DOCUMENTATIE

TEMA 3

NUME STUDENT: PiticTudor

GRUPA: 30245

# CUPRINS

[1. Obiectivul temei 3](#_Toc95297885)

[2. Analiza problemei, modelare, scenarii, cazuri de utilizare 3](#_Toc95297886)

[3. Proiectare 3](#_Toc95297887)

[4. Implementare 3](#_Toc95297888)

[5. Rezultate 3](#_Toc95297889)

[6. Concluzii 3](#_Toc95297890)

[7. Bibliografie 3](#_Toc95297891)

# Obiectivul temei

Obiectivul acestei sarcini este de a proiecta și dezvolta o aplicație pentru gestionarea comenzilor clienților într-un depozit folosind tehnici fundamentale de programare. Aplicația trebuie să fie proiectată conform modelului de arhitectură stratificată și să utilizeze principii de programare orientată pe obiecte. Aplicația trebuie să folosească baze de date relaționale pentru stocarea datelor și trebuie create cel puțin trei tabele: Client, Produs și Comandă. Aplicația trebuie să includă o interfață grafică de utilizator cu ferestre pentru operațiunile client și produse, precum și pentru crearea comenzilor de produse.

Atribuirea necesită utilizarea diferitelor tehnici de programare, cum ar fi tehnici de reflecție, pentru a crea clase generice pentru accesarea bazei de date și pentru generarea dinamică de interogări pentru anumite obiecte. În plus, o clasă Bill trebuie să fie definită în pachetul Model folosind înregistrări Java, care sunt imuabile și generează un obiect factura pentru fiecare comandă care este stocată într-un tabel *Log*.

# Analiza problemei, modelare, scenarii, cazuri de utilizare

Cerințele funcționale ale proiectului sunt:

1. Limitarea lungimii claselor la maximum 300 de linii și lungimea metodelor la maximum 30 de linii.
2. Urmați convențiile de denumire Java.
3. Utilizați Java Doc pentru documentarea claselor și generați fișierele Java Doc corespunzătoare.
4. Folosiți baze de date relaționale pentru stocarea datelor pentru aplicație.
5. Minim trei tabele: Client, Produs și Comanda.
6. Crearea unei interfețe care să includă:
   1. O fereastră pentru operațiunile client: adăugați un client nou, editați clientul, ștergeți clientul, vizualizați toți clienții într-un tabel (JTable).
   2. O fereastră pentru operațiunile produsului: adăugați un produs nou, editați produsul, ștergeți produsul, vizualizați toate produsele într-un tabel (JTable).
   3. O fereastră pentru crearea comenzilor de produse:
      1. Utilizatorul va putea să selecteze un produs existent, să selecteze un client existent și să introducă o cantitate dorită pentru produs pentru a crea o comandă validă.
      2. În cazul în care nu sunt suficiente produse, va fi afișat un mesaj de stoc insuficient.
      3. După finalizarea comenzii, stocul de produse este diminuat.
7. Folosiți tehnici de reflectare pentru a crea o metodă care primește o listă de obiecte și generează antetul tabelului prin extragerea prin reflectare a proprietăților obiectului și apoi populează tabelul cu valorile elementelor din listă.
8. Creați o clasă Bill imuabilă în pachetul Model folosind înregistrări Java.
9. Un obiect Bill va fi generat pentru fiecare comandă și va fi stocat într-un tabel Log.
10. Facturile pot fi introduse și citite numai din tabelul Log; nu sunt permise actualizări.
11. Utilizați tehnici de reflecție pentru a crea o clasă generică care conține metodele de accesare a DB (toate tabelele cu excepția Log):
    1. Creați obiect.
    2. Editați obiectul.
    3. Ștergeți obiectul.
    4. Găsiți obiect.
12. Interogările pentru accesarea DB pentru un anumit obiect care corespunde unui tabel vor fi generate dinamic prin reflecție.
13. Urmați modelul Layered Architecture cu cel puțin patru pachete: dataAccessLayer, businessLayer, model și prezentare.

# Proiectare

Ideea din spatele proiectării a acestei aplicații a fost de a folosi pachete si clase care descriu cat mai bine funcționalitatea părților din aplicație, dar in același timp pentru a păstra o eficienta in scrierea de cod. Din acest motiv, aplicația conține cinci pachete: *model*, *presentation*, *service*, *dao* și *connection*.

Pentru a evita scrierea decod boiler plate, s-a folosit dependenta de biblioteca „Lombok” pentru a scurta scrierea de constructori, metode getter si metode setter.

Pentru implementarea soluției, s-au creat 3 tabele care reprezintă entitățile din aplicație: *clients*, *orders* și *products*. Tabelele conțin date legate de parametrii fiecărei entități: pentru clienți se înregistrează numele și email-ul; pentru produse se înregistrează numele, prețul și cantitatea din inventar; iar pentru comenzi se înregistrează clientul comenzii, produsele din comandă și prețul total. Pentru a crea o relație many-to-many între tabela *orders* și tabela *products*, s-a creat o tabelă intermediară numită *order\_products* care înregistrează care produse se află într-o comandă. Pe lângă id-ul produsului, tabela intermediară conține și cantitatea de produs introdus într-o comandă.

Diagrama UML a bazei de date:

Diagram

Description automatically generated

# Implementare

Pentru implementarea soluției, trebuie creată o aplicație care să poată comunica cu baza de date. Pentru a face acest lucru este necesar un driver care sa permită conexiunea la baza de date prin porturile specifice.

**ConnectionFactory**:

Pentru a stabili o conexiune, se folosește biblioteca JDBC. Cu ajutorul acestei biblioteci, se creează o clasă *ConnectionFactory* aflată în pachetul *connection* care stabilește conexiunea la baza de date conform unor constante care reprezintă portul, adresa și driver-ul bazei de date. Această clasă este proiectată conform modelului *Singleton*, așadar o singură conexiune poate fi stabilită la un moment dat. Clasa conține metode de creare și închidere a conexiunii.

**Pachetul model**:

Pachetul model conține clasele care reprezintă entitățile din aplicație, în acest caz conține clasele *Client*, *Product*, *Order* și *Bill*. În aplicație, obiectele din aceste clase reprezintă entitățile logice care se manipulează in pachetul de servicii, iar apoi se salvează în baza de date folosind obiectele pentru acces la date

**Data Access Objects**:

Principala clasă din pachetul *dao* este *AbstractDAO* care implementează metode generice de salvare, vizualizare, actualizare și ștergere a datelor din baza de date folosind tehnici de reflexie pentru a extrage numele parametrilor și datele aferente acestora. Doar clasa OrderDAO are metode implementate, nu moștenite, pentru că entitatea care reprezintă o comenzile are o relație many-to-many. Aceste clase conțin și metode ajutătoare care sunt mai particulare pentru baza de date aleasă (PostgreSQL), cum ar fi:

/\*\*  
 \* This is a helper method when using Postgresql database. Postgresql uses snake\_case for the column names, but the  
 \* java convention is camelCase. This method converts a camelCase string to a snake\_case string.  
 \* @param camelCase The camelCase string to be converted  
 \* @return The snake\_case string  
 \*/  
protected String camelCaseToSnakeCase(String camelCase) {  
 String regex = "([a-z])([A-Z]+)";  
 String replacement = "$1\_$2";  
 return camelCase.replaceAll(regex, replacement).toLowerCase();  
}

/\*\*  
 \* This is a helper method when using Postgresql that returns the name of the table in the database. The name of  
 \* the table is the plural of the entity name, represented by the entity class in the model package, in lowercase.  
 \* For example, if the entity is "User", the table name is "users".  
 \* @return The name of the table in the database  
 \*/  
protected String getTableName() {  
 return type.getSimpleName().toLowerCase() + "s";  
}

# Rezultate

Aplicația funcționează predictibil, aceasta are 4 pagini: un dashboard și cate o pagină pentru fiecare entitate. Pentru entitatea de vizualizare a comenzilor există și o zonă în care se poate genera chitanța pentru fiecare comanda.

Graphical user interface

Description automatically generated

# Concluzii

În concluzie, proiectul Managementul Comenzilor prezintă o oportunitate provocatoare, dar plină de satisfacții de a aplica tehnici fundamentale de programare într-o aplicație din lumea reală. Prin implementarea unui model de arhitectură stratificat, folosind baze de date relaționale și proiectând clase cu principii OOP bune, am câștigat experiență valoroasă în dezvoltarea de software care poate fi întreținut, scalabil și eficient.

# Bibliografie

*Se vor adauga referintele care au fost consultate de student pe parcursul implementarii temei .*

*Exemplu:*

1. *Bruce Eckel, Thinking in Java (4th Edition), Publisher: Prentice Hall PTRUpper Saddle River, NJUnited States, ISBN:978-0-13-187248-6 Published:01 December 2005.*
2. *What are Java classes? -* [*www.tutorialspoint.com*](http://www.tutorialspoint.com)