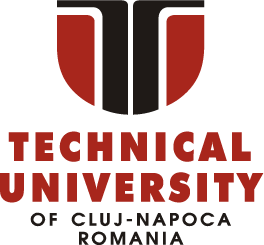
Database Management System

**

Student: Suciu Radu

Group: 30229

1. Obiectiv

Luați în considerare o aplicație OrderManagement pentru procesarea comenzilor clienților pentru un depozit. Bazele de date relaționale sunt folosite pentru stocarea produselor, clienților și comenzilor. În plus, aplicația folosește (minim) următoarele clase:

• Model classes - reprezintă modelele de date ale aplicației (de exemplu Order, Client, Product)

• Business Logic classes conțin logica aplicației

• Presentation classes clase care conțin interfața grafică cu utilizatorul

• Data access classes - clase care conțin accesul la baza de date

Alte clase și pachete pot fi adăugate pentru a implementa funcționalitatea completă a aplicației.

a. Analizați domeniul aplicației, determinați structura și comportamentul claselor sale și desenați o diagramă de clasă UML extinsă.

b. Implementați clasele de aplicație. Utilizați javadoc pentru documentarea claselor.

c. Utilizați tehnici de reflectare pentru a crea o metodă care primește o listă de obiecte și generează antetul tabelului prin extragerea prin reflectare a proprietăților obiectului și apoi populează tabelul cu valorile elementelor din listă.

1. Analiza problemei

* Scurta analiza

Această aplicație ar trebui să poată îndeplini toate cerințele pentru a afișa, modifica și ține evidența comenzilor, clienților și produselor. Acestea sunt stocate într-o bază de date relațională MySQL, împreună cu informațiile despre utilizatorii care au acces la sistem. În acest fel, toate datele sunt mai ușor de preluat și de accesat de pe diferite computere.

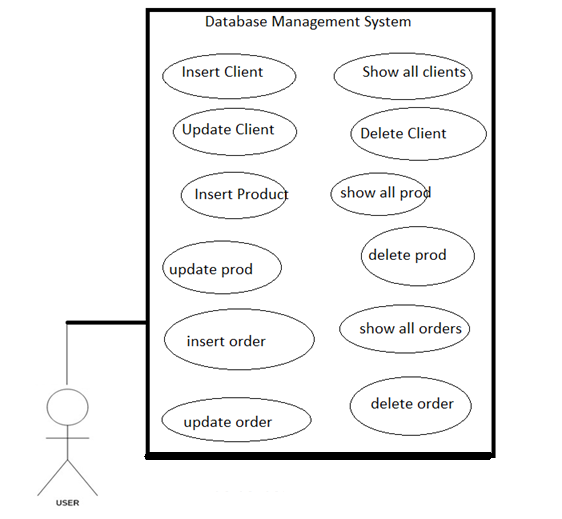
* Input si Output

Când vorbim despre intrarea în aplicație, utilizatorul poate alege să gestioneze 3 tabele, Client, Produs, Comandă. Toate tabelele au opțiunile definite ca operațiuni CRUD, Creare (inserați intrarea), Citire (afișați toate intrările), Actualizare (modificați intrarea) și Ștergere, iar utilizatorul poate introduce valorile în câmpurile specifice pentru oricare dintre cele trei tabele diferite.

De exemplu, pentru tabelul Client utilizatorul poate introduce un ID pentru client (dacă nu este furnizat un ID, aplicația va genera un ID auto-incrementat), numele clientului, adresa de e-mail și numărul de telefon. Toate câmpurile trebuie să respecte regulile standard, cum ar fi numărul de telefon trebuie să aibă 10 cifre (numai cifre), numele clientului poate conține doar litere și spații, adresa de e-mail trebuie să conțină @ și . simboluri .

Pentru următoarele tabele, intrarea este similară, dar cu câmpuri diferite.

* Use Cases



Titlu: Adaugă client

Actiune: Utilizatorul poate adăuga un nou Client introducând detaliile clientului, cum ar fi id-ul, numele, adresa de e-mail și numărul de telefon în câmpurile specificate și apoi ar trebui să apese butonul Inserare. Dacă clientul este creat cu succes, va apărea un pop-up care va spune că clientul a fost creat, iar în caz de eroare un alt pop-up va arăta că a existat o eroare undeva.

Actori: utilizator

Titlu: Delete Client

Actiune: Utilizatorul poate elimina un client specificând ID-ul clientului în câmpul dedicat și apoi apăsând butonul Ștergere. Dacă operațiunea a avut succes, va apărea o fereastră pop-up care menționează asta. În caz de eroare, un alt pop-up va indica eroarea.

Actori: utilizator

Titlu: Update Client

Actiune: utilizatorul poate edita detaliile unui client existent. El/Ea trebuie să completeze toate câmpurile care reprezintă detaliile clientului cu noile valori și să apese butonul Actualizare. Dacă operațiunea a avut succes, va apărea o fereastră pop-up care menționează asta. În caz de eroare, un alt pop-up va indica eroarea.

Actori: utilizator

Titlu: Adaugă produs

Actiune: Utilizatorul poate adăuga produse pentru a fi cumpărate de către clienți, produsele trebuie specificate prin Id, nume, preț și stoc, adică cantitatea totală de bucăți din acel anumit produs. Fiecare comandă scade stocul deoarece oamenii cumpără acele produse de pe piața noastră. Orice comandă anulată/ștersă va „repune” cantitatea de produse implicată în acea comandă. Dacă operațiunea a avut succes, va apărea o fereastră pop-up care menționează asta. În caz de eroare, un alt pop-up va indica eroarea.

Actori: utilizator

Title: Delete Product  
Actiune: Utilizatorul poate elimina un produs introducând ID-ul produsului pe care dorește să îl ștergă și apoi apăsând butonul de ștergere. Dacă operațiunea a reușit, va apărea o fereastră pop-up și va indica asta. În caz de eroare, un alt pop-up va indica eroarea.

Actori: utilizator

Titlu: Actualizare produs

Reluare: utilizatorul poate edita detaliile unui produs existent. El/Ea trebuie să introducă toate detaliile produsului pe care dorește să îl modifice și apoi să apese butonul Actualizare. Dacă operațiunea a avut succes, va apărea o fereastră pop-up care menționează asta. În caz de eroare, un alt pop-up va indica eroarea.

Actori: utilizator

Titlu: Show Product

Reluare: Utilizatorul poate afișa toate produsele apăsând butonul Show All. Dacă un produs este modificat/creat/șters și utilizatorul dorește să vadă acea modificare, trebuie să apese din nou butonul Show All.

Actori: utilizator

Titlu: Show Client

Reluare: Utilizatorul poate afișa toți clienții apăsând butonul Show All. Dacă un client este modificat/creat/șters și utilizatorul dorește să vadă acea modificare, el/ea trebuie să apese din nou butonul Show All.

Actori: utilizator

Titlu: Show Order

Reluare: utilizatorul poate afișa toate comenzile apăsând butonul Afișare tot. Dacă o comandă este modificată/creată/ștersă și utilizatorul dorește să vadă acea modificare, trebuie să apese din nou butonul Show All.

Actori: utilizator

Titlu: Insert Order

Rezumat: Utilizatorul poate introduce o nouă comandă specificând id-ul comenzii, selectând id-ul clientului din lista de clienți, selectând id-ul produsului și introducând cantitatea care va fi cumpărată de client. Dacă operațiunea a avut succes, va apărea o fereastră pop-up care menționează asta. În caz de eroare, un alt pop-up va indica eroarea.

Actori: utilizator

Titlu: Delete Order

Reluare: Utilizatorul poate elimina o comandă introducând ID-ul comenzii pe care dorește să o ștergă și apoi apăsând butonul de ștergere. Dacă operațiunea a avut succes, va apărea o fereastră pop-up care menționează asta. În caz de eroare, un alt pop-up va indica eroarea.

Actori: utilizator

Titlu: Comanda de actualizare

Reluare: utilizatorul poate edita detaliile unei comenzi existente. El/Ea trebuie să introducă toate detaliile comenzii pe care dorește să o modifice și apoi să apese butonul Actualizare. Dacă operațiunea a avut succes, va apărea o fereastră pop-up care menționează asta. În caz de eroare, un alt pop-up va indica eroarea.

Actori: utilizator

1. Implementare

Class Diagram

A picture containing graphical user interface

Description automatically generated

* Data Structures

ArrayList – Am ales să folosesc această structură pentru a-mi păstra cozile și pentru a-mi reține temporar clienții, deoarece acest tip de liste sunt utile pentru stocarea și accesarea datelor

ArrayList utilizează intern matricea dinamică pentru a stoca elementele. Manipularea cu ArrayList este lentă, deoarece utilizează intern matrice. Dacă orice element este eliminat din matrice, toți biții sunt mutați în memorie. Această clasă poate acționa ca o listă numai pentru că implementează numai Listă.

* Packages

Pachetele Java ajută la organizarea mai multor module și grupează clase și interfețe asociate.

În dezvoltarea de programare orientată pe obiecte, model-view-controller (MVC) este numele unei metodologii sau model de proiectare pentru a lega cu succes și eficient interfața cu utilizatorul la modelele de date subiacente. Modelul MVC este utilizat pe scară largă în dezvoltarea de programe cu limbaje de programare precum Java, Smalltalk, C și C++.

Modelul MVC a fost anunțat de mulți dezvoltatori ca un model util pentru reutilizarea codului obiect și un model care le permite să reducă semnificativ timpul necesar pentru a dezvolta aplicații cu interfețe de utilizator.

Modelul MVC propune trei componente sau obiecte principale pentru a fi utilizate în dezvoltarea de software:

- Model, care reprezintă structura logică de bază a datelor dintr-o aplicație software și clasa de nivel înalt asociată acesteia. Acest model de obiect nu conține nicio informație despre interfața cu utilizatorul.

- View, care este o colecție de clase care reprezintă elementele din interfața cu utilizatorul (toate lucrurile pe care utilizatorul le poate vedea și la care poate răspunde pe ecran, cum ar fi butoanele, casetele de afișare și așa mai departe)

- Controller, care reprezintă clasele care conectează modelul și vizualizarea și este folosit pentru a comunica între clasele din model și din vedere.

Proiectul meu se bazează pe Model – View – Controller Pattern, dar am adăugat câteva pachete pentru componenta de acces la date și conexiunea la baza de date. Deci proiectul meu constă din șapte pachete

* Class Design

Întreaga idee de a vă împărți programul în clase se bazează pe o regulă generală numită împărțiți și învingeți. Această paradigmă poate fi folosită aproape peste tot: împărțiți o problemă în probleme mai mici și apoi rezolvați aceste probleme mici, simple și binecunoscute.

Împărțirea programului în clase este unul dintre tipurile de diviziune care a început să devină comună în ultimul deceniu. În această paradigmă de programare modelăm problema noastră după unele obiecte și încercăm să rezolvăm problema prin trimiterea de mesaje între aceste obiecte.

Aplication- Aplicație – pornește aplicația

Connection- ConnectionFactory- este responsabil cu conexiunea cu baza de date, creând o declarație. De asemenea, are metode de închidere a conexiunii/ declarației.

Controller -MainController- Inițializează butoanele UserInterface și configurează mediul aplicației.

DAO

GenericDAO - contine metode generice de a obtine un tabel dintr-o lista de obiecte, o metoda de a insera in baza de date un obiect anume, o metoda de a sterge din baza de date si una de a actualiza o intrare existenta a bazei de date. Toate metodele generice folosesc tehnici de reflexie.

ClientDAO – extinde DAO generic și folosește metodele generice pentru a implementa operațiuni CRUD pe tabelul Client

OrderDAO – extinde DAO generic și folosește metodele generice pentru a implementa operațiuni CRUD pe tabelul Order

ProductDAO – extinde DAO generic și folosește metodele generice pentru a implementa operațiuni CRUD pe tabelul de produse

Model

Client – ​​modelează tabelul client din baza de date

Comanda – modelează tabelul de comenzi din baza de date

Produs – modelează tabelul de produse din baza de date

Validators

ClientValidator – validează informațiile introduse în câmpurile de text din Interfața Client

ProductValidator validează informațiile introduse în câmpurile text din ProductInterface

Vedere

ClientInterface – Cadrul client, oferă butoane, etichete, câmpuri de text pentru a interacționa cu tabelul clienți

OrderDetailsInterface Cadrul pentru detaliile comenzii, furnizează butoane, etichete, câmpuri de text pentru a interacționa cu tabelul cu detaliile comenzii

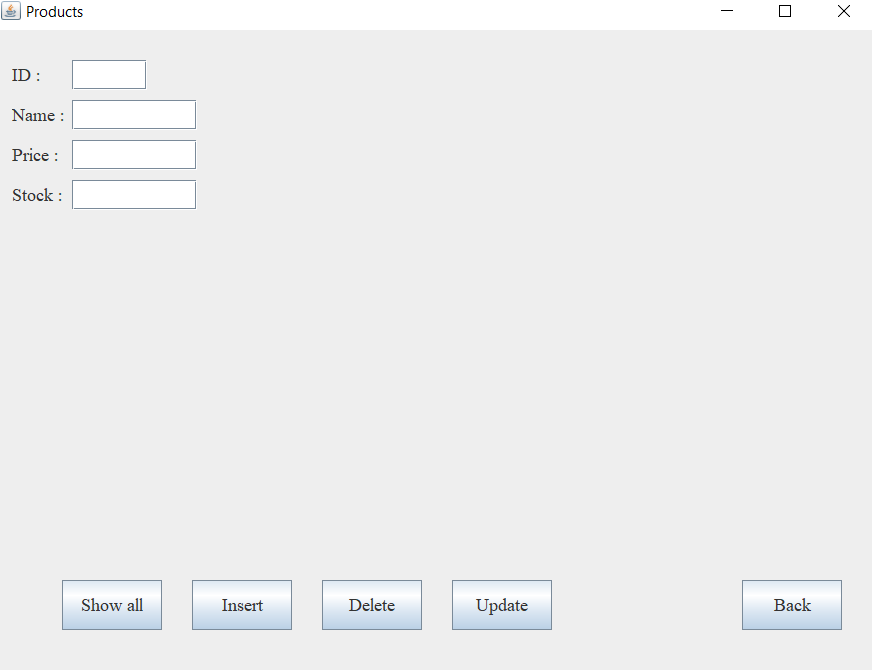
OrderInterface Cadru de comandă, oferă butoane, etichete, câmpuri de text pentru a interacționa cu tabelul de comenzi

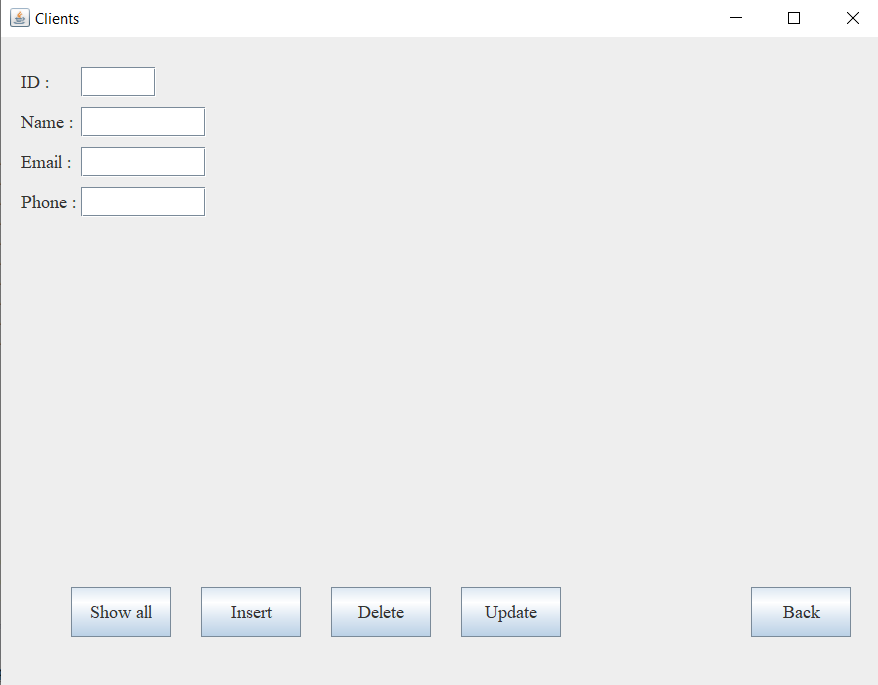
ProductInterface Cadrul de produs, oferă butoane, etichete, câmpuri de text pentru a interacționa cu tabelul de produse

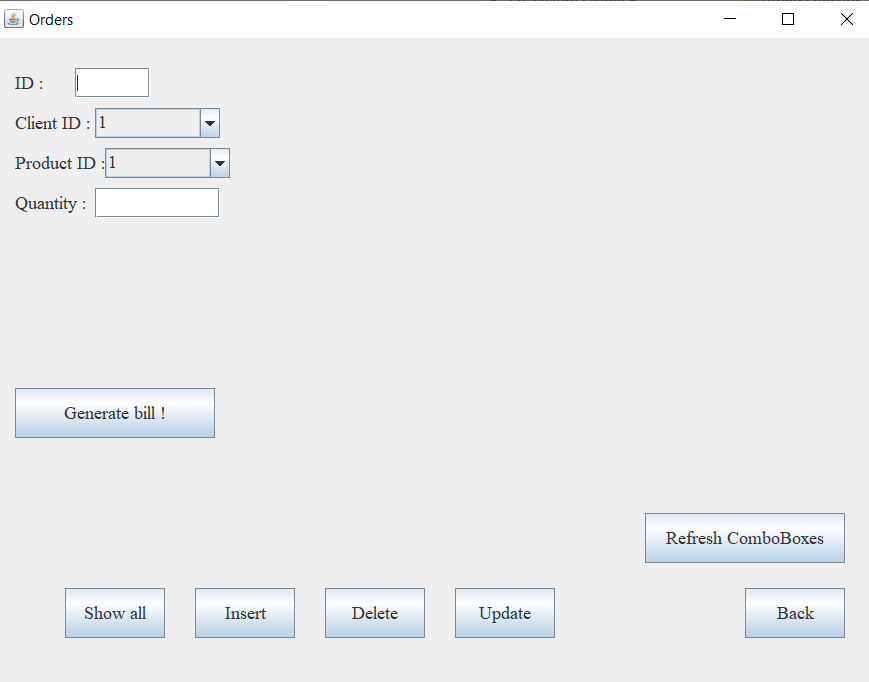
UserInterface - cadrul principal care este „mama” tuturor celorlalte cadre din această aplicație. Oferă acces la celelalte cadre

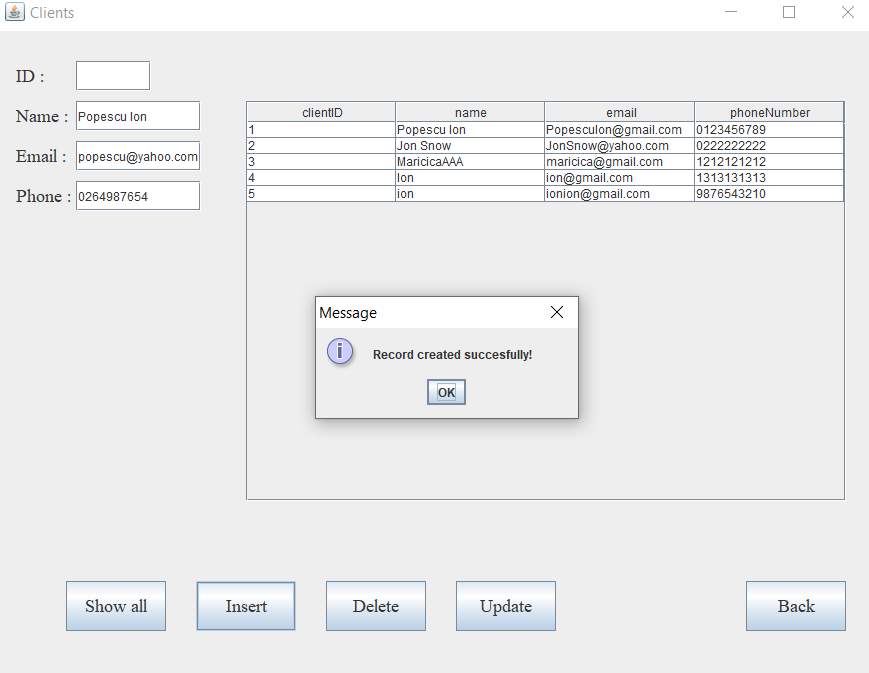
1. Graphical User Interface

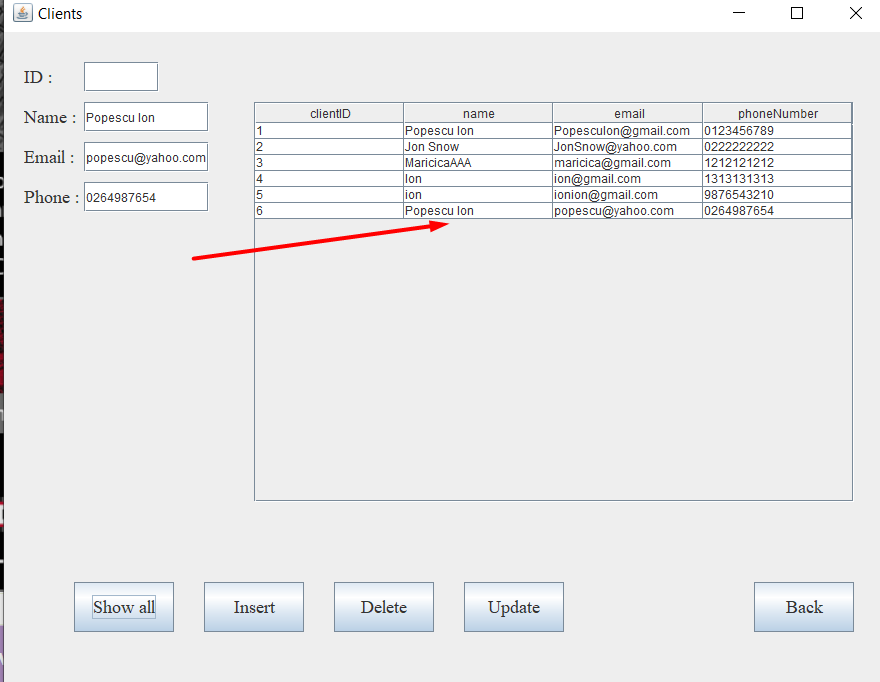
Interfața este una foarte prietenoasă, deoarece toate butoanele cu care interacționează utilizatorul spun foarte clar ce ar trebui să facă.







1. Results



1. Conclusions

Acest proiect a fost un exercițiu bun de reamintire a conceptelor POO învățate în primul semestru, dar și de învățare a altora noi, mi s-a părut foarte util și provocator la început. Sunt câteva lucruri învățate pe care le-aș prezenta în continuare.

În primul rând, managementul timpului este foarte, foarte, foarte, crucial, pentru că un bun spirit organizatoric te ajută să vezi lucrurile treptat și să faci lucruri din timp te ajută foarte mult.

În al doilea rând, modelarea problemei într-un mod corect de la început vă ajută să o implementați mai rapid.

Prin intermediul acestui proiect am invatat cum se poate realiza conexiunea cu o baza de date folosind Java si cum se realizeaza accesul la datele acesteia cat si manipularea lor. De asemenea am aplicat pentru prima data o arhitectura de tip Layered pentru proiectul meu, am aflat cum se pot genera documente de tip PDF si cum se realizeaza lucrul cu tabele in Java. O tehnica foarte importanta pe care o voi aplica si in proiectele mele viitoare ce se foloseste foarte mult in cazul acestei aplicatii este reflection si folosirea tipurilor generice.

În al treilea rând, am ajuns la concluzia că a înfrunta probleme cu codul tău și a încerca să-l faci să funcționeze singur, prin intermediul cercetării, are avantajul de a învăța concepte noi și de a folosi mai bine cele cunoscute. Până la urmă, unul dintre cele mai importante lucruri pe care le-am învățat este să îmi fac interfețele din cod, pentru că semestrul trecut am folosit WindowBuilder și mi-am dat seama că crearea unei interfețe nu este atât de ușoară pe cât pare folosind WindowBuilder. După acest proiect pot spune că abilitățile mele de construire a interfeței sunt la zi.

1. Bibliography

Object-Oriented Programming - Lecture Slides of prof. Marius JOLDOS

Programming Techniques – Lectures of prof. Ioan SALOMIE

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

[www.wikipedia.org](http://www.wikipedia.org)

<http://whatis.techtarget.com/definition/model-view-controller-MVC>

<https://www.draw.io/> (for diagrams)