ORDER MANAGEGEMENT

*Logo, company name

Description automatically generated*

**Profesor: Tudor Cioara**

**Asistent profesor: Alexandru Rancea**

**Student: Stratulat Ildiko Vivien**

**Grupa : 30221**

**1.Obiectivul temei**

Obiectivul principal in constituie implementarea bazei de date simplificate. In acest sens, am considerat cele trei tabele: client,produs si commanda ,care descriu cele 3 entitati implicate in procesul de cumparare. Obiectivele secundare sunt urmatoarele:

-implementarea operatiei de insert in table

-implementarea operatiei de delete in table

-implementarea operatiei de update a unui element in table

-implementarea operatiei de afisare toate elementele dintr un table

-implementarea unei interfere grafice pentru utilizator

-implementarea operatiei de realizare a unei comenzi cu modificarile de rigoare aduse tabelului produsului (modificare stocului dupa o comanda pusa)

-generare Javadoc

-generarea unui fisier sql de creare tabele

Userul va putea interactiona cu baza de date prin intermediul unei interfete grafice, care ii va permite sa selecteze operatia,, rezultatele aparand in tabelul corespunzatoare.

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

* **Analiza problemei:**

Pentru a putea implementa operatiile, am utilizat tehnica Reflection: am implementat operatiile add,insert,update,findAll generalizat, in clasa AbstractDAO, apoi le-am apelat pentru tipul de obiect necesar noua (Client,Produs).

* **Modelarea problemei:**

Utilizatorul va avea posibilitatea de a folosi funcțiile simulatorului de comenzi prin intermediul interfeței grafice, unde va introduce date și va putea alege ce operații să efectueze cu acestea:

* Inserare client nou in baza de date
* Editare client deja existent din baza de date
* Stergere un client existent din baza de date
* Inserare produs nou in baza de date
* Editare produs deja existent in baza de date
* Sterge produs din baza de date
* Inserare comanda in baza de date
* Vizualizare toti clientii
* Vizualizare toate produsele

Rezultatul obținut în urma efectuării operației alese va fi afișat în interfața grafica

**3.Scenarii și cazuri de utilizare:**

Un caz de utilizare reprezintă o colecție de scenarii posibile, referitoare la comunicarea între aplicație și utilizatori, caracterizate cu anumite scopuri. Cazurile de utilizare arată ce trebuie să facă sistemul și nu cum.

Deoarece cazurile de utilizare sunt puternic conectate cu pașii utilizatorului am încercat să fac interfața grafică cât mai user-friendly:

Chart, bar chart

Description automatically generated

-adaugarea unui element in baza de date, in tabelul Client sau Produs

• Actorul principal:userul

• Scenariul de succes:

userul completeaza campurile pt elemental de adaugat

userul seleteaza optiunea de a insera un nou produs

aplicatia introduce elementul in baza de date

• Scenariu alternativ: Utilizatorul insereaza valori invalide pentru Client sau Produs.

• Pentru Client, nu ii permit sa introduca un client cu varsta sub 18 ani (adica nu accepta interactiunea cu posibil cumparatori minori) sau peste 99 de ani.

• Pentru Produs, nu ii permit sa introduca un stoc numar negativ sau 0.

Pentru Comanda (functionalitate implementata direct la nivelul Controllerului pentru Comanda), nu ii permit sa cumpere 0 produse (comanda fara sens) => afisez un mesaj de eroare “Va rugam nu incercati sa cumparati 0 produse..” si resetez interfata cu userul. De asemenea, nu permit sa incerce sa cumpere un produs cand nu exista o cantitate suficienta in stoc=>s-ar ajunge la o cantitate negativa.

**3.Proiectare:**

* *Diagrame*
* Diagrame Use Case

Diagram

Description automatically generated

* Diagrama claselor
* Text

  Description automatically generated with medium confidence

Graphical user interface

Description automatically generated

-Diagramele de clase sunt folosite în modelarea orientată pe obiect pentru a a descrie structura statică a sistemului, modului în care este el structurat

-Oferă o notație grafică pentru reprezentarea: claselor, entități ce au caracteristici comune și a relațiilor dintr două sau mai multe clase.

* **Structuri de date**

Structurile de date care au fost folosite în cadrul acestui proiect sunt ori tipuri de date primitive, double și int, ori un tip mai complex: ArrayList și obiectele create de mine, Client,Produs și Comanda,.

Am ales să folosesc ArrayList în locul obișnuitelor Array-uri din cauza funcționalității și perfomanței acestuia, fiind mai eficient în momentul în care dorim să accesăm elementele lui. Totodată, operația de adăugare de termeni este mai ușoară la ArrayList-uri față de cele clasice la care trebuie să predefinim o anumită lungime.

* **Pachete**

Pachetele sunt folositoare în menținerea proiectului sub o formă mai organizată, fapt ce facilitează înțelegerea acestuia. Prin pachete se înțelege gruparea a multiple clase și interfețe.

Astfel, în programarea orientată pe obiecte se folosește în zilele de astăzi metodologia MVC(model-view-controller) pentru a gestiona eficient interfețele grafice cu modele de date.

MVC este considerat de mulți specialiști în domeniu drept un pattern folositor în cazul reutilizării codului și datorită căruia dezvoltarea aplicațiile cu interfețe grafice pentru utilizatori durează mai puțin.

Acest MVC este alcătuit din trei mari componente care vor fi utilizate în dezvoltare software:

* *MODELUL*: -reprezintă structura logică a datelor într-o aplicație software, acesta nu deține nicio informație despre interfața grafică.
* *VIEW-UL*: -alcătuit dintr-o mulțime de clase care reprezintă diferite elemente în interfața grafică, elementele vizible pentru utilizator și cu care poate interacționa, acestea fiind: butoane, textfield-uri, combo box-uri ș.a.m.d
* *CONTROLLER-UL*: reprezintă legătura din model și view, are drept scop permiterea comunicării dintre clasele din model și din view

**Layered Architecture**

Am structurat codul conform arhitecturii Layered, astfel avand 4 pachete mari:

-Connection- pentru realizarea conextiunii cu baza de date

-Presentation- interfata grafica pentru utilizator, contine si elementele de controller pt fiecare interfata

-Business- contine clasele care incapsuleaza logica de aplicatie

-DAO- contine clase ce au interogarile si conexiunea cu baza de date

**4.Implementare:**

Datorită beneficiilor utilizării metodologiei MVC, am decis să o folosesc în proiectul meu, astfel acesta conține următoarele pachete:

* App – conține o singură clasă, în interiorul căreia se afla metoda principala main()
* Model – conține clasele care rezolvă problema propusă (operațiile pe polinoame)
* Monom
* Polinom
* CalcModel
* View - conține o singură clasă care reprezintă interfața grafică pentru utilizator
* Controller -creează legătura dintre model și view

**Descrierea claselor:**

* **Client**

**-**clasa client are urmatoarele atribute: idClient,nume,prenume,adresa bani si age. Pentru fiecare atribut, am definit settere si gettere, corespunzator strategiei de incapsulare. Am definit atat constructor cu parametrii, cat si fara. De asemenea, tot aici am suprascris si metoda toString pt a a afisa datele unui client sub o forma mai usor de inteles

* Produs

-clasa produs client are urmatoarele atribute: idProdus,nume,producator si stocul disponibil. Pentru fiecare atribut, am definit settere si gettere, corespunzator strategiei de incapsulare. Am definit atat constructor cu parametrii, cat si fara. De asemenea, tot aici am suprascris si metoda toString pt a a afisa datele unui produs sub o forma mai usor de inteles

* Comanda

-clasa comandaclient are urmatoarele atribute: numeClient,prenumeClient,numeProdusp si cantitatea cumparata.Pentru fiecare atribut, am definit settere si gettere, corespunzator strategiei de incapsulare. Am definit atat constructor cu parametrii, cat si fara. De asemenea, tot aici am suprascris si metoda toString pt a a afisa datele uneiui comenzi sub o forma mai usor de inteles

* ClientBLL,ProdusbLL,ComandaBLL

-fiecare clasa din pachetul BLL verifica daca s-a reusit inserarea corespunzatore in baza de date, respecitv daca rezultatul constructorului din pachetul dao, clasa corespunzatoare este sau nu null, iar caz de null se afiseaza mesaj de eroare

* ComandaDAO,ProdusDAO,ClientDAO

-acestea doar apeleaza metodele din superclasa AbstractDAO, aici intervine strategia Reflection. Logica pentru inserare, stergere, update, findAll e scrisa o singura data, in clasa parametrizata, apoi in functie de tipul obiectului T, devine specifica pentru Client,Produs.

* Clasa AbstractDAO

-Clasele ComandaDAO,ProdusDAO si ClientDAO doar apeleaza metodele din superclasa Abstr5actDAO.

-constructoul imi face rost de clasa specifica pe care lucrez: produs, client, comanda, adica tabelul map in pachetul model

-metoda createSelectQuery – genereaza o interogare de tipul „select \* from <nume\_tab> where <>=?”

-metoda createSelectAllQuery- genereaza un statement care face rost de toate tuplele din tabelul respectiv

-metoda createInsertStatement- returneaza statementul construit sub forma de String, generat cu ajutorul lui StringBuilder, folosind un obiect de tip Field trec prin toate coloanele tabelului, setam field acces true, pentru ca fiecare atribut e privat pentru clasa si altfel nu l-am putea accesa

-metoda createObjects -avand un obiect de tip ResultSet genereaza o lista de obiecte de tip T

-metoda createDeleteStatement- analog createInsert

-metode createUpdate -analog

-metoda findAll- realizeaza conexiune la baza de date folosind un obiect din clasa Connection, apoi se foloseste de de statementul creat selectAllQuery. ResultSet reprezinta rezultatul executiei statement-ului pentru care apelam metoda createObjects pentru a genera lista de obiecte necesara ca return value.

-metoda findById- similar cu findAll, doar ca trebuie setat parametrul idClient sau idProdus: folosim metoda statement.setInt(param,valoare) sau setString in fucntie de tipul parametrului, pentru ca numele field ului e variabil dupa tabel, in transmitem ca parametru

-metodaUpdate- creeaza o ferestra noua pt introducerea noilor date.

* View,ViewClienti,ViewProduse,ViewComenzi

-responsabile cu afisarea interfetelor

* ControllerMain,ControllerView,ControllerClienti,ControllerComenzi

- genstioneaza actiunile pe butoane, controller comenzi genereaza si bill ul

* Clasa creareFisier

-creeaza factura

* Clasa ReflectionStrategy

-metoda retrieveHeader- o metoda statica care accesand fieldurile listei de tuple de tip T, returneaza o lista care contine informatiile necesare pt generarea capului de tabel

* Clasa createTable- contine constructorul, care primind o lista de tuple genereaza un tabel nou intr-o noua fereastra.

-se foloseste metoda retrieveHeader pt a face rost de capul de tabel

-pentru fiecare element din lista primita ca parametru fac rost de valorile din fields, pe care le punn in linia corespunzatoare din lista ( matrice de string)

**Interfața grafică**

-are rolul de a conecta utilizatorul cu aplicația

-acest utilizator poate să introducă date, în cazul de față date despre anumie produse sau anumiti clienti

-totodata acesta poate sa modifice aceste date, sau chiar sa le si stearga

Interfata Client

Graphical user interface, application

Description automatically generated

**Interfata Produs**

**Graphical user interface, application

Description automatically generated**

**Graphical user interface, application

Description automatically generatedInterfata Comanda**

**Interfata pt editare produse sau clienti**

**Graphical user interface

Description automatically generated**

**CLASE SINGLETON**

O clasa Singleton e o clasa care poate avea un singur obiect(instanta) in acelasi timp. Dupa prima instantinere, daca mai incercam o data, noua instanta va pointa la prima instanta creata. Deci, orice modificari facem la orice variabila din interiorul clasei prin orice instanta, modifica aceleasi atribute si e vizibila prin orice instanta.

Puncte cheie:

1. Constructorul trebuie sa fie privat

2.Se va scrie o metoda statica care are ca return type obiectul acestei clase (Connection pt noi)=>conceptul de LAZY INITIALIZATION

3. Instanta trebuie sa fie unica si sa fie stocata ca un atribut private static.

De ce avem nevoie de o clasa Singleton?

In principal, pentru ca vrem sa restrangem numarul de obiecte create la unul singur(folosit pt multithreading si database connection in Java). Clasa Singleton se asigura ca la un moment de timp un singur thread/o singura conexiune poate accesa conexiunea realizata.

• Exemple de clase Singleton: Runtime class, Action Servlet, Service Locator; constructori private si factory methods sunt alte exemple.

Cum diferentiem o clasa normala de o clasa Singleton?

-pentru a instantia o clasa normala, folosim constructorul; pt a initializa o clasa Singleton folosim metoda getInstance();

-o clasa normala vanishes la finalul executiei unei aplicatii, dar una Singleton nu se distruge la final singura

-Exista 2 forme a ale desgin patternului Singleton:

-early instantiation= instantierea are loc la load time

-lazy instantiation=instantierea e facute conform requirementului

**5.Rezultate:**

O modalitate de testare o reprezinta verificare grosiera in mySql Workbench a functionarii operatiilor implementate, precum si a generarii istoricului comenzilor in fisierul cu proiectul.

**6.Concluzii:**

Acest proiect a reprezentat un exercițiu absolut necesar pentru aprofundarea noțiunilor studiate în primul semestru despre Programarea orientată pe obiecte, pe parcursul realizării acestuia am întâmpinat anumite dificultăți, dar datorită acestora am fost nevoită să mă documentez mai mult, astfel înțelegând mult mai bine aceste noțiuni.

O astfel de noțiune a fost MVC-ul despre care doar am auzit în primul semestru și nu înțelegeam neapărat care este rostul în a împărți în atâtea clase lucruri care se puteau la fel de bine scrie într-una, dar am înțeles acum că astfel se obține o organizare mult a bună a proiectului.

Din punctul de vedere al cunostiintelor dobandite, personal, am invatat cum sa lucrez cu o clasa generic, cum sa apelez metodele din ea pentru diferite clase specific via Reflection. De asemenea, am inteles cum se realizeaza conexiunea cu baza de date, precum si importanta deschiderii unei singure conexiuni o data (motive de securitate) via conceptul Singleton

O imbunatatire viitoare ar fi sa se poate inregistra timpul realizarii comenzii, precum si o comunicare cu providerul astfel incat userul sa poata primi informatia despre cat de rapid va primi comanda, eventual si cu firma de curierat pentru a putea urmari drumul pachetului spre sine.

**7.Bibliografie:**

1. [www.wikipedia.org](http://www.wikipedia.org)
2. [www.geeksforgeeks.com](http://www.geeksforgeeks.com)
3. [www.w3school.com](http://www.w3school.com)
4. [www.stackoverflow.com](http://www.stackoverflow.com)
5. Cursurile de la POO