**DOCUMENTAȚIE TEMA 3**

**ORDER MANAGEMENT**

**Ovreiu Auraș Dănuț**

**Grupa 30229**

**Profesor Laborator Assist Antal Marcel**

# 

Contents

[1. Cerințe funcționale 3](#_Toc69978219)

[2. Obiective 3](#_Toc69978220)

[2.1. Obiectivul principal 3](#_Toc69978221)

[2.2. Obiective secundare 3](#_Toc69978222)

[3. Analiza problemei 4](#_Toc69978223)

[4. Proiectare 4](#_Toc69978224)

[4.1. Structuri de date 4](#_Toc69978225)

[4.2. Algoritmi 5](#_Toc69978226)

[4.3. Diagrame de clase 5](#_Toc69978227)

[5. Implementare 6](#_Toc69978228)

[5.1. Clase 6](#_Toc69978229)

[5.2. Metode 8](#_Toc69978230)

[6. Testare 9](#_Toc69978231)

[7. Concluzii și Dezvoltări Ulterioare 10](#_Toc69978232)

[8. Bibliografie 10](#_Toc69978233)

# 

# 

**1. Cerințe funcționale**

La această temă se cere implementarea unui program care să simuleze comenzile clienților către un depozit. Acest program trebuie să conțină minim 3 tabele, și anume Client, Product și Order. Se cere de asemenea ca datele să fie extrase din tabele create într-o bază de date(în mysql de exemplu), iar vizualizarea evoluției clienților, order-urilor și a produselor să se realizeze prin câte o fereastră în interfața grafică. La crearea unui nou order valid(clientul și produsul specificat să existe în baza de date, iar cantitatea din order să fie mai mică sau egală decât cea din produs) va trebui actualizată cantitatea rămasă din produs.

# 2. Obiective

## 2.1. Obiectivul principal

Obiectivul principal al acestei teme este proiectarea și implementarea unei aplicații care să simuleze servirea unor clienți cu niște produse, evoluția acestora trebuind să fie vizibilă atât în baza de date, cât și în interfața grafică.

## 2.2. Obiective secundare

|  |  |  |
| --- | --- | --- |
| **Obiectiv Secundar** | **Descriere** | **Capitol** |
| Descrierea cerinței | se prezintă sumar funcționalitatea proiectului | 3 |
| Alegerea structurilor de date | se aleg structuri de date care permit stocarea clienților, produselor și a order-urilor, precum și gestionarea acestora | 4 |
| Împărțirea pe clase | proiectul se împarte pe clase grupate în pachete astfel încât să se poată afișa rezultatele obținute în simulare pe interfața grafică, dar și în baza de date în timp real, totul petrecându-se prin apăsarea unui buton dintr-o fereastră | 4 |
| Dezvoltarea algoritmilor | se vor implementa algoritmi de reflexie pentru o abordare mai generală a problemei, clasele reprezentative tabelelor fiind tratate ca o singură clasă | 4 |
| Implementarea soluției | se va descrie fiecare clasă cu metode si câmpuri importante, precum și implementarea interfeței utilizator | 5 |
| Testare | se verifică funcționarea corectă a aplicației prin vizualizarea tabelelor din baza de date în interfața utilizator(JTable) | 6 |

# 

# 3. Analiza problemei

În cadrul acestei teme sunt impuse o serie de specificații, după cum urmează:

* folosirea javadoc pentru documentarea claselor
* crearea unei baze de date care să conțină câte un tabel pentru clienți, produse și respectiv order-uri
* crearea unei interfețe grafice care să cuprindă câte o fereastră pentru fiecare tabel și care să permită efectuarea principalelor operații pe baza de date(inserare, ștergere, actualizare, selecție)
* utilizarea tehnicilor de reflexie pentru generarea tabelelor, precum și pentru tratarea mai generală a soluției(crearea unei clase care să fie moștenită de clasele corespunzătoare bazei de date)
* respectarea arhitecturii layered
* crearea unei facturi pentru fiecare comandă și afișarea detaliilor despre aceasta într-un fișier pdf sau text

Modelarea problemei a fost facută în mare parte după exemplul alăturat prezentării temei. Fiecare tabel are asignată o clasă care are ca atribute coloanele tabelului. Modelarea a fost facută pe straturi, astfel încât sunt 4 nivele de execuție, fiecare fiind reprezentat printr-un pachet:

* Model classes – clasele care poartă numele tabelelor din baza de date(Client, Product, Orders)
* Business Logic Classes – conține clase ale căror metode pun în execuție principalele operații pet abele, folosite de altfel la interogarea acestora
* Presentation classes – conțin clasele responsabile de afișarea interfeței grafice pe ecran și de vizualizarea modificării acesteia la apăsarea unor butoane.
* Data access classes – acestea sunt clasele care realizează accesul la baza de date, prin implementarea explicită a query-urilor aplicate pet abele

# 4. Proiectare

## 4.1. Structuri de date

Principalele structuri de date folosite sunt lista(List<T> list=new ArrayList<T>()) și array-ul(String []column). Deși lista este mai flexibilă și eficientă în cele mai multe situații, doar array-ul poate fi folosit la apelarea anumitor metode sau instanțierea unor clase, cum ar fi Jtable, care primește ca parametri un array de coloane și o matrice de date(valorile din tabel). Alte structuri de date folosite pentru a reține mai ușor clienți, produsele și comenzile au următoarea formă:

public Client(int id, String name, String address, String email, int age) {  
 super();  
 this.id = id;  
 this.name = name;  
 this.address = address;  
 this.email = email;  
 this.age = age;  
}

public Product(int id, String name, int quantity, int price){  
 super();  
 this.id=id;  
 this.name=name;  
 this.quantity=quantity;  
 this.price=price;  
}

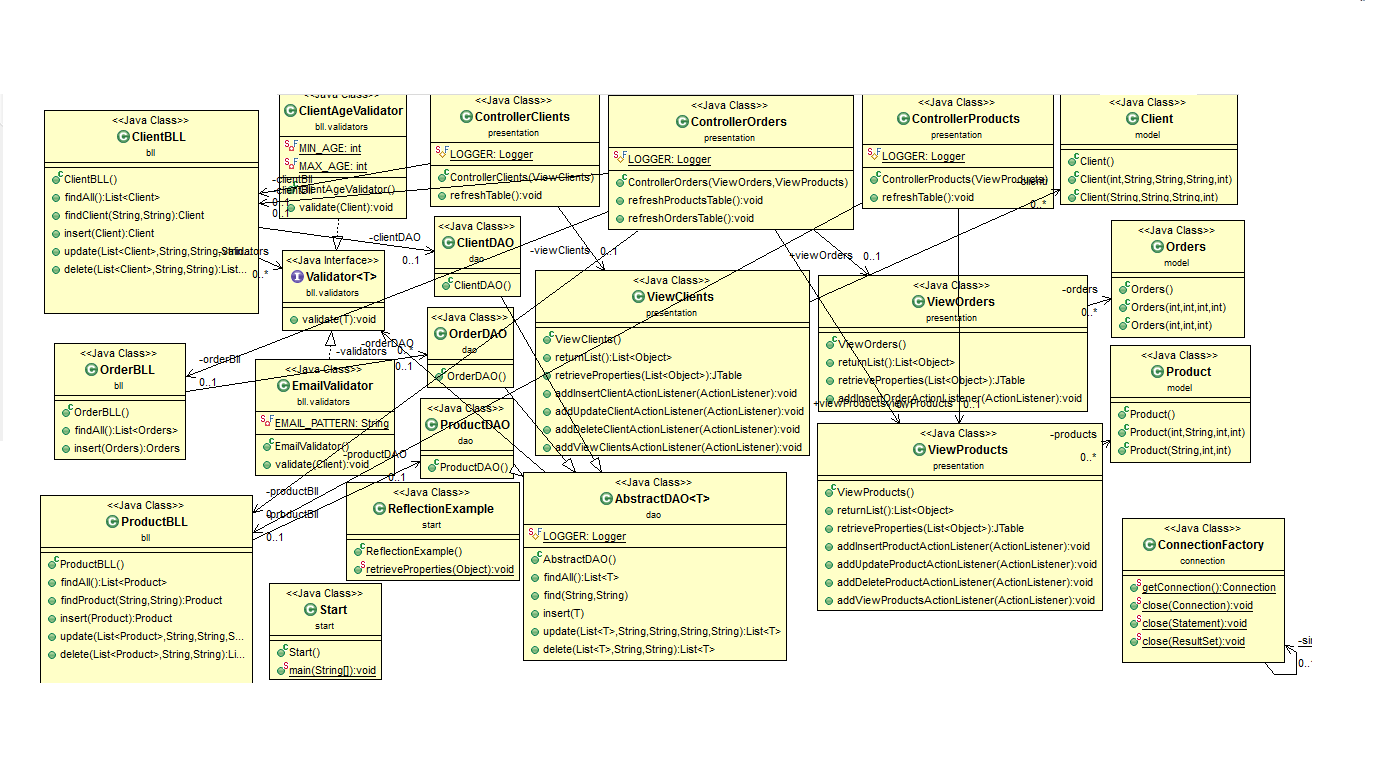
public Orders(int id, int id\_client, int id\_product, int quantity){  
 super();  
 this.id=id;  
 this.id\_client=id\_client;  
 this.id\_product=id\_product;  
 this.quantity=quantity;  
}

## 4.2. Algoritmi

Algoritmii utilizați la această temă sunt cei de reflexie, prin care afișăm proprietățile obiectului. De exemplu, prin utilizarea clasei Field putem parcurge câmpurile clasei de tipul căreia este obiectul primit ca parametru, modificându-le vizibilitatea. Această clasă conține metode prin care se pot extrage atât numele câmpurilor, dar și valorile pe care le iau acestea.

## 4.3. Diagrame de clase

Unified Modeling Language sau UML pe scurt este un limbaj standard pentru descrierea de modele si specificații pentru software. UML a fost la bază dezvoltat pentru reprezentarea complexității programelor orientate pe obiect, al căror fundament este structurarea programelor pe clase, și instanțele acestora ( numite și obiecte ). Cu toate acestea, datorită eficienței și clarității în reprezentarea unor elemente abstracte, UML este utilizat dincolo de domeniul IT.



# 5. Implementare

## 5.1. Clase

***Clase din pachetul start***

***Clasa Start:***

Este clasa în care se realizează deschiderea interfeței grafice prin instanțierea celor 3 ferestre.

***Clasa ReflectionExample:***

Este clasa care face posibilă reflectarea proprietăților obiectelor afișate în interfața grafică, dar și în consolă și in baza de date mysql.

***Clase din pachetul model***

***Clasa Client:***

Este clasa în care ne creăm clienții reprezentați de un ID și caracterizați prin nume, adresă, email și vârstă.

***Clasa Orders:***

Este clasa în care ne creăm comenzile reprezentate de un ID și caracterizate de id-urile unui client și ale unui produs(pentru a fi valide) și de cantitatea de produs cumpărată de către client.

***Clasa Product:***

Este clasa în care ne creăm produsele reprezentate de un ID și caracterizate prin nume, cantitate și preț.

Toate aceste clase conțin getteri și setteri pentru reținerea și respectiv posibila modificare a câmpurilor sale, precum și metoda toString, care are rolul de a afișa cât mai clar și complet informațiile despre clase.

***Clase din pachetul connection***

***Clasa ConnectionFactory:***

Este clasa în care se realizează conexiunea propriu-zisă la baza de date. Informațiile de conectare precum numele de utilizator sau parola sunt reprezentate ca string-uri, iar fiecare conexiune deschisă inițial se închide la finalul clasei prin metode specifice.

***Clase din pachetul dao***

***Clasa AbstractDAO:***

În această clasă se realizează conectarea la baza de date prin query-uri. Fiecare query are în comun faptul că este creat printr-o appenduire de string-uri. În continuare, metodele menționate anterior sunt apelate în alte metode în care query-urile sunt aplicate pe tabelele din baza de date, conexiunea la aceasta din urmă făcându-se prin clasa ConnectionFactory. La fel ca mai înainte, orice conexiune începută trebuie și încheiată, fapt ce este realizat tot prin intermediul clasei anterioare.

Celelalte clase din acest pachet moștenesc clasa AbstractDAO, cea din urmă fiind de fapt forma generală a acestora, fiind mai eficient de lucrat mai general.

***Clase din pachetul bll***

Acest pachet conține clase în care se apelează metodele de prelucrare a tabelelor din clasele incluse în pachetul dao, iar dacă acestea nu se efectuează cu success se va arunca o excepție de tipul NoSuchElementException(nu s-a găsit niciun element în urma execuției unui query).

La una dintre clasele acestui pachet se remarcă existent unei liste care este instanțiată în constructorul implicit. Această listă are rolul de a valida formatul, respectiv valorile anumitor câmpuri. Este vorba despre câmpul email, care trebuie să aibă o structură bine definite și de câmpul age, care poate lua valori în intervalul 7 și 90 pentru ca clientul să poată fi procesat.

***Clase din pachetul presentation***

În acest pachet se regăsesc clasele de tip view și controller, care fac posibilă afișarea pe ecran a interfeței grafice, dar și a evoluției aplicației.

La începutul claselor de view se remarcă o mulțime de variabile instanță declarate private, care sunt de fapt componente ale ferestrelor. Selectarea operației de prelucrare asupra unui table se face relative simplu, prin apăsarea unui buton de tip JButton în interfața grafică, iar mesajele de eroare care pot apărea sunt generate tot din clasele de view, prin textfield-uri. Acestea vor fi completate de un mesaj de eroare corespunzător prin intermediul setterilor.

Clasele de controller pun în execuție ceea ce afost început în clasele de view. La apăsarea butonului, se execută bucata de cod din metoda actionPerformed aferentă butonului, modificările putând fi vizibile imediat după aceea.

## 5.2. Metode

***Metode utilizate în clasa Client***

În această clasă se regăsesc metode pentru reținerea și modificarea caracteristicilor clientului(getteri și setteri). Tot aici întâlnim și metoda toStrting pentru afișarea cât mai corectă a informațiilor despre client(ID-ul său, numele, adresa, email-ul și vârsta).

***Metode utilizate în clasa Product***

În această clasă se regăsesc metode pentru reținerea și modificarea caracteristicilor produsului(getteri și setteri). Tot aici întâlnim și metoda toStrting pentru afișarea cât mai corectă a informațiilor despre produs(ID-ul său, numele, cantitatea și prețul).

***Metode utilizate în clasa Orders***

În această clasă se regăsesc metode pentru reținerea și modificarea caracteristicilor comenzii(getteri și setteri). Tot aici întâlnim și metoda toStrting pentru afișarea cât mai corectă a informațiilor despre comandă(ID-ul său, ID-ul clientului și al produsului selectate, cantitatea cumpărată-în caz că este validă și disponibilă în stock).

***Metode utilizate în clasa ConnectionFactory***

Această clasă conține metode pentru crearea conexiunilor la baza de date(createConnection(), getConnection()), dar și pentru închiderea tuturor conexiunilor existente(close(Connection connection), close(Statement statement),

close(ResultSet resultSet)). Crearea conexiunii la baza de date se realizează prin transmiterea variabilelor reprezentând string-uri cu datele despre conectare ca parametri ai funcției getConnection(). Închiderea conexiunilor se realizează simplu, prin apelarea metodei close() din clasa java.sql.Connection în metodele denumite close.

***Metode utilizate în clasa AbstractDAO***

În această clasă regăsim metodele care realizează prelucrarea datelor din baza de date prin crearea și apelarea diverselor interogări. Acestea din urmă sunt realizate prin crearea unei instanțe a clasei StringBuilder, la care se concatenează string-uri pentru a forma interogarea dorită.

Luăm de exemplu metoda createSelectQuery care crează interogarea ce selectează anumite rânduri dintr-un tabel ce conține un cîmp de o anumită valoare. Numele câmpului este primit ca parametru ca un string, iar numele tabelului se extrage prin apelarea metodei getSimpleName() din clasa T, care poate fi orice clasă. Valoarea field-ului este notată cu “?”, valoarea acesteia urmând să fie dată de către utilizator în interfața grafică.

Printre metodele de realizare a unui query se deosebește metoda createInsertQuery, abordarea acesteia fiind una mai generală decât celelalte. Numele coloanelor tabelului în care se face inserarea se obțin prin reflexie, parcurgându-se array-ul de field-uri printr-o buclă for-each și modificându-se vizibilitatea acestora. Numele coloanelor tabelului se concatenează într-o variabilă la fiecare iterație, punând în variabila care reține valorile de fiecare data semnul întrebării, acestea urmând să fie introduse de utilizator în interfața grafică.

În metodele ce urmează se apelează metodele de conectare la baza de date definite în clasa ConnectionFactory și cele de crearea a interogărilor. Metodele de acest tip care sunt mai atipice sunt cele care șterg și cele care actualizează, fiind create astfel încât să fie posibilă ștergerea și actualizarea mai multor rânduri care respectă conditițiile din query.

În cadrul metodei update se primește ca parametru o listă conținând toate obiectele din baza de date și string-uri cu câmpurile și valorile care sunt luate în calcul la operația de update. Elementele din listă se parcurg printr-o buclă, apoi prin două bucle for-each imbricate se verifică dacă au apărut schimbări la unul din obiecte în urma actualizării. Dacă da, obiectele respective se vor adăuga într-o nouă listă a cărei conținut va fi afișat în consolă pentru a vedea mai bine obiectele care s-au actualizat.

În mod asemănător, în metoda delete se parcurg câmpurile iterativ, iar când se ajunge un câmp al cărui nume coincide cu string-ul primit ca parametru se adaugă într-o nouă listă, pentru a afișa în consolă rândurile pe care le-am șters din tabel.

***Metode utilizate în clasele de view***

În aceste clase realizarea ferestrelor se realizează în principiu în constructor, metodele întâlnite aici fiind getterii și setterii, dar și cele care crează un tabel de tipul JTable.

În metoda returnList() se rețin obiectele a căror informații ar trebui să apară în tabel, iar în metoda retrieveProperties introducem informațiile din listă în tabel prin parcurgerea iterativă a câmpurilor obiectului și reținerea câmpurilor într-un array, iar a valorilor într-o matrice. Tabelul va fi creat pe baza array-ului și a matricii.

***Metode utilizate în clasele de tip controller***

În aceste clase se crează listeneri pentru butoane, prin apăsarea acestora executându-se o operație în baza de date, modificările putând fi observate atât în interfața grafică, dar și în consolă.

O metodă care apare în fiecare clasă de acest gen este refreshTable(), care șterge toate rândurile dintr-un tabel, urmând ca apoi să adauge în JTable toate obiectele din lista modificată în urma operației selectate. Modificările puteau fi văzute fără existența acestei metode doar în baza de date, nu și în JTable.

# 

# 6. Testare

p Testarea în cadrul acestui proiect nu este una precisă și bine definite, verificarea funcționării interogărilor putând fi văzută în baza de date, interfața grafică și în consolă, iar afișarea corectă a informațiilor în JTable-uri este verificată prin vizualizarea tabelelor din baza de date.

# 7. Concluzii și Dezvoltări Ulterioare

Acestă temă m-a ajutat să îmi dezvolt cunoștințele în lucrul cu bazele de date, materie pe care am avut-o semestrul anterior și la care am lucrat cu interogări în sql, conexiunile la baza de date realizându-le însă în php printr-o abordare relativ diferită. Ca și dezvoltări ulterioare, ar putea fi implementate query-uri mai complexe, care nu pot fi generalizate la fel de ușor pentru fiecare clasă în celelalte clase din pachetul dao.

# 8. Bibliografie

1. <https://docs.oracle.com/javase/tutorial/uiswing/components/table.html>

2. Pdf-ul Support Presentation