

MINISTERUL EDUCAȚIEI ȘI CERCETĂRII ȘTIINȚIFICE



UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

**FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE DEPARTAMENTUL
CALCULATOARE**

Simularea activitatii unui depozit Documentație

Berea Roxana

Grupa 30226 | An 2 semestrul 2

Cuprins

1. Obiective
2. Analiza problemei
- 3.Exemplu de lucru
- 4.Implementare
- 5.Proiectare
- 6.Testare si rezultate
- 7.Concluzii
- 8.Bibliografie

1.Obiectiv

Obiectivul acestei teme de laborator a fost sa proiectam si sa implementam un sistem de management al unui deposit, avand scopul de a analiza procesarea comenzilor si a clientilor. Produsele din deposit sunt stocate intr-o baza de date relationala, la fel si clientii si comenzile primite.

Sistemul va avea o interfata grafica, care poate fi utilizata usor si confortabil de catre orice utilizator, unde vor putea fi introduce datele clientilor, detaliile produselor si datele necesare pentru procesarea comenzilor, fiecare dintre acestea intr-o fereastră diferita ce apare pe ecran la rularea programului.

Pentru fiecare client se cunoaste id-ul, care este unic, numele, adresa fiecaruia de email si comapia. Pentru produse se retine in baza de date un id, care de asemenea este unic, constituind cheia primara a tabelului, denumirea produsului, cantitatea aflata in stoc in depozit si pretul pe bucata. Pentru realizarea unei comenzi si retinerea cu success a acesteia in baza de date sunt necesare: id-ul clientului, id-ul produsului si cantitatea cumparata pentru produsul respectiv.

Această temă are ca scop simularea unui magazin, care preia comenzi, pe baza unui client și a unui produs ales și cantitatea dorită din produsul respectiv. Aplicația este realizată printr-o conexiune la o bază de date, creată de noi. Aplicația permite adăugarea de noi clienți, noi produse, actualizarea acestora și preluarea de comenzi .

Java oferă JDBC (Java DataBase Connectivity) ca parte a Java SDK (Software Development Kit). Folosind acest API, este foarte ușor să vă conectați la o bază de date relațională. Deci, ce este o bază de date, oricum? Pentru un programator sau un entuziast al tehnologiei, conceptul de bază de date este ceva care poate fi cu adevărat de acordat. Cu toate acestea, pentru mulți oameni, conceptul de bază de date în sine este puțin strain...

În timp ce API-ul JDBC de bază este inclus în java, conectarea la o bază de date particulară, cum ar fi MySQL sau SQL Server, necesită o componentă suplimentară cunoscută sub denumirea de driver de bază de date. Acest driver de bază de date este o componentă software care traduce apelurile JDBC de bază într-un format înțeles de acea bază de date.

Pentru simularea activitatii descrise se va folosi interfata grafica, utilizatorul introducand datele necesare simulării si se va putea vizualiza procesul descris.

De asemenea, aplicatia trebuie sa respecte paradigmele Programarii Orientate pe Obiect, clasele si metodele trebuie sa respecte numarul de linii impus (maxim 300, respectiv 30) si sa se respecte conventiile Java legate de denumirea claselor, metodelor si variabilelor.

2. Analiza problemei

Analiza problemei presupune identificarea legaturilor dintre clasele proiectului, precum si functionalitatea acestuia. In acest sens, programarea orientata pe obiect ne permite implementarea aplicatiei cunoscand un numar minim necesar de informatii.

Programarea orientată pe obiect (Programare Orientată Obiectual) este unul din cei mai importanți pași făcuți în evoluția limbajelor de programare spre o mai puternică abstractizare în implementarea programelor. Ea a apărut din necesitatea exprimării problemei într-un mod mai simplu, pentru a putea fi înțeleasă de cât mai mulți programatori. Astfel unitățile care alcătuiesc un program se apropie mai mult de modul nostru de a gândi decât modul de lucru al calculatorului.

Bazele de date orientate pe obiecte permit crearea de obiecte complexe din componente mai simple, fiecare avand propriile attribute si propriul comportament. Printre domeniile care se preteaza in mod deosebit la o tartare orientate pe obiecte se afla si: proiectarea, fabricarea si ingineria asistate de calculator; simularea si modelarea; sistemele informationale spatiale; multimedia si multe altele. Obiectivele principale ale acestor sisteme de gestiune fiind: puterea de modelare superioara a datelor, posibilitati de deductive superioara, ameliorarea interfetei cu utilizatorul.

Pentru asigurarea corectitudinii datelor de intrare se vor face anuimite verificari pe datele introduse de utilizator si se vor afisa mesaje corespunzatoare pentru eventualele greseli de scriere. Pentru ca aplicatia sa functioneze corect datele introduse trebuie sa fie ori numere intregi reprezentand valori corespunzatoare etichetei fiecarui camp(id-ul pentru client si produs, pretul unui produs, cantitatea disponibila pe stoc, cantitatea dorita pentru cumparare) ori String(sir de caractere), reprezentand numele clientului, respectiv adresa acestuia, numele produsului sau descrierera produsului.

In decursul implementarii am luat in calcul diferite scenarii de utilizare a aplicatiei, fie ca este vorba despre introducerea corecta/ incorecta a datelor, fie ca ne referim la afisarea rezultatelor in campurile corespunzatoare. In cazulul in care nu au fost completate toate campurile inainte de efectuarea operatiilor, editarea datelor din tabele, precum si inserarea nu vor fi posibile si va aparea pe ecran un pop-up cu un mesaj sugestiv, astefl incat utilizatorul sa realizeze care este problema si sa poata folosi in continuare aplicatia cu usurinta. Daca utilizatorul doreste sa starga un rand din tabel, aceasta operatie este posibila si daca nu sunt toate campurile completate, insa field-ul 'id' este necesar in orice situatie, acesta fiind cheia primara, adica caracteristica unica pentru fiecare client, respectiv produs. De asemenea vor fi afisate mesaje corespunzatoare pentru incercarea de a sterge un client care nu exista, pentru incercarea de a introduce un client sau produs deja existent sau pentru crearea de comenzi pentru clienti sau produse care nu exista in baza de date sau care nu se mai afla pe stoc.

3. Exemplu de lucru

Use Case: simularea activitatii de creare comanda

In aceasta situatie, actorul va fi utilizatorul, adica persoana care doreste simularea crearii unei comenzi. Acesta trebuie sa aleaga un client, un produs si sa plaseze comanda. Inafara de acest

caz, actorul ar mai putea sa insereze un client, sa insereze un produs, sa stearga un client sau produs si sa editeze un rand deja existent in baza de date. Dupa introducerea datelor, utilizatorul va apasa butonul "add" de adaugare a comenzi, unde se pot adauga mai multe produse, care vor fi afisate intr-un tabel dupa apasarea butonului "orders" din partea dreapta-jos a ferestrei. Dupa realizarea acestei actiuni, se va observa actualizarea in timp real a tabelului cu comenzile plasate, dar si a numarului de produse ramase pe stoc in tabelul din fereastra paginii de produse, cat si daca verificam baza de date direct in MySQL.

Un exemplu functional de lucru, in care toate datele sunt introduse corect este urmatorul:

La prima fereastra aparuta selectam varianta CLIENT prima data pentru a ne asigura ca tabela clientilor este populata, iar in caz contrar putem adauga un client nou prin completarea casutelor libere din patratul "ADD". Dupa completarea tuturor spatiilor cerute va trebui sa apasam butonul "ADD" din partea de jos si putem verifica daca clientul s-a adaugat corect apasand ultimul buton din partea de jos, din dreapta, de pe fereastra. Astfel ne va aparea tabelul cu clientii pe care depozitul i-a avut pana acum.

Urmatorul pas este sa mergem inapoi la pagina de pornire si sa ne asiguram ca tabelul de produse este populat. Pentru aceasta apasam butonul "BACK" de doua ori, alegem "PRODUCT" din fereastra de pornire si repetam pasii pe care i-am efectuat pentru a adauga un client, de data aceasta cu date specifice unui produs. Atat la client cat si la produs trebuie sa ne asiguram ca am completat toate spatiile libere si ca am introdus tipul de date corect pentru ca aplicatia sa functioneze corect.

Dupa ce ne-am asigurat ca exista cel putin un client si un produs in tabelele corespunzatoare acestora putem trece la efectuarea comenzii. Pentru aceasta mergem inapoi la pagina principala si apasam butonul "ORDER". Aici, la fel ca inainte, putem adauga o comanda completand campurile cerute din prima parte de "ADD", dupa care putem verifica daca comanda a fost plasata corect prin deschiderea tabelului de comenzi. Daca este necesar, daca o greseala a fost facuta sau clientul doreste o cantitate mai mare de produs, se poate modifica comanda, fara a trebui sa facem una noua. Completam in patratelul de editare datele comenzii initiale, cu acelasi ID si pastram datele anterioare, modificand doar datele pe care doream sa le editam. Apasam butonul de jos din patratel pentru a efectua editarea, dupa care, din nou, putem verifica tabelul pentru a ne asigura ca totul a functionat corect.

La plasarea unei comenzi se va crea automat si o factura. Astfel, dupa plasarea unei comenzi vom putea vizualiza datele comenzii in fisierul text "bill.txt", in care se scrie automat.

4. Implementare

Pentru inceput, am ales sa impart proiectul in pachete, respectand conventiile programarii pe obiect si grupand clasele astfel:

- 🚦 Pachetul businessLogicLayer

- avand clasele ClientBll, ProductBll, OrderBll, FileWriterr.

Clasa ClientBll gestioneaza interogarile necesare pentru obiecte de tip Client. Drept variabila instanta aceasta clasa are un obiect de tipul clientDAO, necesar pentru apelarea metodei din pachetul DAO.

In aceasta clasa este descrisa o metoda care returneaza lista de clienti, cu ajutorul careia este realizat tabelul pentru interfata grafica, o metoda care se ocupa de inserarea clientilor in baza de date, o metoda folosita la editarea datelor despre clienti si o metoda folosita pentru a sterge un client din baza de date in functie de id-ul acestuia.

Clasa ProductBll gestioneaza interogarile necesare pentru obiecte de tip Product. Drept variabila instanta aceasta clasa are un obiect de tipul productDAO, necesar pentru apelarea metodelor din „Data Access Operation”.

In aceasta clasa gasim metodele pentru returnarea listei de produse, necesara pentru crearea tabelului ce va fi afisat in interfata grafica, metoda care se ocupa de inserarea produselor in baza de date, metoda folosita la editarea datelor despre produse, precum si metodele pentru stergerea unui produs din baza de date in functie de id-ul acestuia si metoda care cauta in baza de date produsul in functie de id si il returneaza.

Clasa OredrBll care gestioneaza interogarile necesare pentru obiecte de tip OrderR. Drept variabila instanta aceasta clasa are un obiect de tipul orderDAO, necesar pentru apelarea metodelor din „Data Access Operation”.

Aceasta clasa implementeaza metode care se ocupa de returnarea listei de comenzi, , necesara pentru crearea tabelului ce va fi afisat in interfata grafica si o metoda de inserare a comenzilor in baza de date.

Clasa FileWriter

Aceasta clasa realizeaza scrierea intr-un fisier a carui cale este data unei variabile de tip String. Astfel, intr-un fisier numit “bill.txt” se scriu datele unei comenzi, care reprezinta factura comenzii respective. Sunt afisate pe factura: id-ul comenzii, id-ul clientului, id-ul produsului si cantitatea cumparata.

Tot in clasele din pachetul BLL sunt puse comentariile necesare pentru a putea genera fisierele javaDoc.

Pachetul dao

-care contin clasele ClientDAO, ProductDAO, OredrDAO

Fiecare clasa face legatura intre aplicatie si baza de date, adica cu tabelul corespunzator fiecareia din baza de date. In aceste clase este utilizata tehnica de reflexie pentru crearea metodelor necesare accesarii bazei de date.

Metodele implementate in aceste clase realizeaza conexiunea la baza de date, returneaza un obiect de tipul dorit in urma interogarii bazei de date, insereaza in baza de date obiectul trimis ca si parametru, actualizeaza obiectul cu id-ul trimis ca si parametru si sterge obiectul din baza de date in functie de id-ul introdus.

Pachetul connection

Continue **Clasa ConnectionFactory**, care realizeaza conexiunea la baza de date si foloseste pattern-ul singleton.

Pachetul presentation

-contine clasele ClientGUI, ProductGUI, OrderGUI, MainGUI, Controller si ShowCGUI.

Clasa MainGui constituie prima fereasta care apare la rularea programului si aceasta ne permite sa selectam ce tip de operatie dorim sa efectuam in continuare si pe ce tabel. Astfel, aceasta clasa descrie fereastra principala, "Welcome", care are trei butoane: Client, Order si product, care ne vor trimite mai departe la ferestrele in care putem efectua operatiile dorite pe fiecare tabel.

Celelalte clase care contin "GUI" in denumire se ocupa de interfata grafica, acestea constituind celelalte trei ferestre ce vor aparea dupa apasarea butonului de deschidere de pe prima fereasta. Ca variabile instantate avem JLabel-uri, JTextField-uri, in care utilizatorul va introduce datele corespunzatoare etichetelor, JButton-uri pentru inserarea datelor introduse si pentru a putea efectua diferite operatiuni.

Clasa Controller are rolul de a descrie actiunile efectuate de butoanele descrise in celelalte clase ale interfetei, in functie de datele completate in textfield-uri.

Pachetul model

-contine clasele Order, Client, Product si Except

Clasa Client- clientul este entitatea principala in modelarea aplicatiei.

Clasele din acest pachet au drept field-uri exact coloanele tabelurilor corespunzatoare din baza de date. Toate aceste clase contin metode de get si set pentru campurile de id, nume, denumire, adresa etc.

Clasa Except descrie exceptia de care ne vom folosi pentru a anunta utilizatorul ca datele completate nu sunt corecte sau pentru a semnala o oarecare problema in timpul utilizarii aplicatiei.

5. Proiectare

Pentru proiectarea aplicatiei am organizat programul in 6 pachete, asa cum le-am descris in Implementare la punctul 4, in acest sens se poate observa mai jos diagrama UML de clase corespunzatoare.

Clasele au fost numite sugestiv si sunt implementate toate aspectele necesare bunei functionarii a simularii activitatii depozitului.

Interfata grafica a fost conceputa folosind JFrame, reusind astfel sa implementez o GUI cu care orice utilizator se poate familiariza in scurt timp, fiind usor de utilizat, campurile fiind numite explicit si avand butoane de pornire a executiei.

In urmatoarele fraze urmeaza sa prezint rolurile fiecarui pachet:

Presentation este reprezentat de interfata grafica, care faciliteaza interactiunea utilizatorului cu aplicatia. Tot in gui sunt implementati si ascultatorii pentru butoanele din interfata. Modelul este reprezentarea obiectuala a clientilor, connection realizeaza legatura cu baza de date, dao face legatura intre cerintele programului si interogari pentru baza de date. Pachetul bll face legatura intre cerintele utilizatorului transmise prin intermediul interfetei grafice si pachetul dao care se ocupa cu interogari pentru baza de date.

Pachetul Model contine clasele reprezentative Client, Product, Order, Except. Acest pachet

Reprezinta modelul conceptual pe care se bazeaza aplicatia.

Pachetul Bll pachet realizeaza logica care conecteaza interfata grafica cu interogari necesare din baza de date.

Pachetul Connection realizeaza conectarea aplicatiei la baza de date. In cadrul acestei aplicatii am folosit MySQL server pentru baza de date, aceasta fiind de tip relational.

Pachetul Presentation realizeaza implementarea interfetei grafice, implementarea claselor care se ocupa de ascultatori.

Si pachetul DAO cu clasele Client DAO, ProductDAO, OrderDAO.

Diagrama UML de clase arata in felul urmator:



6.Testare si rezultate

Regulile pentru testarea aplicatiei au fost descrise in detaliu la punctul 3, unde am dat si un exemplu de introducere a datelor. Aici voi atasa o imagine in care se poate observa felul in care vor fi afisate rezultatele finale.

ORDER

ADD

ID: 6

Client ID: 3

Product ID: 1

Quantity: 21

Add

EDIT

ID:

Client ID:

Product ID:

Quantity

Edit

DELETE

ID:

Delete

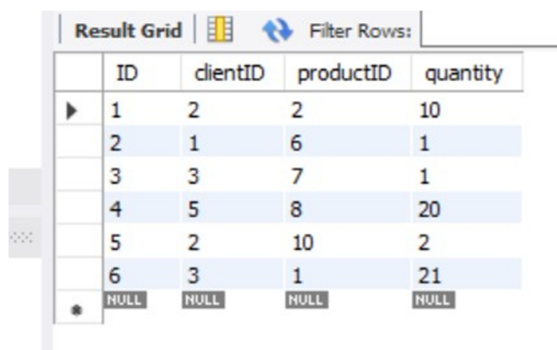
Orders

Back

ID	clientID	productID	quantity
1	2	2	10
2	1	6	1
3	3	7	1
4	5	8	20
5	2	10	2
6	3	1	21

Back

Totodata, informatiile sunt actualizate sin in baza de date:



	ID	clientID	productID	quantity
▶	1	2	2	10
	2	1	6	1
	3	3	7	1
	4	5	8	20
	5	2	10	2
	6	3	1	21
*	NULL	NULL	NULL	NULL

7.Concluzii

In concluzie, prin realizarea acestui proiect mi-am fixat mult mai bine informatia invatata in primul semestru la programarea orientata pe obiect, dar am vazut si metode noi de realizare a interfetei grafice si de testare a programului si de asemenea, acest assignment m-a ajutat s ama familiarizez cu conexiunea programarii orientate pe obiect si a bazelor de date.

Pot fi aduse imbunatatiri la aceasta aplicatie, atat in implementarea operatiilor, cat si la aspectul interfetei grafice si la posibilitatea introducerii datelor intr-un mod mai la indemana utilizatorilor. Dar am reusit intr-un final sa duc proiectul pana la capat, documentandu-ma chiar si de lucruri elementare ale programarii orientate pe obiect, cat si din proiecte realizate si distribuite de alte persoane cu experienta in domeniu.

8.Bibliografie

- ✚ Resursele de curs si laborator
- ✚ Oracle's java documentation<https://docs.oracle.com/javase/tutorial/java/index.html>
- ✚ Youtube tutorials
- ✚ Github projects
- ✚ Stackoverflow
- ✚ <https://www.scrigroup.com/calculatoare/baze-de-date/BAZE-DE-DATE-ORIENTATE-PE-OBIE33193.php>
- ✚ https://ro.wikipedia.org/wiki/Programare_orientat%C4%83_pe_obiecte
- ✚ <https://ro.ephesossoftware.com/articles/programming/how-to-connect-to-a-mysql-database-with-java.html>