Facultatea de Automatică și Calculatoare

Departamentul Calculatoare

TEMA 3

la disciplina

TEHNICI DE PROGRAMARE

- MANGEMENTUL COMENZILOR-



Proiect realizat de : Moloce Sabina-Maria

An: 2

Grupa: 30225

ANUL ȘCOLAR 2020-2021

Cuprins

1. Obiectivul temei..................................................................................................................................pag 3
2. Analiza problemei, modelare, scenarii, cazuri de...............................................................................pag 4
3. Proiectare (decizii de proiectare, diagrame UML, structuri de date, proiectare clase, interfete, relatii, packages, algoritmi, interfata utilizator) ............................................................................................pag 5
4. Implementare .....................................................................................................................................pag 10
5. Rezultate ............................................................................................................................................pag 15
6. Concluzii ............................................................................................................................................pag 15
7. Bibliografie.........................................................................................................................................pag 15

# 1. Obiectivul temei

1.1 Obiectivul principal

Obiectivul acestei teme de laborator este să proiectăm și să implementăm o aplicatie Java care proceseaza comenzile unor clienti pentru un depozit. Se vor folosi baze de date reloattinale pentru a stoca produsele, clientii si comenzile cu care lucreaza aplicatia.

Se doreste o aplicatie prin care sa putem sa interactionam cu tabelele din baza de date, sa adaugam, sa stergem, sa facem update si sa vizualizam toate datele din baza de date. Interactiunea trebuie sa fie usoara, interfata grafica fiind destul de explicita si pentru utilizatorii care nu sunt cunoscatori ai domeniului.

1.2 Obiectivele secundare

Dezvoltarea de use case-uri – capitolul doi

Use case-urile reprezintă un set de scenarii legate de modul în care este utilizat sistemul. Daca le utilizam putem sa descoperim:

* entități de sistem
* actori de sistem (roluri)
* comportament
* atribute
* cum interacționează actorii cu resursele sistemului

* Dezvoltarea de diagrame UML pentru pachete și clase – capitol trei

Vom utiliza diagrama de pachete pentru a asigura un „good practice”. Cele 6 parți ale acestei arhitecturi fiind separate în câte un pachet, conform modelului Layered Architecture care contine cel putin patru pachete(dataAccessLayer, businessLayer, model si presentation). Vom utiliza diagramele UML de obiecte pentru a obține clasele necesare pentru rezolvarea obiectivului principal.

* Dezvoltarea algoritmilor – capitol trei

Pentru a ne atinge obiectivul principal, avem nevoie să dezvoltam algoritmi sau să căutam algoritmi ce realizeaza funcționalitatea dorită. Specific pentru acest proiect vor fi algoritimi care folosesc tehnici de reflectie pentru a genera tabele si queries generice pentru obiecte.

* Implementarea soluției – capitol patru

Vom descrie clasele si ineterfata, si motivele pentru alegerea acestei implementari.

* Testarea – capitol cinci

Testarea consta in testarea prin intermediul interfetei grafice si prin intermediul MySql Workbench, unde se va putea oberva consistenta datelor, cu datle ce se afiseaza in interfata.

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

# 

Se dorește o soluție capabilă să proceseze corect comenzile pe care le primeste depozitul. Interfata trebuie sa fie intuitiva si usor de folosit, pentru ca o mare parte din aplicatie tine de introducere corecta a datelor de catre utilizator. Programul poate fi predispus erorilor, datorate introducerii gresite a dateleor de catre utilizator.

Modelarea problemei este extrem de importanat pentru a ajunge la soluția dorita. Structura aleasa fiind detaliata in capitolul trei si patru.

Caz de utilizare: procesarea comenzilor

Actor principal: utilizator calculator Scenariul principal de succes:

1. Utilizatorul apasa pe unul dintre cele trei butoane(Clients, Products, Orders) din interfata principala care contine meniul.
2. Se deshide interfata aleasa de catre utilizator.
3. Se introduc datele corecte si se selecteaza randurile corespunzatoare pentru operatia dorita.
4. Se apasa butonul corespunzator pentru operatia dorita.
5. Se afiseaza rezultatele dorite, care sunt in concordanta cu datele din baza de date.
6. Se apasa butonul de back si se revine la interfta principala care contine meniul.

Secvențe alternative:

Au fost lasate casute necompletate:

- se afiseaza un mesaj corespunzator si se cere utilizatorului sa completeze casutele

Datele nu sunt corecte:

- se afiseaza un mesaj corespunzator si se cere utilizatorului sa introduca date corecte

Nu a fost selectat randul corespunzator din tabel:

- se afiseaza un mesaj corespunzator si se cere utilizatorului sa selecteze un rand din tabel

Stocul nu este suficient:

- se afiseaza un mesaj corespunzator si comnda nu este plasata, deci baza de date ramane nemodificata

3. Proiectare (decizii de proiectare, diagrame UML, structuri de date, proiectare clase, interfețe, relații, packages, algoritmi, interfață utilizator)

O arhitectură stratificată(Layered Architectures) este organizarea structurii proiectului în patru categorii principale: prezentare, business, domeniu și model. Fiecare dintre straturi conține obiecte legate de preocuparea specială pe care o reprezintă.

Stratul de prezentare(presenatation) - conține toate clasele responsabile de interfata prezentata utilziatorului final si de trimiterea a rezultatului inapoi la client.

Stratul business - conține logica pe care se bazeaza aplicație pentru a functiona corect, conform cerintelor cerute.

Stratul de data access – contine clasele care se ocpua de queries si de conexiunea cu baza de date. In implementarea propriu-zisa am impartit acest strat in doua pachete, unul care se opcipa de queries si unul pentru conexiunea la baza de date.

3.1 Diagrame UML

* Diagrama de pachete

Diagramele de pachete ne ajută să decompunem sistemul în subsisteme, astfel

descompunem sistemul nostru in sase pachete presentation: dao, connection start, model, presentation si bll. Deși nu sunt reprezentate în imagine, există relații de asociere între cele sase pachete.

Diagram

Description automatically generated

* Diagrama de clase

Diagrama de clase descrie clasele din sistem și relațiile dintre ele, ce pot fi asocieri, generalizari, de realizare, de dependenta. Clasele vor fi doar enumerate în aceast capitol, detaliile despre câmpuri și metode se vor dezvolta în capitolul cinci.

Clasele din pachetul model: Client, Product, Order (care modeleaza tabelele din baza de date)

Letter

Description automatically generated

Clasa din pachetul dao: AbstarctDAO, ClientDAO, OrderDAO, ProductDAO (creeaza queries pentru intergoarae bazei de date)

Diagram

Description automatically generated

Clasa din pachetul connection: ConnectionFactory (realizeaza conexiuni la baza de date)

A picture containing diagram

Description automatically generated

Clasele din pachetul bll: ClientBLL, ProductBLL, OrderBLL ( business logic)

Diagram

Description automatically generated

Clasele din pachetul presentation: GUI, OrderFrame, ProductFrame, ClientFrame, JtableManagement

Diagram

Description automatically generated

Clasa din pachetu start: Start (contine main-ul)

A picture containing text

Description automatically generated

* 1. Structuri de date folosite

Structurile de date folosite preponderent sunt listele (List<>).

3.3 Detalii legate de implementare (generics, wildcards, reflectie)

Ca si detalii legate de structura implementarii mentionam o folosire deasa a genericelor. In Java metodele generice și clasele generice permit programatorilor să specifice, cu o declarație cu o singură metodă, un set de metode conexe sau, cu o declarație cu o singură clasă, un set de tipuri conexe, respectiv. Genericele oferă, de asemenea, o siguranță a tipului de compilare, care permite programatorilor să prindă tipuri nevalide în timpul compilării. Folosirea genericelor se remarca preponderent in pachetul dao, in clasa abstracta AbstractDAO , care este o clasa generica, permitandu-ne astfel sa generam diferite queries(insert, update, delete, findAll) pentru diferite tipuri de tabele, care au atributele variate ca numar si ca tip.

Reflecţie este capacitatea unui program în desfășurare de a se examina (i) pe sine, (ii) mediul său software și (iii) de a-și schimba acțiunile în funcție de ceea ce găsește. Folosim acesta carcteristica din Java pentru a genera queires pentru baza de date, necunoscand cum arata tabelul, ci doar ca are un priamary key numit id, care nu trebuie introdus pentru ca atributul este setat auto-increment De asemenea, folosim reflectia si pentru a crea Jtable, in care vom afisa rezultatele primite de la baza de date sub forma de lista de obiecte, deci nu cunoastem ce fel de obiecte sunt in lista (List<?> objects).

Semnul de întrebare (?) este cunoscut sub numele de wildcard în programarea generică si reprezintă un tip necunoscut. Acesta poate fi utilizat într-o varietate de situații, cum ar fi tipul unui parametru, câmp sau variabilă locală; uneori ca tip retur.

# Implementare

4.1 Clasele din pachetul model

Text

Description automatically generated

* Clasa Client

Clasa Client modeleaza tabelul client din baza de date warehousedb. Tabelul are patru atribute: id, name, address, email

care se regasesc si in campurile acestei clase, cu exact aceleasi nume pentru ca maparea intre tabelul din baza de date si obiectele

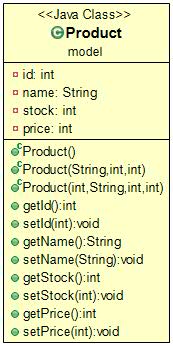
de tipul Client.

Clasa are trei constructori, folositi in diferite instante: cel fara parametrii este folosit in AbstractDAO in metoda createObjects(), unde avem nevoie de un obiect Client, a caror campuri vor fi setate pe rand. Constructorul fara id se foloseste cand luam date de

la utilizator si vrem sa inseram un rand nou, insa baza de date va

fi responsabila de generarea id-ului.

De asemenea, avem gettersi si settersi pentru fiecare camp din clasa.

* Clasa Product

Clasa Product modeleaza tabelul product din baza de date warehousedb. Tabelul are patru atribute: id, name, stock si price

care se regasesc si in campurile acestei clase, cu exact aceleasi nume pentru ca maparea intre tabelul din baza de date si obiectele

de tipul Product.

Clasa are trei constructori, folositi in diferite instante: cel fara parametrii este folosit in AbstractDAO in metoda createObjects(), unde avem nevoie de un obiect Product, a caror campuri vor fi setate pe rand. Constructorul fara id se foloseste cand luam date de

la utilizator si vrem sa inseram un rand nou, insa baza de date va

fi responsabila de generarea id-ului.

De asemenea, avem gettersi si settersi pentru fiecare camp din clasa.

* Text

  Description automatically generatedClasa Order

Clasa Ordet modeleaza tabelul order din baza de date warehousedb. Tabelul are patru atribute: id, idClient, idProduct si quantity care se regasesc si in campurile acestei clase, cu exact aceleasi nume pentru ca maparea intre tabelul din baza de date si obiectele de tipul Order. Tebuie acordata atentie faptului ca idClient si idProduct trebui sa existe in tabele corespunzatoare pentru ca acestea reprezinta foreign key-uri.

Clasa are trei constructori, folositi in diferite instante: cel fara parametrii este folosit in AbstractDAO in metoda createObjects(), unde avem nevoie de un obiect Order, a caror campuri vor fi setate pe rand. Constructorul fara id se foloseste cand luam date de la utilizator si vrem sa inseram un rand nou, insa baza de date va fi responsabila de generarea id-ului.

De asemenea, avem gettersi si settersi pentru fiecare camp din clasa.

* 1. Clasa din pachetul connection
* Clasa ConnectionFactory

Clasa este responsabila pentru a creea conexiuni la baza de date, atunci cand avem nevoie, dar si pentru a

conexiunea, statementuri sau resultset-uri.

Diagram

Description automatically generated with medium confidence

* 1. Clasele din pachetul dao

Text

Description automatically generated

* Clasa AbstractDAO<T>

Este cea mai importanta clasa din acest pachet, restul

claselor din acest pachet doar implementeaza aceasta clasa

Clasa este un abstracta care are atribute generice. Atributul

type este de tipul Class<T> si reprezinta un obiect de tipul

Class care are un tip specific T. Acesta ne permite de fapt

generam queries care sa fie aplicabile pe orice tabel, care

are o primary key si mai multe atribute.

Metodele publice sunt: insert, delete, update,findById

(care ne permite sa cautam un anumit rand din tabel dupa

id) si findAll( care ne returneaza o lista cu toate randurile

dintru-un tabel, sub forma de obiecte Java).

De asemenea, avem mai multe metode private care ne creeaza querie-urile cu carcter generic, dar

si metoda createObjects care este cea mai importanta, deoarece construieste obiectele din baza de

date din resultset-ul unui querie.

* 1. Text

     Description automatically generatedClasele din pachetul bll

Se ocupa de incapularea logicii aplicatiei. Pentru fiecare clasa din model creeam o clasa de business logic.

Fiecare va avea un dao, prin care va accesa operatiile generice ce se pot face pe tabele. Clasele vor avea metode specifice tipului de tabel, care vor fi apelate din interfata. OrderBLL are in plus cateva metode:

getPossibleClientsIds si getPossibleProductsIds, care ne vor ajuta in interfata pentru comenzi, pentru ca utilizatorul sa poata selecta doar id-uri valide si metoda privata createBill care se apleaza la fiecare insert pentru a genera un fisier .txt ce contine factura comenzii.

* 1. Clasele din pachetul presentation

Diagram

Description automatically generated

Avem un gui principal care este meniul de start si alte trei frame-uri pentru a lucra pe fiecare tabel din baza de date. Din punctul de vedere al codului, prezinta inters clasa JtableManagement, care foloseste wildcards si tehnici de reflectie pentru a genera Jtable-uri pentru orice tabel din baza de date, care prezinta un primary key si un numar si tip variabil de atribute.

Text

Description automatically generated

Prezentarea interfetei aplicatiei:

Graphical user interface, website

Description automatically generated

Graphical user interface, application

Description automatically generatedMeniul ne permite sa selectam pe ce tabel vrem sa lucram: Clients, Products, Orders.

Pentru tabelul Clients putem sa aduagam noi clienti, sa editam informatiile despre clientii deja existenti

in baza si sa stergem clienti. Toti clientii din baza vor aparea in tabelul generat cu functia createTable().

Daca vrem sa revenim la meniul principal apasam butonul back.

Graphical user interface, text, application, email

Description automatically generated

Pentru tabelul Products putem sa aduagam noi produse, sa editam informatiile despre produsele deja existentein baza si sa stergem produse. Tote produsele din baza vor aparea in tabelul generat cu functia createTable(). Daca vrem sa revenim la meniul principal apasam butonul back.

Graphical user interface, text, application

Description automatically generated

Interfata pentru Orders este prevazuta doar partea de adaugare a comenzilor si vizualizarea comenzilor.

Este selecti id-urilor clientilor se poate face doar dintr-o gama valida de id-uri care chiar exista in baza

de date. De asemenea, la fiecare comanad reusita se adauag o factura in Order.txt.

# 5. Rezultate

Pentru a ne asigura că programul functioneaza cum ne dorim vom efectua mai multe teste pentru diferitele operatii puse la dispozitie prin interfata utilizatorului. Verificam daca ce se afiseaza in interfata

si ce este in baza de data este in conordanta.

6. Concluzii

Prin implementarea acestei teme mi-am aprofundat cunostințele de programare in Java, cunostințele despre tehnici de reflectie si despre baze de date realtionale, sintaxa sql si java. De asemenea, am exersat și construirea unei interfețe, si am intrat in detalii privind lucrul cu tabelele JTabel.

Ca și posibilitați de dezvoltarea ulterioară, se poate lucra la interfața, o realizare mai „user friendly”, care pune la dispozitie utiliatorului mai multe functionalitati, iar modul de operarea asupra datelor s-ar putea face strict pe JTable . De asemenea, mesajele de eroare la introducerea datele ar putea fi mult mai explicite. La nivel de proiectare se pot aduce înbunătațiri, iar codul poate fi optimizat.

# 7.Bibliografie

[1] Cursurile de tehnici de progarmare

[2] Cursurile de POO

[3] Wikipedia

[4] https://stackoverflow.com

[5] <https://www.geeksforgeeks.org/>

[6] <https://gitlab.com/utcn_dsrl/pt-reflection-example>

[7] Prezentarea suport

[8] https://gitlab.com/utcn\_dsrl/pt-layered-architecture