



UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

DOCUMENTAȚIE

APLICAȚIE PENTRU GESTIONAREA COMENZILOR UNUI DEPOZIT

BUMBU RĂZVAN-VASILE

GRUPA 30223

PROFESOR COORDONATOR: ALEXANDRU RANCEA



CUPRINS

1. Obiectivul temei
2. Analiza problemei, modelare, scenarii, cazuri de utilizare
3. Proiectare
4. Implementare
5. Rezultate
6. Concluzii
7. Bibliografie



1. OBIECTIVUL TEMEI

Obiectivul principal al acestei teme consta in proiectarea si implementarea unei aplicatii care simuleaza un sistem de gestionare a comenzilor unor clienti intr-un depozit. De asemenea folosirea bazei de date este necesara pentru stocarea datelor necesare pentru client, produs si comanda.

Cerintele obligatorii pentru aceasta tema sunt: respectarea paradigmelor programarii orientate pe obiecte, realizarea unei interfete grafice in care sa existe cel putin trei tabele: unul pentru client, unul pentru produs si unul pentru comanda. De asemenea este necesara folosirea tehnicii reflexiei pentru crearea tabelului.

Pentru obtinerea notei maxime, se cere crearea unei facturi pentru fiecare comanda, calitatea documentatiei si folosirea arhitecturii pe layere. Sunt necesare patru pachete: `dataAccess`, `businessLayer`, `model` si `presentation`. De asemenea se pot obtine puncte suplimentare daca se mai adauga un tabel si se realizeaza structura corecta a bazei de date, precum si folosirea tehnicii reflexiei pentru crearea de metode de accesare a bazei de date.

Una dintre imbunatatirile aduse limbajului Java dupa versiunea 1.0 a fost introducerea reflexiei, cunoscuta si sub numele de introspectie. Sub orice nume s-ar folosi, reflexia permite unei clase Java – cum sunt toate programele scrise pana acum – sa afle detalii despre orice alta clasa. Prin reflexie un program Java poate incarca o clasa despre care nu stie nimic, sa afle despre variabilele, metodele si constructorii clasei si apoi sa lucreze cu ele.

Arhitectura de tip layer se refera la folosirea unor pachete specifice pentru a imparti clasele dupa functionalitatea lor. Astfel in pachetul de model avem clasele care modeleaza obiectele necesare. In pachetul presentation avem clasele pentru interfata grafica. In pachetul `dataAccess` avem clasele necesare accesarii obiectelor din baza de date. In pachetul `businessLogic` avem clasele necesare executarii anumitor interogari pe baza de date si obtinerea rezultatelor necesare din aceasta.

Printre obiectivele secundare se numara urmatoarele:

- i. Alegerea structurilor de date: se aleg astfel incat acestea sa faciliteze operatiile de adaugare, de extragere precum si de sortare a datelor.
- ii. Impartirea pe pachete: organizarea claselor inrudite in pachete pentru mentinerea lizibilitatii aplicatiei.
- iii. Impartirea pe clase: fiecare clasa manipuleaza un singur obiect
- iv. Implementarea: descrierea in limbaj Java a algoritmilor utilizati



2. ANALIZA PROBLEMEI, MODELARE, SCENARIU, CAZURI DE UTILIZARE

Analiza problemei presupune identificarea claselor proiectului și a funcționalităților acestuia, precum și legăturile existente între acestea. Asadar, folosind un număr minim de informații, programarea orientată pe obiecte ne oferă perspectiva implementării unei aplicații folosindu-ne doar de informații de suprafață.

Împreună cu cunoașterea paradagimelor programării orientate pe obiect mai sunt necesare și cunoașterea unor minime tehnici de programare a bazelor de date MySQL.

O **bază de date**, uneori numită și **bancă de date** (abreviat **BD**), reprezintă o modalitate de stocare a unor informații și date pe un suport extern (un dispozitiv de stocare), cu posibilitatea extinderii ușoare și a regăsirii rapide a acestora. La prima vedere sarcina poate părea banală. Totuși, în condițiile în care este vorba de a lucra cu milioane de elemente, fiecare putând consta din cantități de date care trebuie accesate simultan prin internet de către mii de utilizatori răspândiți pe întreg globul; și în condițiile când disponibilitatea aplicației și datelor trebuie să fie permanentă (de ex. pentru a nu pierde ocazia de a încheia afaceri), soluțiile bune nu sunt de loc simple.

De obicei o bază de date este memorată într-unul sau mai multe fișiere. Bazele de date sunt manipulate cu ajutorul sistemelor de gestiune a bazelor de date.

Cel mai răspândit tip de baze de date este cel relațional, în care datele sunt memorate în tabele. Pe lângă tabele, o bază de date relațională mai poate conține: indici, proceduri stocate, declanșatori, utilizatori și grupuri de utilizatori, tipuri de date, mecanisme de securitate și de gestiune a tranzacțiilor etc.

Astfel cu ajutorul bazelor de date, stocăm informațiile necesare și prin interogări putem obține informațiile necesare.

În timpul utilizării aplicației, la început se va deschide o fereastră prin intermediul căreia utilizatorul este lăsat să aleagă între trei opțiuni:

- Deschiderea unei noi ferestre în cadrul căreia se pot realiza acțiuni asupra tabelii de client (inserare, ștergere și actualizarea datelor).
- Deschiderea unei noi ferestre în cadrul căreia se pot realiza acțiuni asupra tabelii de produs (inserare, ștergere și actualizarea datelor).
- Deschiderea unei noi ferestre în cadrul căreia se poate realiza cumpărarea unui produs pe baza id-ului sau într-o anumită cantitate specificată de client.

Ca și cazuri de utilizare există următoarele posibilități:

- Clientul dorește să cumpere o cantitate dintr-un produs mai mică decât stocul acestuia => stocul este actualizat, în cadrul interfeței grafice se afișează totalul de plată, iar în



- Clientul doreste sa cumpere o cantitate dintr-un produs egala custocul acestuia => produsul este sters din tabela, in cadrul interfetei grafice se afiseaza totalul de plata, iar in documentul "receipt.txt" este afisat bonul care contine id-ul clientului, id-ul produsului, numele produsului, pretul produsului, cantitatea cumparata si pretul total platit.
- Clientul doreste sa cumpere o cantitate dintr-un produs mai mare decat stocul acestuia => in cadrul interfetei grafice se afiseaza un mesaj prin care clientul este informat ca produsul respective nu dispune de un stoc suficient de mare.

Unified Modeling Language (UML) este un limbaj standard pentru descrierea de modele și specificații pentru software. UML a fost la bază dezvoltat pentru reprezentarea complexității programelor orientate pe obiect, al căror fundament este structurarea 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.

```

classDiagram
    class Orders {
        clientId int
        id int
        productId int
        quantity int
        totalPrice int
    }
    class ProductView {
        productIdField JTextField
        productNameField JTextField
        productPriceField JTextField
        productQuantityField JTextField
    }
    class OrderView {
        clientField JTextField
        productField JTextField
        quantityField JTextField
        receipt JTextField
    }
    class Product {
        id int
        name String
        price int
        quantity int
    }
    class AbstractDAO_T["AbstractDAO<T>"]
    class ClientDAO
    class ProductDAO
    class OrdersDAO
    class ClientView {
        clientAgeField JTextField
        clientIdField JTextField
        clientNameField JTextField
    }
    class Client {
        age int
        id int
        name String
    }
    class ConnectionFactory {
        connection Connection
    }
    AbstractDAO_T <|-- ClientDAO
    AbstractDAO_T <|-- ProductDAO
    AbstractDAO_T <|-- OrdersDAO
  
```

The diagram illustrates the following classes and their attributes:

- Orders**:
 - clientId int
 - id int
 - productId int
 - quantity int
 - totalPrice int
- ProductView**:
 - productIdField JTextField
 - productNameField JTextField
 - productPriceField JTextField
 - productQuantityField JTextField
- OrderView**:
 - clientField JTextField
 - productField JTextField
 - quantityField JTextField
 - receipt JTextField
- Product**:
 - id int
 - name String
 - price int
 - quantity int
- AbstractDAO<T>**:
 - Abstract base class for ClientDAO, ProductDAO, and OrdersDAO.
- ClientDAO**:
 - Implements AbstractDAO<T>.
- ProductDAO**:
 - Implements AbstractDAO<T>.
- OrdersDAO**:
 - Implements AbstractDAO<T>.
- ClientView**:
 - clientAgeField JTextField
 - clientIdField JTextField
 - clientNameField JTextField
- Client**:
 - age int
 - id int
 - name String
- ConnectionFactory**:
 - connection Connection



4. IMPLEMENTARE

● Pachetul CONNECTION

Contine clasa ConnectionFactory care este responsabila pentru realizarea conexiunii dintre aplicatie si baza de date.

➤ CLASA ConnectionFactory:

Este responsabila pentru realizarea conexiunii dintre aplicatie si baza de date.

Metode implementate:

- ✓ ConnectionFactory () – constructor
- ✓ CreateConnection () – pentru realizarea conexiunii dintre aplicatie si baza de date
- ✓ GetConnection () – pentru gasirea conexiunii dintre aplicatie si baza de date
- ✓ Close () – pentru terminarea conexiunii dintre aplicatie si baza de date

● Pachetul DAO

Contine clasele AbstractDAO, ClientDAO, OrdersDAO, ProductDAO.

➤ CLASA AbstractDAO

Este responsabila pentru crearea de interogari asupra bazei de date.

Metode implementate:

- ✓ createSelectQuery (String s) – creeaza interogari pentru selectia unui camp din tabel
- ✓ createInsertQuery (String s) – creeaza interogari pentru inserarea unui camp in tabel
- ✓ createDeleteQuery (String s) – creeaza interogari pentru stergerea unui camp din tabel
- ✓ createUpdateQuery (int id, String s) – creeaza interogari pentru actualizarea unui camp din tabel
- ✓ createSelectAllQuery () – creeaza interogari pentru selectia tuturor campurilor din tabel
- ✓ findById (int id) – realizeaza returnarea unui obiect dupa id
- ✓ insert (String s) – realizeaza inserarea unui obiect in tabel
- ✓ delete (int id) – realizeaza stergerea unui obiect din tabel
- ✓ update (int id, String s) – realizeaza actualizarea unui obiect in tabel
- ✓ findAll () – realizeaza returnarea tuturor obiectelor din tabel



➤ **CLASA ClientDAO**

Extinde clasa AbstractDAO pentru obiectele de tip Client.

➤ **CLASA ProductDAO**

Extinde clasa AbstractDAO pentru obiectele de tip Product.

➤ **CLASA OrdersDAO**

Extinde clasa AbstractDAO pentru obiectele de tip Orders.

● **Pachetul MODEL**

Contine clasele Client, Orders si Product.

➤ **CLASA Client**

Este responsabila pentru definirea si crearea obiectelor de tip Client.

Metode implementate:

- ✓ Client () – constructor
- ✓ Getter – pentru returnarea parametrilor obiectului de tip Client
- ✓ Setter – pentru setarea parametrilor obiectului de tip Client

➤ **CLASA Product**

Este responsabila pentru definirea si crearea obiectelor de tip Product.

Metode implementate:

- ✓ Product () – constructor
- ✓ Getter – pentru returnarea parametrilor obiectului de tip Product
- ✓ Setter – pentru setarea parametrilor obiectului de tip Product



➤ **CLASA Orders**

Este responsabila pentru definirea si crearea obiectelor de tip Orders.

Metode implementate:

- ✓ Orders () – constructor
- ✓ Getter – pentru returnarea parametrilor obiectului de tip Orders
- ✓ Setter – pentru setarea parametrilor obiectului de tip Orders

● **Pachetul PRESENTATION**

Contine clasele ClientView, ModelView, OrderView, ProductView (responsabile pentru crearea ferestrelor interfetei grafice) si clasa Controller (responsabila pentru controlul aplicatiei).

➤ **CLASA ClientView**

Este responsabila pentru lucrul asupra datelor din tabela Client.

Metode implementate:

- ✓ ClientView () – constructor
- ✓ AddActionListener () – pentru adaugarea de actiuni la click asupra butoanelor ferestrei
- Getter – pentru returnarea parametrilor

➤ **CLASA MainView**

Este responsabila pentru a face trecerea la una din cele trei ferestre principale de lucru.

Metode implementate:

- ✓ MainView () – constructor
- ✓ AddActionListener () – pentru adaugarea de actiuni la click asupra butoanelor ferestrei
- ✓ Getter – pentru returnarea parametrilor



➤ **CLASA ProductView**

Este responsabila pentru lucrul asupra datelor din tabela Product.

Metode implementate:

- ✓ ProductView () – constructor
- ✓ AddActionListener () – pentru adaugarea de actiuni la click asupra butoanelor ferestrei
- ✓ Getter – pentru returnarea parametrilor

➤ **CLASA OrderView**

Este responsabila de crearea de noi comenzi.

Metode implementate:

- ✓ OrderView () – constructor
- ✓ AddActionListener () – pentru adaugarea de actiuni la click asupra butoanelor ferestrei
- ✓ Getter – pentru returnarea parametrilor

➤ **CLASA Controller**

Este responsabila de gestionarea tuturor operatiilor realizate de aplicatie.

Metode implementate:

- ✓ Controller () – constructor
- ✓ ButtonListener () – pentru crearea de actiuni la click asupra butoanelor ferestrei
- ✓ readFromFile () – pentru citirea din fisierul text
- ✓ writeToFile () – pentru scrierea in fisierul text



• Pachetul START

Contine clasa Main,

➤ CLASA Main

Este responsabila de crearea aplicatiei

Metode implementate:

- ✓ Main () – realizeaza crearea aplicatiei

5. REZULTATE

Ca și rezultate, am obținut o aplicație care comunică cu baza de date pentru a returna informații despre clienți și produse, dar totodată și pentru a insera, șterge și actualizare clienți și produse.

6. CONCLUZII

În concluzie, această temă m-a ajutat să aprofundez paradigmele Programării Orientate pe Obiect. Astfel, am reușit să îmi dezvolt abilitățile de a lucra cu liste, de a lucra cu interfața grafică. De asemenea, consider că am reușit să îmi îmbunătățesc și abilitățile gândirii logice și matematice. Cel mai important lucru pe care am reușit să-l învăț de la această temă a fost utilizarea reflexiei în Java.

Acest proiect a avut ca și scop înțelegerea mai bine a bazelor de date și de a genera rapoarte specific pentru anumite cerințe ale unui utilizator. Ca dezvoltări ulterioare se pot genera o multitudine de proiecte din domeniul bazelor de date. Aceasta baza de date este una simplă, specifică unui mic magazin de cartier sau a unui aprozar, dar poate fi dusă mai departe la un lanț de hypermarketuri ce au nevoie de o baza de date foarte bine pusă la punct pentru gestionarea clienților, furnizorilor, produselor detinute pe stocuri, atât alimentare cât și nealimentare. Atributele pot fi mult mai multiple și de multe ori pot fi generate mesaje prin care să se atenționeze neregulile cu privire la anumite produse sau comenzi. Prin această modalitate se poate verifica dacă a avut loc o eroare umană în realizarea unei facturi, în cazul în care cantitatea produselor din stoc nu corespunde cu cea a produselor din baza de date a magazinului. Acest proiect m-a învățat de asemenea cum să generăm rapoarte despre anumite obiecte despre care doresc să aflu tot ceea ce mă interesează. În generarea notelor de plată în toate magazinele este necesar un astfel de program, mult mai complex, ce trebuie să fie extrem de lizibil și de ușor de înțeles pentru orice client. Bazele de date sunt un concept vital în viața unui programator și este necesară cunoașterea lor pe deplin pentru a se putea lucra elegant pe ele în ideea realizării unor programe ce pot ajuta utilizatorii chiar



UNIVERSITATEA TEHNICĂ

DIN CLUJ-NAPOCA

sim ai puțin experimentați. Se poate ca pentru niste proiect mai complexe a apăra diverse erori din cauza unor utilizatori ce nu țin cont de condițiile standard de a se insera date în baza de date, deci ținerea sub control a acestora este de un efort mult mai ridicat decât în cazul proiectului nostru și o mică eroare introdusă în baza de date o poate corupe și să fie necesară refacerea ei și implicit a întregului program pentru a se evita pe viitor o asemenea problemă similară.

7. BIBLIOGRAFIE

- <https://www.baeldung.com/>
- https://ro.wikipedia.org/wiki/Pagina_principal%C4%83
- <https://www.w3schools.com/java/>
- https://www.youtube.com/watch?v=duEkh8ZsFGs&ab_channel=CodeJava
- <https://jenkov.com/tutorials/java-reflection/index.html>