Tema 3

Student:Oprean Andrei

Grupa 30228

1. Obiectivul temei

Obiectivul acestei teme este de a crea o aplicatie care sa gestioneze comenzile primite de un depozit. Aceste comenzi sunt definite de un client, un produs si o anumita cantitate. Aceasta aplicatie va folosi o baza de date relationala si tehnici de reflexie pentru a rezulta o solutie generala si usor de intretinut.

2. Analiza problemei, modelare, scenarii , cazuri de utilizare

2.1 Analiza problemei

Rezolvarea acestei probleme se poate imparti in mai multe etape :

- Accesul la baza de date
- Extragerea informatilor relevante din baza de date respectiva
- Procesarea acestor informatii in functie de scenariu
- Afisarea rezultatelor procesate cu ajutorul interfetei grafice cu utilizatorul

De aici rezulta ca aplicatia noastra va trebui sa mentina aceasta structura segmentata.

Un alt motiv pentru care am dori sa mentinem aceasta structura este nevoie de securizare a datelor. Dorim ca utilizatorul sa aiba acces doar la datele de care are nevoie si care ii sunt in acelasi timp permise. De exemplu nu vrem ca orice anajat al depozitului sa aiba acces la emailul clientului sau clientul sa aiba acces la datele altor clienti.

2.2 Modelare

Informatiile despre clienti, produse si tranzactii vor fi stocate intr-o baza de date de tip relational. Modelarea datelor se va face de la un layer la altul, informatia fiind mai intai luata din baza de date de catre primul layer, apoi procesata in layerul pentru “Business Logic” iar la sfarsit sa fie afisata cu ajutorul interfetei grafice cu utilizatorul. Datele de baza vor fi modelate cu ajutorul unor clase de baza, de exemplu clasa “Client” .

2.3 Scenarii

In cazul acestei aplicatii ar putea exista 3 scenarii

- Clientul face o comanda pentru un produs pentru care nu exista o cantitate suficienta, caz in care un mesaj adecvat va fi afisat pentru a il anunta acest lucru.
- Clientul face o comanda pentru un produs care nu exista,caz in care va fi nevoit sa introduca un id de produs valid
- Clientul face o comanda pentru un produs valid care exista intr-o cantitate suficienta pentru a satisface comanda, caz in care baza de date va fi actualizata corespunzator.

Aceste cazuri se pot extinde si la celalalte operatiuni cum ar fi stergerea, introducerea si actualizarea datelor. In majoritatea cazurilor, introducerea unei valori intr-un camp necorespunzator va genera o exceptie care il va informa pe utilizator de greseala.

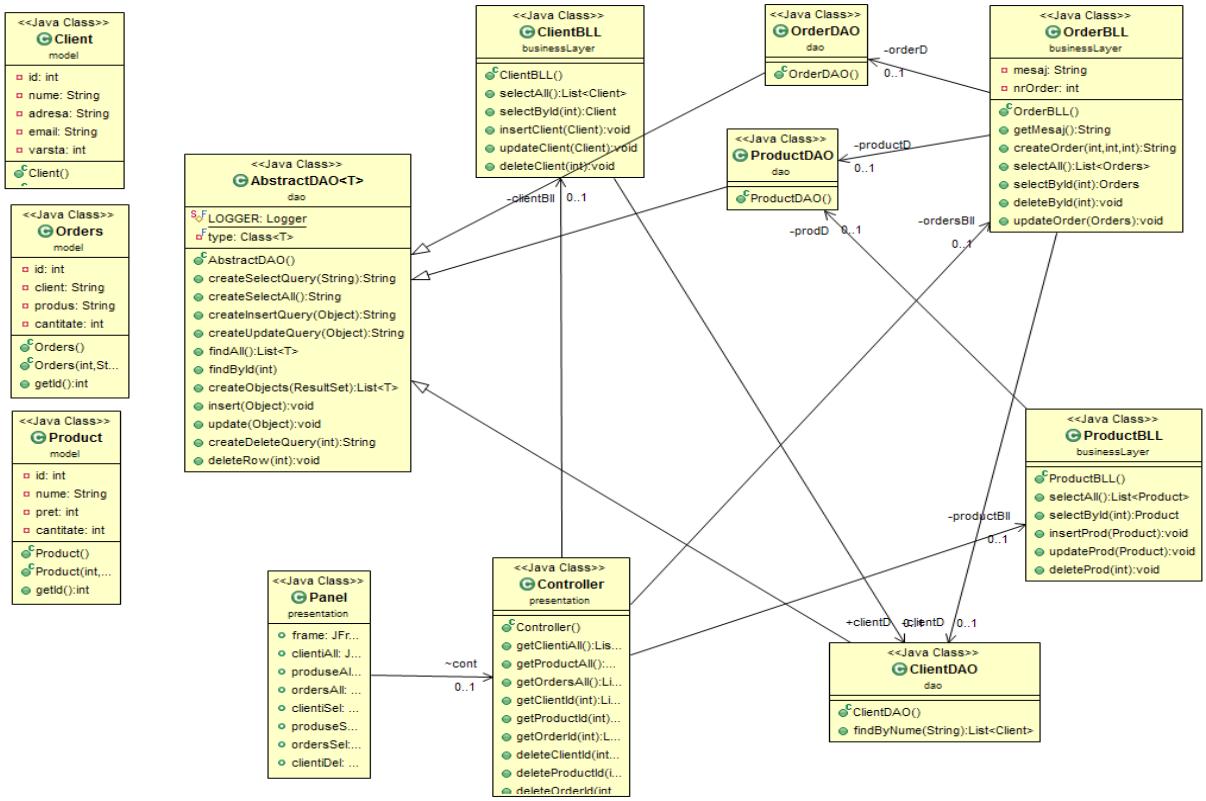
2.3 Cazuri de utilizare

Exista 5 cazuri de utilizare pentru fiecare tip de data de baza din aplicatie

- Selectare tuturor liniilor din tabel
- Selectarea unui numar specific de linii din tabel
- Inserarea unei linii noi in tabel
- Actualizarea unei linii din tabel
- Stergerea unei linii din tabel

Pentru fiecare caz elementele pe care utilizatorul este nevoit sa le introduca sunt specificate clar in interfata grafica.

3. Proiectare



Pentru aceasta aplicatie am folosit un model “layerd” care se bazeaza pe transmiterea datelor de la un layer la altul. Acest model usureaza intretinerea programului si totodata asigura integritatea datelor si mentinerea securitatii.

Pentru accesul la date am folosit reflexia, implementata intr-o clasa abstracta care este extinsa de orice clasa care modeleaza un tip de tabel. Astfel oricand este nevoie de a adauga sau a scoate un anumit tip de date nu este nevoie de modificari majore aduse programului. Tot aceasta metoda este folosita si pentru creeare tabelelor din interfata grafica cu utilizatorul, asa ca procesul de input – output poate integra orice tip de date care este creat dupa modelul unei baze de date relationale. Din punct de vedere al generalizarii, reflexia ajuta mult, deoarece folosind aceasta metoda nu trebuie sa aducem modificari logicii de accesare a datelor ci doar a datelor propriu – zise deoarece metodele din clasa abstracta extrag automat metodele de “set” si “get” invocandule cu datele curente.

Structurile de date sunt grupate in layerul “model” al proiectului ele fiind implementarea elementelor bazei de date. Aceste clase au in componenta lor exact aceleasi tipuri primitive de date care sunt accesate prin reflexie cu ajutorul metodelor de “get” si “set”.

Pentru interfata grafica am ales un design simplist care implementeaza 16 butoane caracteristice operatiilor pe fiecare tabel, astfel ca utilizatorul poate accesa si actualiza rapid orice informatie disponibila la momentul dat. Pentru afisarea tabelelor folosesc o interfata custom pentru modelul tabelului, "RowTableModel" care, cua ajutorul reflexiei poate extrage dintr-o clasa si o lista de obiecte de tipul acelei clase, toate campurile si valorile necesare pentru popularea tabelului.

Principala structura de date pe cam am folosit-o este lista de obiecte "custom".

4.Implementare

Pentru a implementa modelul "layerd" am impartit clasele aplicatiei in mai multe pachete

-dataAccessLayer: acest pachet este folosit pentru a stabili o conexiune cu baza de date implementata in MySQL Workbench cu ajutorul clasei ConnectionUnit. Pentru a stabili aceasta conexiune am declarat 5 constante statice de tip String care reprezinta adresa bazei de date, driver-ul, username-ul si parola. In aceasta clasa sunt implementate metode pentru creeare conexiunii si inchiderea conexiunii. Metodele acestei clase sunt disponibile doar pachetului din acest layer, "dao" (data access objects).

-dao: in acest este definit o clasa abstracta "AbstractDAO" in care sunt implementate metodele de baza pentru selectare, stergerea, actualizarea si inserarea datelor noi in tabelele din baza de date. Aceste operatii sunt facute stabilind conexiunea cu baza de date cu ajutorul pachetului descris mai sus, iar apoi cu ajutorul unor metode care returneaza un String echivalent cu o instructiune in limbajul SQL, interogarea este trimisa catre baza de date, care returneaza rezultatul sub forma unui resultSet. Din acest resultSet, cu ajutorul metodei createObjects, este returnat in final o lista de obiecte sau un singur obiect, dupa caz. Aceasta clasa abstracta este implementata de un numar de clase egal cu numarul de tabele din baza de date, iar in aceste clase putem defini pe langa metodele de baza anumite metode specifice pentru fiecare tip de tabel.

Impreuna aceste doua pachete formeaza layer-ul pentru accesul la baza de date, clasele specifice acestuia fiind singurele care pot accesa baza de date.

-businessLayer: acest pachet este implementarea in sine a urmatorului layer, acela care va lua informatiile trimise de catre "data access objects" si le va procesa in functie de cerintele aplicatiei. In interiorul metodelor din clasele acestui pachet putem implementa logica aplicatiei si mai ales modul in care sunt transmise datele catre utilizator prin interfata grafica.

-presentationLayer: aceasta parte a aplicatiei se ocupa de preluarea datelor transmise de layer-ul anterior si procesarea acestora astfel incat sa ia o forma usor de inteles de catre utilizator. In cazul acestei aplicatii, datele transmise sub forma de lista de obiecte sunt introduse intr-un tabel, si apoi afisate intr-un panel separat. Utilizatorul poate folosi o multime

de butoane pentru a selecta tipul de operatie pe care doreste sa o efectueze. Dupa ce a selectat o operatie, utilizatorul poate introduce orice valori in campurile specifice fiecarui tabel. De exemplu daca utilizatorul doreste sa introduca in baza de date un client nou, el va apasa pe butonul “Inserare clienti” iar mai apoi va putea introduce fiecare camp din tabel intr-o serie de JTextField-uri. Datele sunt validate doar dupa apasarea butonului “ok”, moment in care instructiunea SQL este generata si transmisa programului MySQLWorkbench pentru a fi executata.

Crearea tabelelor se bazeaza pe reflexie, constructorul clasei responsabile pentru aceasta activitate acceptand o clasa din pachetul “model” din care va extrage metodele de scriere si citire a datelor (“getters” si “setters”) si o lista de obiecte de tipul clasei model din care se vor extrage datele propriu zise.

5. Rezultate

Ca rezultat final am obtinut o aplicatie care se poate conecta la orice tip de baza de date si poate extrage, insera, updata si sterge orice tip de date in functie de cerintele utilizatorului.

Din punctul de vedere al utilizatorului, aplicatia este usor de folosit avand o serie de butoane cu notatie explicita din punctul de vedere al functionarii si cu celalalte campuri folosite pentru introducerea datelor , la fel de explicit evidentiate. Rezultatele sunt usor de interpretat, ele fiind afisate sub forma de tabele intr-un format relativ fidel fata de baza de date, fiecare coloana fiind afisata cu numele propriu pe primul rand. Ordinea coloanelor poate fi modificata usor, prin simpla tragere cu mouse-ul in locul potrivit. Unul dintre dezavantajele acestei aplicatii este faptul ca permite selectia datelor doar pe baza cheii primare “id” .

Din punctul de vedere al unui programator, aplicatia este usor de mentinut datorita designului de tip “layerd” astfel permitand detectarea unei probleme direct de la sursa si rezolvarea acesteia fara modificari aditionale aduse altor clase sau pachete. Totodata, aplicatia este usor de adaptat nevoilor oricarui mediu prin simpla adaugare a unor metode specifice in clasele de acces la baza de date sau procesarea datelor intr-un mod specific prin modificare logicii din pachetul “businessLayer” . Interfata grafica simpla este usor de adaptat la aceste schimbari, mai ales prin faptul ca tabelele cu rezultate sunt generate folosind doar rezultatul interogarii si clasa model. Interogarile pot fi si ele modificate usor, acestea fiind generate automat de catre metode care returneaza doar String-ul specific interogarii. Astfel, daca programatorul vrea, de exemplu, sa faca interogarea SELECT in functie de alt camp in afara de “id” trebuie doar sa modifice in String “id” cu campul respectiv.

6. Concluzii, dezvoltari ulterioare

In concluzie, dupa terminarea acestei teme pot spune ca am invatat bazele folosirii limbajului Java in paralel cu o baza de date. Prin folosire se intelege:

-stabilirea unei conexiuni cu o baza de date folosind adresa acesteia, driverul, username-ul si parola cu ajutorul metodei getConnection, inchiderea acestei conexiuni si folosirea acestor doua metode pentru a stabili conexiunea cu orice baza de date.

-creare si executarea interogarilor specifice limbajului SQL pentru a extrage datele din tabelele bazei de date.

-procesarea acestor date si modelarea lor dupa o serie de obiecte model cu ajutorul carora se va crea mai departe logica aplicatiei.

-folosirea reflexiei pentru a crea un model general de generarea a tabelelor folosind doar o clasa model si lista de obiecte care contine datele.

Pe langa acestea, am invatat importanta partitionarii unui proiect in mai multe "layere" care sa indeplineasca functii specifice si care sa fie vizibile doar altor "layere" direct implicate in functionarea acestuia sau care depind de datele continute de acesta. Motivatia pentru folosirea acestui design este evidenta luand in considerare usurinta mentinerii functionarii optime a programului si a depanarii in cazul erorilor sau a generarii de rezultate neasteptate. Un caz special in folosirea acestui design este pachetul care contine clasele care modeleaza fiecare tabel din baza de date, acesta fiind accesibil tuturor celorlalte pachete, ele constituind baza pe care este construit programul.

Se pot remarca o gama larga de dezvoltari ulterioare, aceasta aplicatie reprezentand doar un schelet al unei aplicatii apropiate de o folosinta comerciala. Totusi, aducand imbunatatiri interfetei cu utilizatorul astfel facilitand o manipulare mai rapida a datelor si crescand numarul de moduri in care pot fi manipulate datele, aceasta aplicatie ar putea fi folosita de orice companie care se ocupa cu gestionarea unui numar mare produse, clienti si comenzi, o centralizare a datelor fiind evident necesara. Pe langa acestea, adaugand suport pentru manipulari simultane a datelor folosind threaduri putem facilita folosirea aplicatiei de un numar mare de utilizatori simultan, singura restrictie fiind capacitatea serverelor care gazduiesc datele. De exemplu, pornind de la ideea de baza a temei, gestionarea unui depozit, putem facilita folosirea aplicatiei de catre fiecare angajat al depozitului simultan, astfel permitand o vizualizare in timp real a stocului de produse, a clientilor depozitului si adaugarea si executarea tranzactiilor. Utilitatea aplicatiei de fata nu este limitata la gestionarea unui depozit, deoarece cu ajutorul designului "layerd" putem modifica foarte usor specificatiile modelului si logicii astfel asigurand portabilitatea.

7.Bibliografie

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

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

http://www.mkyong.com/jdbc/how-to-connect-to-mysql-with-jdbc-driver-java/

http://christoph-burmeister.eu/?p=1556

http://theopentutorials.com/tutorials/java/jdbc/jdbc-mysql-create-database-example/

http://tutorials.jenkov.com/java-reflection/index.html

https://tips4java.wordpress.com/2008/11/27/bean-table-model/