

Facultatea de Automatică și Calculatoare

Tehnici de programare fundamentale

Laborator – Tema 3

Order Management

Profesor: Dr. Cristina Pop

Student: Talpoș Andreea Georgiana

Grupa: 30228

1. Obiectivul temei

Obiectivul temei este proiectarea și implementarea unei aplicații care să se ocupe de gestionarea unor comenzi. Această aplicație trebuie să rezolve problema procesării comenzilor unor clienți către un depozit de produse. Detaliile despre clienți, produse și comenzi vor fi stocate într-o bază de date, între program și sursa de date fiind stabilită o conexiune.

Aplicația trebuie să aibă o arhitectură stratificată, fiecare parte având un scop bine definit. Partea de **Model** conține clasele mapate la tabelele bazei de date, **Presentation Layer** conține clasele care definesc interfața cu utilizatorul**, Business Layer** care este format din clasele care încapsulează logica aplicației și **Data Access Layer** care conține clasele în care sunt definite interogările și conexiunea cu baza de date.

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

Această aplicație trebuie să îndeplinească toate cerințele necesare pentru a insera, modifica, șterge și afișa detaliile despre clienți, produse și comenzi. Aceste informații trebuie să fie stocate într-o bază de date relațională MySQL la care pot avea acces mai mulți utilizatori, astfel toate datele sunt mai ușor de preluat și analizat de pe diferite computere.

Mecanismul de reflecție ne este oferit de către pachetul java.lang.reflect. Reflecția reprezintă un mecanism ce permite unui program să-și examineze și să-și modifice structura in timpul rulării. În limita securității, putem afla metodele clasei, constructorii acesteia și restul membrilor clasei. Prin apelul metodelor specifice, putem modifica starea obiectului sau putem construi obiecte noi.

Pentru această problemă se recomandă să se folosească tehnici de reflecție pentru crearea unei metode care primește o listă de obiecte de un anumit tip și generează antetul tabelului prin extragerea prin reflecție a proprietăților obiectului și apoi populează tabelul cu valorile elementelor din listă.

Tot prin folosirea tehnicilor de reflecție se va crea o clasă generică care va conține metode pentru accesarea bazei de date precum: crearea unui obiect, editarea unui obiect, ștergerea și găsirea obiectelor. Interogările pentru accesarea bazei de date pentru un anumit obiect care se găsește într-o anumită tabelă vor fi generate în mod dinamic tot prin reflecție.

1. **Modelarea problemei**

Aplicația va dispune de o interfață cu utilizatorul care îi va oferi utilizatorului posibilitatea de a alege ce acțiuni dorește să execute asupra tabelei cu clienți, tabelei cu produse sau a celei cu comenzi. Asupra fiecărei tabele se pot executa operații de creare (inserarea unui nou obiect), citire ( afișarea tuturor datelor), actualizare (modificarea detaliilor despre o anumită intrare din tabelă) și ștergerea unei anumite intrări.

De exemplu, dacă clientul vrea să acceseze tabela cu clienți, din interfața principală va apăsa butonul corespunzător pentru a avea acces la interfața care se ocupă cu gestionarea clienților. După, va introduce datele necesare precum: un ID, numele clientului, email-ul, și adresa. Apoi, are la dipoziție câteva butoane cu ajutorul cărora poate alege ce acțiune dorește să fie executată.

1. **Scenarii și cazuri de utilizare**

Scenariile și cazurile de utilizare sunt metode folosite in analiza sistemelor pentru a identifica, clarifica și a organiza cerințele sistemului. Un caz de utilizare reprezintă un set de posibile interacțiuni dintre utilizator și aplicație. Deoarece cazurile de utilizare reprezintă, de fapt, pașii pe care utilizatorul îi are de parcurs, am încercat să proiectez o interfață cât mai ușor de utilizat (“User Friendly”).

**Diagrama UML – Use-case**

Diagram

Description automatically generated

Titlu: Order Management System

Rezumat: Primul pas pe care utilizatorul trebuie să îl facă este să aleagă tabela pe care vrea să o vizualizeze apăsând în interfața principală butonul corespunzător, va apărea o altă interfață corespunzătoare tabelei alese. Acesta va trebui să introducă datele de intrare în casetele text prezente pe interfață, date necesare efectuării operațiilor: insert, update, delete și, în plus, poate vizualiza conținutul tabelei apăsând butonul “List all”. În cazul în care drește să depună o comandă, acesta poate alege să primească sau nu factura.

Actori: utilizatorul

Scenarii de utilizare:

1. Condiții inițiale: Utilizatorul trebuie să aleagă tabela asupra căreia vrea să execute prima acțiune, în cazul în care dorește să adauge sau să modifice detaliile despre vreun client va accesa interfața corespunzătoare gestionării tabelei Client. Utilizatorul va introduce în câmpurile pentru text datele cerute: ID, nume, email și adresa apoi va alege operația care vrea să fie efectuată. În cazul în care dorește să insereze un nou client, acesta trebuie să îi asocieze un ID care nu e asociat altui client, altfel inserarea nu se va executa. După executarea operațiilor dorite, pentru a se întoarce la interfața principală trebuie să apese butonul “Back”. În cazul tabelei asociată produselor pașii sunt asemănători, trebuie să introducă datele și să solicite executarea operației dorite. În interfața corespunzătoare comenzilor, la executarea funcției “List all” se vor afișa toate cele trei tabele din baza de date, pentru a se putea vizualiza toate detaliile despre clienți și produse. Tot aici există butonul “Bill” prin care utilizatorul solicită factura.

În cazul în care utilizatorul introduce nume sau adrese de email invalide, numere negative, sau o cantitate care depășește stocul unui anumit produs, acesta va fi atenționat, iar operația nu se va efectua.

1. Scenariu normal: Utilizatorul a introdus corect toate datele necesare carespunzătoare fiecărei interfețe, alege operația pe care vrea să o execute și datele din tabele se modifică corespunzător. La final, utilizatorul poate vizualiza factura generată, în cazul în care a depus o comandă și a solicitat factura. Dacă se verifică datele din baza de date, se va putea observa că s-au modificat.
2. Scenariu alternativ: Utilizatorul introduce un nume sau o adresă de email invalidă, introduce numere negative sau cere o cantitate mai mare decât stocul unui produs. În acest caz va fi atenționat printr-o fereastră de tip pop-up și, evident, operația nu se va executa.
3. Proiectare

Proiectarea aplicației a fost realizată folosind un model arhitectural stratificat (Layered Architecture). Fiecare pachet are propriul scop și apelează funcții ale pachetelor de sub acesta.

**Presentation** – conține clasele care implementează interfața grafică a aplicației și interacțiunea dintre exterior( ceea ce vede utilizatorul) și logica interioară. Acesta are un subpachet numit “View’’ căruia îi corespunde reprezentarea grafică, are rolul de a prelua datele introduse de către utilizator și de a afișa rezultatul.

**BLL (business logic layer)** – conține clasele care încapsulează logica aplicației. Are un subpachet “Validators” în care sunt grupate clasele care au rolul de a valida datele introduse de către utilizator

**DAO (data access object)** – conține clasele care realizează conexiunea cu baza de date și care efectuează diferite operații asupra tabelelor din baza de date

**Model** - conține clasele mapate la tabelele bazei de date

**Connection** – conține clasa care realizează cone

**Diagrama UML de pachete**

Chart, diagram, box and whisker chart

Description automatically generated

**Diagrama UML de clase**

Diagram, schematic

Description automatically generated

**Interfața cu utilizatorul**

Interfața cu utilizatorul a acestei aplicații reprezintă o simulare a unui sistem de gestionare a unei baze de date a unui depozit. Conține spații pentru text destinate pentru introducerea datelor de intrare și mai multe butoane: pentru selectarea tabelei asupra căreia să se execute comenzile și butoane pentru diferite operații. Pe interfețele destinate clienților, produselor și comenzilor se vor afișa tabelele cu datele din baza de date.

1. Implementare

**Pachete**

**Pachetul Model** – conține clasele reprezentante ale tabelelor bazei de date, respectiv Client, Product și Orders, fiind denumite la fel ca tabelele bazei de date pentru a se putea realiza interogările. De asemenea, variabilele instanță ale acestor clase reprezintă atributele tabelelor.

**Pachetul BLL** – cuprinde clasele care implementează operațiile care sunt executate. Are un subpachet **Validator** care conține clasele cu rol de a valida datele de intrare introduse de către utilizator, care urmează să fie introduse în tabele, doar dacă sunt valide. Clasele din subpachet sunt: AddressValidator, EmailValidator, NameValidator, PositiveNumberValidator, toate aceste clase implementează interfața Validator.

**Pachetul Connection** – conține clasa CnnectionFactory care realizează conexiunea aplicației cu baza de date.

**Pachetul DAO** - conține clasele care realizează conexiunea cu baza de date și care efectuează diferite operații asupra tabelelor din baza de date

**Pachetul Presentation** - conține cele patru clase care implementează ferestrele interfeței grafice: MainInterface, Clientinterface, PorductInterface și Orderinterface și clasa Controller care realizează legătura dintre interfața grafică și logica aplicației.

**Implementarea claselor**

Clasele din pachetul **Model:**

După cum am precizat și mai sus, în pachetul Model se găsesc clasele : **Client, Product și Orders .**

* Client – modelează tabela client din baza de date, conținând detaliile despre clienți
* Product – modelează tabela product din baza de date, în care se găsesc datele despre produsele disponibile
* Orders – modelează tabela orders din baza de date, în care sunt stocate datele despre comenzile efectuate

Clasele din pachetul **DAO:**

* Principala clasă din acest pachet este clasa **AbstractDAO**, o clasă generică în care sunt definite operațiile care trebuie executate pentru accesarea unei tabele: Insert, Update, Delete, FindById, FindAll, createObjects și createTable și, de asemenea, metode care generează interogările care fac posibilă manipularea bazei de date.

Clasele **ClientDAO, OrderDAO și ProductDAO** extind clasa AbstractDAO, utilizează metodele generice pentru implementarea operațiilor de accesare a tabelelor.

* Clasele din pachetul **BLL**: -**ClientBLL, ProductBLL, OrderBLL**

Validează datele preluate din clasele pachetului DAO, și implementează operațiile corespunzătoare gestionării fiecărei tabele.

Clasele din pachetul **Validators,** subpachet al lui BLL:

* **AddressValidator** validează adresa introdusă de către utilizator, aceasta nu poate fii nulă, poate conține atât litere cât și cifre
* **EmailValidator** validează adresa de email introdusă în interfață, aceasta trebuie să corespundă cu formatul unei adrese obișnuite de email, poate avea litere, cifre și anumite caractere speciale.
* **NameValidator** – un nume poate conține doar litere mari și mici, iar de validarea lor se ocupă această clasă
* **PositiveNumberValidator** – asigură faptul că sunt introduse doar numere pozitive pentru preț, cantitate și stoc

Toate clasele enumerate implementează interfața **Validator**, inclusă, de asemenea, în pachetul Validators.

* În pachetul **Connection** există o singură clasă : ConnectionFactory care asigură realizarea conexiunii cu baza de date. De asemenea, are metode și de întrerupere a conexiunii.
* Clasele din pachetul **Presentation**

Pachetul Presentation are un subpachet numit **View** în care am definit clasele **MainInterface, ClientInterface, PorductInterface și OrderInterface**. Am creat o fereastră principală în MainInterface prin intermediul căreia se alege tabela( adică ferestrele create în celelalte trei clase) asupra căreia se vor executa diferitele operații.

Clasa **Controller** – asigură conexiunea dintre View și logica internă a aplicației.

**Clasa App** – este folosită pentru a da start aplicației

Conține metoda statică main() care pornește controller – ul, deci pornește întreaga aplicație

1. Interfața grafică cu utilizatorul

Am încercat să realizez o interfață cât mai ușor de utilizat și care să și aibă un aspect plăcut. După cum am mai precizat și anterior, interfața grafică are mai multe ferestre, fiecare având asociată o tabelă din baza de date, exceptând fereastra principală, care doar face conexiunea cu restul ferestrelor.

Fereastra principală:

A picture containing diagram

Description automatically generated

Pentru a alege tabela asupra căreia să opereze, utilizatorul va apăsa butonul corespunzător.

Fereastra pentru tabela client:

Graphical user interface

Description automatically generated with medium confidence

Datele de intrare trebuie introduse în casetele destinate pentru text, apoi se apasă un buton pentru a se executa o anumită operație. Dacă e apăsat butonul „Back” se revine la fereastra principală.

Fereastra pentru tabela product:

Graphical user interface, table

Description automatically generated

Fereastra pentru tabela orders:

Table

Description automatically generated

1. Rezultate

Rezultatele acestei probleme ar putea fi noile date introduse în baza de date. La fiecare operație executată de aplicație se modifică automat și informațiile din tabelele bd. Aceste informații sunt afișate în interfață la executarea comenzii „ListAll”.

După fiecare nouă comandă depusă se poate solicita generarea unei facturi care va conține toate detaliile referitoare la clientul care depus comanda, produsul comandat și prețul acestuia.

În plus, am generat o documentație Javadoc în care se găsesc detalii despre majoritatea claselor și metodelor implementate.

1. Concluzii

Acest proiect a fost primul în care am lucrat cu bazele de date în Java. La partea de interogări SQL mi-a fost destul de ușor deoarece am avut în prumul semestru o materie dedicată doar pentru lucrul cu bazele de date. Am avut câteva probleme la început când a trebuit să realizez conexiunea dar într-un final am reușit să o rezolv.

A fost un exercițiu util pentru aprofundarea cunoștințelor legate de paradigmele programării POO și de asemenea am învățat altele noi. În ceea ce privește o dezvoltare ulterioară, cred că există și implementări mai eficiente decât cele propuse de mine, și ar aduce un plus aplicației.

1. Bibliografie

<https://creately.com>

<https://junit.org/junit5/docs/current/api/org.junit.jupiter.api/org/junit/jupiter/api/Assertions.html>

<https://www3.ntu.edu.sg/home/ehchua/programming/java/j4a_gui.html>

<https://regex101.com>

[www.stackoverflow.com](http://www.stackoverflow.com)

<https://www.wikipedia.org>

<https://www.youtube.com>