Order Management

*Documentatie*

Nume: Samoila Alexandra

Grupa: 30228

Disciplina : Tehnici de programare

Data: 5.07.2019

Cuprins:

1. Obiectivul temei

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

3. Proiectare

4. Implementare

5. Rezultate

6. Concluzii

7. Bibliografie

1. Obiectivul temei

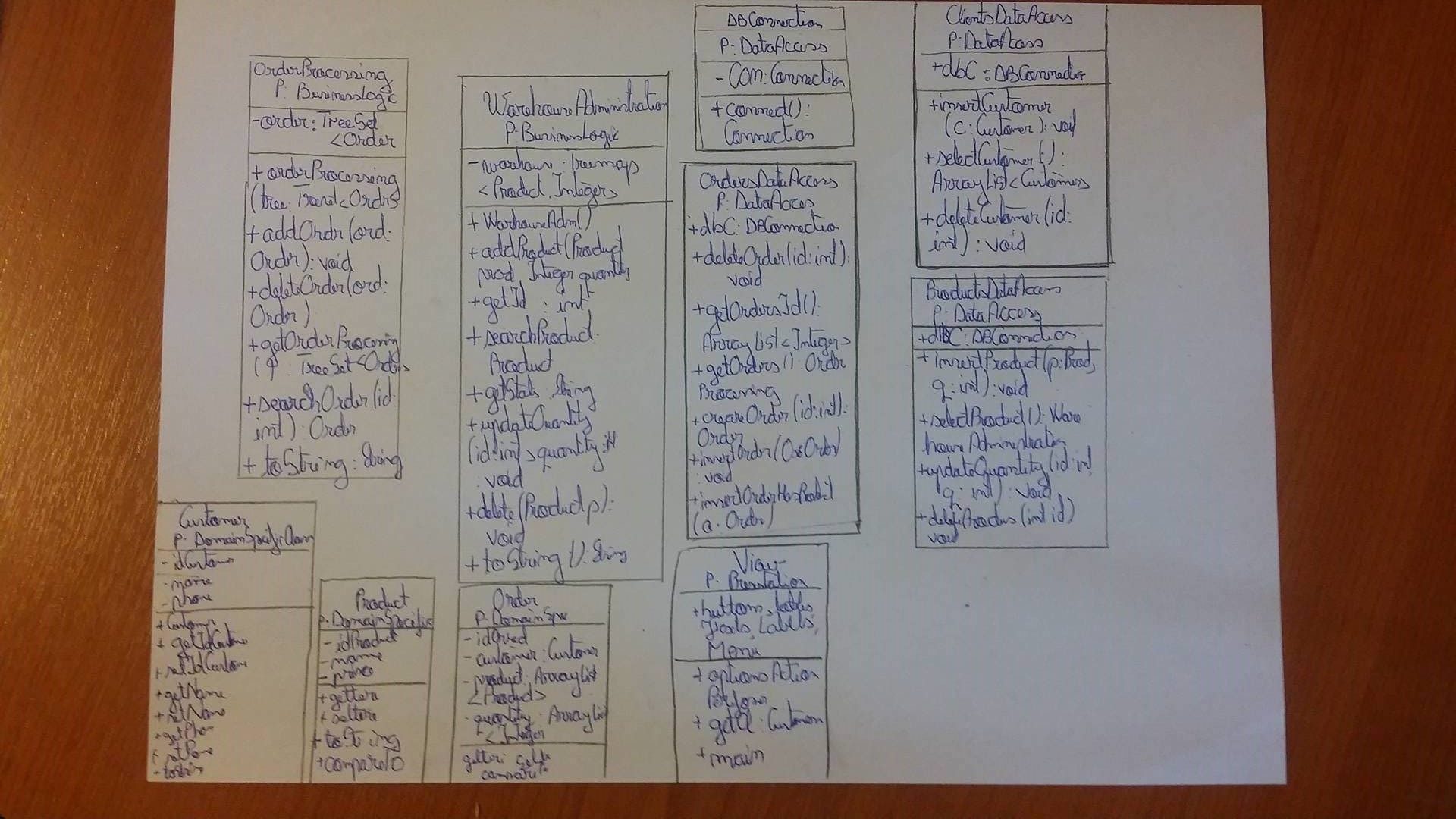
Aceasta tema are ca obiectiv realizarea unui proiect care efectueaza operatii atat asupra unor obiecte care sunt construite prin extragerea lor dintr-o baza de date cat si direct asupra bazei de date astfel incat sa se respecte o concordanta 1 la 1 intre obiectele java si continutul bazei de date.

Proiectul acesta presupune realizarea unei aplicatii de *Order Management* care sa fie utila in gestionarea unui depozit de produse diverse. Aplicatia ofera suport pentru adaugare, stergere, modificare de produse, tinerea unei evidente detaliate a clientilor ce cumpara produse si comenzile acestora. O caracteristica important a aplicatiei este aceea de notificare cand una sau mai multe produse au stocul depasit sau prea putine produse in stoc.

Toate operatiile pe care un utilizator le poate efectua cu ajutorul aplicatiei se realizeaza prin intermediul unei interfete foarte intuitive. Aceasta este structurata in 3 parti. Operatii asupra produselor din depozit, operatii asupra clientilor si gestiunea comenzilor. Alegerea intre asupra careia dintre aceste se doreste a se efectua operatii se realizeaza prin intermediul meniului din interfata java. In fiecare panel al interfetei apare si un tabel cu produsele/clientii/comenzile deja existente in baza de date care dupa fiecare operatie efectuata va fi actualizat corespunzator.

In vederea realizarii acestui proiect am utilizat si unele structure noi pe langa ArrayList-urile deja utilizate in proiectele anterioare. Aceste structure sunt TreeMap si TreeSet care vor fi detaliate ulterior.

1. Analiza problemei, modelare, scenarii, cazuri de utilizare
2. Proiectare
   1. Diagrama UML:



* 1. Structure de date, proiectare clase, interfete, relatii

O colectie este de fapt un obiect care grupeaza mai multe elemente intr-o singura unitate. Colectiile se utilizeaza pentru stocarea, regasirea si manupularea datelor. Deci, colectiile reprezinta obiecte care in mod natural formeaza un grup. In JDK 1.1 am avut trei tipuri de colectii:

* Vector
* Hashtable
* Array

Incepand cu JDK 1.2. s-a introdus un cadru nou pentru lucrul cu colectii. Orice cadru contine urmatoarele trei lucruri esentiale:

* interfete: tipuri de date abstracte (TAD=ADT Abstract Data Type) pentru reprezentarea diferitelor colectii. Aceste interfete permit colectiilor sa fie manipulabile intr-un mod unitar indiferent de implementarii.
* implementari: implementarile concrete ale tipurilor de date abstracte din interfete
* algoritmi: o colectie de metode care permit operatii asupra colectiilor (cautare,  sortare). Acesti algoritmi sunt polimorfici, adica aceasi metoda se utilizeaza pentru diferite tipuri de date (ex. metoda sort(), ordoneaza orice colectie ale caror elemente implementeaza interfata Comparable).

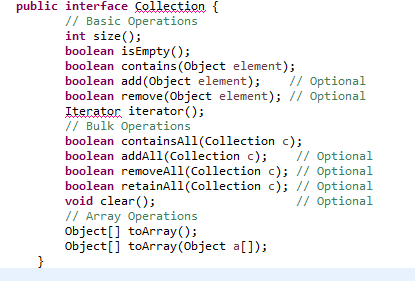
Acestea fiind spuse, in realizarea acestui proiect am utilizat: Interfata Collection (ArrayList), Interfata Set (TreeSet) si Interfata Map (TreeMap):

**Interfata Collection**

Aceasta interfata este interfata de baza din care sunt specializate celelalte interfete. O colectie reprezinta un grup de obiecte numite si elemente. Anumite colectii permit redundanta (acelasi element sa apara de doua ori), altele nu. Unele sunt ordonate, altele nu. Nu exista nicio clasa care sa implementeze in mod direct aceasta interfata, doar interfetele Set si List vor avea implementari.

Orice implementare indirecta a acestei interfete  poseda un constructor  cu un argument de tip Collection si care initializeaza colectia respectiva cu cea specificata. De exemplu:   
fie c o colectie (List, Set sau un alt tip), atunci   
List  l = new ArrayList( c );   
creaza un obiect de tip ArrayList initializat cu colectia c.

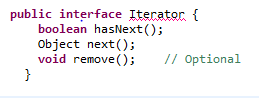
Definitia interfetei Collection: 



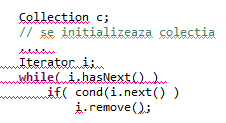
**Operatii elementare:**

* a afla numarul elementelor din colectie ( size )
* a verifica daca lista este vida ( isEmpty )
* a verifica daca un anumit obiect apartine colectiei
* adaugarea si stergerea elementelor
* a obtine un iterator pentru parcurgerea elementelor colectiei

Iteratorul este un obiect similar cu Enumeration, dar pe langa parcurgerea colectiei poate sa permita si stergerea elementelor. Interfata Iterator este declarat in java.util in modul urmator:



Metoda hasNext() este similara cu metoda hasMoreElements() a interfetei Enumeration, iar metoda next() cu nextElement() a aceleiasi interfete. Metoda remove() sterge elementul curent (elementul obtinut prin ultimul apel next()).  Urmatoarea portiune de cod parcurge o colectie stergand elementele care respecta o anumita conditie.



Exemplul precedent functioneaza pentru orice colectie, deci algoritmul de mai sus este unul polimorfic.  Ar fi fost imposibil de implementat acest algoritm cu Enumeration deoarece interfata nu permite stergerea sigura a elementelor pe parcursul traversarii.

**Operatii in masa**

Sunt operatii care executa o anumita operatie asupra mai multor elemente dintr-o colectie. Aceste operatii pot fi executate si  altfel (prin repetarea aceleiasi operatii elementare), dar probabil  intr-un mod mai putin eficient. 

* boolean containsAll( Collection c):  returneaza true, daca c este continuta in intregime in colectie
* boolean addAll( Collection c): adauga la colectie toate elementele colectiei c si returneaza true daca colectia s-a schimbat dupa adaugare
* boolean removeAll( Collection c ): sterge toate elemntele care apartin colectiei c si returneaza true daca colectia s-a schimbat dupa stergere
* boolean retainAll( Collection c ): sterge toate elementele care nu apartin colectiei c si returneaza true daca colectia s-a schimbat dupa stergere
* void clear(): sterge toate elementele colectiei

**Operatii cu array**

Metoda toArray() s-a introdus pentru a pastra compatibilitatea cu versiunile precedente, permite transformarea unei colectii intr-un array. Fara nici un argument se obtine un array format din elemente de tip Object.

  **Interfata Set**

Interfata Set reprezinta o colectie care nu permite duplicate. Modeleaza multimea din matematica. Nu contine metode noi, doar cele mostenite de la interfata Collection, la care adauga restrictia cu duplicatele. Doua obiecte de tip Set sunt egale daca ele contin aceleasi elemente.

Pachetul java.util contine doua implementari a interfetei Set:

* *HashSet*: stocheaza elementele colectiei intr-un tablou hash
* *TreeSet*: stocheaza elementele intr-un arbore "red-black"

Utilizare:

Collection c;   
...   
Collection c1 = new HashSet(c);

Exemplul urmator afiseaza duplicatele dintr-o colectie. (Colectia va fi formata din argumentele liniei de comanda)

**Executie:** java Duplicate am venit am vazut am plecat   
**Rezultat:**   
Duplicat: am   
Duplicat: am   
4 cuvinte distincte: [ vazut, venit, plecat, am]

Schimband *HashSet* la *TreeSet*vom obtine cuvintele in ordine alfabetica, adica   
Duplicat: am   
Duplicat: am   
4 cuvinte distincte: [ am, plecat, vazut, venit ]

In exemplele din acest capitol intotdeanu ne-am referit la o colectie printr-o referinta la o interfata in loc de o referinta la o implementare. Este o tehnica foarte buna, astfel schimband implementarea nu va trebui sa rescriem codul, deoarece nu contine referinte la implementare,ci doar la interfata.

Operatii cu multimi:

*Reuniune*

Set union = new HashSet(s1);   
union.addAll(s2); 

*Intersectie*

Set intersection = new HashSet(s1);   
intersection.retainAll(s2);

*Diferenta*

Set difference = new HashSet(s1);   
difference.removeAll(s2);

In exemplul cu duplicatele am obtinut la sfarsit multimea care contine fiecare element o singura data, fara duplicate. Vom modifica exemplul astfel incat sa afiseze doua multime, una care contine doar cuvintele care apar o singura data, iar o alta multime care contine duplicatele. Pentru a realiza acest lucru vom lucra cu doua multimi (doua obiecte de tip Set). La prima multime se adauga toate cuvintele, iar la a doua multime doar duplicatele si la sfarsit se considera diferenta celor doua multimi astfel obtinand unicatele.

**Interfata Map**

Map este un obiect care asocieaza chei la valori si nu permite duplicate pentru cheie. Deci fiecare cheie are asociata o singura valoare. 

In JDK avem doua implementari pentru aceasta interfata:

* HashMap, care stocheaza elementele intr-un tabel hash
* TreeMap, care stocheaza elementele intr-un arbore "red-black"

Interfata Map generalizeaza clasa Hashtable existenta si in versiunile precedente. Permite iterarea cheilor, valorilor si a perechilor cheie-valoare. Mai mult permite si operatii de stergere pe parcursul iterarii. Programul urmator calculeaza frecventa cuvinteleor in lista argumentelor liniei de comanda.

*Metode pentru obtinerea unor view-uri din colectie*

keySet: este o multime (Set) care contine toate cheile din colectie

values: colectia valorilor atasate cheilor, nu este o multime adevarata deoarece aceasi valoare poate fi atasata mai multor chei

entrySet: multimea (set) perechilor cheie-valoare continute in Map.

Interfata Map contine si o interfata incuibata Entry pentru tipul elementelor din multime.

Iterarea peste elementele colectiei se poate efectua intr-un singur mod

• Proiectarea Claselor:

  Proiectul contine 10 clase, impartite in 4 pachete. Clasele sunt:

* + **OrderProcessing, WharehouseAdministration – BusinessLogic Package;**
  + **ClientsDataAcces, DBConnection, OrdersDataAcces, ProductsDataAccess – DataAccess Package;**
  + **Customer, Order, Product – DomainSpecificClasses Package**
  + **Presentation – View.**

Clasa **OrderProcessing** este clasa care contine un obiect de tip TreeMap in care sunt stocate comenzile. Aceste comenzi pot fi accesate prin intermediul unei chei care este reprezentat de identificatorul unic atribuit ficarei comenzi. Printre operatiile care se pot efectua pe obiectele de tipul OPDept se numara adaugarea unei noi comenzi si vizualizarea unei comenzi indicate prin identificatorul unic precum si stergerea unei comenzi existente in baza de date dupa id-ul comenzii. In plus, aceasta clasa contine o metoda ce suprascrie metoda toString pentru o reprezentare mai usoara si mai rapida a datelor stocate si testarea metodelor inainte de a le apela in interfata.

Clasa **WarehouseAdministration** este clasa care contine obiectul de tip **TreeMap** in care se vor stoca produsele. Functionalitatile pe care le indeplineste aceasta clasa sunt in principal cele de adaugare a unui nou produs, de a cauta un produs dupa id pentru a garanta accesul in vederea actualizarii datelor si stergerea unui produs indicat prin id-ul acestuia. Tot in aceasta clasa implementez metode pentru modificarea cantitatii produsului, o metoda pentru obtinerea starii produsului in momentul apelului metodei. Aceasta metoda returneaza un String care va aparea in interfata si va “spune” starea produsului in functie de cantitate. Aceasta poate fi : “ UnderStock “ - cand cantitatea produsului respectiv este mai mica de 50 de bucati si respective “ OverStock “ - cand cantitatea produsului respectiv este mai mare de 300 de bucati.

Clasele din DataAcces Package, sunt clasele in care se realizeaza conexiunea cu baza de date, si astfel am ales sa impart in Clienti, Produse si Comenzi, clase care contin codul pentru metodele de inserare, selectare, stergere, update-uri din MySQL. Toate aceste clase utilizeaza o instant a clasei DBConnection, care realizeaza legatura la Baza mea de date, utilizandu-se de parola si localhost-ul meu.

Clasa **Customer** este clasa care reprezina un client si detine ca variabile de instanta numele, prenumele, si numarul de telefon al clientului inscris in modelul aplicatiei. Aceasta contine metode de tip getter si setter pentru toate variabilele de instanta precum si o metoda “ toString “ pe care am folosit-o in vederea testarii metodelor inaintea implementarii interfetie.

Clasa **Order** este abstractizarea unei comenzi. Ea contine informatiile esentiale specifice unei comenzi, avand ca atribute un id de comanda, un obiect de tip client care pastreaza toate informatiile despre clientul respectiv, un ArrayList de obiecte de tip produs si un ArrayList de cantitati. Tot aici suprascriu metoda “ compareTo “ in vederea ordonarii comenzilor dupa id.

Clasa **Product** este clasa care reprezinta un produs, avand ca variabile de instant id-ul produsului, numele produsului, si pretul. Accesul la aceste variabile se face prin intermediul unor metode de tip getter si setter astfel incat se poate actualiza orice informatie ce are legatura cu produsele. Pe de alta parte, aceasta clasa implementeaza interfata **Comparable** pentru a permite ordonarea produselor in ordinea crescatoare a id-urilor produsului.

Obiectele de tip *Product* sunt salvate intr-un **TreeMap** aflat in clasa **WarehouseAdministration**. In aceasta clasa se afla metodele pentru operatiile aferente produselor: adaugare, actualizare si stergea produselor. In plus, in clasa *Product* se suprascrie metoda *public int compareTo().* Avem nevoie de aceasta metoda, deoarece in decursul implementarii proiectului avem nevoie de a compara doua produse.

Clasa **View** este clasa care implementeaza partea cu care ultilizatorul aplicatiei ia contact direct. Aceasta clasa extinde clasa JFrame si contine constructorul in interiorul caruia sunt create si pozitionatele obiectele care sunt vizibile in interfata, metoda “optionsActionPerformed “ in interiorul careia sunt implementate functionalitatile butoanelor si metoda “ main “ in interiorul careia este instantiatata clasa “ View“ prin intermediul construtorului acesteia . Pentru implementarea clasei “ Interfata “ am utilizat si obiecte noi fata de proiectele trecute : printre acestea se numara obiectele de tip JMenu si JTable .

* 1. Algoritmi:

Pentru implementarea acestui proiect nu a fost necesara utilizarea unor algoritmi complecsi ci doar efectuare unor operatii simple asupra bazei de date de unde initial sunt extrase datele dupa care operatiile urmatoare care pot fi de adaugare, modificare, stergere se efectueaza atat asupara bazei de date cat si asupra obiectelor corespunzatoare create anterior in java in vederea pastrarii relatiei 1 la 1 intre baza de date si obiectele java.

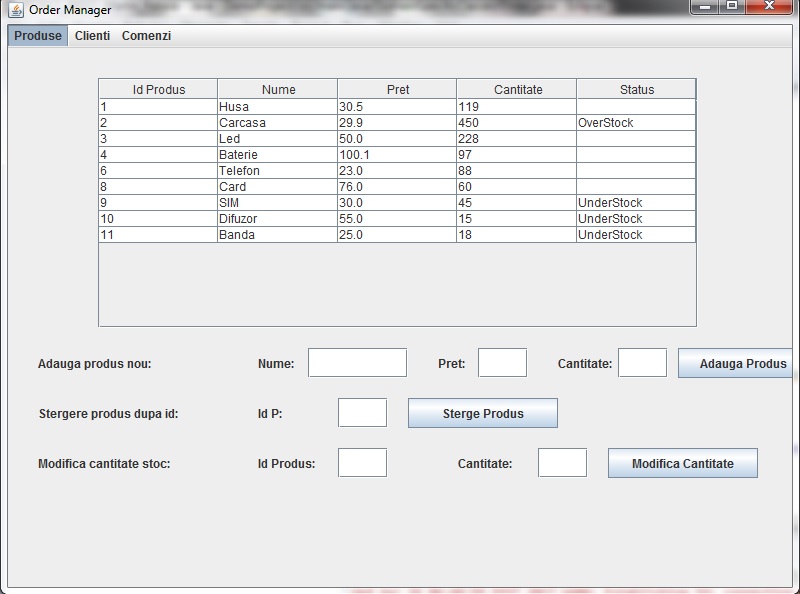
Pentru realizarea acestei relatii 1 la 1 am utilizat operatii simple pe ArrayList-uri, TreeMap-uri si TreeSet-uri si functii specifice MySql cum ar fi “ SELECT “ , “ INSERT “ , “ UPDATE “ si “ DELETE “ .

Pentru implementarea acestui proiect am utilizat structure de tip TreeSet si TreeMap. Pentru aceste structuri nu a fost nevoie sa le implementez eu personal deoarece acestea sunt gata implementate in Java avand metode gata implementate care sunt relativ complexe insa datorita limbajului Java care este suficient de bine dezvoltat astfel incat sa ne usureze munca intelectuala depusa pentru a implementa aceste metode.

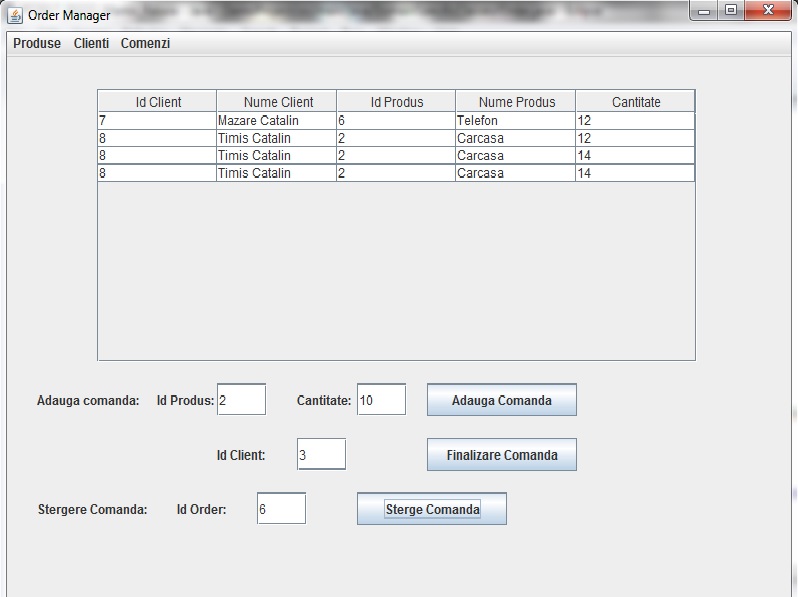
* 1. Interfata utilizator:

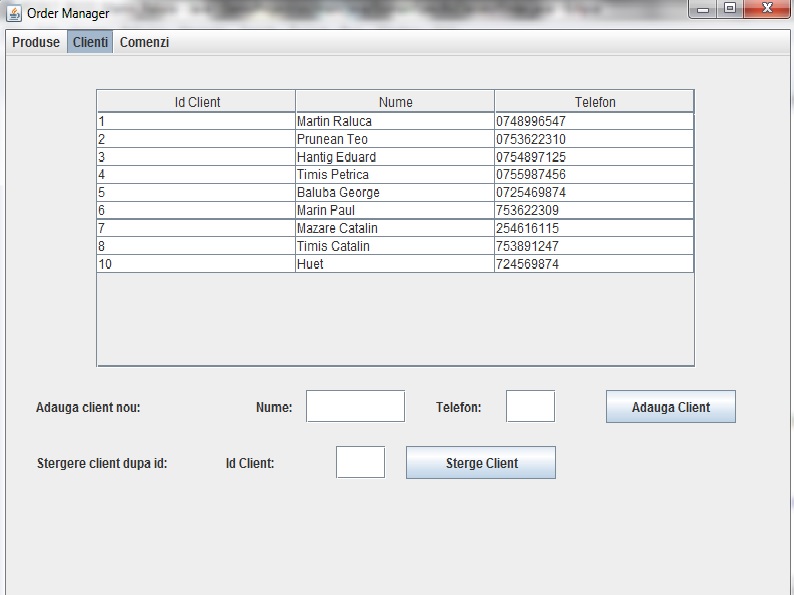
Interfata este una prietenoasa, usor de utilizat, intuitiva si destul de complexa, cuprinzand elemente de tip JMenu, JTable precum si clasicele JLabel-uri, JTextField-uri si JButton-uri. Toate aceste sunt utilizate pentru a prezenta cat mai bine informatiile care apoi vor fi procesate precum si a implementa cat mai simplu si intuitiv toate operatiile asupra bazei de date.

Elementele de tip JTable sunt utilizate pentru a vizualiza cat mai usor datele asupra carora efectuez mai apoi operatii iar elementele de tip JMenu sunt utilizate pentru a realiza o separere a managementului Clientilor de ce al Produselor si de cel al comenzilor. Uneori modificarile asupra unora dintre acestea, modificarile se rasfrang si asupra alteia. De exemplu: efectuarea unei comenzi presupune modificarea cantitatii produselor din depozit informatie vizibila in interfata in primul cam din meniu la sectiunea Produse. Odata cu aceasta se poate modificara si starea produsului respectiv, adica sa scada prea multa cantitatea si sa ajunga UnderStock. Sau daca era OverStock sa ajunga la o valoare “ ok ”.



Pentru a introduce o comanda, se introduce datele in primele 3 TextFielduri, ca mai apoi sa se apese “Adauga Comanda” si apoi “Finalizare COmanda” urmand update-ul tabelului si a cantitatii de produse, sau “Cantitate Depasita!”.



Tabela Clientilor

1. Implementare si testare

La implementarea acestei teme, am inceput prin a creea modelul acestei clase si logica din spatele acestei aplicatii. Pentru a verifica functionalitatea modelului, am folosit o funtie “main” unde am verificat corectitudinea metodelor definite in modelul aplicatiei. In final, dupa ce am considerat faptul ca modelul creat in pasul precedent este functionabil si utilizabil, am trecut la pasul urmator, cel in care se ofera o interfata grafica aplicatiei.

In stadiul final al proiectului, interfata este una placuta si foarte responsiva la comenzile executate de utilizator, elementele grafice sunt foarte bine definite si delimitate intr-o structura bine ganditasi care nu conduce utilizatorul la confuzii datorita claritatii (in general) a aplicatiei.

Un rol important in implementarea acestei aplicatii o joaca structura de date in care sunt stocate produsele si comenzile. Faptul ca acestea sunt salvate intr-un arbore binar de cautare semnifica faptul ca accesul la ele se face in cel mai scurt timp posibil, acest fapt datorandu-se numarului mic de cautari care trebuie facute. In plus, dupa efectuarea unei operatii de adaugare sau de stergere a unui element din aceasta structura de date, ea isi pastreaza proprietatea de arbore binar de cautare. Toate aceste caracteristici sunt oferite de clasa TreeMap care stocheaza date intr-un arbore rosu-negru care prin definitie este un arbore binar de cautare echilibrat.

1. Rezultate :

Rezultatele sunt vizibile prin intermediul interfetei.

Rezultatele provenite in urma etapei de testare a acestei aplicatii de gestiune a comenzilor confirma faptul ca aplicatia isi indeplineste sarcina, adica reuseste sa implementeze un mecanism de management a comenzilor. Astfel managementul comenzilor are 3 componente : managementul produselor , managementul clientilor si managementul propriu – zis al comenzilor. Managementul produselor presupune : adaugare unui produs nou, modificarea cantitatii unui produs ales pe baza id-ului produsului caruia dorim sa-i modificam cantitatea, dar si stergerea unui produs ales tot in functie de id-ul produsului care dorim sa il stergem.

Managementul clientilor presupune : adaugarea unui client nou cu toate informatiile de contact ale acestuia, si stergerea unui client din baza de date. Clientul este selectat in functie de id.

Managementul comenzilor presupune adaugarea unei noi comenzi. O comanda poate contine unul sau mai multe produse. Dupa alegerea fiecarui produs dupa id se apasa butonul adauga produs iar la final se apasa butonul finalizare comanda care va adauga toate produsele in lista de comenzi si updateaza cantitatea de produse ramase in Warehouse dupa adaugarea unei comenzi.

Pentru a usura alegerea produselor / clientilor si respectiv comenzilor asupra carora se efectueaza operatiile de adaugare / stergere / modificare am ales ca in interfata sa afisez intr-un tabel datele deja existente in baza de date. Aceste informatii sunt updatate dupa fiecare modificare asupra bazei de date si a obiectelor din Java care detin informatiile din baza de date.

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

Sistemele de management a comenzilor reprezinta un subiect atat de comun in zilele noastre incat eficientizarea modului in care sunt gestionate capata din ce in ce mai multa atentie.

Aplicatia vine astfel ca o metoda interesanta de eficientizare a managementului comenzilor, simplasi usor de utilizat, care realizeaza functionalitatile de baza a unui magazin online.

Realizarea aplicatiei nu este dificila din punct de vedere algoritmic, dar pentru a o realiza cat mai eficient, a fost necesara o atentie foarte mare la detalii – inserarea/stergerea produselor corect in/din arbore, serializarea si deserializarea arborelui.

Personal, tema m-a ajutat la formarea unei idei si mai bune asupra a ceea ce inseamna programarea OOP. Am invatat cum trebuie manipulate datele folosind JTable, am invatat conceptele si modul de lucru folosind clasele TreeMap si TreeSet.

O idee de dezvoltare ulterioara ar putea fi introducerea de thread-uri si in acest proiect. Acesta ar avea rolul de a reactualiza informatiile din interfata in momentul in care sa presupunem ca 2 clienti efectueaza concomitant in modificari asupra bazei de date, fiecare avand acces la servarul pe care lucreaza aplicatia. Modificarile aduse de cei doi in baza de date nu vor fi vizibile in informatiile din interfata, in cazul in care nu avem thread-uri insa in cazul in care avem, in fiecare secunda se face o verificare a informatiilor si in cazul in care s-au efectuata modificari fata de ceea ce exista in interfata, informatiile din interfata sa se actualizeze. Verificare existentei sau nu a modificarilor se poate implementa prin adaugarea unui camp nou in tabele, de tip Boolean care devine “ true ” daca informatiile de pe linia respective au fost modificate, iar in rest false.

1. Bibliografie :

1. “Thinking in Java” , Fourth Edition , Bruce Eckel, President, MindView, Inc. teochew;

2. “Java de la 0 la expert”, Stefan Tanasa, Cristian Olaru, Stefan Andrei, Polirom, 2003;

3. <http://stackoverflow.com/>;

4. Tutoriale swing.