*Tehnici de programare Universitatea Tehnică din Cluj -Napoca*

*Tema 3*

**Warehouse management**

**-documentație-**

*Nimigean Emanuela*

*CTI, Grupa 30226*

♦Cerințe funcționale

Dezvoltarea unui program de gestiune a clientilor, produselor si comenzilor ce tin de un depozit de marfuri , care să permită efectuarea următoarelor operații:

* Inserarea, editarea, stergerea din/in baza de date a clientilor, respectiv a produselor (pentru aceasta se selecteaza operatia dorita din fereastra Client/ Produs, insotita de completarea campurilor aferente operatiunii dorite .
* Vizualizarea clientilor/ produselor aflate in baza de date a depozitului, sub forma de tabel .
* Crearea de comenzi prin selectarea unui client existent in baza de date, a unui produs existent in baaza de date si a cantitatii dorite de client din respectivul produs, una mai mica sau egala cu cea aflata pe stoc si afisarea unei chitante aferente comenzii, in format txt.
* Obiectivul temei

Obiectiv principal: propunerea, proiectarea și implementarea unui sistem de procesare a comenzilor efectuate intr-un depozit si gestiunea clientilor, respectiv a produselor .

Obiective secundare:

* Dezvoltarea de use-case -uri și scenarii -detalii în cap. următor
* Împărțirea pe clase : proiectul este structurat pe cinci clase, împărțite în sase pachete astfel: clasa Start e inclusa în pachetul start, clasa COnnectionFactory este inclusă în pachetul connection, clasele Client, Order, OrderDetail si Product sunt incluse în pachetul model, clasele ClientView, OrderDetailView, MainnView si ProductView sunt incluse in pachetul presentation, AbstractDAO, CLientDAO, ProductDAO si OrderDetailDAO sunt incluse in pachetul dao, clasele ClientBLL, EmailValidator, OrderDetailBLL, ProductBLL si Validator sunt incluse in pachetul bll -detalii în cap . Proiectare și Implementare
* Implementarea soluției -detalii în cap . Implementare
* Testare -detalii în cap . Rezultate
* Analiza problemei, modelare, scenarii, cazuri de utilizare

Se cere implementarea unui program de gestiune a stoculului si a comenzilor aferente unui depozit de marfuri care să permită efectuarea următoarelor operații:

-Citirea de la tastatura a datelor corespunzatoare tabelului din baza de date astfel: id, nume, adresa si email pentru clienti, id, nume, pret, numar pentru produse, respectiv id\_comanda, id\_client, id\_produs si cantitate pentru comenzi .

-Inserarea, editarea, stergerea din/in baza de date a clientilor, respectiv a produselor (pentru aceasta se selecteaza operatia dorita din fereastra Client/ Produs, insotita de completarea campurilor aferente operatiunii dorite. Vizualizarea clientilor/ produselor aflate in baza de date a depozitului, sub forma de tabel .

- Crearea de comenzi prin selectarea unui client existent in baza de date, a unui produs existent in baaza de date si a cantitatii dorite de client din respectivul produs, una mai mica sau cel mult egala cu cea aflata pe stoc si afisarea unei chitante in format txt.

Use case name : Warehouse management

Actors :

Triggers : user-ul indică faptul că dorește efectuarea unei anumite operații asupra bazei de date a depozitului, cu datele introduse

Preconditions : user-ul a introdus datele pe care vrea să le folosească

Post-conditions :

- programul va citi datele

- va fi selectată și efectuată operația cerută

- se furnizează un rezultat

Normal flow :

- user-ul introduce datele

- user-ul selectează operația dorită prin apăsarea butonului corespunzător acesteia

- datele sunt citite și sunt validate și transpuse în obiecte de clasă Client/Product/OrderDetail

- se apelează metoda corespunzătoare operației alese pe aceste date date ca parametri

- se calculează rezultatul operației, un set de elemente

- rezultatul este afișat într-un JTable

- dacă se dorește efectuarea unei alte operații , use case-ul se reia

Alternate flows :

1. - user-ul introduce datele

- user-ul selectează operația dorită prin apăsarea butonului corespunzător acesteia

- datele sunt citite, însă nu respectă formatul impus, deci nu pot fi procesate

- use case-ul se reia

1. - user-ul introduce doar unele date, dar introduce totusi id-ul/ id-urile necesare

- user-ul selectează una din operații

- programul foloseste datele primite și use case-ul se reia daca se doreste efectuarea unei alte operatii

1. user-ul nu introduce id-ul/ id-urile clientilor/produselor/comenzilor fara de care nu se pot efectua operatii si use-case-ul se reia.
2. User-ul introduce id-uri inexistente in baza de date pentru clienti sau produse atunci cand doreste crearea unei comenzi si use-case-ul se reia

* Proiectare ( decizii de proiectare, diagrame UML, structuri de date, proiectare clase, interfețe, relații, packages, algoritmi, interfața utilizator )

Împărțirea proiectului pe pachete am făcut-o separând componentele acestuia: modelul ( package model - clasele Client, Product, Order si OrderDetail) , interfața (package presentaion - clasele ClientView, ProductView, OrderDetailView si Mainn ), logica operatiilor(package bll- clasele ClientBLL, OrderDetailBLL, ProductBLL), implementarea operatiunilor(package dao- clasele AbstractDAO, ClientDAO, ProductDAO si OrderDetailDAO), realizarea conexiunii cu baza de date(package connection - clasa COnnectionFactory) si clasa de pornire Start din pachetul start..

În ceea ce privește clasele, cele sase pachete conțin 19 clase : ClientBLL, EmailValidator, OrderDetailBLL, ProductBLL, Validator, ConnectionFactory, AbstractDAO, CLientDAO, OrderDetailDAo, ProductDAO, Client, Order, OrderDetail, Product, CLientView, Mainn, OrderDetailView, ProductView, Start .

Diagrama de pachete:

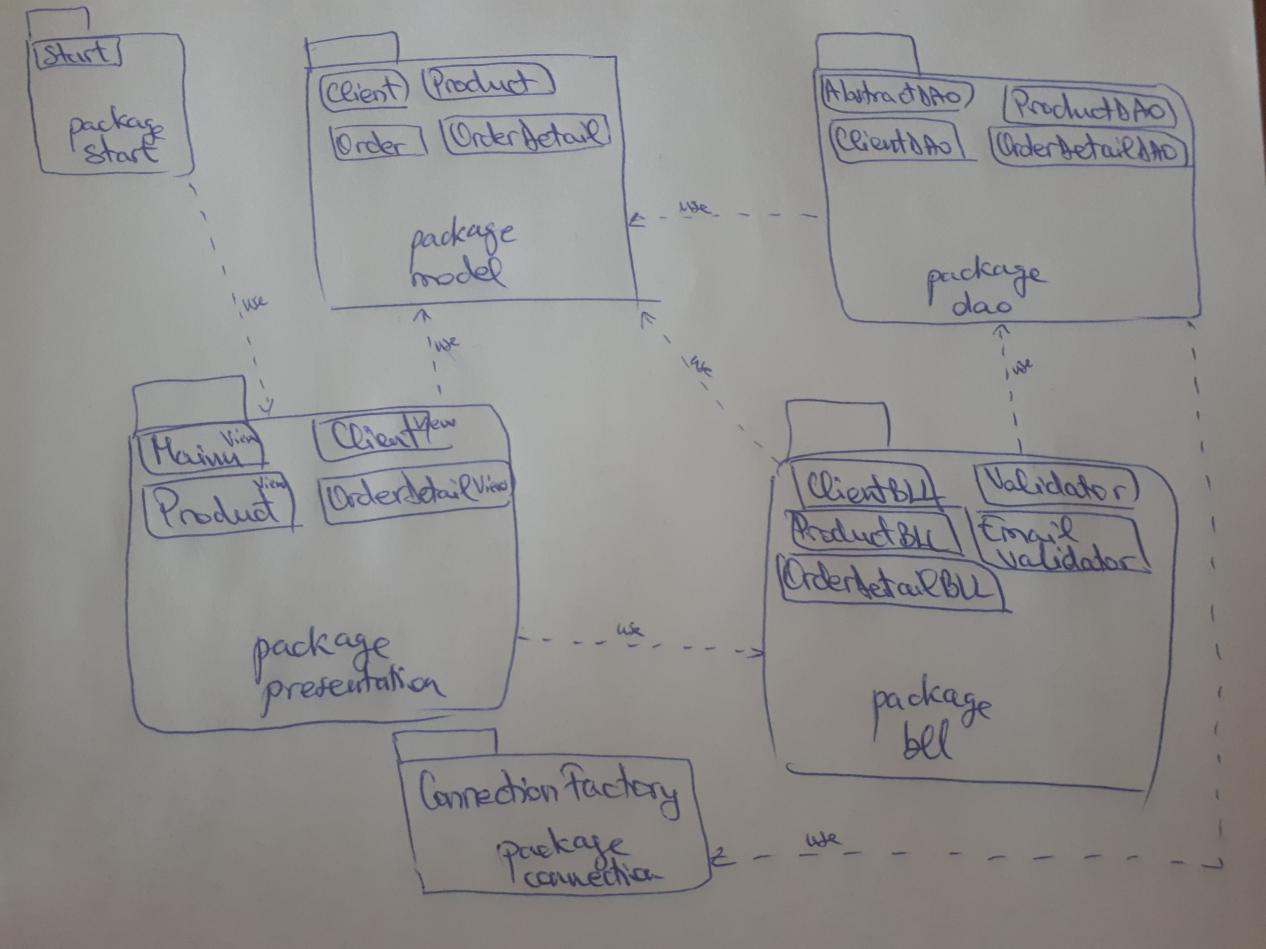
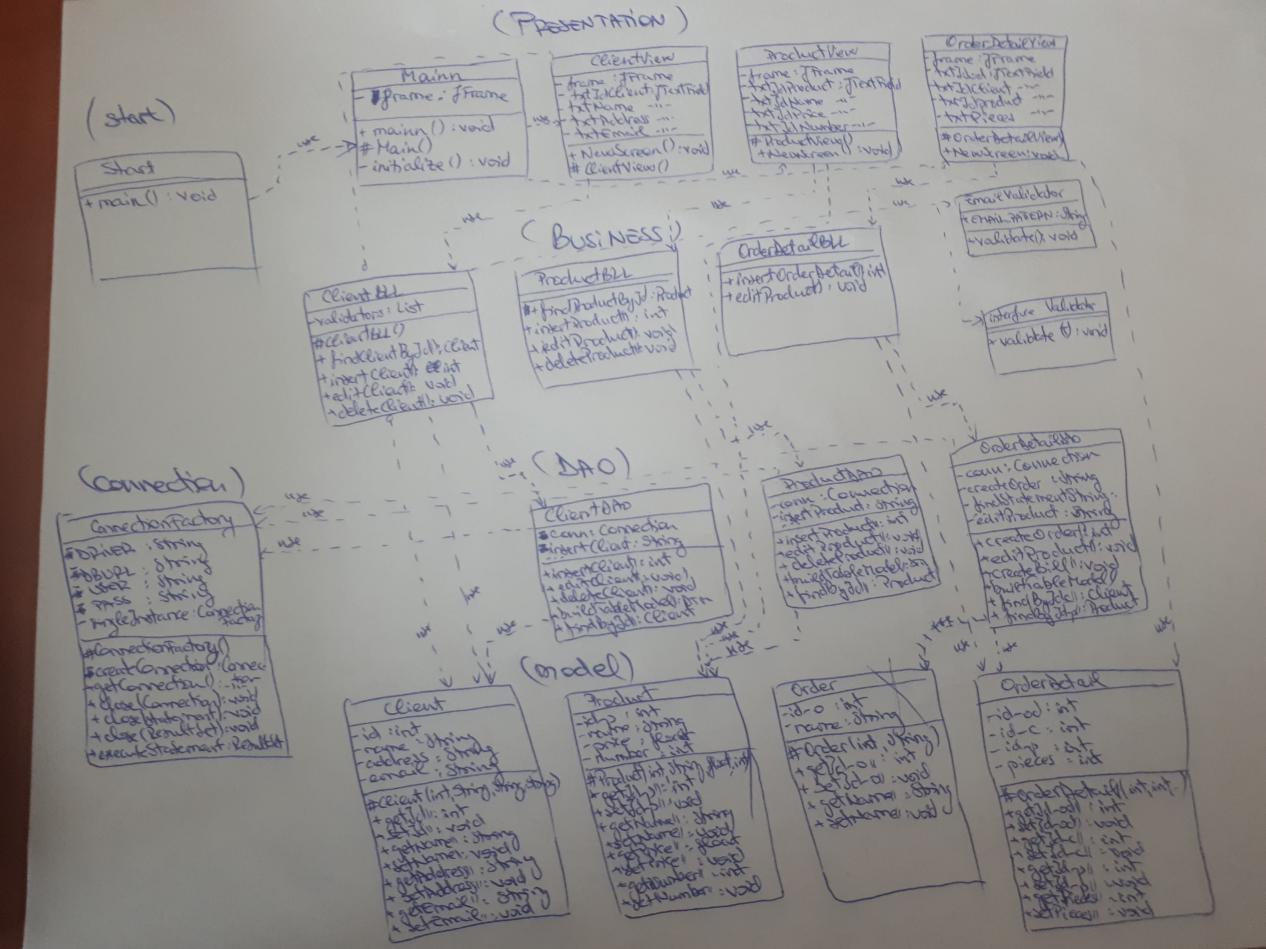


Diagrama de clase:



Interfața utilizator este una simplă, clasică și ușor de utilizat. Compusă dintr-o fereastră în cadrul căreia am pus un panou cu elementele sale ( etichete, butoane, text fields ), din care se pot accesa alte 3 ferestre corespunzatoare operatiunii dorite(cu clientii, cu produsele, cu comenzile), permite o interacțiune facilă a utilizatorului cu programul ce gestioneaza stocul si comenzile corespunzatoare depozitului .

* Implementare

Proiectul este structurat in 19 clase , după cum urmează:

- 4 clase aparținând modelului

- 4 clase aparținând interfeței cu utilizatorul

- o clasă în care am dat ‘start-ul’ aplicatiei

- 5 clase corespunzatoare business-logicu-ului

- 4 clase corespunzatoare implementarii operatiilor

- o clasa aferenta realizarii conexiunii cu baza de date din MySql .

**Clasa Client** descrie clientii din cadrulbazei de date ( care sunt de forma id, nume, adreasa, email ), având acestea 4 ca atribute . Id-ul este definit ca număr intreg , iar numele, adresa si email-ul ca String-uri. Contructorul cu parametri explicit permite construirea unui client cu datele oferite de noi . Am implementat și metode accesoare și mutatoare pentru a putea avea acces laatributele clientului și pentru a-l putea modifica, altfel fiind imposibil, atributele fiind declarate private . Am suprascris metoda toString() în această clasă pentru a putea afișa clientii din tabelul din baza de date cu toate atributele lor).

**Clasa Product** , similara clasei Client, descrie un produs, având ca atribute id-ul acestuia , numele, pretul si numarul de bucati pe stoc . Metoda de afișare este realizata tot prin suprascrierea metodei toString() si in aceasta clasa .

De asemenea, am implementat si aici metode accesoare si mutatoare pentru a avea acces la atribute produselor atunci cand e nevoie .

**Clasa OrderDetail,** similara celor doua clase de mai sus, descrie comenzile realizate la depozitul nostru, inregistrand id-ul comenzii, cel al clientului ce a realizat-o si cel al produsului dorit, precum si numarul de bucati din acesta .

In clasele **ClientDAO si ProductDAO** avem metode ce realizeaza inserarea, editarea, stergerea si vizualizarea clientilor, respectiv a produselor corespunzatoare depozitului. In cadrul acestora, se citesc datele introduse de user in text-fields, se realizeaza conexiunea la baza de date si se executa inserarea, editarea si stergerea sub forma unor query-uri de tipul ‘insert’, ‘delete’ si ‘update’ . Dupa executarea lor, rezultatul de afiseaza intr-un tabel de tipul JTable, folosind metoda buildTableModel care primeste un result set provenit din executarea unei interogari asupra bazei de date si creeaza un tabel de tip default care va fi populat cu datele existente in baza de date si apoi afisat pe ecran . De asemenea, am implementat metoda de cautare a unui client sau a unui produs dupa id-ul sau , fapt realizat tot prin executarea unui query adecvat de forma ‘select \* from table where id=-’ .

Clasa **OrderDetailDAO** are ca metoda principala metoda createOrder ce primeste datele introduse de utilizator in campurile din panoul OrderDetailView si cu ajutorului unui query , adauga o noua linie in tabelul destinat comenzilor si il afiseaza, iar in acelasi timp, decrementeaza cantitatea din produsul comandat cu numarul de bucati comandate si afiseaza tabelul produselor .

Pentru interfața utilizator, am ales împărțirea ei în patru clase și anume : clasa **Mainn** ce prezinta panoul principal din care se poate alege daca se doreste realizarea unei operatii referitoare la clienti sau la produse sau daca se vrea realizarea unei comenzi .

Butoanele Client, Product si Order fac legatura dintre aceasta fereastra principala si cele trei ferestre corespunzatoare fiecarui tip de operatiune ce se poate realiza: cu clientii, cu produsele sau inregistrarea unei comenzi . Fereastra Client prezinta cate un buton pentru fiecare operatie posibila (inserare, editare, stergere, vizualizare), dar si unul ce permite revenirea in fereastra principala( back ). Pe langa acestea, ea prezinta si patru text-field-uri in care utilizatorul poate insera id-ul, numele, adresa si email-ul clientului cu care doreste sa lucreze . Similar acesteia, Fereastra Product prezinta cate un buton pentru fiecare operatie posibila (inserare, editare, stergere, vizualizare), dar si unul ce permite revenirea in fereastra principala( back ). Pe langa acestea, ea prezinta si patru text-field-uri in care utilizatorul poate insera id-ul, numele, pretul si numarul de bucati din stoc ale produsului cu care doreste sa lucreze . Fereastra Order prezinta patru text-field-uri in care se pot insera id-ul comenzii, cel al clientului si cel al produsului, dar si cantitatea dorita din acel produs . Butonul Create order creeaza comanda , adica o insereaza ca linie noua in tabelul comenzilor din baza de date , butonul Bill genereaza o chitanta pentru acea comanda intr-un fisier text, iar butonul Back permite revenirea in meniul principal pentru a putea selecta un alt tip de operatie daca se doresste acest lucru .

Clasa **ConnectionFactor**y, realizata dupa modelul primit in cadrul laboratorului, este destinata realizarii conexiunii cu baza de date a depozitului, din MySQL , conexiune ce va fi folosita apoi de clasele ce implementeaza operatiile ce pot fi executate , pentru a putea folosi si modifica datele din tabele . Are o metoda ce returneaza o instanta a conexiunii create , dar si metode ce inchid atat conexiune, cat si statement-urile folosite si result-set-urile obtinute in urma acestora . Ultima metoda din aceasta clasa este executeStatement ce are rolul de a pune in executie query-ul transmis ca parametru .

Unicul rol al clasei **Start** este cel de a apela in metoda sa main, metoda mainn din clasa Mainn, ce deschide meniul principal, lansand in executie programul prin interactiunea cu utilizatorul prin intermediul intefertei utilizator.

In clasele din pachetul bll, am scris logica operatiilor cerute a fi implementate , a caror implementare propriu-zisa se afla insa in clasele corespondente din pachetul dao .

* Rezultate

Prin testarea programului pentru mai multe variante de intrare , am verificat funcționarea acestuia atât la introducerea unor forme neacceptate de intrare , precum și a celor permise .

Astfel , operatiile se vor executa corect in momentul in care datele introduse in text-field-uri respecta cerintele atribuite lor cum ar fi de exemplu: id-urile la inserare sa nu fie duplicate ale celor deja existente in tabele, id-urile in general sa nu fie nule , sa se respecte formatul intrarii: String pentru String (nume, email etc ), numar intreg pentru numar intreg (id), numar real pentru numar real (in cazul pretului) . In plus id-urile nu pot lipsi sub nicio forma din datele introduse , chiar daca, spre exemplu, la inserare, nu se introduce nimic in campul aferent numelui sau al adresei (fiind vorba de un client e exemplu) .

In cazul epuizarii stocului unui anumit produs dupa repetate comenzi, se va afisa un mesaj ce indica acest fapt, la fel si in cazul unora din situatiile enumerate mai sus . In caz contrar epuizarii stocului, va fi afisat atat tabelul comenzilor ce confirma efectuarea comenzii cerute, cat si tabelul produselor ce atesta faptul ca stocul produsului respectiv a fost decrementat cu cantitatea comandata .

* Concluzii

Implementarea unui program care realizează efectuarea operațiilor legate de managementul unui depozit realizat prin interactiunea cu o baza de date a acestuia prezinta unele dificulati in ceea ce priveste interactiunea cu baza de date si afisarea acesteia in interfata, dar si in privinta structurii pe care a trebuit s-o respectam . Totuși, tema aceasta contribuie semnificativ la exersarea atât a limbajului de programare Java, cât și a conceptelor ce țin de programarea orientată pe obiect .

Ca dezvoltare ulterioară, programului i se pot adăuga și alte funcții mai complexe precum crearea unor comenzi multiple de catre acelasi client si tinerea unei evidente a acestora, dar si oferirea posibilitatii de a rezerva produse, nu doar de a le comanda direct, acestea fiind pastrate in depozit pentru perioada aleasa, insa marcate pentru a fi recunoscute si scoase din stocul principal.

* Bibliografie

-www.coned.utcluj.ro/~salomie/PT\_Lic/

-http://tynerblain.com/blog/2007/04/09/sample-use-case-example/

-http://users.utcluj.ro/~igiosan/teaching\_poo.html

-https://stackoverflow.com/questions/2885173/how-do-i-create-a-file-and-write-to-it-in-java

-https://alvinalexander.com/java/joptionpane-showmessagedialog-examples-1

-https://stackoverflow.com/questions/39272302/getting-data-from-defaulttablemodel-to-a-jtable-in-the-mvc-architecture

-http://www.java-blog.com/mapping-javaobjects-database-reflection-generics

-https://utcn\_dsrl@bitbucket.org/utcn\_dsrl/pt-layered-architecture.git