Order Management

(documentatie)

Disciplina: Tehnici de programare fundamentale

Student: Mihaila Alexia

Grupa: 30229

**Cuprins**

1. **Obiectivul temei**
2. **Analiza problemei, modelare, scenarii, cazuri utilizare**
3. **Proiectare**
4. **Implementare**
5. **Rezultate**
6. **Concluzii**
7. **Bibliografie**
8. **Obiectivul temei**

Se considera o aplicatie *OrderManagement* pentru procesarea comenzilor date de clientii unui depozit.

Ca obiective secundare care vor fi detaliate mai departe in capitolul de **Implementare,** se prezinta urmatoarele conditii : - stocarea produselor, clientilor si comenzilor va fi realizata intr-o baza de date

relationala.

- aplicatia va trebui structurata in pachete utilizandu-se arhitectura de tip “layered

architecture” si va utiliza minimul prezentat de urmatoarele clase:

1. *Model classes*: - modelul de date al aplicatiei
2. *Business Logic classes*: - implementarea logica a aplicatiei
3. *Presentation classes*: - implementarea unitatii de intrare/iesire pentru utilizator
4. *Data access classes*: - implementarea unitatii de acces la baza de date
5. **Analiza problemei, modelare, scenarii, cazuri utilizare**

Aplicatia va prelua anumite comenzi ce doresc sa fie aplicate asupra unei baze de date prin intermediul unei interfete grafice, care face usoara legatura dintre utilizator si realizearea operatiilor dorite.Scopul principal al acestei aplicatii este acela de a manipula datele dintr-o baza de date cu scopul de a adauga,sterge edita sau afisa ( prelua ) obiecte.

Pasii urmati in cadrul utilizarii aplicatiei sunt: la pornirea aplicatiei, utilizatorului I se va deschide o fereastra principala care va permite alegerea unui tip de obiecte ce corespunde unui tabel din baza de date astfel incat utilizatorul sa isi poata alege campul dorit. In urma alegerii uneia dintre optiuniile dorite (client, produs, comanda), in fereastra sav or deschide optiuniile posibile de realizat asupra tipului de obiect ales.De exemplu daca este ales client sau produs, utilizatorul va putea adauga un client sau produs nou, va putea edita sau sterge un client deja existent sau va putea vizualiza toti clientii sau toate produsele existente in baza de date sub forma unui tabe. Daca acesta selecteaza optiunea comanda, acesta isi va putea plasa o comanda noua doar daca clientul este deja inregistrat in baza de date iar produsul ce doreste sa il selecteze exista de asemenea. In caz afirmativ se va plasa comanda si se vor scadea numarul produselor din baza de date. De asemenea in cazul optiunii comanda, se poate genera si o facture in format pdf cu datele importante din cadrul facturii.

Acest tip de aplicatie este util in cazul unor magazine online pentru a retine atat evidenta produselor si comenziilor dar si pentru a putea face mai usoara interactiunea utilizatorilor cu baza de date si de asemenea de a retine clientii si a face comenzile si generarea facturilor mai utila.

1. **Proiectare**

In urma deciziei de proiectare (proprie si impuse) s-au format urmatoarele aspecte ale caror totalitate asambleaza structura aplicatiei, acest fapt fiind observabil in diagrama UML din *Fig. 1*, impartite pe mai multe straturi (layers):

**Clase**: (pkg.: bll)ClientBLL, ProductBLL, OrderBLL, (pkg. bll.validators) EmailValidators, PriceValidators (pkg.connection) ConnectionFactory, (pkg. dao) AbstractDAO, ClientDAO, ProductDAO, OrderDAO

(pkg. main) Start, (pkg. modelClasses) Bill, Client, Order, Product,

(pkg. presentationClasses) Data, (pkg.presentation.tables) AbstractJTable, ClientTable, ProductTable (pkg.presentation) StartPage, ClientPage, ProductPage, OrderPage

**Structuri de date**: LinkedList<?>, File, Scanner, FileOutputStream, Connection, ResultSet, Statement,

TimeStamp, Iterator, Chunk, Document, Paragraph, PdfWriter, DriverManager.

**Relatii**: asociere, agregare, compozitie, dependenta

**Pachete**: java.io.\*, java.util.\*, java.sql.\*, com.itextpdf.text.\*

Începem implementarea prin crearea clasei care realizează conexiunea cu baza de date. Aceasta este clasa ConnectionFactory care oferă un getter pentru conexiunea creată, precum și metode pentru închiderea conexiunii, declarației și resultSet.

Apoi trecem la clasele care definesc baza de date în aplicație. Acestea sunt clasele care construiesc modelul și fiecare trebuie să fie mapat la unul dintre tabelele din baza de date. Identificăm toate tabelele din baza de date și creăm clasele: Clienți, Produse, Comenzi de comandă având fiecare ca câmpuri atributele din tabelul pe care le mapează.O ultimă clasă particulară este clasa Bill care se mapează pe o tabelă virtuală care este utilizată pentru generarea datelor pentru raportul comenzilor. Aceste 4 clase constituie modelul aplicației și toate celelalte clase se bazează pe ele.

Următorul pas este implementarea claselor care se ocupă de operațiunile de acces la date. Vizualizând cele mai des operații necesare, se poate vedea cu ușurință că urmează un model foarte similar. În acest caz, operațiunile de bază, cum ar fi identificarea, inserarea, actualizarea și ștergerea, ar putea fi implementate mai eficient într-un mod generic. Definim o clasă abstractă AbstractDAO care oferă o implementare generică (realizata prin reflexive) pentru operațiunile menționate mai sus și reprezintă baza pentru toate clasele DAO specifice. Intuitiv, creăm o clasă pentru fiecare tabel care va implementa operația necesară care trebuie efectuată pe tabela respectivă. Toate clasele: ClientDAO, ,ProductDAO și OrderDAO extind clasa AbstractDAO care le oferă operațiunile de bază și fiecare dintre ele definește implementarea operațiunilor specifice necesare pentru fiecare tabelă. Clasele DAO definesc pachetul de acces la date (DAO).

Continuând, observăm necesitatea unor clase care funcționează cu clasele DAO la un nivel mai abstract. Aceste clase ar trebui să definească logica pentru gestionarea operațiunilor de pe tabele. Aici sunt validate datele care intră în baza de date și sunt prelucrate datele obținute din baza de date. Este creat un nou set de clase care construiesc stratul logicii de afaceri: ClientsBLL, ProductsBLL, OrderItemsBLL și OrdersBLL fiecare având grijă de toate operațiunile posibile efectuate pe tabele, precum și gestionarea fluxului de date. Cu toate acestea, este necesar un nou validator de clasă care să ofere metode pentru validarea tuturor câmpurilor din clasele de model, reducând astfel inconsecvența care poate apărea în datele stocate în baza de date EmailValidators, PriceValidators. Aceste clase implementează împreună logica de afaceri a aplicației.

Ultimul lucru pe care trebuie să îl poată face aplicația este să ofere o modalitate de comunicare cu utilizatorul. De obicei, acest lucru se face prin interfața grafică cu utilizatorul.In cadrul interfetei grafica am relizat un frame principal StartPage in cadrul careia se poate alege tipul de obiect asupra caruia dorim sa realizam operatiile CRUD,iar in urma butonului apasat in cadrul acestei interfete se va elibera panel-ul si se vor adauga campurile necesare pentru realizarea operatiilor posibile asupra obiectului. Acest lucru se realizeaza in cadrul claselor ClientPage, ProductPage, OrderPage. De asemenea pentru realizarea optiunii de afisare a obiectelor din baza de data a fost necesar sa cream clasa AbstractJTAble in care realizam in med generic (prin reflexive) crearea unui Jtable in care ne sunt afisate field-urile si datele din baza de date corespunzatoare obiectului. Apoi cream clasele ProductTable si ClientTable prin care ne generam tabelele in mod particular.

Întregul design al aplicației va fi mai bine înțeles din diagramele de mai jos care conțin toate clasele împreună cu dependențele lor.

1. **Implementation**

**ConnectionFactory**:

***Campuri***: - “*DRIVER*” - sir de caractere constant utilizat la crearea unei noi instante Driver care il va inregistra cu scopul de a permite conectarea la baza de date.

- “*DBURL*” - sir de caractere constant care reprezinta adresa server-ului si a bazei de date la care se va efectua conectarea.

- “*USER*” - sir de caractere constant care reprezinta numele utilizator pentru conectarea la server-ul in care s-a creat baza de date.

- “*PASS*” - sir de caractere constant care reprezinta parola pentru conectarea la server-ul in carea s-a creat baza de date.

***Constructor***: - “*ConnectionFactory*” - inregistreaza Driver-ul pentru a se putea realiza conexiunea la baza de date.

***Metode***: - “*createConnection*” - returneaza conexiunea unica la baza de date sau genereaz un mesaj de eroare in lipsa realizarii acesteia si opreste executia.

- “*getConnection*” - returneaza unica conexiune la baza de date.

- “close” - aceste metode supraincarcate efectueaza pe rand operatia de inchidere a conexiunii la baza de date, a optiunii de interogare a bazei de date sau a optiunii de captare a rezultatelor din baza de date in urma interogarii.

**Bill**:

***Campuri***: - “*id*” - reprezinta ID-ul clientului pentru care s-a inregistrat chitanta.

- “*clientName*” - reprezinta numele clientului pentru care s-a inregistrat chitanta.

- “productType” - reprezinta tipul produsului pentru care s-a inregistrat chitanta.

- “total” - reprezinta suma totala de plata pentru care s-a inregistrat chitanta.

***Constructor***: - “*Bill*” - instantieaza campurile obiectului creat cu valorile transmise ca parametrii.

***Metode***: - “*toArrayString*” - returneaza un tablou de siruri de caractere care contine campurile si numele tabelei din care face parte obiectul in baza de date cu scopul efectuarii de operatii cu acesta.

**Order**:

***Campuri***: - “*id*” - reprezinta ID-ul produsului pentru care s-a inregistrat factura.

- “*orderQuantity*” - reprezinta cantitatea de produse pentru care s-a inregistrat factura.

- “product” – numele produsului comandat

-“clientName” – numele clientului ce comanda

- “timestamp” - reprezinta data exacta cand s-a generat factura.

***Constructori***: - “*Order(id, orderQuantity)”* - instantieaza campurile obiectului creat cu valorile transmise ca parametrii iar pentru campul “*timestamp*” se genereaza data curenta pentru factura.

**Client:**

***Campuri***: - “*id*” - reprezinta ID-ul unic clientului cu care a fost inserat in baza de date.

- “*name*” - reprezinta numele clientului cu care a fost inserat in baza de date.

- “*address*” - reprezinta adresa clientului cu care a fost inserat in baza de date.

- “email” – reprezinta adresa de email a clientului care a fost inserat

***Constructor***: - “*Client*” - instantieaza campurile obiectului creat cu valorile transmise ca parametrii.

**Product**:

***Campuri***: - “*id*” - reprezinta ID-ul unic produsului cu care a fost inserat in baza de date.

- “type” - reprezinta tipul produsului cu care a fost inserat in baza de date.

- “*quantity*” - reprezinta cantitatea produsului cu care a fost inserat in baza de date.

- “*price*” - reprezinta valoarea produsului (1 bucata) cu care a fost inserat in baza de date.

***Constructor***: - “*Product*” - instantieaza campurile obiectului creat cu valorile transmise ca parametrii

• Clasa AbstractDAO

Aceasta este o clasă abstractă menită să implementeze generic operațiile de bază efectuate pe baza de date, cum ar fi găsiți prin id, introduceți, actualizați și ștergeți prin id. Oferă metoda pentru toate operațiunile specificate mai sus implementate ca implicite. Pentru fiecare clasă de model ar trebui creată o nouă clasă DAO care va extinde clasa AbstractDAO. Noua clasă DAO ar trebui să completeze setul de operații necesare pentru tabelul reprezentat.

* findbyId(id) – O metoda prin care vom cauta un obiect de tip type, in baza de date, care va avea ca id parametrul dat si vom returna obiectul gasit sau null, daca acesta nu se afla in baza de date
* findAll() - O metoda prin care vom prelua toate obiectele de tip type din baza de date si vom returna o lista de obiecte de acelasi tip.
* insert () - O metoda prin care se realizeaza un statement, dupa care se executa cu scopul de a insera intr-un tabel (type), un obiect de tip T. Am folosit reflexia astfel incat putem utiliza aceasta metoda pentru a insera orice tip de obiect (existent in baza de date) in orice tabel (atata timp cat corespund)
* update () - In aceasta metoda realizam si executam un statement prin care vom face update unui obiect de tip T existent in baza de date. Am folosit reflexia pentru a putea folosi aceasta metoda pentru a face update tuturor obiectelor de un anumit tip dat (atata tip cat exista in baza de date)
* delete () - In aceasta metoda realizam si executam un statement prin care vom sterge un obiect de tip T

existent in baza de date. Am folosit reflexia pentru a putea folosi aceasta metoda pentru a realiza stergerea tuturor obiectelor de un anumit tip dat (atata tip cat exista in baza de date)

• ClientDAO

Această clasă extinde clasa AbstractDAO

• ProductDAO

Această clasă extinde clasa AbstractDAO

• OrderDAO

Această clasă extinde clasa AbstractDAO și implementează operațiunile necesare rămase necesare pentru tabelul comenzilor. Metoda getMessage imi genereaza factura pentru o comanda in format pdf.

• ClientBLL

Această clasă funcționează cu clasele ClientsDAO și OrdersDAO descrise anterior și oferă o vedere mai abstractă a operațiunilor. Rolul acestei clase este de a controla accesul la date și de a filtra datele procesate astfel încât să nu apară incoerențe în baza de date. Acesta oferă metodele necesare pentru inserarea, ștergerea și actualizarea tabelului clienților, precum și metode pentru obținerea id-ului unui client și generarea raportului clienților.

• ProductBLL

Această clasă funcționează cu clasele ProductsDAO și OrdeItemsDAO descrise anterior și oferă o vedere mai abstractă a operațiunilor. Acesta oferă metodele necesare pentru inserarea, ștergerea și actualizarea tabelului de produse, precum și metode pentru obținerea id-ului unui produs și generarea raportului clienților.

• OrderBLL

Această clasă funcționează cu clasele DAO descrise anterior și oferă o vedere mai abstractă a operațiunilor. Acesta oferă metodele necesare pentru inserarea, ștergerea și actualizarea tabelului de comenzi, precum și metode de generare a raportului comenzilor și a raportului pentru factură.

* StartPage, ClientPage, ProductPage, OrderPage

Aceste clase imi realizeaza partea de interfata grafica prin realizarea de “initializare” a frame-ului principal (StartPage) dupa care in functie de butonul selectat se va “goli” panel-ul actual, si se vor apela metodele initialize din cadrul clasei corespunzatoare.

De asemenea in pachetul presentation avem si clasele AbstractJTable in cadrul carora vom genera un JTable ce va contine field-urile si datele unui anumit obiect si este afisat in cadrul interfetei grafice.De aici clasele ProductTable si ClientTable vor extinde aceasta clasa pentru a putea particulariza acest tabel.

1. **Concluzii**

Din aceasta tema s-au putut invatat conceptele de baza in legatura cu gestionarea unei baze de date prin intermediul limbajului de programare JAVA, cu ajutorul unori comenzi simple prin care se realizeaza conectare la baza de date, selectia, stergerea, improspatarea sau inserarea unor tuple de date. De asemenea s-au stabilit concepte noi in legatura cu termeni precum “layerd architecture” sau “reflection techniques” care conduc la perfectionarea crearii unui program JAVA din punct de vedere al eficientei metodelor dar si lizibilitatea programului care conduce la un debugging mai usor.

Acest program poate fi dezvoltat mai departe prin introducerea unor instructiuni care sorteaza si grupeaze tuplele rezultate, optiunea de creare a triggerelor si a procedurilor dar si a generarii unei interfete grafice cu utilizatorul pentru a usura realizarea acestor procese.

1. **Bibliografie**

- notitele din cursurile si laboratoarele materiei “Tehnici de programare fundamentale”

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

- http://tutorials.jenkov.com/java-reflection/index.html

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

- https://dev.mysql.com/doc/workbench/en/wb-admin-export-import-management.html

-<http://tutorials.jenkov.com/java-reflection/index.html>