DOCUMENTATIE

TEMA Menagement-ul unui depozit

NUME STUDENT: Sipos Bogdan
GRUPA: 30225

CUPRINS

1. Obiectivul temei.....3

2. Analiza problemei, modelare, scenarii, cazuri de utilizare 3

3. Proiectare.....10

4. Implementare.....10

5. Concluzii 17

6. Bibliografie.....17

1. Obiectivul temei

Obiectivul principal este de a proiecta si implementa o aplicatie pentru gestionarea comenzilor clientilor pentru un deposit.

Obiectiv Secundar	Descriere	Capitol
Analiza Problemei și Identificarea Cerințelor	Definirea cerințelor funcționale și non-funcționale. Identificarea funcționalităților cheie necesare pentru gestionarea comenzilor clienților în depozit.	2
Proiectarea Aplicației de Gestionare a Comenzilor	Dezvoltarea designului orientat pe obiecte (OOD) al aplicației. Crearea diagramelor UML pentru clase și pachete. Definirea structurilor de date și a interfețelor necesare pentru gestionarea comenzilor.	3
Implementarea Aplicației de Gestionare a Comenzilor	Dezvoltarea aplicației bazate pe OOD. Implementarea claselor și metodelor pentru gestionarea comenzilor clienților, inventarului depozitului și procesării comenzilor. Integrarea componentelor interfeței de utilizator pentru gestionarea eficientă a comenzilor.	4
Testarea Aplicației de Gestionare a Comenzilor	Dezvoltarea scenariilor de testare pentru diferite funcționalități ale aplicației. Executarea testelor unitare pentru a asigura corectitudinea și fiabilitatea funcționalităților implementate. Realizarea testării de integrare pentru a valida interacțiunile dintre diferitele componente ale aplicației.	5

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

Scenariu de Utilizare: Adăugare Produs

Actor Principal: Angajat

Scenariul Principal:

1. Angajatul selectează opțiunea de adăugare a unui produs nou.

2. Aplicația afișează un formular în care trebuie introduse detaliile produsului.

3. Angajatul introduce numele produsului, prețul și cantitatea.

4. Angajatul apasă pe butonul "Add New Product".

5. Aplicația salvează datele produsului în baza de date și afișează un mesaj de confirmare.

Secvență Alternativă: Valori nevalide pentru datele produsului

- Utilizatorul introduce o valoare negativă pentru stocul produsului.

- Aplicația afișează un mesaj de eroare și solicită utilizatorul să introducă un stoc valid.

- Scenariul se întoarce la pasul 3.

Scenariu de Utilizare: Stergere Produs

Actor Principal: Angajat

Scenariul Principal:

1. Angajatul selectează opțiunea de stergere a unui produs nou.

2. Aplicația afișează un formular în care trebuie introduse detaliile produsului.

3. Angajatul introduce numele clientului, email-ul și adresa.

4. Angajatul apasă pe butonul "Delete Product".

5. Aplicația sterge datele produsului în baza de date și afișează un mesaj de confirmare.

Secvență Alternativă: Valori nevalide pentru datele produsului

- Utilizatorul nu alege un produs pe care sa il stearga.

- Scenariul se întoarce la pasul 3.

Scenariu de Utilizare: Update Product

Actor Principal: Angajat

Scenariul Principal:

1. Angajatul selectează opțiunea de updatare a unui produs nou.

2. Aplicația afișează un formular în care trebuie introduse detaliile produsului.

3. Angajatul introduce numele produsului, prețul și cantitatea.

4. Angajatul apasă pe butonul "Update Product".

5. Aplicația salvează datele produsului în baza de date și afișează un mesaj de confirmare.

Secvență Alternativă: Valori nevalide pentru datele produsului

- Utilizatorul introduce o valoare negativă pentru stocul produsului.

- Aplicația afișează un mesaj de eroare și solicită utilizatorul să introducă un stoc valid.

- Scenariul se întoarce la pasul 3.

Scenariu de Utilizare: Adăugare Client

Actor Principal: Angajat

Scenariul Principal:

1. Angajatul selectează opțiunea de adăugare a unui client nou.

2. Aplicația afișează un formular în care trebuie introduse detaliile clientului.

3. Angajatul introduce numele clientului, adresa și email-ul.

4. Angajatul apasă pe butonul "Add New Client".

5. Aplicația salvează datele clientului în baza de date și afișează un mesaj de confirmare.

Secvență Alternativă: Valori nevalide pentru datele clientului

- Utilizatorul nu introduce un email valid.

- Aplicația afișează un mesaj de eroare și solicită utilizatorul să introducă un email valid.

- Scenariul se întoarce la pasul 3.

Scenariu de Utilizare: Ștergere Client

Actor Principal: Angajat

Scenariul Principal:

1. Angajatul selectează opțiunea de ștergere a unui client existent.

2. Aplicația afișează o listă cu clienții existenți din care trebuie selectat clientul ce urmează să fie șters.

3. Angajatul selectează clientul dorit pentru ștergere.

4. Angajatul apasă pe butonul "Delete Client".

5. Aplicația șterge datele clientului din baza de date și afișează un mesaj de confirmare.

Secvență Alternativă: Client inexistent

- Utilizatorul încearcă să șteargă un client care nu există în baza de date.

- Aplicația afișează un mesaj de eroare și solicită utilizatorul să selecteze un client existent.

- Scenariul se întoarce la pasul 2.

Scenariu de Utilizare: Actualizare Client

Actor Principal: Angajat

Scenariul Principal:

1. Angajatul selectează opțiunea de actualizare a datelor unui client.

2. Aplicația afișează o listă cu clienții existenți din care trebuie selectat clientul ce urmează să fie actualizat.

3. Angajatul selectează clientul dorit pentru actualizare.

4. Angajatul modifică informațiile clientului cum ar fi numele, adresa sau email-ul.

5. Angajatul apasă pe butonul "Update Client".

6. Aplicația actualizează datele clientului în baza de date și afișează un mesaj de confirmare.

Secvență Alternativă: Valori nevalide pentru datele clientului

- Utilizatorul introduce un email invalid.

- Aplicația afișează un mesaj de eroare și solicită utilizatorul să introducă un email valid.

- Scenariul se întoarce la pasul 4.

Scenariu de Utilizare: Adăugare Comandă

Actor Principal: Angajat

Scenariul Principal:

1. Angajatul selectează opțiunea de adăugare a unei noi comenzi.

2. Aplicația afișează un formular în care trebuie introduse detaliile comenzii.

3. Angajatul introduce numele clientului, denumirea produsului și cantitatea comandată.

4. Angajatul apasă pe butonul "Adăugare Comandă".

5. Aplicația salvează datele comenzii în baza de date și afișează un mesaj de confirmare.

Secvență Alternativă: Cantitate invalidă pentru comandă

- Utilizatorul introduce o cantitate negativă sau zero pentru produsul comandat.

- Aplicația afișează un mesaj de eroare și solicită utilizatorul să introducă o cantitate validă.

- Scenariul se întoarce la pasul 3.

Scenariu de Utilizare: Ștergere Comandă

Actor Principal: Angajat

Scenariul Principal:

1. Angajatul selectează opțiunea de ștergere a unei comenzi existente.

2. Aplicația afișează o listă cu comenzile existente din care trebuie selectată comanda ce urmează să fie ștearsă.

3. Angajatul selectează comanda dorită pentru ștergere.

4. Angajatul apasă pe butonul "Ștergere Comandă".

5. Aplicația șterge datele comenzii din baza de date și afișează un mesaj de confirmare.

Secvență Alternativă: Comandă inexistentă

- Utilizatorul încearcă să șteargă o comandă care nu există în baza de date.

- Aplicația afișează un mesaj de eroare și solicită utilizatorul să selecteze o comandă existentă.

- Scenariul se întoarce la pasul 2.

Scenariu de Utilizare: Actualizare Comandă

Actor Principal: Angajat

Scenariul Principal:

1. Angajatul selectează opțiunea de actualizare a datelor unei comenzi.

2. Aplicația afișează o listă cu comenzile existente din care trebuie selectată comanda ce urmează să fie actualizată.

3. Angajatul selectează comanda dorită pentru actualizare.

4. Angajatul modifică informațiile comenzii, cum ar fi numele clientului, denumirea produsului sau cantitatea comandată.

5. Angajatul apasă pe butonul "Actualizare Comandă".

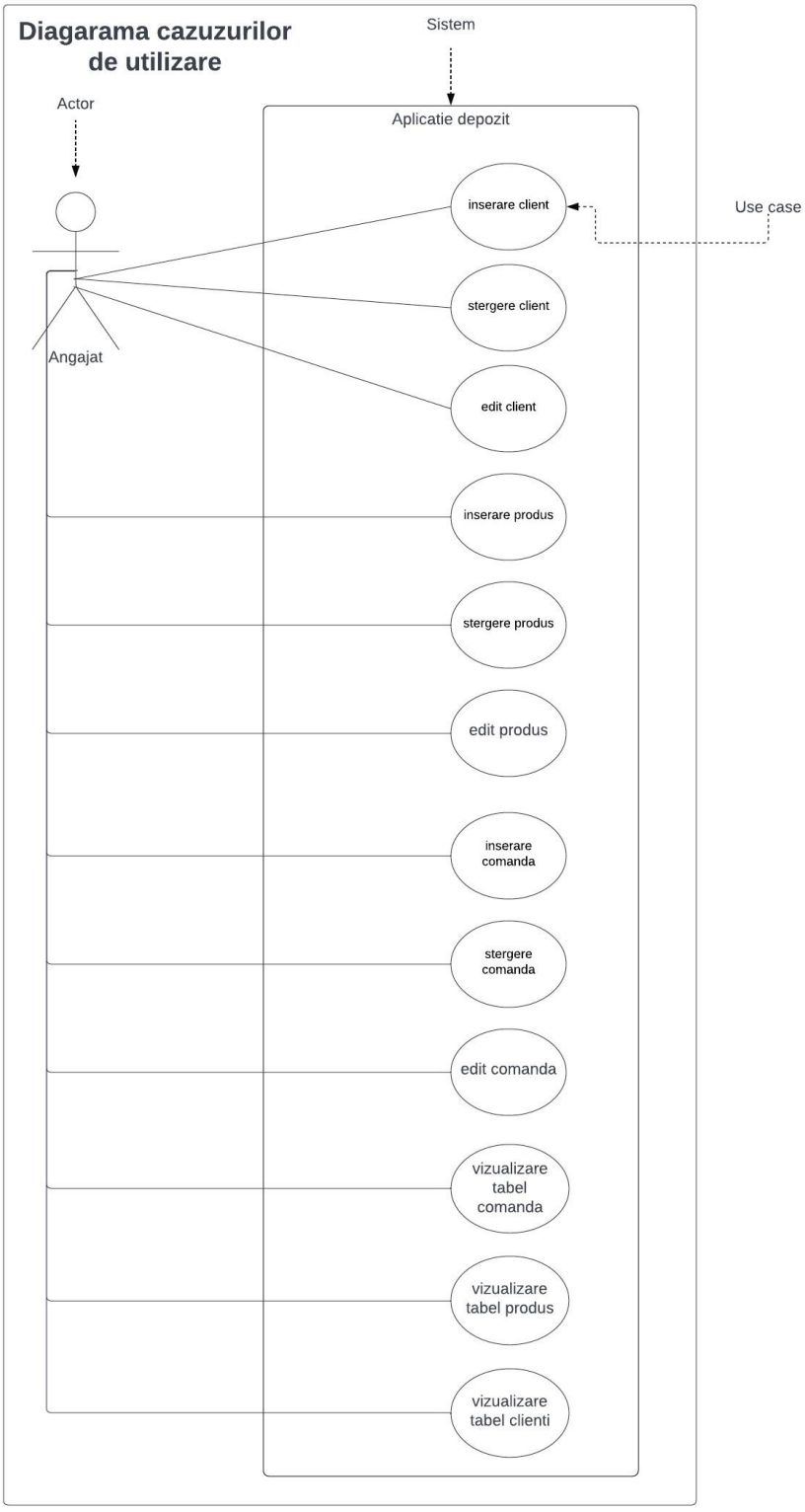
6. Aplicația actualizează datele comenzii în baza de date și afișează un mesaj de confirmare.

Secvență Alternativă: Cantitate invalidă pentru comandă

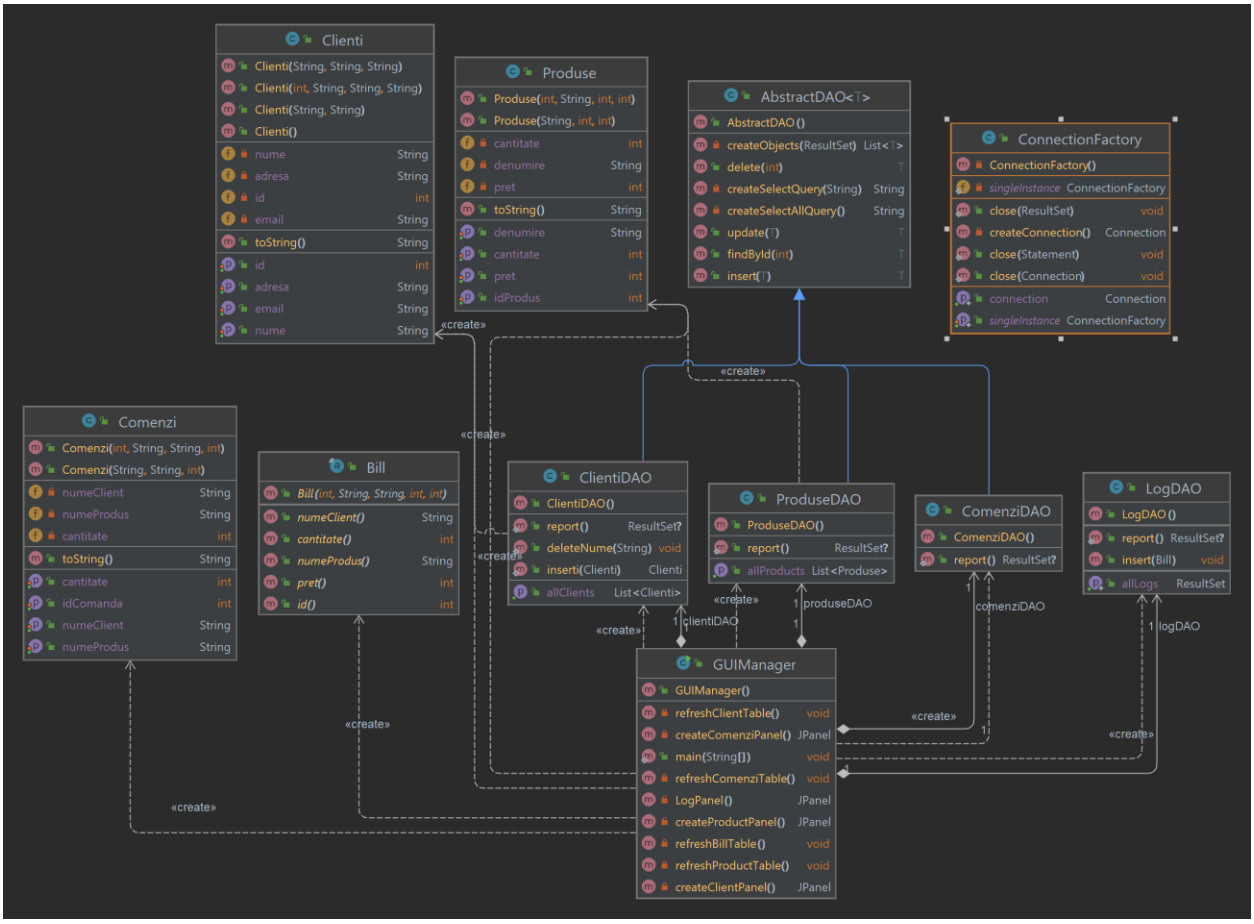
- Utilizatorul introduce o cantitate negativă sau zero pentru produsul comandat.

- Aplicația afișează un mesaj de eroare și solicită utilizatorul să introducă o cantitate validă.

- Scenariul se întoarce la pasul 4.


Diagarama cazuzurilor de utilizare
Sistem
Actor
Aplicatie depozit
Use case
inserare client
stergere client
edit client
Angajat
inserare produs
stergere produs
edit produs
inserare comanda
stergere comanda
edit comanda
vizualizare tabel comanda
vizualizare tabel produs
vizualizare tabel clienti

3. Proiectare



4. Implementare

Clasa **`ConnectionFactory`** este responsabilă pentru crearea și gestionarea conexiunilor la baza de date folosind JDBC (Java Database Connectivity). Iată o prezentare detaliată a acestei clase:

Câmpuri:

- `LOGGER`: Un obiect `Logger` pentru înregistrarea mesajelor de avertizare și erori.

- `DRIVER`: Șirul de caractere care indică clasa driverului JDBC pentru baza de date MySQL.

- `DBURL`: Adresa URL pentru conexiunea la baza de date MySQL.

- `USER`: Numele utilizatorului pentru autentificarea în baza de date MySQL.

- `PASS`: Parola utilizatorului pentru autentificarea în baza de date MySQL.

- `singleInstance`: Instanța unică a clasei `ConnectionFactory`, implementată ca singleton.

Metode:

1. `getSingleInstance()`: Returnează instanța unică a clasei `ConnectionFactory`.

2. `setSingleInstance(ConnectionFactory singleInstance)`: Permite setarea unei alte instanțe pentru singleton, dacă este necesar.

3. `ConnectionFactory()`: Constructorul privat al clasei, care încarcă driverul JDBC.

4. `createConnection()`: Metodă privată care creează și returnează o conexiune la baza de date.

5. `getConnection()`: Metodă statică pentru obținerea unei conexiuni la baza de date. Aceasta folosește instanța unică a clasei `ConnectionFactory` pentru a apela `createConnection()`.

6. `close(Connection connection)`: Metodă statică pentru închiderea unei conexiuni deschise. Primeste ca parametru conexiunea care trebuie închisă.

7. `close(Statement statement)`: Metodă statică pentru închiderea unei declarații SQL deschise. Primeste ca parametru declarația care trebuie închisă.

8. `close(ResultSet resultSet)`: Metodă statică pentru închiderea unui set de rezultate deschis. Primeste ca parametru setul de rezultate care trebuie închis.

Clasa **`GUIManager`** reprezintă interfața grafică a sistemului de gestionare a magazinului online. Aceasta conține panouri pentru gestionarea clienților, produselor, comenzilor și a facturilor. Iată o prezentare detaliată a clasei și a funcționalităților sale:

Metode și câmpuri importante:

1. `clientiDAO`, `produseDAO`, `comenziDAO`, `logDAO`: Obiecte ale claselor DAO (Data Access Object) utilizate pentru interacțiunea cu baza de date pentru gestionarea clienților, produselor, comenzilor și a facturilor.

2. `createClientPanel()`, `createProductPanel()`, `createComenziPanel()`, `LogPanel()`: Metodele care creează panourile pentru gestionarea clienților, produselor, comenzilor și a facturilor, respectiv. Aceste panouri sunt adăugate la un `JTabbedPane` pentru navigare.

3. `refreshClientTable()`, `refreshProductTable()`, `refreshComenziTable()`, `refreshBillTable()`: Metodele pentru reîmprospătarea datelor în tabelele de afișare a clienților, produselor, comenzilor și a facturilor. Aceste metode interacționează cu obiectele DAO pentru a obține datele actualizate din baza de date și le afișează în tabelele corespunzătoare.

4. Listeneri pentru butoanele de adăugare, ștergere și actualizare: Acești listeneri sunt asociați butoanelor din interfață și realizează acțiunile corespunzătoare când utilizatorul interacționează cu aceste butoane. De exemplu, atunci când utilizatorul apasă butonul "Add New Client", este deschis un dialog pentru introducerea datelor unui client nou, iar datele introduse sunt salvate în baza de date prin intermediul obiectului `ClientiDAO`.

5. Declarații ale tabelelor (`JTable`) și ale modelelor acestora (`DefaultTableModel`): Acestea sunt utilizate pentru afișarea datelor în tabele și sunt actualizate periodic prin apelurile la metodele `refresh...Table()`.

6. `main()`: Metoda principală care pornește aplicația prin crearea unei instanțe a clasei `GUIManager` și afișarea ferestrei principale.

Clasa **`AbstractDAO`** este o clasă generică abstractă care oferă funcționalități comune pentru operațiuni CRUD (Creare, Citire, Actualizare, Ștergere) utilizând JDBC și reflecție. Iată o prezentare detaliată a clasei și a funcționalităților sale:

Câmpuri:

- `type`: Este tipul generic al clasei. Este determinat utilizând reflecția în constructorul clasei.

Constructor:

- `AbstractDAO()`: Este constructorul clasei. Utilizează reflecția pentru a determina tipul generic al clasei.

Metode importante:

1. `findById(int id)`: Găsește un obiect de tip `T` după ID din baza de date.

2. `createObjects(ResultSet resultSet)`: Creează o listă de obiecte de tip `T` dintr-un `ResultSet`.

3. `insert(T t)`: Inserează un obiect de tip `T` în baza de date.

4. `update(T t)`: Actualizează un obiect de tip `T` în baza de date.

5. `delete(int id)`: Șterge un obiect de tip `T` din baza de date după ID.

6. Metodele private `createSelectQuery(String field)` și `createSelectAllQuery()`: Creează interogări SQL pentru operațiile de citire.

Clasa **`ClientiDAO`** extinde funcționalitățile generice ale clasei `AbstractDAO`, oferind funcționalități specifice pentru gestionarea obiectelor de tip `Clienti` în baza de date. Iată o prezentare detaliată a clasei și a funcționalităților sale:

Câmpuri:

- `LOGGER`: Este folosit pentru înregistrarea mesajelor de log specific clasei `ClientiDAO`.

- `insertStatementString`: Este șirul de instrucțiune SQL pentru inserarea unui client în baza de date.

- `deleteStatementString`: Este șirul de instrucțiune SQL pentru ștergerea unui client din baza de date pe baza numelui.

- `reportStatementString`: Este șirul de instrucțiune SQL pentru generarea unui raport cu toate înregistrările din tabelul `Clienti`.

Metode importante:

1. `inserti(Clienti client)`: Inserează un obiect de tip `Clienti` în baza de date și returnează obiectul cu ID-ul generat.

2. `report()`: Generează un raport cu toate înregistrările din tabelul `Clienti` și returnează un `ResultSet` care conține rezultatele.

3. `deleteNume(String nume)`: Șterge o înregistrare din tabelul `Clienti` pe baza numelui clientului.

4. `getAllClients()`: Returnează o listă cu toți clienții din tabelul `Clienti`.

Clasa **`ComenziDAO`** extinde funcționalitățile generice ale clasei `AbstractDAO`, oferind funcționalități specifice pentru gestionarea obiectelor de tip `Comenzi` în baza de date. Iată o prezentare detaliată a clasei și a funcționalităților sale:

Câmpuri:

- `LOGGER`: Este folosit pentru înregistrarea mesajelor de log specific clasei `ComenziDAO`.

- `insertStatementString`: Este șirul de instrucțiune SQL pentru inserarea unei comenzi în baza de date.

- `findStatementString`: Este șirul de instrucțiune SQL pentru găsirea unei comenzi în baza de date pe baza ID-ului clientului.

- `deleteStatementString`: Este șirul de instrucțiune SQL pentru ștergerea unei comenzi din baza de date pe baza numelui clientului.

- `reportStatementString`: Este șirul de instrucțiune SQL pentru generarea unui raport cu toate înregistrările din tabelul `Comenzi`.

Metode importante:

1. `report()`: Generează un raport cu toate înregistrările din tabelul `Comenzi` și returnează un `ResultSet` care conține rezultatele.

Clasa **`LogDAO`** oferă funcționalități pentru gestionarea obiectelor de tip `Bill` în tabelul `Log` din baza de date. Iată o prezentare detaliată a clasei și a funcționalităților sale:

Câmpuri:

- `LOGGER`: Este folosit pentru înregistrarea mesajelor de log specifice clasei `LogDAO`.

- `reportStatementString`: Este șirul de instrucțiune SQL pentru generarea unui raport cu toate înregistrările din tabelul `Log`.

Metode importante:

1. `insert(Bill bill)`: Inserează un obiect `Bill` în tabelul `Log`. Metoda primește un obiect `Bill` și efectuează o inserare în baza de date, utilizând valorile obiectului `Bill`. Dacă inserarea nu reușește, se lansează o excepție `IllegalAccessException`.

2. `getAllLogs()`: Returnează un `ResultSet` care conține toate înregistrările din tabelul `Log`.

3. `report()`: Generează un raport cu toate înregistrările din tabelul `Log` și returnează un `ResultSet` care conține rezultatele.

Clasa **`ProduseDAO`** oferă funcționalități pentru gestionarea obiectelor de tip `Produse` în baza de date. Iată o prezentare detaliată a clasei și a funcționalităților sale:

Câmpuri:

- `LOGGER`: Este folosit pentru înregistrarea mesajelor de log specifice clasei `ProduseDAO`.

- `insertStatementString`: Reprezintă șirul de instrucțiune SQL pentru inserarea unui produs în tabelul `Produse`.

- `findStatementString`: Reprezintă șirul de instrucțiune SQL pentru găsirea unui produs din tabelul `Produse` după ID.

- `deleteStatementString`: Reprezintă șirul de instrucțiune SQL pentru ștergerea unui produs din tabelul `Produse` după denumire.

- `reportStatementString`: Reprezintă șirul de instrucțiune SQL pentru generarea unui raport cu toate produsele din tabelul `Produse`.

- `selectStatementString`: Reprezintă șirul de instrucțiune SQL pentru selectarea prețului unui produs din tabelul `Produse` după denumire.

- `updateComenziProduse`: Reprezintă șirul de instrucțiune SQL pentru actualizarea cantității unui produs după ce a fost comandat.

- `gasesteDenumireProdus`: Reprezintă șirul de instrucțiune SQL pentru găsirea cantității unui produs după denumire.

- `updateStatementString`: Reprezintă șirul de instrucțiune SQL pentru actualizarea cantității unui produs după comandă.

Metode importante:

1. `getAllProducts()`: Returnează o listă cu toate produsele din tabelul `Produse`.

2. `report()`: Generează un raport cu toate produsele din tabelul `Produse` și returnează un `ResultSet` care conține rezultatele.

Clasa **`Bill`** reprezintă o factură înregistrată în sistem și este definită ca o clasă înregistrare (record) în Java. Această clasă conține informații despre o factură, cum ar fi ID-ul, numele clientului, numele produsului, cantitatea și prețul.

Clasa **`Clienti`** reprezintă o entitate pentru clienții înregistrați în sistem. Această clasă definește atributele și comportamentul asociat clienților. Iată o descriere a fiecărui membru al clasei:

- `id`: Identificatorul clientului.

- `nume`: Numele clientului.

- `email`: Adresa de email a clientului.

- `adresa`: Adresa fizică a clientului.

Clasa `Clienti` definește mai multe constructori pentru a crea obiecte `Clienti` cu diferite combinații de atribute. Acestea includ:

- Constructor cu parametri pentru un client cu toate atributele specificate.

- Constructor cu parametri pentru un client nou, fără adresă de email.

- Constructor cu parametri pentru un client nou, cu nume și adresă specificate.

- Constructor implicit pentru un client nou, cu valori implicite pentru nume, adresă și email.

De asemenea, clasa `Clienti` oferă metode de acces pentru a obține și seta valorile atributelor.

Clasa **`Comenzi`** reprezintă o entitate pentru comenzile efectuate de clienți în sistem. Această clasă definește atributele și comportamentul asociat comenzilor. Iată o descriere a fiecărui membru al clasei:

- `id`: Identificatorul comenzii.

- `numeClient`: Numele clientului care a plasat comanda.

- `numeProdus`: Numele produsului comandat.

- `cantitate`: Cantitatea de produs comandată.

Clasa `Comenzi` definește doi constructori pentru a crea obiecte `Comenzi` cu diferite combinații de atribute. Acestea includ:

- Constructor cu parametri pentru o comandă cu toate atributele specificate.

- Constructor cu parametri pentru o comandă nouă, fără atributul de identificator.

De asemenea, clasa `Comenzi` oferă metode de acces pentru a obține și seta valorile atributelor.

În plus, a fost suprascrisă metoda `toString()` pentru a oferi o reprezentare text a obiectului `Comenzi`.

Clasa **`Comenzi`** reprezintă o entitate pentru comenzile efectuate de clienți în sistem. Această clasă definește atributele și comportamentul asociat comenzilor. Iată o descriere a fiecărui membru al clasei:

- `id`: Identificatorul comenzii.

- `numeClient`: Numele clientului care a plasat comanda.

- `numeProdus`: Numele produsului comandat.

- `cantitate`: Cantitatea de produs comandată.

Clasa `Comenzi` definește doi constructori pentru a crea obiecte `Comenzi` cu diferite combinații de atribute. Acestea includ:

- Constructor cu parametri pentru o comandă cu toate atributele specificate.

- Constructor cu parametri pentru o comandă nouă, fără atributul de identificator.

De asemenea, clasa `Comenzi` oferă metode de acces pentru a obține și seta valorile atributelor.

În plus, a fost suprascrisă metoda `toString()` pentru a oferi o reprezentare text a obiectului `Comenzi`.

5. Concluzii

Tema ne-a oferit o vedere detaliată asupra gestionării datelor în Java, prin intermediul claselor DAO. Am învățat să gestionăm conexiunile la baza de date, să utilizăm interfețele și moștenirea pentru organizarea codului, precum și să tratăm excepțiile și să înregistrăm loguri pentru diagnosticarea problemelor. Utilizând șabloanele de proiectare, cum ar fi DAO, am putut organiza și structura codul într-un mod modular și ușor de întreținut. Există perspective ample de dezvoltare ulterioară, cum ar fi extinderea funcționalității, validarea datelor, integrarea cu interfețe grafice și îmbunătățirea securității. În concluzie, tema ne-a dotat cu cunoștințele și instrumentele necesare pentru a construi aplicații Java robuste și scalabile, care interacționează eficient cu bazele de date relaționale.

6. Bibliografie

1. *https://dsrl.eu/courses/pt/materials/PT2024_A3_S1.pdf*
2. *https://gitlab.com/utcn_dsrl/pt-reflection-example*
3. *https://gitlab.com/utcn_dsrl/pt-layered-architecture*

4. *https://www.baeldung.com/java-jdbc*
5. *https://mkyong.com/jdbc/how-to-connect-to-mysql-with-jdbc-driver-java/*
6. *https://dzone.com/articles/layers-standard-enterprise*
7. *https://jenkov.com/tutorials/java-reflection/index.html*
8. *https://www.baeldung.com/javadoc*
9. *https://dev.mysql.com/doc/workbench/en/wb-admin-export-import-management.html*