Order Management

Țîmpău Georgiana

Grupa 30228

Cuprins

1.Obiectivul temei

2.Analiza problemei

2.1 Modelarea problemei

2.2 Cazuri de utilizare

3.Proiectare

3.1 Diagrama de clase

3.2 Diagrama de pachete

3.3 Structuri de date folosite

3.4 Algoritmi

3.5 Interfata

4. Implementare

5. Rezultate

6. Concluzie. Dezvoltari ulterioare

7. Bibliografie

1. Obiectivul temei

Luați în considerare o cerere OrderManagement pentru procesarea comenzilor clienților pentru un depozit. Bazele de date relaționale sunt utilizate pentru a stoca produsele, clienții și comenzile.

a. Analizați domeniul aplicației, determinați structura și comportamentul clasei sale și desenați o extensie Diagrama claselor UML.

b. Implementați clasele de aplicații. Utilizați javadoc pentru a documenta clasele.

c. Utilizați tehnici de reflecție pentru a crea o metodă createTable care primește o listă de obiecte și generează antetul tabelului, prin extragerea prin reflecție a proprietăților obiectului și apoi a populației cu tabelul valorile elementelor din listă: JTable createTable (obiecte listate <Object>);

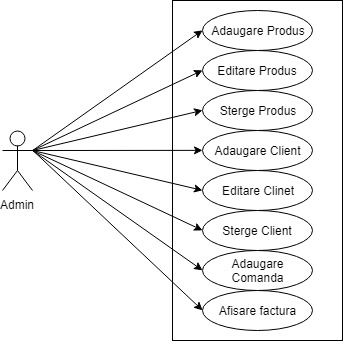
d. Implementați un sistem de programe de utilitate pentru raportare, cum ar fi: stocuri, totaluri, filtre etc.

1. Analiza problemei
   1. Modelarea problemei

Managementul comenzilor se referă la stocarea comenzilor și efectuarea de operații pe ele. Desigur, acestea reprezintă o reprezentare virtuală a unei operațiuni în lumea reală. Un produs virtual enumerat în aplicație este un obiect real și un client înregistrat este o persoană care necesită un serviciu. El își poate alege produsul dintr-o listă de produse. Desigur, ele trebuie stocate fizic într-un depozit, care are un stoc de fiecare produs.

2.2 Cazuri de utilizare

Am simplificat problema luand in considerare doar cazul in care aplicatie aste folosita de aministratorul depozitului de produse. El are accesul la fiecare caracterisitca a plicatiei. Poate manipula produsele in clinetii in toate modurile posibile.



In acest proiect sunt doua scenarii principale: scenariul de introducere a erorii și simularea normală.

Exista 2 scenarii normale de utilizare.

Scenariul 1

Titlu: Adăugarea unui produs în depozit de către manager

Rezumat: Acest caz de utilizare permite administratorului să specifice toate atributele unui anumit produs și apoi să îl adauge la depozit

Actor: manager

1. Administratorul se va afla in panel-ul dedit produselor.
2. Va apasa pe butonul „Add Product”.
3. Se deschide o nouă fereastră și managerul introduce numele, producătorul, prețul produsului, observatii si numarul de produse din depozit.
4. Administratorul apasa pe „Ok”.
5. Aplicația verifică corectitudinea datelor de intrare și adaugă produsul în depozit.
6. Managerul poate alege acum să adauge un alt produs (treceți la pasul 1).

Editarea si stergerea unui produs sunt asemanatoare, ceea ce difera este modul in care utilizatorul solicita comanda. Administratorul apasa click dreapta, va aparea un meniu din care el poate selecta editarea, respectiv stergerea unui produs.

Stergerea, editarea si adaugarea unui client este identica, cu mentiunea ca administratorul trebuie sa se asigura sa fie in panelul dedicat clientilor.

Scenariul 2

Tilul: Crearea unei noi comeni

Rezumar: Acest caz de utilizare permite administratorului crearea unei noi comenzi si selecta.

1. Adminisratorul se afla in panelul de comenzi.
2. Va selecta un produs pe care sa il adauge in comanda.
3. Va selecta numarul de produse pentru comanda.
4. Va asocia un client comenzii.
5. Managerul va apasa pe butonul de “Add Order”.
6. Aplicația verifică corectitudinea datelor de intrare și adaugă comanda în baza de date.
7. Managerul poate alege acum să adauge un alta comanda (treceți la pasul 1).

Scenarii alternative

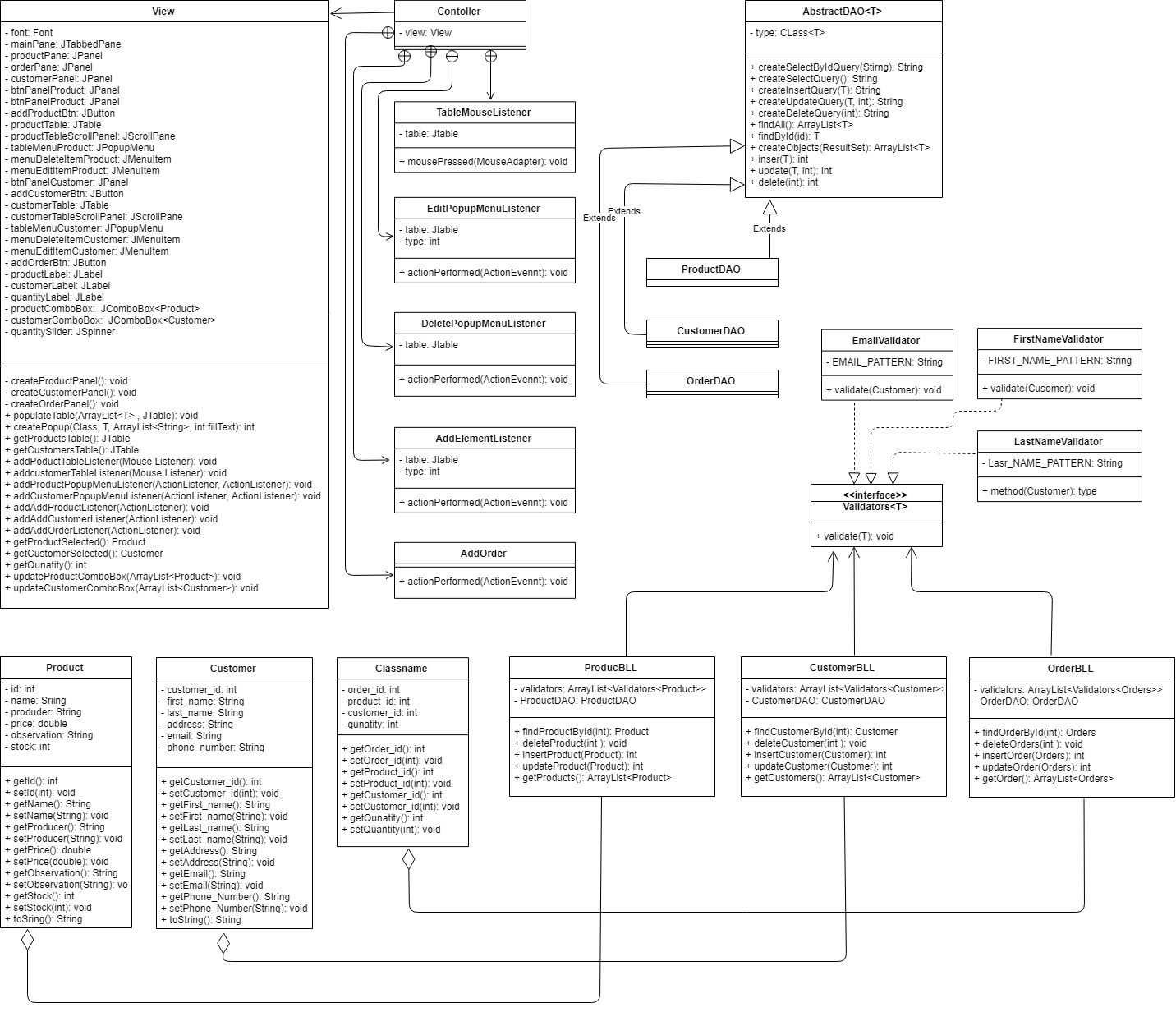
Dacă utilizatorul încearcă să introducă date care nu sunt coerente ca format, el are încă timp să-și repare greșelile până când apasă butonul ”Ok”. În acest stadiu, datele sunt verificate (pentru structura minimă solicitată). Va fi afișat un mesaj de informare și utilizatorul poate modifica ceea ce este necesar pentru ca câmpurile să treacă prin validare.

Scenarii de eroare

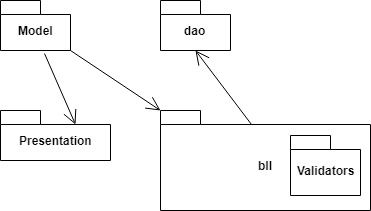
Singurul scenar de eroare este atunci cand aplicatia nu reuseste sa se conecteze la baza de date.

1. Proiectare

3.1 Diagrama de clase



3.2 Diagrama de pachete



3.3 Structuri de date folosite

Cea mai importantă structură de date utilizată în această aplicație este ArrayList utilizat pentru stocarea comenzilor, produsele din deposit si a clientilor. Am ales acesta structura de date pentru ca in momentul de fata aplicatia, respectiv baza de date este de dimensiuni mici. Procesul de cautare este relativ scurt O(n). In cazul in care volumul datelor creste putem imbunatati programul prin conversia ArrayList-utrilor la BinetySearchTree (acestia avand o complexitate logaritmica pentru cautari).

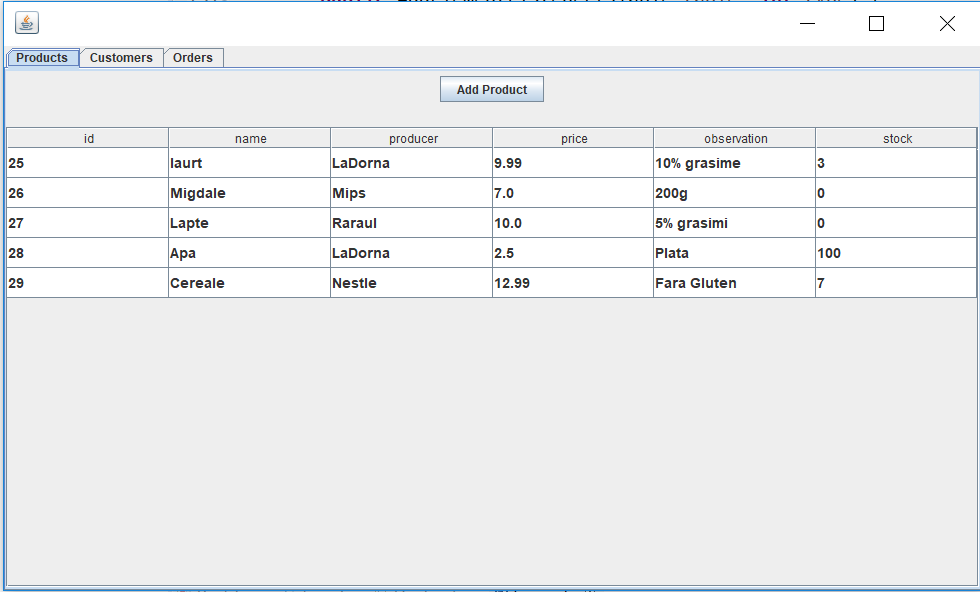
De asemenea, în acest program au fost utilizate Vectori obișnuiti pentru a forma tabelele în care sunt afișate comenzile și produsele.

3.4 Algoritmi

Algoritmii folositi in acest proiect s-au folosit algortimii care au legatura cu structura ArrayList. Acestia sunt implementati de clasa List. Atructura de date care sta la baza clasei List sunt listele dublu inlantuite. Cand de gandim la liste trebuie sa avem:

* Insertie
* Cautare
* Stergere

3.5 Interfata



In momentul in care va incepe rularea aplicatiei ne vom afla in Panel-ul dedicate prdiselor. In table sunt afisare toate produsele din deposit. Pentru adaugarea unui nou produs se apasa butonul “Add Product”.

Pentru editarea sau stergerea unui produs utilizatorul va plasa cursorul pe randul dorit, va apasa click dreapta, iar apoi va aparea un Popup Menu cu obtiunile de editare, reaspectiv stergere.

## Setarea Tabbed Panes-ului dorit se face din coltul stanga sus prin apasarea textului asociat.

## Tabbed Panes-ul „ Customers” este asemanator.

## 

## Tabbed Panes-ul „Orders” este diferit, prezinta 2 ComboBox-uri. Primul este folosit pentru a asocia comenzii un client, iar al doilea pentru a selecta produsul de comandat. Se observa si un JSpinner pentru selectarea numarului de produse. Pentru plasarea comenzii se apasa butonul „Add Order”, iar daca datele introduse trec validarile se adauga comanda in baza de data si in caz contrar se afiseaza un mesaj. Daca comanda a fost adaugata un baza de se va creea o factura.

## Implementare

## Implementarea a fost făcută folosind Eclipse IDE, în limbajul de programare Java. Clasele rezultate sunt o cartografiere a ceea ce a fost descris în diagrama claselor UML și designul clasei. Mai departe, voi prezenta cele mai relevante funcții din aplicații. Metodele Getter și Setter vor fi ignorate din cauza lipsei de complexitate.

## Proiectul a fost realizat dupa arhitectura de tipul Layered. Componentele din modelul de arhitectură Layered sunt organizate în straturi orizontale, fiecare strat realizând un anumit rol în cadrul aplicației (de exemplu logica de prezentare sau logica de afaceri).

## Alt Text

La baza implementarii programului sta tehnica de Reflection. Reflection este un API care este folosit pentru a examina sau a modifica comportamentul metodelor, claselor, interfețelor la timpul de execuție. Prin aceasta tehnica putem invoca metode în timpul rulării, indiferent de specificatorul de acces utilizat cu acestea.

Reflection poate fi utilizată pentru a obține informații despre:

* Clasa – getClass()
* **Constructori -** getConstructors()
* **Metode -**  getMethods()

## In continuarea vom prezenta toate pachetele aplicatie de jos in sus in arhitectura de Layered.

## In pachetul Model e afla clasele care corespund tabelelor din baza de date – Product, Customers si Orders. Practic, in aceste clase se stocheaza informatiile preluate din baza de date. Variabilele instanta a acestor clase corespund cu coloane asociate tabelei respective.

Pachetul dao - Data Access Layer contine toate query-urile pentru prelucrarea datelor din baza de data. Clasele acestui pachet sunt AbstratDAO, ProductDAO, CustomerDAO si OrderDAO. Prima clasa este generica, iar urmatoarele trei o exting pe prima, inlocuid tipul de date T cu Product, Customer, respectiv Orders.

Clasa AbstractDAO este implementata pe baza proprietatiilot tehnicii de Reflection. Pentru a usura scrierea metodelor clase are o variabila instanta de tipul Class care refera la tipul variabilei generice T.

* Private String createSelectByIdQuery(Sring field) – creeaza query-ul care selecteza un element din baza de date pe baza id-ului. Dupa cum am mentionat mai sus compilatorul stie la ce tabela de referm prin intermediul variabilelor genrice.
* Private String createSelectQuery() – creeaza query-ul care returneaza toate interogarile dint-un tabel.
* Private String createInsertQuery(T obj) - creeaza query-ul care insereaza intr-un tabel toate campurile obiectului T, exceptang ID-ul (acesta este auto increment in baza de date). Mentionam ca validarile s-au creeat inainte apelarii acestei metode.
* Private String createUpdatetQuery (T obj, int id) - creeaza query-ul care face update la interogarea cu ID-ul id, iar noile valori ale campurilor sunt stocate in obiectul obj.
* Private String createDeletetQuery (int id) - creeaza query-ul care sterge interogarea cu ID-ul id.
* Punblic ArrayList<T> findAll () – se folosete de prima metoda pentru a converti fiecare data dintr-o table in obiecte de tipul T. Ne bazam pe faptul ca rezultatul unui query este stocat intr-un obiect de tipul ResulSet.
* Public T finById (int id) – se foloseste de a doua metoda descrisa anterior pentru a returna data care are id-ul egal cu id-ul dat ca parametru al functiei
* Private ArrayList<T> createObjects(ResultSet resultSet) – este una din principalele metode. Primeste ca argument un rezultatul unui query si il converteste in obiectele corespunzatoare in java. Este o metoda esentiala, pentru ca fara acesta nu am putea manipula in aplicatia nostra datele din baza de date.
* Public int Insert(T obj) – se foloseste de metoda de a crea un query pentru insertie a unui obiect in tabel, cu specificarea ca aceata metoda face conexiunea cu baza de date si implicit executa query-ul. Returneaza numarul de randuri inserate, preluat din valoarea rezultata de excutia query-ului. Mentionam ca in baza de date nu se insereaza ID-ul unui element deoarece ID-urile in baza de date a plicatie sunt auto\_incremental.
* Public int update(T t, int id) – se foloseste de metoda care creeaza un query pentru editarea unui obiect din tabel, cu specificarea ca aceata metoda face conexiunea cu baza de date si implicit executa query-ul. Returneaza numarul de randuri editate, preluat din valoarea rezultata de excutia query-ului.
* public int delete (int id) - – se foloseste de metoda care creeaza un query pentru stergerea unui obiect din tabel, cu specificarea ca aceata metoda face conexiunea cu baza de date si implicit executa query-ul. Returneaza numarul de randuri sterse, preluat din valoarea rezultata de excutia query-ului.

Dupa cum am mai mentionat clasele ProductDAO, CustomerDAO, OrderDAO extind clasa AbstractDAO. In momentul extinderii se inlocuieste tipul de date T cu Product, Customer, respectiv Orders. Folosirea tehnii de reflexion face posibila folosirea metodelor din clasa parinte, in clasele pe care o mostenes indiferent de variabilelel instanta a claselor. Clasele copil nu contin proprii. In aceasta aplicatie folosim doar metodele de mostenite.

Urmatorul pachet din erarhie este pachetul bll - Business Logic. Acesta contine apelul metodelor de select, insert, update si delete din pachetul dao. In alte pachete ale aplicatie cand dorim sa facem operati cu baza de date vom apela metodele din acest pachet. Practic acest pachet poate fi considerata o „interfata” a bazei de date si operatiilor pe aceasta.

Tot in acest pachet avem si un subpachet validator care contine validatori pentru datele care se folosesc la operatiile de insert si update. Daca datele introduse de utilizator nu trec validarile operatia de insert/update este suspendata.

Pachetul Presentation este cel care constuieste interfata cu utilizatorul . Aici avem doua clase View si Contoller.

Clase View contine toate componentele din interfata grafica (butoane, tabele, etc). In aceasta clasa avem doua metode care folosesc tehnica de refexion impreuna cu variabilele generice. Prima metoda o folosim pentru popularea tabelelor, iar a doua are rolul de formarea ferestrei popup. Aceasta o folosim la operatiile de editare unui camp si adaugare unui nou Produs / Clinet in baza de date.

Clasa Controller contine toti ascultatorii elementelor din interfata cu utilazatorul. Practic, daca nu am avea aceasta clasa interfata noasta grafica nu ar face nimic nimic la apasarea unui buton sau la solicitarea oricarei comenzi. Clasa contine doar clase interioare, aceastea la randul lor implementaza interfata ActionListener, cu metode de implementat actionEvent. Aceasta metoda se va executa in momenul in care utilizatoul apasa elementul grafic care are asculatorul respectiv.

Tot in cardul proiectului mai avem si pachetul App care contine clasa Start. Aceasta este contine metoda *main*, fara de care programul nu se poate executa. Complexitatea metodei etse detul de redusa, acem o instantiere a Clasei View si o instantiere a clasei Contoller.

1. Rezultate

O prezentare rapidă a aplicației arată că am ajuns la următoarele rezultate:

• O aplicație cu acces ușor

• Vizualizați produsele disponibile

• Comandarea produselor

• Posibilitatea de a adăuga un produs nou

• Posibilitatea de a adăuga un client nou

• Posibilitatea de a edita un produs

• Posibilitatea de a edita un client

• Posibilitatea de a sterge un produs

• Posibilitatea de a sterge un cient

• Posibilitatea de a crea o factura in momentul adaugarii unei noi comenzi

1. Concluzie. Dezvoltari ulterioare

Pe parcursul acestei teme am învățat:

• Cum să citiți și să scrieți obiecte dintr-un fișier

• Cum să folosiți tehnica de reflection

• Cum se utilizează JTables și JComboBoxes

• Cum să utilizați keyEvents

Deși proiectul functioneaza, este doar baza cadrului privind gestionarea comenzilor. Ca evoluții ulterioare, se poate observa cu ușurință următoarele:

• Posibilitatea de a anula o comandă

• Utilizatori cu priorități diferite

• Separarea timpului de comandă și a timpului de expediere

• Informații suplimentare privind comenzile și produsele

• O interfață grafică mai bună

1. Bibliografie

<https://www.codejava.net/java-se/swing/jtable-popup-menu-example>

<http://baseprogramming.com/blog1/2017/08/24/automating-jdbc-crud-operations-with-reflection/>

<http://www.coned.utcluj.ro/~salomie/PT_Lic/4_Lab/HW3_Tema3/Tema3_HW3_Indications.pdf>

<https://utcn_dsrl@bitbucket.org/utcn_dsrl/pt-layered-architecture.git>

<https://utcn_dsrl@bitbucket.org/utcn_dsrl/pt-reflection-example.git>