Universitatea Tehnică din Cluj-Napoca

Facultatea de Automatizări și Calculatoare

Student : Stanciu Ioan-Octavian

An: 2

Grupa: 30223

Profesor: Dorin Vasile Moldovan

**TEHNICI DE PROGRAMARE FUNDAMENTALE**

**TEMA 3**

**Order Management**

**CUPRINS:**

* Obiectivul temei
* Analiza problemei, modelare, scenarii, cazuri de utilizare
* Proiectare
* Implementare
* Rezultate
* Concluzii
* Bibliografie

***1) Obiectivul temei***

Obiectivul acestei teme a fost sa se creeze o aplicatie care sa menegerieze comenzi. In aceasta aplicatie a fost nevoie sa folosim o baza de date relationala si sa folosim Layered Architecture.

Obiective secundare:

* Crearea bonului unui client la selectarea acestuia si alegerea acestei optiuni. De asemenea se creeaza cate un fisier txt cu numele acestuia care contine numarul produselor si pretul fiecaruia, iar in final suma totala pe care o are de platit
* Cand se efectueaza stergerea unui client se sterg toate comenzile ce au id-ul acelui client si se incrementeaza la loc numarul de produse
* Cand se efectueaza stergerea unui produs se sterg toate comenzie ce au id-ul acelui produs si se decrementeaza, la fiecare client ce a comandat acel produs, cat are de platit in total
* Determinarea numarului de pachete si numele acestora
* Plasarea claselor in pachetele corespunzatoare

***2) Analiza problemei, modelare, scenarii, cazuri de utilizare***

In aceasta aplicatie se poate simula in interfata grafica aceasta aplicatie de menegiere a unor comenzi, insa cu cateva reguli (care vor fi prezentate in capitolele urmatoare).

Descriere use-case pentru simulare in GUI:

* Se deschide aplicatia
* Se ruleaza metoda main din clasa Controller
* In interfata de clienti si in interfata de produse trebuie respectate urmatoarele reguli:
* Pentru a se introduce unul din acestea in interfata corespunzatoare nu trebuie sa se introduca id-ul lor, iar la clienti nu trebuie introdus nimic in campul tota\_bill
* Pentru a sterge unul din acestea in interfata corespunzatoare trebuie introdus doar id-ul
* Pentru a edita unui din acestea in interfatele lor trebuie introduse toate campurile (daca la unele nu se doresc modificarea atunci se vor introduce valorile deja introduse
* In interfata grafica pentru comenzi daca se doreste introducerea unei comenzi trebuie selectat un client, un produs si trebuie sa se introduca numarul de produse care se doresc sa fie comandate dupa care se apasa butonul de order
* In interfata grafica pentru comenzi daca se doreste bonul unui client atunci se selecteaza un client dupa care se apasa butonul "bill" care genereaza in pachetul cu acelasi nume bonul clientului selectat
* In toate cele 3 interfete grafice daca se doreste vizualizarea clientilor/produselor/comenzilor se vor apasa numele de view din interfata grafica corespunzatoare

***3) Proiectare***

Text

Description automatically generated

* Prima decizie pe care am luat-o a fost sa ma ajut de prezentarea oferita de domul profesor
* Dupa am implementat clasele prezentate si am urmat instructiunile
* Dupa mi-am creat cele 3 clase pentru clienti, produse si comenzi, numele lor avand acelasi nume cu numele tabelelor din baza de date. Variabilele instanta au acelasi nume cu coloanele din tabelele corespunzatoare si am gettere si settere pentru fiecare
* Dupa mi-am creat clasele ClientsDAO, ProductsDAO si OrdersDAO care reprezinta clase care ajuta la accesarea datelor din baza de date
* Dupa mi-am creat clasa Commands care ajuta la comunicarea dintre acestea 3 pentru a aduce modificari la baza de date
* Intr-un final mi am creat cele 4 clase pentru interfetele grafice: una pentru clienti, una pentru produse, una pentru comenzi si una pentru a initializa tabelul, esential pentru afisarea informatiilor dintr-un anumit tabel

***4) Implementare***

* Clasa ConnectionFactory:

In aceasta clasa am ca variabile instanta informatiile necesare pentru a ma putea conecta la propria baza de date precum numele user-ului si parola

In constructor imi initializez driver-ul

In metoda createConnection imi creez conexiunea la baza de date folosindu-ma de DBURL, numele user-ului si parola lui

In metoda getConnection este un getter pentru a-mi putea returna conexiunea

Dupa acestea am 3 metode denumite toate close care au rolul de a inchide conexiunea unei variabile de tip Connection, Statement respectiv ResultSet

* Clasa AbstractDAO<T>

Aceasta clasa are parametrii generici deoarece este folosita pentru cele 3 clase de clienti, produse si comenzi folosind metodele cu tehnici de reflexie

In constructor imi retin in variabila instanta type care este de tip Class<T> tipul clasei care o extinde

In metoda createSelectQuery se returneaza un String necesar pentru a construi query-ul de selectare a unui obiect dintr-un tabel cu un anumit id

In metoda createObjects se returneaza un List<T> ce contine toate obiectele create ce contin informatiile dintr-ul ResultSet transmis ca si parametru. De aceea a fost necesar ca pentru fiecare clasa de clienti, produse si comenzi sa imi construiesc si gettere si settere si sa am cate un constructor fara parametrii si fara corp

In metoda findByID se returneaza obiectul de tip T ce are id-ul id, acesta fiind un parametru transmis metodei si este de tip int. Se folosesc aici cele 2 metode descrise anterior pentru query si pentru returnarea obiectului ( se returneaza primul obiect din lista)

In metoda deleteObject se v-a sterge un obiect din tabela respectiva care are id-ul id, parametru transmis functiei de tip int)

In metoda stringInsert se returneaza un String care reprezinta query-ul necesar pentru inseararea unui obiect in tabela respectiva. Se trateaza de ce tip este type, iar in functie de acesta se construieste String-ul cu campurile necesare pentru obiectul ce urmeaza sa fie inserat.

In metoda setInsertStatementFields se seteaza in parametru s de tip PreparedStatement valorile necesare pentru obiectul ce urmeaza sa se insereze, valori ce sunt stocate in String[] alLFields care este si el la randul lui transmis ca parametru. Din nou se verifica de ce tip este type pentru a putea determina ce fel de obiect urmeaza sa se insereze

In metoda newID se determina un nou ID, id care nu a fost deja inserat in tabela table care este transmis ca si parametru si este de tip String. Query-ul fiind unul usor se construieste direct in metoda

In metoda insertObject este metoda care face inserarea unui obiect in tabela corespunzatoare. Ca parametru se transmit valorile ce trebuie inserate stocate intr-un String. Se da un split(" ") pentru a putea crea allFields. Dupa se apeleaza metoda de stringInsert() pentru a putea crea query-ul esential pentru inserarea obiectului. Dupa se apeleaza metoda setInsertStatementFields care seteaza valorile din query cu cele din all fields, iar la final se executa statement-ul.

In metoda stringUpdate se construieste query-ul necesar pentru actualizarea unui obiect. Se verifica de ce tip este type. Trebuie mentionat ca se poate face actualizare doar pe un client sau produs nu si pe o comanda.

In metoda setupdateStatementFields se seteaza in variabila transmisa metodei de tip PreparedStatement valorile noi ce urmeaza sa fie actualizate si care sunt stocate in parametrul allFields de tip String[].

In metoda updateObject se actualizeaza obiectul cu valorile stocate in parametrul fields de tip String. Se face un split(" ") dupa care se apeleaza metoda de stringUpdate pentru a putea construi query-ul, iar la final se apeleaza metoda setUpdateStatementFields si se executa statement-ul

In metoda listForJTabel se returneaza un ArrayList<T> cu toate obiectele dintr-un ResultSet pentru a popula campurile tabelului

In metoda getFieldsName se returneaza un obiect de tip DefaultTableModel care are rolul de a fi initializat cu numele campurilor stocate intr-un resultSet ( esentiala pentru clasa ViewJTabel - o sa se faca o detaliere cand se ajunge la acea clasa)

* Clasa ClientsDAO

Aceasta clasa extinde clasa AbstractDAO<Clients>

Este doar un constructor fara parametrii si fara corp

Exista doar o metoda numita updateClientTotalBill care are rolul de a actualiza pretul total pe care il are de platit un client cu id-ul id transmis ca si parametru de tip int. Acest camp este actualizat cu valoarea stocata in parametrul tPrice de tip float. Metoda este folositoare atunci cand se sterge un produs din baza de date

* Clasa ProductsDAO

Aceasta clasa extinde clasa AbstractDAO<Products>

Este doar un constructor fara parametrii si fara corp

Exista doar o metoda numita updateProductInStock care are rolul de a actualiza numarul total pe care il are un produs si reprezinta numarul total de produse valabile in stoc. Produsul cu id-ul id, care este transmis ca si parametru si este de tip int, o sa fie actualizat cu valoarea number de tip int care este de asemenea si ea transmisa ca si parametru. Metoda este folositoare atunci case se sterge un client din baza de date

* Clasa OrdersDAO

Aceasta clasa extinde clasa AbstractDAO<Orders>

Este doar un constructor fara parametrii si fara corp

In metoda returnOrdersClient se returneaza un ArrayList<Orders> care contine toate comenzile ce au ca si client cel cu id-ul idClient transmis ca si parametru. Metoda folositoare atunci cand se sterge un client

In metoda returnOrdersProduct se returneaza un ArrayList<Orders> care contine toate comenzile ce au ca si produs cel cu id-ul idProduct transmis ca si parametru. Metoda folositoare atunci cand se sterge un produs.

In metoda returnSingleProduct se returneaza un obiect de tip Products ce reprezinta produsul cu id-ul idProduct care este transmis ca parametru si este de tip int.

* Clasa Commands

In aceasta clasa sunt 3 variabile instanta: una de tip ClientsDAO, una de tip ProductsDAO si una de tip OrdersDAO

In constructor se initializeaza toate acestea 3

In metoda createOrder se creeaza o comanda cu id-ul clientului idClient, id-ul produsului idProduct si numarul de produse care se doresc sa fie comandate nrOfProducts. Daca numarul de produse care a fost introdus pentru a fi comandate este mai mare decat numarul de produse in stoc atunci se afiseaza un mesaj de eroare. Daca nu, atunci se incrementeaza total\_bill al clientului cu vechea valoare plus numarul de produse inmultit cu pretul produsului, se decrementeaza numarul de produse din stoc si dupa se insereaza comanda

In metoda deleteClient se sterge clientul cu id-ul idclient. Se stocheaza mai intai toate comenzile cu idClient egal cu acelui care urmeaza sa fie sters, iar pentru fiecare comanda se incrementeaza la loc numarul de produse in stoc al fiecarui produs si se sterg comenzile. Intr-un final se sterge si clientul

In metoda deleteProduct se sterge produsul cu id-ul idproduct. Se stocheaza mai intai toate comenzile cu idproduct egal cu acelui care urmeaza sa fie sters, iar pentru fiecare comanda se decrementeaza total\_bill al fiecarui client care a comandat acel produs ( se decrementeaza cu pretul produsul inmultit cu numarul de produse comandate). Se sterge fiecare comanda in parte care contine acest id, iar la final se sterge produsul

In metoda getClients se returneaza un String[] ce contine toti clientii. Necesar pentru popularea tabelului in interfata ViewClients

In metoda getProducts se returneaza un String[] ce contine toate produsele. Necesar pentru popularea tabelului in interfata ViewProducts

In metoda getOrdersWithClientID se returneaza un ArrayList<Orders> cu toate comenzile efectuate de un client cu id-ul idClient transmis ca si parametru si de tip int

In metoda getProduct se returneaza un obiect de tip Products ce are id-ul idProduct transmis ca si parametru si este de tip int.

* Clasa ViewClients:

Clasa de interefata grafica pentru clienti

Cand se apasa pe butonul de add se iau informatiile introduse din ultimele al 2-lea , al 3-lea si al 4-lea textField si se insereaza un nou client in baza de date

Cand se apasa pe butonul de edit se iau informatiile din toate textField-urile si se actualizeaza clientul cu id-ul introdus

Cand se apasa butonul de delete se sterge clientul cu id-ul introdus

Cand se apasa pe butonul "VIEW ALL CLIENTS" se deschide un nou frame cu un JTabel populat cu toti clientii

* Clasa ViewProducts:

Clasa de interefata grafica pentru produse

Cand se apasa pe butonul de add se iau informatiile introduse din ultimele al 2-lea , al 3-lea si al 4-lea textField si se insereaza un nou produs in baza de date

Cand se apasa pe butonul de edit se iau informatiile din toate textField-urile si se actualizeaza produsul cu id-ul introdus

Cand se apasa butonul de delete se sterge produsul cu id-ul introdus

Cand se apasa pe butonul "VIEW ALL PRODUCTS" se deschide un nou frame cu un JTabel populat cu toate produsele

* Clasa ViewOrders:

Clasa de interefata grafica pentru comenzi

Cand se apasa pe butonul "VIEW ALL ORDERS" se deschide un nou frame cu un JTabel populat cu toate comenzile

Daca se doreste introducerea unei noi comenzi atunci o sa se selecteze un client, dupa care se selecteaza un produs si se introduce in textField numarul de produse ce se doreste a fi comandate. In final se apasa pe butonul order

* Clasa Controller:

In aceasta clasa sunt 3 variabile instanta.

Una de tip Commands, una de tip ClientsDAO si una de tip ProductsDAO

In contructor se initializeaza acestea 3

In metoda main se creeaza 3 obiecte de tip ClientsDAO, ProductsDAO respectiv OrdersDAO, se creeaza un obiect de tip Commands, iar intr-un final se creeaza cele 3 ferestre de intrefete grafice

***5) Rezultate***

A picture containing text, screenshot, computer, indoor

Description automatically generated

***6) Concluzii***

In concluzie, am implementat tot ce se cerea si cum. Este de recomandat sa se respecte regulile precizate anterior pentru a asigura o functionalitate cat mai buna a aplicatiei

***7) Bibliografie***

Connecting to MySQL: https://www.baeldung.com/java-jdbc

Layered Architecture: https://dzone.com/articles/layers-standard-enterprise

Reflection in Java: http://tutorials.jenkov.com/java-reflection/index.html4

JavaDoc: https://www.baeldung.com/javadoc

SQL dump: https://dev.mysql.com/doc/workbench/en/wb-admin-export-import-management.html