**FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE**

**CATEDRA CALCULATOARE**

Orders Management

Documentație

Poponet Tiberiu-Sergiu

Grupa 30221 | An 2 semestrul 2

Cuprins

1. Obiectiv

2. Studiul probelemei

3.Implementare

3.1 Diagrame UML

3.2 Clase

3.3 GUI

3.4 Metode

4.Concluzii

5.Dezvoltari ulterioare

6.Bibliografie

1. Obiectiv

• Obiectiv principal:

Crearea și implementarea unei aplicații pentru gestionarea comenzilor clienților într-un depozit.

• Sub-obiective:

Analiza problemei și identificarea cerințelor:

Efectuarea unei analize aprofundate a cerințelor și necesităților în ceea ce privește gestionarea comenzilor clienților într-un depozit.

Identificarea funcționalităților cheie ale aplicației și a specificațiilor tehnice necesare.

Proiectarea aplicației de gestionare a comenzilor:

Definirea arhitecturii aplicației și a structurii datelor necesare pentru stocarea informațiilor despre clienți și comenzile acestora.

Proiectarea interfeței grafice pentru a facilita inserarea, actualizarea și ștergerea comenzilor, precum și vizualizarea acestora.

Implementarea aplicației de gestionare a comenzilor:

Dezvoltarea codului sursă al aplicației, respectând specificațiile și cerințele stabilite în faza de proiectare.

Implementarea funcționalităților pentru inserarea, actualizarea, ștergerea și vizualizarea comenzilor clienților.

Integrarea cu baza de date pentru stocarea și accesarea informațiilor despre clienți și comenzile acestora.

Testarea aplicației de gestionare a comenzilor:

Efectuarea testelor unitare pentru a verifica corectitudinea funcționalităților implementate.

Realizarea testelor de integrare și testelor de performanță pentru a asigura funcționarea corectă și eficientă a aplicației.

Remedierea eventualelor erori sau probleme identificate în timpul testelor.

Prin îndeplinirea acestor sub-obiective, se urmărește crearea unei aplicații robuste și eficiente pentru gestionarea comenzilor clienților într-un depozit, asigurând o experiență de utilizare intuitivă și satisfăcătoare.

2.Studiul Problemei

Pentru a îndeplini cerințele temei într-un mod eficient, primul pas esențial este să avem o înțelegere solidă a bazei de date MySQL. Aceasta este de o importanță crucială, deoarece toate operațiunile, precum vizualizarea, ștergerea, actualizarea și plasarea comenzilor, vor fi efectuate direct pe baza de date.

• Cerințe funcționale:

-Utilizatorul trebuie să poată selecta operațiunea dorită: operații pe tabela de clienți, operații pe tabela de produse și operații pe tabela de comenzi.

-Utilizatorul trebuie să poată adăuga un client nou.

-Utilizatorul trebuie să poată edita un client existent.

-Utilizatorul trebuie să poată șterge un client existent.

-Utilizatorul trebuie să poată adăuga un produs nou.

-Utilizatorul trebuie să poată edita un produs existent.

-Utilizatorul trebuie să poată șterge un produs existent

Cerințe non-funcționale

Aplicația trebuie să fie intuitivă și ușor de utilizat.

Aplicația trebuie să aibă o interfață grafică interactivă care să faciliteze utilizarea

Cazuri de utilizare

Utilizatorul trebuie să poată crea o comandă prin introducere unui client și a unui produs și introducerea unei cantități valide, iar această comandă trebuie să fie apoi inserată în tabela de comenzi. Utilizatorul accesează fereastra principală a aplicației și selectează operația dorită: Operații pe tabela de clienți. A picture containing screenshot, diagram, plan, line

Description automatically generated

**Client:**

Utilizatorul introduce datele noului client în câmpurile relevante din interfață.

Utilizatorul apasă butonul "Add" pentru a adăuga noul client în baza de date.

Editare client:

Utilizatorul introduce noile date sau modificările dorite în câmpurile corespunzătoare ale clientului selectat.

Utilizatorul apasă butonul "Edit" pentru a actualiza informațiile clientului în baza de date conform modificărilor introduse.

Ștergere client:

Utilizatorul apasă butonul "Delete" pentru a elimina clientul selectat din baza de date.

Vizualizare clienți:

Pentru operațiile pe tabela de produse:

Utilizatorul accesează fereastra principală a aplicației și selectează operația dorită: Operații pe tabela de produse.

**Produs:**

Utilizatorul introduce datele noului produs în câmpurile relevante din interfață.

Utilizatorul apasă butonul "Add" pentru a adăuga noul produs în baza de date.

Editare produs:

Utilizatorul introduce noile date sau modificările dorite în câmpurile corespunzătoare ale produsului selectat.

Utilizatorul apasă butonul "Edit" pentru a actualiza informațiile produsului în baza de date conform modificărilor introduse.

Ștergere produs:

Utilizatorul apasă butonul "Delete" pentru a elimina produsul dorit din baza de date.

**Comenzi:**

Utilizatorul introduce un client disponibil.

Utilizatorul introduce un produs disponibil.

Utilizatorul introduce o cantitate validă pentru produsul selectat în câmpul corespunzător.

Utilizatorul apasă butonul "Add" pentru a genera o nouă comandă.

Informațiile despre comanda nouă, inclusiv clientul, produsul și cantitatea, sunt inserate în tabela de comenzi din baza de date.

Aceasta permite utilizatorului să creeze și să înregistreze noi comenzi în sistemul de gestionare a comenzilor. Pe măsură ce comenzile sunt adăugate, baza de date se actualizează pentru a reflecta starea curentă a comenzilor existente.

**3.Implementare**

3.1 Diagrame UML

A screenshot of a computer program

Description automatically generated with low confidence

3.2 Clase

**A.bll**

**validators : -Date Validator:** nefolosita.

**ClientBLL:** nefolosita.

**OrderBLL:** nefolosita.

**ProductBLL:** nefolosita.

**B.connection:**

**ConnectionFactory:** realizeaza conexiunea intre baza de date si Java code.

**C.** **dao**

-Clasa **AbstractDAO** oferă operații generice de bază pentru accesul la date pentru un anumit tip de entitate. Această clasă este folosită ca bază pentru clasele DAO specifice care manipulează datele entităților specifice.

Clasele **ClientDAO**, **OrdersDAO**, **ProductDAO** sunt goale si fac extindere la AbstractDAO<Clients/Orders/Products>;

**D.** **model**

Clasa **Client** este o clasă din pachetul "model" care reprezintă un obiect client în cadrul aplicației. Această clasă are următoarele caracteristici:

Atribute:

id (de tip int) - reprezintă identificatorul unic al clientului.

nume (de tip String) - reprezintă numele clientului.

prenume (de tip String) - reprezintă prenumele clientului.

Constructori:

Client() - constructorul fără parametri care inițializează un obiect Client cu valorile implicite pentru atribute.

Client(int id) - constructor care primește ca parametru id-ul clientului și îl atribuie atributului "id".

Client(int id, String nume, String prenume) - constructor care primește ca parametri id-ul, numele și prenumele clientului și le atribuie atributelor corespunzătoare.

Metode:

getId() - returnează id-ul clientului.

setId(int id) - setează id-ul clientului cu valoarea specificată.

getNume() - returnează numele clientului.

setNume(String nume) - setează numele clientului cu valoarea specificată.

getPrenume() - returnează prenumele clientului.

setPrenume(String prenume) - setează prenumele clientului cu valoarea specificată.

toString() - returnează o reprezentare textuală a obiectului Client, care conține valorile atributelor.

Clasa Client este utilizată pentru a crea și manipula obiecte de tip client în cadrul aplicației de gestionare a comenzilor.

Clasa **Comanda** este o clasă din pachetul "model" care reprezintă o comandă plasată de un client pentru un anumit produs în cadrul aplicației. Această clasă are următoarele caracteristici:

Atribute:

id (de tip int) - reprezintă identificatorul unic al comenzii.

NumeClient (de tip String) - reprezintă numele clientului care a plasat comanda.

PrenumeClient (de tip String) - reprezintă prenumele clientului care a plasat comanda.

NumeProdus (de tip String) - reprezintă numele produsului comandat.

Cantitate (de tip int) - reprezintă cantitatea de produs comandată.

Constructori:

Comanda() - constructorul fără parametri care inițializează un obiect Comanda cu valorile implicite pentru atribute.

Comanda(int id) - constructor care primește ca parametru id-ul comenzii și îl atribuie atributului "id".

Comanda(int id, String NumeClient, String PrenumeClient, String NumeProdus, int Cantitate) - constructor care primește ca parametri id-ul comenzii, numele și prenumele clientului, numele produsului și cantitatea comandată, și le atribuie atributelor corespunzătoare.

Metode:

getId() - returnează id-ul comenzii.

setId(int id) - setează id-ul comenzii cu valoarea specificată.

getNumeProdus() - returnează numele produsului comandat.

setNumeProdus(String numeProdus) - setează numele produsului comandat cu valoarea specificată.

getCantitate() - returnează cantitatea de produs comandată.

setCantitate(int cantitate) - setează cantitatea de produs comandată cu valoarea specificată.

getNumeClient() - returnează numele clientului care a plasat comanda.

setNumeClient(String numeClient) - setează numele clientului care a plasat comanda cu valoarea specificată.

getPrenumeClient() - returnează prenumele clientului care a plasat comanda.

setPrenumeClient(String prenumeClient) - setează prenumele clientului care a plasat comanda cu valoarea specificată.

Clasa Comanda este utilizată pentru a crea și manipula obiecte de tip comandă în cadrul aplicației de gestionare a comenzilor pentru un depozit.

Clasa **Produs** este o clasă din pachetul "model" care reprezintă un produs în cadrul aplicației de gestionare a comenzilor pentru un depozit. Această clasă are următoarele caracteristici:

Atribute:

id (de tip int) - reprezintă identificatorul unic al produsului.

NumeProdus (de tip String) - reprezintă numele produsului.

Stock (de tip int) - reprezintă stocul disponibil al produsului.

Constructori:

Produs() - constructorul fără parametri care inițializează un obiect Produs cu valorile implicite pentru atribute.

Produs(int id) - constructor care primește ca parametru id-ul produsului și îl atribuie atributului "id".

Produs(int id, String NumeProdus, int Stock) - constructor care primește ca parametri id-ul produsului, numele produsului și stocul disponibil, și le atribuie atributelor corespunzătoare.

Metode:

getId() - returnează id-ul produsului.

setId(int id) - setează id-ul produsului cu valoarea specificată.

getStock() - returnează stocul disponibil al produsului.

setStock(int stock) - setează stocul disponibil al produsului cu valoarea specificată.

getNumeProdus() - returnează numele produsului.

setNumeProdus(String numeProdus) - setează numele produsului cu valoarea specificată.

Clasa Produs este utilizată pentru a crea și manipula obiecte de tip produs în cadrul aplicației de gestionare a comenzilor pentru un depozit. Aceasta conține informații despre numele produsului și stocul disponibil, permițând utilizatorilor să gestioneze și să actualizeze starea stocului produselor în depozit.

**E.** **presentation**

Cuprinde toate clasele responsabile pentru GUI.

**F.** **start**

Cuprinde clasele RefractionExample o clasa „Start” ce cuprinde metoda „Main” care da drumul aplicatiei.

3.3 GUI

Pachetul „**presentation”** cuprinde pachetele cu clasele specifice interfetei GUI.

**Controller**:

Clasa Controller reprezintă o clasă dintr-un sistem de prezentare care gestionează interacțiunea între interfața grafică și modelele de date. Ea conține mai multe metode, fiecare responsabilă de gestionarea acțiunilor asociate unor evenimente specifice.

Principalele metode ale clasei sunt:

clientPress(): Returnează un obiect ActionListener care va fi utilizat pentru gestionarea evenimentului apăsării unui buton de client. La apăsarea butonului, se va crea un obiect de tip ClientOp.

clientAdd(JTextField jt1, JTextField jt2): Returnează un obiect ActionListener care va fi utilizat pentru gestionarea evenimentului adăugării unui client. La apăsarea butonului de adăugare, se va crea un obiect Client cu informațiile introduse în câmpurile text jt1 și jt2, iar apoi se va insera acest client în baza de date utilizând obiectul AbstractDAO.

clientEdit(JTextField jt1, JTextField jt2, JTextField jt3): Returnează un obiect ActionListener care va fi utilizat pentru gestionarea evenimentului modificării unui client. La apăsarea butonului de modificare, se va crea un obiect Client cu informațiile introduse în câmpurile text jt1, jt2 și jt3, iar apoi se va actualiza acest client în baza de date utilizând obiectul AbstractDAO.

clientDelete(JTextField jt1): Returnează un obiect ActionListener care va fi utilizat pentru gestionarea evenimentului ștergerii unui client. La apăsarea butonului de ștergere, se va crea un obiect Client cu ID-ul introdus în câmpul text jt1, iar apoi se va șterge acest client din baza de date utilizând obiectul AbstractDAO.

comandaPress(), comandaAdd(), comandaEdit(), comandaDelete(): Metode similare celor pentru client, dar pentru gestionarea comenzilor (Comanda).

produsPress(), produsAdd(), produsEdit(), produsDelete(): Metode similare celor pentru client, dar pentru gestionarea produselor (Produs).

viewAllClients(), viewAllCommands(), viewAllProducts(): Returnează obiecte ActionListener pentru afișarea tuturor clienților, comenzilor și produselor. La apăsarea butoanelor corespunzătoare, se vor extrage informațiile din baza de date utilizând obiectul AbstractDAO, iar apoi se vor afișa în interfața grafică folosind clasele ViewAllClients, ViewAllCommands și ViewAllProducts.

ClientOp:

Clasa ClientOp reprezintă o clasă care definește interfața grafică și acțiunile asociate operațiilor cu clienți. Aceasta conține butoane, câmpuri de text și alte elemente grafice necesare pentru interacțiunea cu utilizatorul.

Constructorul clasei ClientOp inițializează interfața grafică, creând un obiect de tip JFrame și adăugând panoul Panou ca conținut al ferestrei. Apoi, se stabilesc proprietățile ferestrei și se afișează pe ecran. Se creează și se inițializează un obiect de tip Controller pentru gestionarea acțiunilor asociate butoanelor.

În constructor, se atașează acțiuni butoanelor prin apelarea metodelor corespunzătoare din clasa Controller, pasând câmpurile de text necesare pentru extragerea datelor introduse de utilizator. Astfel, atunci când utilizatorul apasă unul dintre butoane, se va executa acțiunea asociată din clasa Controller.

În final, clasa ClientOp permite interacțiunea cu utilizatorul prin intermediul interfeței grafice, oferind funcționalități precum adăugarea, modificarea, ștergerea și afișarea clienților.

ComandaOp:

La fel ca la ClientOp, dar pentru clasa Comanda.

ProdusOp:

La fel ca pentru ClientOp, dar pentru clasa Produs.

Clasele ViewAllClients, ViewAllProducts si ViewAllOrders afiseaza tabelele cu toate informatiile stocate in ele.

3.4 Metode

Clasa AbstractDAO<T> este o clasă generică care definește operațiile de bază pentru un Data Access Object (DAO). Aceasta oferă metode pentru a realiza operațiuni CRUD (Create, Read, Update, Delete) asupra unor obiecte de tip T, unde T este un tip generic.

Principalele metode ale clasei AbstractDAO<T> sunt:

Constructorul AbstractDAO():

Acesta este un constructor implicit care obține tipul concret al clasei T utilizând reflexia.

Utilizează getClass().getGenericSuperclass() pentru a obține tipul generic al clasei și getActualTypeArguments()[0] pentru a obține primul argument de tip din lista de argumente de tip.

Este utilizat pentru a asocia tipul clasei T cu obiectele DAO create.

Metoda findAll():

Returnează o listă de obiecte de tip T care reprezintă toate înregistrările din tabela corespunzătoare clasei T.

Metoda poate fi suprascrisă în clasele derivate pentru a implementa funcționalitatea specifică.

Metoda findById(int id, Class<T> objectType)

Primește un ID și un obiect de tip Class care indică clasa T la care se face referire.

Realizează o interogare către baza de date pentru a găsi înregistrarea cu ID-ul specificat în tabela corespunzătoare clasei T.

Utilizează metoda mapResultSetToObject() pentru a transforma rezultatul interogării într-un obiect de tip T.

Metoda mapResultSetToObject(ResultSet resultSet, Class<T> objectType):

Primește un obiect ResultSet care reprezintă rezultatul unei interogări și un obiect de tip Class care indică clasa T la care se face referire.

Realizează maparea valorilor din ResultSet la un obiect de tip T utilizând reflexia.

Metoda utilizează metadatele ResultSet-ului pentru a obține informații despre coloane și apoi utilizează reflexia pentru a seta valorile în câmpurile obiectului.

Metoda getFieldByName(Class<?> objectType, String fieldName):

Primește un obiect de tip Class și numele unui câmp.

Caută și returnează obiectul Field asociat câmpului cu numele specificat în clasa dată.

Metoda este recursivă și verifică și în clasa părinte dacă câmpul nu este găsit în clasa curentă.

Metoda update(T t):

Primește un obiect de tip T și actualizează înregistrarea corespunzătoare în baza de date.

Metoda poate fi suprascrisă în clasele derivate pentru a implementa funcționalitatea specifică.

Metoda insert(T t):

Primește un obiect de tip T și inserează o nouă înregistrare în baza de date.

Metoda generează o interogare de inserare dinamica utilizând reflexia pentru a obține numele câmpurilor și valorile obiectului.

Metoda generateInsertQuery(T t):

Primește un obiect de tip T și generează o interogare de inserare SQL dinamica utilizând reflexia pentru a obține numele câmpurilor și a construi clauza VALUES.

Metoda update(T t, int id):

Primește un obiect de tip T și un ID și actualizează înregistrarea corespunzătoare cu ID-ul specificat în baza de date.

Metoda generează o interogare de actualizare dinamica utilizând reflexia pentru a obține numele câmpurilor și a construi clauza SET.

Metoda generateUpdateQuery(T t):

Primește un obiect de tip T și generează o interogare de actualizare SQL dinamică utilizând reflexia pentru a obține numele câmpurilor și a construi clauza SET.

Metoda delete(T t, int id):

Primește un obiect de tip T și un ID și șterge înregistrarea corespunzătoare cu ID-ul specificat din baza de date.

Metoda generează o interogare de ștergere dinamică utilizând reflexia pentru a obține numele tabelei și a construi clauza WHERE.

Metoda generateDeleteQuery(T t):

Primește un obiect de tip T și generează o interogare de ștergere SQL dinamică utilizând reflexia pentru a obține numele tabelei și a construi clauza WHERE.

Metoda retrieveTableContents(Object object):

Primește un obiect și returnează conținutul tabelei corespunzătoare clasei obiectului într-o matrice de șiruri de caractere.

Metoda realizează o interogare pentru a obține toate înregistrările din tabela corespunzătoare clasei și le stochează într-o listă de șiruri de caractere.

Rezultatul este returnat sub forma unei matrici de șiruri de caractere.

Metoda generateTable(List<?> objects):

Primește o listă de obiecte și generează o reprezentare tabulară a acestora în consolă.

Metoda utilizează reflexia pentru a obține câmpurile obiectelor și afișează valorile lor într-un format tabular.

4.Concluzii

Dezvoltarea unei aplicații sofisticate de gestionare a clienților, produselor și comenzilor într-un depozit diversificat a fost realizată cu succes. Proiectul a avut ca scop principal crearea și implementarea unei soluții profesionale și bine definite pentru gestionarea eficientă a clienților, produselor și comenzilor într-un mediu de depozit diversificat. Iată câteva caracteristici cheie ale aplicației:

Interfață grafică interactivă: Utilizatorii beneficiază de o interfață grafică interactivă care facilitează efectuarea operațiilor de inserare, actualizare și ștergere a clienților și produselor, precum și plasarea comenzilor prin selectarea unui client și a unui produs.

Sistem eficient de notificare în timp real: Aplicația dispune de un sistem eficient de notificare în timp real care alertează utilizatorii cu privire la stocul excedentar sau insuficient al unui produs. Această funcționalitate contribuie la o gestionare mai precisă și eficientă a depozitului.

Interfață intuitivă și modulară: Interfața aplicației este intuitivă și împărțită în module distincte, asigurând un acces rapid și ușor la funcționalitățile dorite. Utilizatorii pot gestiona produsele din depozit, actualiza informațiile despre clienți și administra comenzile într-un mod simplu și eficient.

Actualizare în timp real a informațiilor: Prin intermediul panelurilor interfeței, utilizatorii au acces la tabele actualizate în timp real, oferindu-le informații despre produsele, clienții și comenzile existente în baza de date. Astfel, utilizatorii pot lua decizii în cunoștință de cauză având acces la datele actualizate.

Metodologie riguroasă de dezvoltare: Proiectul a fost abordat cu o metodologie riguroasă, inclusiv etapele de analiză a problemei, modelare, definire a scenariilor și cazurilor de utilizare. S-a acordat o atenție deosebită proiectării aplicației și implementării propriu-zise, respectând standardele și practicile moderne de programare.

În cadrul dezvoltării aplicației, s-au utilizat diverse structuri de date, inclusiv tipuri de date primitive și colecții de date. Pentru afișarea și manipularea datelor, s-a folosit componenta JTable din biblioteca Java Swing, oferind o interfață grafică intuitivă.

Au fost aplicate tehnici de reflexie pentru a accesa și manipula dinamic clase și obiecte în timpul execuției, oferind flexibilitate și extensibilitate.

5.Dezvoltari ulterioare

Pe baza proiectului de gestionare a clienților, produselor și comenzilor într-un depozit diversificat, există câteva dezvoltări ulterioare pe care le puteți lua în considerare: Implementarea unui sistem de raportare avansat: Puteți adăuga funcționalități suplimentare pentru generarea de rapoarte personalizate în aplicație. Aceste rapoarte pot include informații despre vânzări, stocuri, performanța clienților, etc. Aceasta va oferi utilizatorilor o perspectivă mai detaliată asupra afacerii și le va permite să ia decizii informate.

6.Bibliografie

1.Sierra, K., & Bates, B. (2020). OCP Oracle Certified Professional Java SE 11 Programmer I

2.Pratt, P., & Adamski, J. J. (2018). SQL Programming (6th Edition). Cengage Learning.

Oracle Java Documentation: https://docs.oracle.com/en/java/

Java Tutorials - Oracle: https://docs.oracle.com/javase/tutorial/index.html

Baeldung: https://www.baeldung.com/

Java Code Geeks: https://www.javacodegeeks.com/