Documentatie de descriere solutie

Tehnici de Programare

Order Management

Coblisan George, grupa 30225

An academic: 2020 – 2021

Abstract

Acest document urmareste descrierea programului, solutia abordata, tehnicile de programare folosite in elaborarea aplicatiei, utilizarea aplicatiei, analiza problemei, rezultate.

Cuprins

1. Cerinte functionale

Aceasta aplicatie dezvolta un sistem de management cu baze de date relationale pentru procesarea clientilor doritori sa cumpere produse dintr-un depozit. Baza de date relationala este folosita pentru a stoca 3 tabele: Client, Produs si Comanda.

Obiectivul principal al unei baze de date relationale este de a stoca si a oferi acces punctelor de date care sunt legate intre ele, de a urmari inventarele, a procesa tranzactiile de comert electronic, a gestiona cantitati uriase de informatii despre clienti si multe altele. O baza de date relationala poate fi luata in considerare pentru orice nevoie de informatii in care punctele de date se coreleaza si trebuie sa fie gestionate intr-un mod sigur, bazat pe reguli si consecvent.

E de dorit ca aplicatia sa aiba o interfara usor de utilizat si prietenoasa astfel incat orice tip de utilizator sa o foloseasca cu usurinta si cu drag.

2. Constrangeri de implementare

Din punctul meu de vedere, constrangerile fundamentale ale acestei aplicatii sunt urmatoarele:

- Impartirea in 4 pachete importante si esentiale: Model classes (reprezinta modelele de date pentru aplicatie), Business Logic classes (contine logica din spatele aplicatiei), Presentation classes (contine interfata si clasele adiacente acesteia, precum Controller) si Data access classes (contine clasele care acceseaza baza de date).
- Folosirea colectiilor din java (List) in schimbul utilizarii array-urilor in cat mai multe cazuri posibile
- Folosirea buclei foreach in schimbul buclei clasice (for int i=0...)
- Implementarea claselor sa contima maxim 300 de linii (cu exceptia claselor UI) si a metodelor sa contina maxim 30 de linii (cu anumite exceptii)
- Folosirea functiei javadoc pentru documentarea claselor si generarea corespondentelor din aplicatie
- Utilizarea unei baze de date relationale pentru a stoca datele aplicatiei, minim cele 3 tabele: Client, Product si Order
- Afisarea facturii generate intr-un fisier .txt pentru o vizulizare mai rapida si cu usurinta
- Minim 3 ferestre de interfata: una pentru a opera pe tabela Client (adaugare, editare, stergere si vizualizare), una pentru a opera pe tabela Product (adaugare, editare, stergere si vizualizare) si una pentru a selecta un client si un produs si a face o comanda.
- Utilizarea tehnicii Reflection

3. Obiectivul aplicatiei

Principalul obiectiv al acestei aplicatii este ca interfata sa fie usor de folosit, prietenoasa si sa functioneze corect alaturi de afisarea tuturor interfetelor si a mesajelor de eroare. Totodata, este esential ca aplicatia sa dezvolte o arhitectura impartita pe clase specifice si in pachete pentru o mai buna organizare a codului scris si o eleganta a acestuia, dar si pentru a opera cu solutii moderne si demne de un programator.

Aplicatia ar trebui sa functioneze prin definirea urmatoarelor: crearea bazei de date, pachetelor specifice, abordarea tuturor claselor.

Bineinteles ca un alt obiectiv important este ca toate operatiile sa fie functionale pe toate cazurile posibile si sa informeze utilizatorul in cazul in care a introdus date de intrare invalide sau nu a introdus toate datele necesare pentru functionarea simulatorului cu un mesaj de eroare si un mesaj informativ.

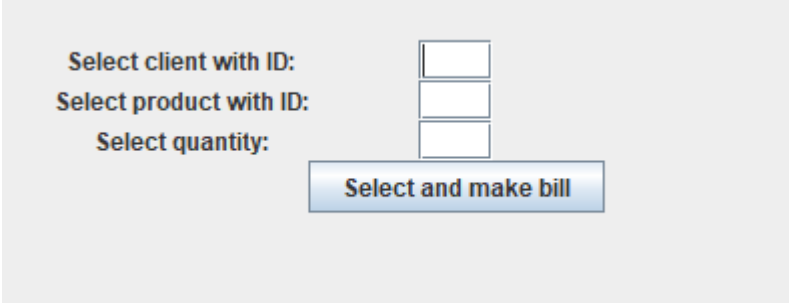
Consider ca obiectivul pentru dezvoltator este sa analizeze toate cazurile posibile si sa adapteze problema la nivel de cod in cel mai simplu mod posibil deoarece acest mediu de dezvoltare ofera o multitudine de avantaje si o infinitate de moduri de a aborda problema, iar pentru utilizator sa inteleaga modul de functionare al aplicatiei si sa faca diferenta daca rezultatul afisat este cel asteptat sau nu.

4. Utilizarea aplicatiei

Meniul aplicatiei este unul primitor, simplu de folosit pentru orice tip de utilizator. La pornirea aplicatiei se va deschide o fereastra pentru a selecta pe ce tabela dorim sa operam. Poza urmatoare afiseaza acest lucru:

Modul de lucru pentru tabelele Client si Product este identic, contine cele 4 operatii de adaugare, editare, stergere si vizualizare. Pentru adaugare este nevoie sa introducem toate datele in text field-uri separate, pentru editare este nevoie sa introducem ID-ul si apoi datele care urmeaza sa fie inlocuite, pentru stergere trebuie sa introducem doar ID-ul si pentru vizualizare doar sa apasam butonul. Voi prezenta interfara doar pentru Client.

In tabela Order trebuie sa introducem ID-ul unui client, ID-ul unui produs si cantitatea, apoi prin apasarea butonului se va genera o factura intr-un fisier txt si se va adauga comanda facuta in tabela Order. Interfata arata astfel:



In continuare voi prezenta pasii care trebuie efectuati de catre utilizator pentru folosirea in mod corect a aplicatiei.

- Dupa pornirea aplicatiei, primul pas este selectarea tabelei dorite, recomandarea mea fiind ca utilizatorul sa intre pe rand in tabela Client, Product si sa vizualizeze clientii si produsele existente in baza de date.
- Dupa ce clientul s-a acomodat, poate daca doreste sa faca modificari in baza de date asupra tabelelor Client si Product.
- In continuare, utilizatorul poate sa efectueze o comanda accesand butonul „Order” si sa genereze o factura pentru clientul si produsul dorit.

5. Proiectare

Etapa de proiectare a aplicatiei consta in algoritmii din spate si modul de gandire a efectuarii tuturor operatiilor. In prima faza am preluat structura proiectului din prezentarea suport si m-am acomodat cu aceasta abordare, urmand sa-mi definesc clasele necesare pentru tabelele din baza de date (bineinteles prima data am creat tabelele in SQL) si clasele corespunzatoare fiecarei tabele pentru a efectua interogarile necesare: cautare, adaugare, stergere, editare si vizualizare.

Accesarea unei baze de date dintr-o aplicatie Java se realizeaza prin intermediul unui program de comanda (driver) specific unui anumit sistem de gestiune a bazelor de date. Un driver intermediaza legatura dintre aplicatii si baze de date.

Java DataBase Connectivity JDBC reprezinta un API care permite lucrul cu baze de date relationale. Prin intermediul JDBC sunt transmise comenzi SQL la un server de baze de date. Folosind JDBC, nu este necesara dezvoltarea mai multor aplicatii pentru a accesa servere de baze de date care utilizeaza sisteme diferite de gestiune a bazelor de date (Oracle, MySQL, Sybase). Este suficienta o singura aplicatie, care sa utilizeze API-ul JDBC, pentru a transmite comenzi SQL la serverul de baze de date dorit. In felul acesta este asigurata portabilitatea aplicatiei.

O conexiune la un server de baze de date reprezinta un canal de comunicatii prin care sunt transmise cereri SQL si sunt returnate raspunsuri corespunzatoare. Stabilirea unei conexiuni dintr-o aplicatie Java presupune inregistrarea (incarcarea) unui driver si realizarea conexiunii propriu-zise prin intermediul clasei DriverManager din pachetul java.sql.

Dupa stabilirea unei conexiuni la serverul de baze de date si selectarea unei baze de date active este necesara crearea unei instante de tip Statement prin metoda createStatement() a clasei Connection. Instanta de tip Statement permite manipularea unor comenzi SQL.

De cele mai multe ori rularea unei comenzi SQL pe o baza de date are ca si rezultat un set de date care apar sub forma tabelara. Pentru a obtine date dintr-o baza de date pot fi rulate comenzi SQL prin intermediul metodei executeQuery(). Aceasta metoda returneaza informatia sub forma unor linii de date (inregistrari), care pot fi accesate prin intermediul unei instante de tip ResultSet.

Inregistrarile dintr-un obiect de tip ResultSet pot fi parcurse cu ajutorul metodei next(). De asemenea, accesarea valorilor corespunzatoare anumitor coloane poate fi realizata prin metode de tipul get<Type>() (getString(), getInt()). Coloanele dintr-un tabel pot fi referite prin intermediul numelui sau prin intermediul pozitiei coloanei in interiorul tabelului (prima coloana din tabel are pozitia 1). Toate tabelele unei baze de date detin meta-date care descriu denumirile si tipurile de date specifice fiecarei coloane. In acest fel poate fi utilizata clasa ResultSetMetadata pentru a obtine numarul de coloane dintr-un tabel sau denumirile coloanelor.

6. Implementare

In acest capitol voi prezenta fiecare clasa in parte alaturi de metodele acestora.

- **Start**

Aceasta clasa este necesara pentru executia programului si apeleaza metoda clasa Controller si View pentru crearea interfetei si creeaza un Client pentru a ilustra modul de functionalitate a tehnicii Reflection.

- **Reflection Example**

Aceasta clasa contine o metoda care abordeaza tehnica Reflection si afiseaza in consola headerul unei tabele (prima linie din aceasta) in momentul in care utilizatorul doreste sa vizualizeze tabela.

- **View**

Aceasta clasa creeaza intreaga interfata si contine metode pentru operatiile necesare de adaugare, stergere, editare si vizualizare, bineinteles metode in care se apeleaza alte metode din clasele specifice acestor operatii, prezentate in paragrafele urmatoare.

- **Controller**

Aceasta clasa contine functionalitatea butoanelor din toata interfata si adauga Action Listener pe acestia, urmand sa apeleze metodele specifice pentru fiecare operatie din interfata, din clasa View.

- **Product**

Aceasta clasa creeaza constructori, getters si setters pentru toate atributele tabelei Product din baza de date.

- **Client**

Aceasta clasa creeaza constructori, getters si setters pentru toate atributele tabelei Client din baza de date.

- **Order**

Aceasta clasa creeaza constructori, getters si setters pentru toate atributele tabelei Order din baza de date.

- **Connection Factory**

Aceasta clasa stablieste conexiunea cu baza de data si o opreste.

- **ProductDAO**

Aceasta clasa efectueaza toate interogarile specifice tabelei Product: cautare, adaugare, stergere, editare si stabileste o conexiune cu baza de date, defineste un statement pentru interogare si preia atributele clasei cu ajutorul getters urmand sa execute interogarea si sa rezulte un mesaj de eroare in cazul in care a esuat.

- **ClientDAO**

Aceasta clasa efectueaza toate interogarile specifice tabelei Client: cautare, adaugare, stergere, editare si stabileste o conexiune cu baza de date, defineste un statement pentru interogare si preia atributele clasei cu ajutorul getters urmand sa execute interogarea si sa rezulte un mesaj de eroare in cazul in care a esuat.

- **OrderDAO**

Aceasta clasa efectueaza toate interogarile specifice tabelei Order: adaugare si stabileste o conexiune cu baza de date, defineste un statement pentru interogare si preia atributele clasei cu ajutorul getters urmand sa execute interogarea si sa rezulte un mesaj de eroare in cazul in care a esuat.

- **AbstractDAO**

Aceasta clasa efectueaza toate interogarile specifice uneia dintre tabelele Client sau Product: cautare, adaugare, stergere, editare si stabileste o conexiune cu baza de date, defineste un statement pentru interogare si preia atributele clasei cu ajutorul getters urmand sa execute interogarea si sa rezulte un mesaj de eroare in cazul in care a esuat. Insa abordeaza tehnica Reflection fara ca, obiectele Client sau Product sa fie instantiate undeva in aceasta clasa.

- **ProductBLL**

Aceasta clasa contine metode pentru a accesa metodele specifice operatiilor pe tabela descrise anterior.

- **ClientBLL**

Aceasta clasa contine metode pentru a accesa metodele specifice operatiilor pe tabela descrise anterior.

- **OrderBLL**

Aceasta clasa contine metode pentru a accesa metodele specifice operatiilor pe tabela descrise anterior.

- **AgeValidator**

Aceasta este o simpla clasa de verificare a varstei unui client, sa nu fie mai mica de 7 si mai mare de 50.

- **EmailValidator**

Aceasta este o simpla clasa de verificare a emailului unui client, daca indeplineste formatul standard al unui email.

7. Rezultate asteptate de utilizator

In acest capitol voi prezenta cateva screenshoturi cu datele rezultatele aplicatiei.
Dupa generarea unei facturi din interfata „Order” selectand ID-ul clientului, ID-ul produsului si cantitatea dorita se va genera o factura numai daca cantitatea dorita este mai mica decat cantitatea existenta in depozit si bineinteles daca exista. Factura generata in fisierul .txt arata in felul urmator:

```
Bill
Client: Edi edi@gmail.com 20
Product: ciment 43 63
```

Vizualizarea unuia dintre tabele direct din interfata arata in felul urmator:

ID	Produs	Quantity	Stock
1		-5	-5
3	ciment	1	41
4	ciment	41	61
5	bca	22	27

8. Concluzii si dezvoltare ulterioara

Aplicatia poate fi imbunatatita din multe puncte de vedere, iar cele pe care as fi dorit eu sa le implementez sunt urmatoarele:

- Un sistem de informare pentru utilizator despre cum ar trebui sa foloseasca si sa introduca datele si un sistem care sa ii afiseze pe langa rezultatul din fisier, adica factura o factura mai stufoasa precum cea din viata reala.
- O interfata mai primitoare, cu un altfel de meniu si totodata usor de utilizat, dar si mai animata referitor la adaugarea, stergerea, editarea si vizualizarea tabelelor.
- Un istoric al datelor introduse de catre utilizator si un mesaj daca pentru acele date aplicatia a functionat corect sau nu alaturi de restul informatiilor deja existente.
- Un sistem si o noua interfata care sa permita utilizatorului sa selecteze mai multe produse la generarea unei comenzi.