

Tehnici de Programare

Tema 3 – Order Management

Profesor curs: Conf. Tudor Cioara Sudent: Varga Alexandra-Ioana

Profesor laborator: Dorin Moldovan Grupa: 30223

Cuprins

1. Obiectivul proiectului
2. Analiza problemei, modelare, scenarii, cazuri de utilizare
   1. Analiza Problemei
   2. Modelare
   3. Cazuri de utilizare
3. Proiectare (diagrama UML, structuri de date, clase, interfete, relatii, pachete, algoritmi)
4. Implementare si testare
5. Rezultate
6. Concluzii
7. Bibliografie
8. Obiectivul temei

Obiectivul acestei teme este in principal de a ne familiariza cu lucrul cu baza de date, de la 0. Atat invatarea conexiunii cu baza de date cat si prelucrarea tuturor informatiilor pe care aceasta ni le ofera. De la inceput se lucreaza cu vizualizarea informatiilor, cu adaugarea de noi informatii, de update respectiv stergere de informatii.

Dupa parerea mea, este foarte important sa fim cat mai bine familiarizati cu conexiunea cu baza de date pentru ca acesta este principiul pe care functioneaza aproximativ totul din jurul nostru, de exemplu: case de marcat in magazine, liste cu studentii dintr-o anumita facultate, etc. Preluarea informatiilor dintr-un loc bine definit este o necesitate in ziua de astazi, totodata manipularea si prelucrarea acestora.

Pe langa aceasta familiarizare, aceasta tema mai are ca si scop si creearea unei legaturi a celui care face proiectul cu o legatura de retea precum acest proiect. Acesta consta in adaugarea de clienti, adaugarea de produse, de comenzi, stergerea de clienti, de produse, update-ul anumtior produse si respectiv vizualizarea tuturor acestora. Dupa finalizarea proiectului, cel care a implementat proiectul va avea cunostinte necesare in scopul lucrarii cu anumtie produse, informatii, si anumite comenzi.

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

2.1 Analiza problemei si modelarea ei

Tema propusa prezinta o complexitate ridicata, cu toate ca modul de implementare a solutiei este lasat la alegerea studentului. Scopul aplicatiei este de a permite utilizatorului sa efectueze diverse operatii de adaugare/ stergere de produse, respectiv comenzi. Aplicatia implementata de mine ofera posibilitatea diferitelor operatii:

Operatii premise pentu administrator:

-> Adaugarea unui nou produs

-> Stergerea unui produs existent

-> Schimbarea pretului unui produs

Operatii premise pentru client:

-> Adaugarea unui client nou

-> Adaugarea unei comenzi pentru un client deja existent

-> Stergerea unui client

De asemenea, sunt permise operatiile de listare a tuturor produselor existente , a tuturor comenzilor, a stock-ului valabil si a tuturor clientilor.

Pentru a permite pastrarea datelor de la o rulare la alta a aplicatiei s-a folosit conexiunea cu o baza de date. Astfel la fiecare adaugare de client, stergere de client sau adaugare si stergere de produse, respectiv comenzi se vor face modificarile corespunzatoare asupra datelor retinute in baza de date.

Ideea principala a implementarii acestui proiect consta in crearea unei baze de date in care sa se pastreze toate informatiile despre obiectele implicate si concetarea acesteia la proiectul efectiv unde se vor desfasura alte operatii, specifice cerintei.

Acest proiect va fi construit dupa arhitectura layered pentru ca aceaste permite cea mai buna mentenata.

Aplicatiile care au la bază arhitectura multi-strat (n-Tier) reprezintă un model flexibil de aplicatie cu cod reutilizabil. Structurând aplicatia în straturi (layere), programatorii trebuie să modifice doar nivelul respectiv, în loc să rescrie întreaga aplicatie pentru cazul în care se hotărăsc să schimbe una sau mai multe tehnologii folosite.

**2.2 Scenarii si cazuri de utilizare**

Aplicatia poate fi folosita pentru gestionarea produselor si a comenzilor dintr-un magazin sau un depozit, oferind posibilitatea de procesare a comenzilor si a operatiilor de adaugare si stergere de noi produse in timp real. De asemenea, datorita utilizarii unei conexiuni cu o baza de date, aplicatia ofera posibilitatea de pastrare a datelor de la o rulare la alta.

3. Proiectare (diagrame UML, structuri de date, proiectare clase, interfete, relatii, packages, algoritmi, interfata utilizator)

Pentru proiectarea acestui proiect se va lua in considerare modul in care se vor face pachetele.

Exista un pachet pentru a pastra clasele POJO in care efectiv se mapeaza fiecare obiect din baza de date intr-o clasa. Aceasta clasa va contine doar atributele obiectului, care defapt desemneaza coloanele tabelului din baza de date, si metode de setare si returnare a acelor atribute.

Al doilea pachet care reprezinta si primul strat al arhitecturii este cel care contine conexiunea la baza de date.

Al treilea pachet presupune un set de clase prin care se accesesaza componentele din baza de date. Am ales metoda mai putin flexibila in care fiecarui element i-am construit cate o clasa in care am implementat operatiile caracterstice bazei de date si anume inserare, gasire, stergere si updatare. Acest pachet putea sa fie contruit si generic astfel incat sa poata primi orice obiect din baza de adate si sa poata realiza operatiile principale.

Al patrulea pachet care reprezinta al treilea strat se refera la clasele care contin logica sistemului de manageriat comenzi. In acest pachet sunt implementate clase care mapeaza pentru a doua oara toate obiectele astfel incat pentru fiecare obiect din baza de date se creaza un obiect cu ajutorul caruia se vor implementa operatiile din interfata. Aceasta decizie are avantajul de a oferi flexibilitate in sensul in care chiar daca se schimba implementarea legata de modul de acces la baza de date logica proiectului sa ramana stabila.

Al cincilea pachet, care defapt reprezinta si ultimul strat al arhitecturii, se refera la modul in care acest sistem de administrat comenzi este efectiv utilizat de catre utilizatori, adica prin interfata.

Structurile de date folosite sunt cele de baza si anume liste sau cel mult vectori de stringuri si matrice de obiecte.

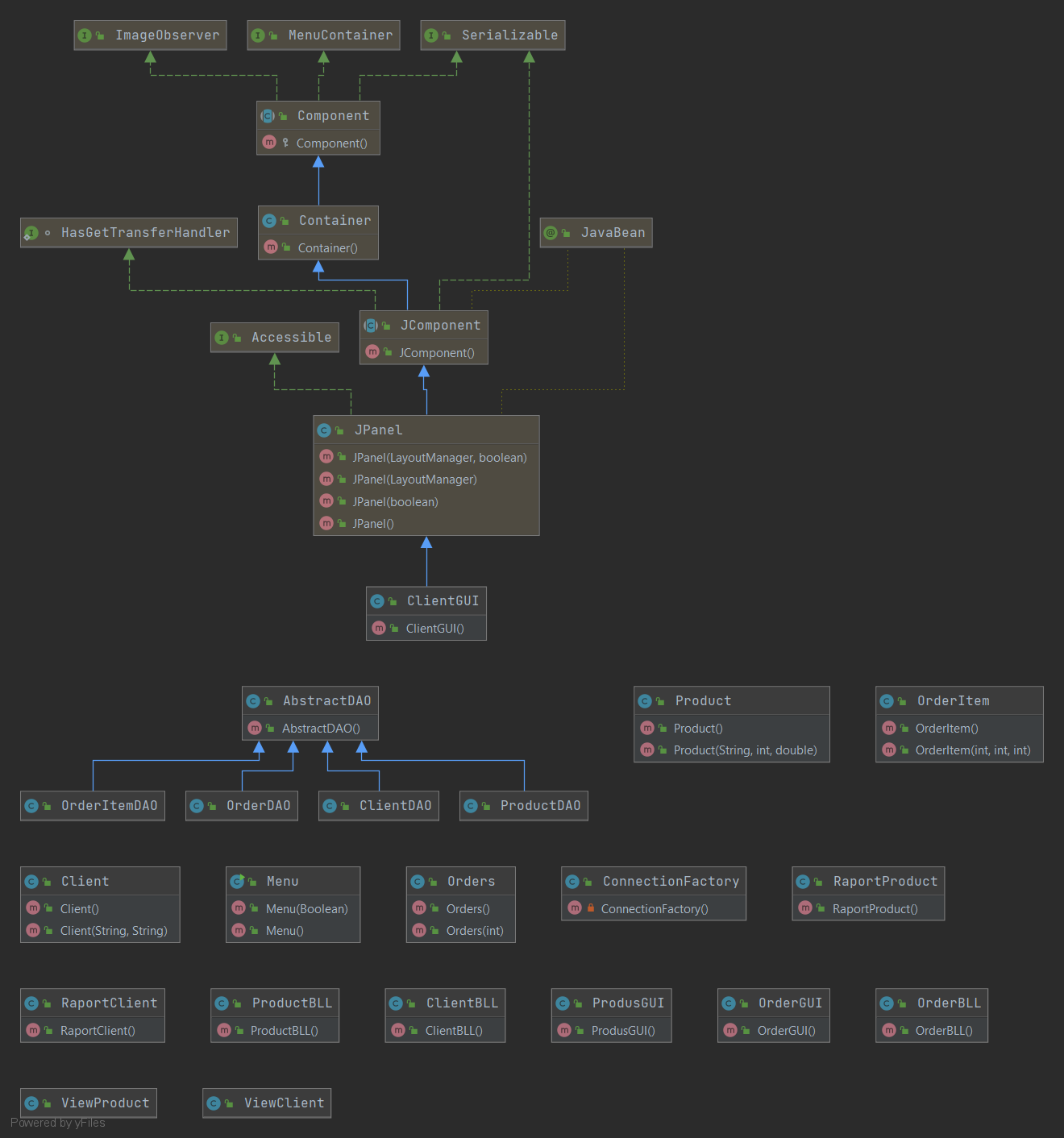
Pentru afisarea obiectelor in tabel se foloseste un tabel generic care primeste o lista de obiecte si le mapeaza intr-un tabel configurat special pentru tipul de obiect folosit.

3.1 Diagrama UML

Pentru a furniza datele corespunzător şi de a separa clar modul în care este structurat programul, am folosit modelul arhitectural stratificat( Layered architecture) pentru a realiza proiectarea acestei aplicaţii pentru WMS.

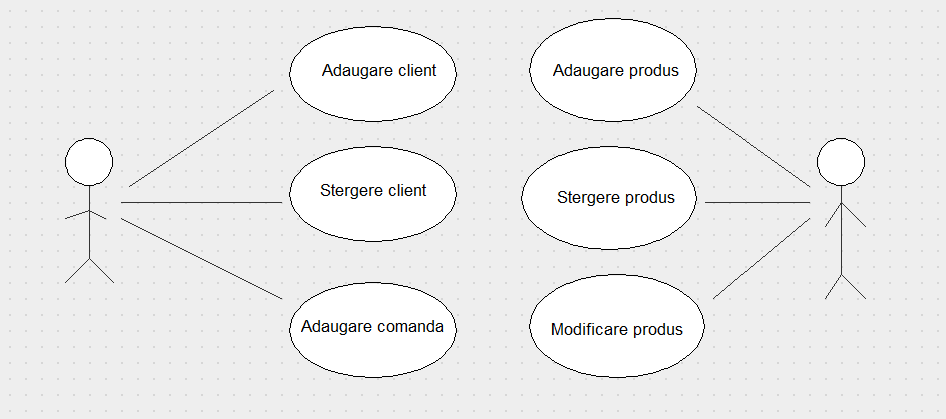
Modelul arhitectural stratificat constă în folosirea a 4 sau mai multe pachete diferite care impart logica aplicaţiei în 4 sau mai multe elemente interconectate. Acest lucru este făcut ca să împartă reprezentarea internă a informaţiei pe straturi care vor conţine fiecare date legate de o anumită categorie de date. Acest model structural este extrem de util atunci când se lucrează cu bazele de date, deoarece toate straturile se construiesc bottom-up de la baza de date spre prezentarea datelor către utilizator. Astfel, cele 4 pachete fundamentale în aplicaţia aceasta sunt următoarele:

* Modelul – componenta care reprezintă o entitate din baza de date în limbajul de programare ales, pe baza căreia se construieşte ulterior funcţionalitatea aplicaţiei; reprezintă structura de date abstractă;
* Data Access Layer– componenta care conţine clasele specifice legăturii dintre aplicaţie şi baza de date folosită pentru a stoca toate informaţiile prelucrate; acest pachet se împarte în două subpachete în aplicaţia curentă:
  + Data Access Objects – DAO – componenta care oferă o clasă “abstractă” generică pentru conexiunea cu datele stocate, aici fiind create propriu-zis interogrările şi prelucrarea datelor existente
  + Connection – componenta care oferă propriu-zis conexiunea cu baza de date, modelată după modelul singleton – în tot programul există o singură instanţă a conexiunii, deoarece mereu este nevoie strict de aceeaşi conexiune în baza de date, ar fi redundant să se creeze câte un obiect nou de fiecare dată când e nevoie de accesul în baza de date
* Business Logic Layer – componenta care conţine clasele specifice logicii aplicaţiei – teoretic aici se dezvoltă partea cea mai importantă: modelează şi determină exact cum se crează, modifică, stochează datele: funcţiile CRUD (create – read – update – delete)
* Presentation Layer – componenta care conţine clasele specifice interfeţei grafice, unde are loc formatarea şi – după cum îi spune şi numele – prezentarea informaţiei pentru procesare ulterioară.



3.2 Diagrama Use-Case (utilizare, functionalitate, descriere)

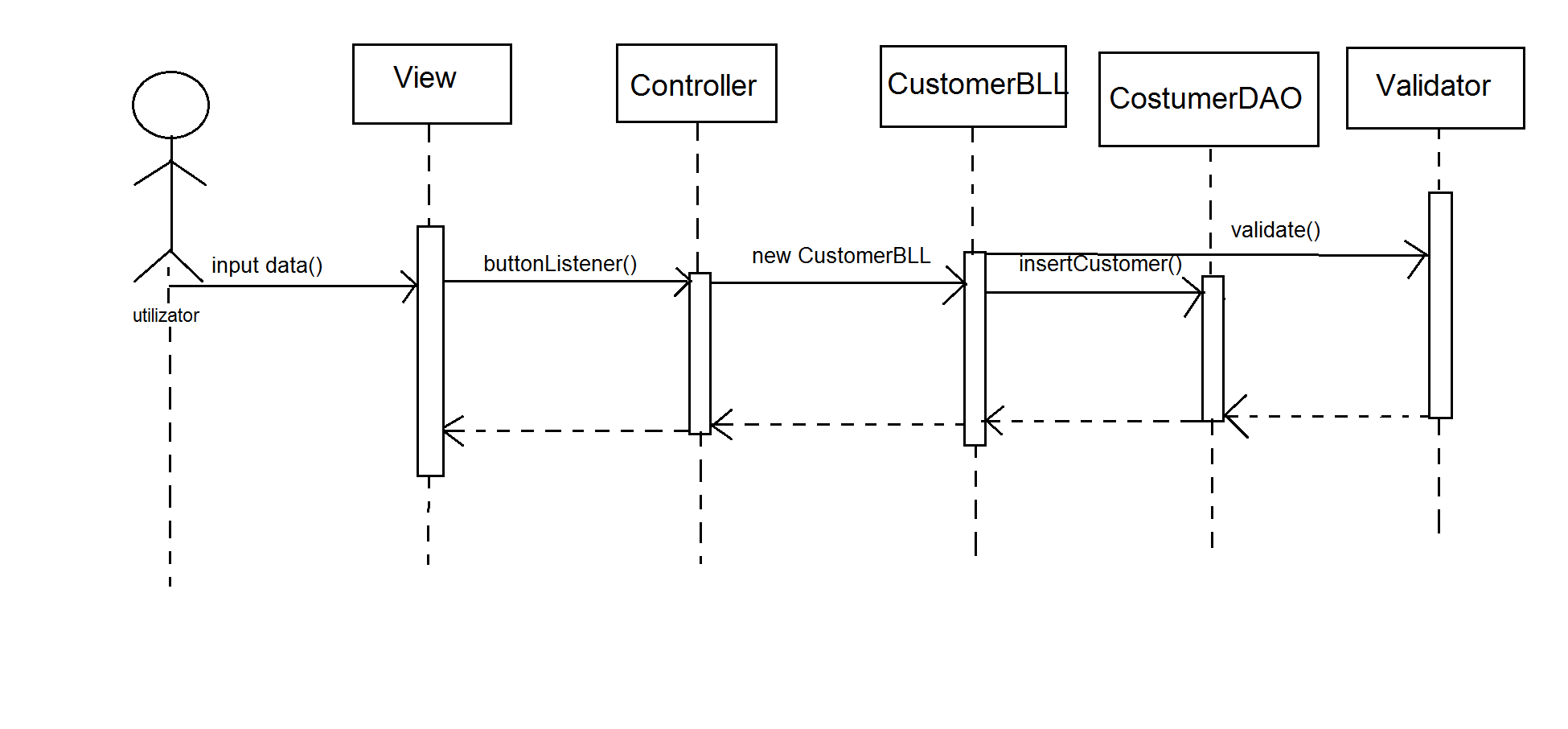
Diagrama *use case* permite prezentarea posibilităților pe care utilizatorul le are atunci când folosește aplicația. Se distinge o singura operație care poate fi executata de utilizator, deoarece aplicația are un rol de simulare demonstrativa. Diagrama este exemplificată în schema următoare.



3.3 Diagrama de secventa (utilizare, functionalitate, descriere)

Diagrama de secvență arată cum interacționează obiectele unul cu celălalt, și în ce ordine. Aceste interacțiuni sunt aranjate în ordine temporală și prezintă scenariile posibile care pot apărea în urma interacțiunii utilizatorului cu aplicația. În cazul sistemului de simulare al cozilor, utilizatorul introduce prin intermediul interfeței grafice datele inițiale, apoi apasă butonul adecvat operatiei dorite.

Diagrama de secvență este reprezentată prin schema următoare.



3.4 Proiectare de clase

Pentru a implementa această aplicaţie WMS am folosit o bază de date MySQL de ultimă versiune, iar ca IDE pentru devzoltarea părţii de SQL am folosit MySQL Workbench. Tabelele au fost create prin interfaţa grafică a IDE-ului.

Pe partea de Java, conexiunea la baza de date s-a facut folosind librărie externă specializată, printr-un driver specific din această librărie.

În clasa ConnectionFactory din pachetul Connection întâlnim metodele specifice legării aplicaţiei de baza de date. Metoda createConnection() va permite deschiderea conexiunii către baza de date. Prin metodele getConnection() se va prelua conexiunea în tot programul ulterior. Deoarece e obligatoriu ca pentru a reaccesa conexiunea în mai multe locuri din program, trebuie sa ne asigurăm că toate obiectele cu care se lucrează sunt închise şi deschise corespunzător, pentru a nu exista obiecte “zombie” care să creeze erori. Se va supraîncărca metoda close() în funcţie de tipul obiectului care va fi închis: Connection, Statement sau ResultSet.

În pachetul DAO, cea mai importantă clasă, după cum am menţionat, este AbstractDAO, o clasă parametrizată cu un tip abstract, care se va plia în funcţie de clasa care extinde AbstractDAO, pe un tip concret. Constructorul acestei clase va prelua tipul concret al clasei T. Următoarele metode care sunt implementate în AbstractDAO sunt generate folosind Java Reflection API, un API care îi permite codului să se analizeze şi să se modifice dinamic în timpul rulării, pentru a elimina redundanţa codului scris. Astfel întâlnim metode care să creeze un query specific pentru baza de date: createSelectQuery(), createInsertQuery(), createUpdateQuery(), createDeleteQuery(), şi metodele care folosesc aceste stringuri de interogare pentru a acţiona direct asupra bazei de date: insert(), update(), select(), delete(). Pe lângă aceste metode, mai apar trei metode care sunt folosite pentru a rezolva problemele inserţiilor cheilor străine: findID(), sau a găsi un obiect cu un anume ID din baza de date: findByID(). Metoda findAll() va returna o listă a obiectelor de tip T şi este folosită pentru a genera raporturile. Celelate clase din pachetulDAO nu au vreo implementare concretă, acestea doar extind clasa AbstractDAO.

În pachetul BLL întâlnim trei clase specifice fiecărei entităţi fundamentale din model: ClientBLL, ProductBLL, OrderBLL. După cum am menţionat anterior, în aceste clase se găsesc metode CRUD. Aici sunt variaţiuni pe aceeaşi temă: insertClient(), insertProduct(), insertOrder(), findClientID(), findProductID(), findOrderID(), updateProduct(), deleteClient(), deleteProduct(). Metoda de READ va genera fişierele PDF specifice fiecărei categorii: generateClientReport(), generateProductReport(), generateBill(), generateOrderReport().

Pachetul Model este probabil cel mai simplu dintre toate, conţine clasele care descriu entităţile din baza de date. Fiecare clasă (Client, Product, Orders, OrderItem) este denumită la fel cu tabela corespunzătoare şi va conţine aceleaşi atribute ca şi câmpurile din tabelă. Pentru fiecare din aceste clase sunt scrise getteri şi setteri specifici, cât şi constructori care vor crea obiectele atunci când este nevoie.

4. Implementare si testare

Testarea aplicatiei a fost facuta prin intermediul unui fisier text. Pentru fiecare operatie (stergere/ adaugare client, stergere/adaugare produs, schimbare pret, listarea tuturor produselor/comenzilor/clientilor/stock-ului disponibil) s-a efectuat testarea tuturor cazurilor posibile.

5.Rezultate

S-a obtinut o aplicatie care ne ofera posibilitatea de a gestiona produsele si comenzile existente la un depozit sau la un anumit magazin. Aplicatia ne ofera doar posibilitatea efectuarii operatiilor de baza si anume: adaugare articol, stergere articol, adaugare client, stergere client, schimbare pret produs, adaugate comanda, listare produse, listare comenzi, listare clienti, listare stock.

6. Concluzii, ce s-a invatat din tema, dezvoltari ulterioare

Din aceasta tema am invatat sa fac legatura dintre o aplicatie Java si o baza de date MySql.

Dezvoltari ulterioare:

-posibilitatea logarii cu parola si username a clientilor;

-in cazul in care clientul doreste sa faca o comanda pentru un anumit produs existent in depozti si cantitatea introdusa de client este mai mare decat cantitatea existent in baza de date sa se afiseze cantitatea existenta in mesajul de atentionare;

-posibilitatea clientului de a obtine o lista cu propriile comenzi;

-interactiune client – GUI .

7. Bibliografie

<http://wikipedia.org>

<http://google.ro>

<http://youtube.com>