluate anterior.

**public** Product insertOrder(Orderr order) {

PrintWriter writer;

ProductBLL p = **new** ProductBLL();

ClientBLL c = **new** ClientBLL();

String pop = **new** String();

pop = "("+order.getIdclient()+","+order.getIdproduct()+","+order.getQuantity()+");";

Product ax=p.findProductById(order.getIdproduct());

Client cx=c.findClientById(order.getIdclient());

String cc =cx.toString();

String pp = ax.toString();

orderDAO.insert(order,pop);

**try** {

writer = **new** PrintWriter("orderInfo.txt");

writer.println("FACTURĂ\r\n");

writer.println("Comanda contine: ");

**if**(order.getQuantity()==1)

writer.println(pp+"\r\nA fost comandata o bucata.");

writer.println(pp+"\r\nAu fost comandate "+order.getQuantity()+" bucati.");

writer.println("Produsul a fost comandat de :");

writer.println(cc);

writer.close();

} **catch** (FileNotFoundException e) {

e.printStackTrace();

}

**return** ax;

}

Pachetul Start

Conține clasa Start care creează un obiect de tipul View și un obiect de tipul Controller, care primește ca și parametru view-ul, astfel la rularea programului apare primul frame, iar apoi, în funcție de butoanele apăsate, se execută restul codului.

1. Rezultate

Rezultatele sunt vizibile în fiecare tabelă, se pot verifica operațiile de inserare, actualizare și ștergere, atât pentru clienți cât și pentru produse. Pentru fiecare comandă se pot observa, în tabel, clienții care au procesat comanda și ce produse au comandat, și de asemenea se pot observa informațiile detaliate ale comenzii procesate anterior, în fișierul text orderInfo .txt.

1. Concluzii

În concluzie, această aplicație simulează modul în care se realizează o comandă, în care putem selecta ce client dorim să comande, ce anume să comande și avem posibilitatea de a adăuga atât clienți cât și produse, de a-i actualiza, adică de a modifica doar anumite valori la ei, sau de a-i șterge cu totul din tabelă.

Din această temă, am învățat să lucrez cu tehnica de Reflection, care preia informațiile dintr-o clasă, fără să știe ce tip de obiect este, am mai învățat să lucrez generic, adică sa implementez o metodă ”la general”, apoi să o apelez în cadrul mai multor clase, respectiv metode, pentru a nu scrie același cod în fiecare clasă, am creat o clasă abstractă, care a implementat generic toate metodele care trebuiau aplicate pe mai multe tipuri de obiecte. Astfel am economisit timp si memorie.

Dezvoltări ulterioare ar putea fi, adăugarea unui coș de cumpărături, în care să putem adăuga mai multe produse într-o singură comandă, deoarece momentan într-o comandă se poate alege un singur tip de produs, dar la cantitatea dorită de client, respectiv utilizator. Aplicația mai poate fi modificată astfel încât o comandă să se realizeze alegând produsele direct din tabela în care sunt prezentate, să fie cât mai real, utilizatorul alege produsul și îl introduce în coș. Pentru fiecare produs s-ar putea adăuga și o descriere, care sunt atributele produsului, calitatea acestuia, poate și recenzii de la alți clienți, automat adăugând și o secțiune de recenzii la finalul unei comenzi, sau o poza cu produsul ar fi o altă idee, ca utilizator, clientul să vadă cum arată produsul pe care dorește să și-l achiziționeze.

1. Bibliografie

* <http://www.rasfoiesc.com/educatie/informatica/baze-de-date/Interfatarea-cu-bazele-de-date16.php>
* <https://profs.info.uaic.ro/~acf/java/slides/ro/jdbc_slide.pdf>
* <http://coned.utcluj.ro/~salomie/PT_Lic/4_Lab/HW3_Tema3/Tema3_HW3_Indications.pdf>