DOCUMENTAȚIE

TEMA 3

NUME SI PRENUME: MOȘILĂ LUCIANA

GRUPA: 30226

CUPRINS

1.Obiectivul temei

Obiectivul temei este să proiectăm și să implementăm o aplicație de management al comenzilor pentru procesarea comenzilor clienților într-un depozit. Aceasta va utiliza baze de date relationale pentru a stoca informații despre produse, clienți și comenzi. Aplicația va fi proiectată conform pattern-ului de arhitectură stratificată și va utiliza, în mod minim, următoarele clase:

- Clasele model - reprezintă modelele de date ale aplicației

- Clasele de logica de afaceri - conțin logica aplicației

- Clasele de prezentare - clase legate de interfața grafică a utilizatorului (GUI)

- Clasele de acces la date - clase care facilitează accesul la baza de date.

2. Analiza problemei, modelare, scenarii, cazuri de utilizare

Cadrul de cerințe funcționale și non-funcționale al aplicației Orders Management este următorul:

Cerințe funcționale:

1. Gestionarea clienților:

   a. Adăugare client nou.

   b. Editare informații client.

   c. Ștergere client.

   d. Vizualizare toți clienții într-un tabel (JTable).

2. Gestionarea produselor:

a. Adăugare produs nou.

b. Editare informații produs.

c. Ștergere produs.

d. Vizualizare toate produsele într-un tabel (JTable).

3. Crearea comenzilor de produse:

a. Selectarea unui produs existent.

b. Selectarea unui client existent.

c. Introducerea unei cantități dorite pentru produs pentru a crea o comandă validă.

d. Verificarea disponibilității în stoc a produsului. În cazul în care nu există suficiente produse, se va afișa un mesaj de sub-stoc.

e. După finalizarea comenzii, stocul produsului va fi decrementat.

4. Generarea automată a interfeței grafice:

a. Generarea unei ferestre pentru operațiile clientului.

b. Generarea unei ferestre pentru operațiile produsului.

c. Generarea unei ferestre pentru crearea comenzilor de produse.

Cerințe non-funcționale:

1. Implementarea design-ului orientat pe obiecte.

2. Respectarea convențiilor de denumire Java.

3. Utilizarea javadoc pentru documentarea claselor și generarea fișierelor JavaDoc corespunzătoare.

4. Utilizarea bazelor de date relaționale pentru stocarea datelor aplicației, cu cel puțin trei tabele: Client, Product și Order.

5. Implementarea unei interfețe grafice care să includă ferestrele menționate anterior.

6. Utilizarea tehnicilor de reflexie pentru generarea automată a metodei care primește o listă de obiecte și generează antetul tabelului, extrăgând prin reflexie proprietățile obiectului, iar apoi populează tabelul cu valorile elementelor din listă.

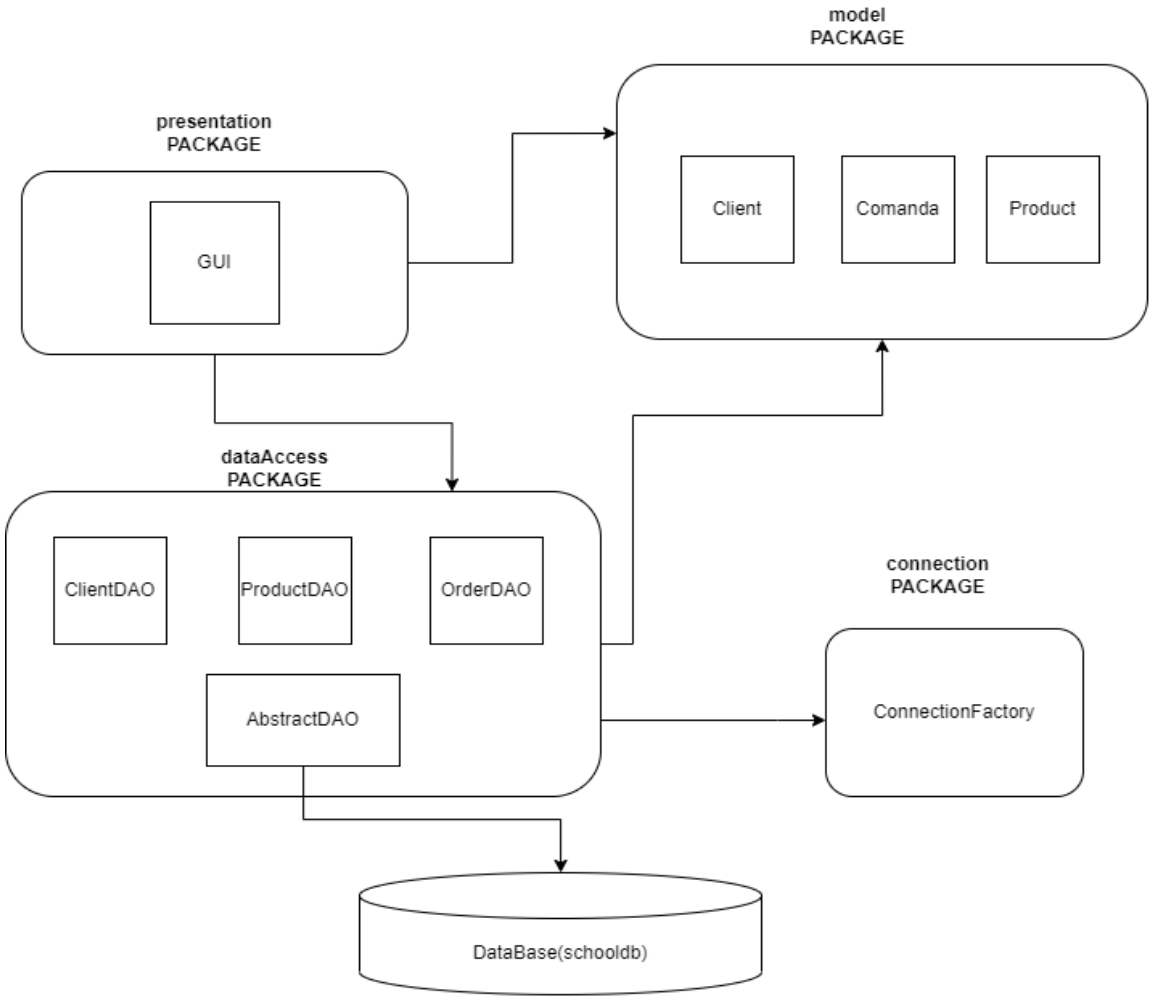
7. Documentație de înaltă calitate care acoperă secțiunile din șablonul de documentație.

8. Arhitectură stratificată a aplicației, cu cel puțin patru pachete: dataAccessLayer, businessLayer, model și presentation.

9. Definirea unei clase Bill imutabile în pachetul Model, utilizând înregistrările Java. Un obiect Bill va fi generat pentru fiecare comandă și va fi stocat într-un tabel Log. Facturile pot fi doar inserate și citite din tabelul Log, nu sunt permise actualizări.

3. Proiectare

Diagrama UML de clase este urmatoarea:



Package dataAccess

Class AbstractDAO<T>

java.lang.Object
    dataAccess.AbstractDAO<T>

Direct Known Subclasses:

ClientDAO, OrderDAO, ProductDAO

```
public class AbstractDAO<T>
extends Object
```

Constructor Summary

Constructors

Constructor	Description
AbstractDAO()	

Method Summary

All Methods | Instance Methods | Concrete Methods

Modifier and Type	Method	Description
List<T>	createObjects(ResultSet resultSet)	
String	createUpdateQuery()	
void	delete(int id)	
List<T>	findAll()	
T	findById(int id)	
T	insert(T t)	
T	update(T t)	

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Details

AbstractDAO

```
public AbstractDAO()
```

Method Details

insert

```
public T insert(T t)
```

delete

```
public void delete(int id)
```

4. Implementare

Clasa `AbstractDAO<T>` este o clasă generică care furnizează metode generice pentru a accesa și manipula date într-o bază de date. Această clasă servește ca o clasă de bază pentru alte clase DAO (Data Access Object) specifice entităților.

Campuri:

- `LOGGER`: Un obiect de tip `Logger` folosit pentru înregistrarea mesajelor de jurnalizare. Este declarat ca `protected` pentru a fi accesibil în clasele derivate.

- `type`: Un obiect de tip `Class<T>` care reprezintă clasa generică `T` specificată la instanțierea clasei `AbstractDAO`.

Metode importante:

- `AbstractDAO()`: Constructorul clasei. Acesta obține tipul generic `T` din clasa derivată și îl stochează în câmpul `type`.

- `insert(T t)`: Inserează un obiect de tip `T` în baza de date. Extrage informațiile despre obiectul `t` și generează și execută un query de inserare. Returnează obiectul inserat cu ID-ul generat.

- `createInsertQuery()`: Generează și returnează un string care reprezintă un query de inserare pentru clasa `T`.

- `delete(int id)`: Șterge un obiect din baza de date pe baza unui ID dat. Generează și execută un query de ștergere.

- `createDeleteQuery()`: Generează și returnează un string care reprezintă un query de ștergere pentru clasa `T`.

- `findAll()`: Returnează o listă cu toate obiectele de tip `T` din baza de date. Generează și execută un query de selectare.

- `findById(int id)`: Găsește și returnează un obiect de tip `T` din baza de date pe baza unui ID dat. Generează și execută un query de selectare după ID.

- `createSelectQuery(String field)`: Generează și returnează un string care reprezintă un query de selectare pentru clasa `T`, filtrat după un câmp specificat.

- `update(T t)`: Actualizează un obiect de tip `T` în baza de date. Extrage informațiile despre obiectul `t` și generează și execută un query de actualizare.

- `createUpdateQuery()`: Generează și returnează un string care reprezintă un query de actualizare pentru clasa `T`.

- `createObjects(ResultSet resultSet)`: Creează obiecte de tip `T` pe baza rezultatelor obținute dintr-un `ResultSet`. Folosește reflexia pentru a inițializa câmpurile obiectelor.

Aceasta este clasa `AbstractDAO` care oferă funcționalitate generică pentru accesul la baza de date. O clasă derivată ar trebui să specifice tipul generic `T` prin extinderea acestei clase și să implementeze metode specifice pentru operațiunile pe care le necesită entitatea respectivă.

Clasa "Client" este un model care reprezintă obiectele de tip client în aplicație. Aceasta conține variabilele de instanță clientId, name, contact și address, care stochează informațiile relevante despre un client. Clasa oferă metode pentru accesarea și modificarea acestor variabile.

Clasa "Order" este un model care reprezintă obiectele de tip comandă în aplicație. Aceasta conține variabilele de instanță orderId, client, product și quantity, care rețin informațiile specifice unei comenzi. Clasa oferă metode pentru accesarea și modificarea acestor variabile.

Clasa "Product" este un model care reprezintă obiectele de tip produs în aplicație. Aceasta conține variabilele de instanță productId, name, price și stock, care conțin informații despre un produs specific. Clasa oferă metode pentru accesarea și modificarea acestor variabile.

Clasa "ConnectionFactory" este responsabilă de gestionarea conexiunii la baza de date. Aceasta utilizează driverul JDBC pentru a realiza conexiunea la o bază de date MySQL locală. Clasa oferă metode pentru crearea unei conexiuni, închiderea conexiunii, închiderea declarațiilor SQL și închiderea setului de rezultate.

Clasa GUI este responsabilă de implementarea unei interfețe grafice pentru un sistem de comenzi de produse. Aceasta facilitează interacțiunea utilizatorului cu sistemul și permite efectuarea diverselor acțiuni, cum ar fi adăugarea, modificarea și ștergerea de produse.

5.Rezultate

Am testat și verificat aplicația de mai multe ori pentru fiecare scenariu și caz de utilizare, examinând toate funcționalitățile pe care le are, atât în baza de date cât și în interfața utilizatorului.

6.Concluzii

Din acest proiect, am învățat următoarele aspecte importante:

Integrarea dintre MySQL și Java, în special lucrul cu baze de date.

Am asimilat noul design conceptual al programului.

Am dobândit cunoștințe în utilizarea tehnicilor de reflexie.

Am învățat cum să generăm documentație Javadoc.

Dezvoltări posibile pentru proiect:

O îmbunătățire ar fi ca aplicația să primească confirmarea comenzii prin intermediul metodei de contact a clientului, indiferent dacă comanda a fost trimisă, anulată sau modificată.

Eficientizarea metodelor existente.

Îmbunătățirea aspectului vizual al interfeței.

Concluzie:

Proiectul implementează o aplicație de gestionare a produselor cu un design bazat pe programarea orientată pe obiecte. S-au respectat practici precum limitarea dimensiunii claselor și metodelor, utilizarea convențiilor de denumire în Java și documentarea codului utilizând Javadoc, cu generarea fișierelor JavaDoc corespunzătoare.

Pentru stocarea datelor aplicației, s-a folosit o bază de date relațională care conține cel puțin trei tabele: Client, Product și Order. Interfața grafică include ferestre dedicate pentru operațiuni cu clienți și produse, permitând adăugarea, modificarea și ștergerea acestora, precum și afișarea lor în tabele (JTable). De asemenea, a fost creată o fereastră pentru crearea comenzilor de produse, în care utilizatorul poate selecta un produs existent, un client existent și poate introduce cantitatea dorită pentru a crea o comandă validă. În cazul în care nu există suficiente produse în stoc, se afișează un mesaj de sub-stocare. După finalizarea comenzii, stocul produsului este actualizat prin decrementare.

S-a utilizat tehnica de reflexie pentru a crea o metodă care primește o listă de obiecte și generează antetul tabelului, extrăgând prin reflexie proprietățile obiectului și populând tabelul cu valorile elementelor din listă.

7. Bibliografie

1. https://dsrl.eu/courses/pt/materials/PT2023_A3_S1.pdf
2. https://dsrl.eu/courses/pt/materials/PT2023_A3_S2.pdf
3. https://users.utcluj.ro/~igiosan/teaching_poo.html
4. https://gitlab.com/utcn_dsrl/pt-layered-architecture
5. https://jenkov.com/tutorials/java-reflection/index.html