# DOCUMENTATIE TEMA 3: ORDER MANAGEMENT

**AGHENITEI BIANCA-IOANA**

CALCULATOARE ROMANA GRUPA 30225



# Obiectivul temei

Luați în considerare o aplicație OrderManagement pentru procesarea comenzilor clienților pentru un depozit.

Bazele de date relaționale sunt utilizate pentru a stoca produsele, clienții și comenzile. În plus,

aplicația ar trebui să fie structurată în pachete folosind o arhitectură stratificată prezentată în suport

material (Assignment\_3\_Indications.pdf) și ar trebui să utilizeze (minim) următoarele clase:

• Clasele de modele - modelele de date ale aplicației

• clase de logică de afaceri - implementează logica aplicației

• clase de prezentare - implementează intrarea / ieșirea utilizatorului

• Clasele de acces la date - implementează accesul la baza de date

Aplicația ar trebui să permită procesarea comenzilor dintr-un fișier text dat ca argument, efectuați

operațiunile solicitate, salvați datele din baza de date și generați rapoarte în format pdf. Alte clase

și pachete pot fi adăugate pentru a implementa funcționalitatea completă a aplicației.

Comenzi valide:

Insert client: Ion Popescu, Bucuresti

Insert client: Luca George, Bucuresti

Report client

Insert client: Sandu Vasile, Cluj-Napoca

Report client

Delete client: Ion Popescu, Bucuresti

Report client

Insert product: apple, 20, 1

Insert product: peach, 50, 2

Insert product: apple, 20, 1

Report product

Delete Product: peach

Insert product: orange, 40, 1.5

Insert product: lemon, 70, 2

Report product

Order: Luca George, apple, 5

Order: Luca George, lemon, 5

Order: Sandu Vasile, apple, 100

Report client

Report order

Report product

**Obiective secundare:**

* 1. Analzia problemei si dezvoltarea de scenarii de utilizare
  2. Alegerea structurilor de date
  3. Impartirea pe pachete si clase, alegerea strcuturii proiectului
  4. Generarea unui fisier .jar
  5. Generarea fisierelor de PDF si verificarea corectitudinii lor
  6. Generarea unui Dump file

# Analiza problemei, modelare, scenarii, cazuri de utilizare

Acest proiect va o aplicatie de management al comenzilor unui depozit. Utilizatorul va rula in linia de comanda fiserul .jar , trimitand ca argumente calea fisierului de intrare. Programul va procesa datele de intrare din fisier, va face modificarile corespunzatoare in baza de date si va genera raporte in format PDF, in functie de cerinta.

**Cazuri de utilizare:**

Daca fisierul de intrare este corect, programul isi va continua executia dupa cum urmeaza:

-se analizeaza secvential comezile din fiserul de intrare si se incearca executia lor. In cazul in care se precizeaza inserarea unui produs sau client care exista deja in baza de date, se arunca o exceptie, nu se intampla modificari la aceasta si se continua cu urmatoarea comanda. In cazul in care se incearca operatii pe tabele care nu respecta structura ei, se arunca exceptii si nu se executa modificari. Baza de date se va modifica corespunzator la orice comanda din fisierul exemplu, comenzi considerate corecte.

-se vor genera rapoarte noi,in format pdf, la curent cu baza de date, la fiecare comanda de tip report;

-de va genera un fisier pdf de eroare doar cand se doreste cumpararea unui produs care nu este pe stoc in cantitatea dorita.

In cazul in care calea fisierului de intrare este invalida, programul va arunca o exceptie care va fi semnalata in terminal. De asemenea, el nu va rula daca argumentele nu sunt trimise corect. Formatul apelului corect este:

*java -jar PT2020\_Group\_FirstName\_LastName\_Assignment\_2.jar command.txt*

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

# 

Baza de date a fost creata cu toolul drag and drop din MySql. Pe langa cele 3 tabele oarecum evidente, Client, Product si Order am optat pentru folosirea inca unei tabele, Order\_item, care rezolva problema many-to-many dintre Order si Product si permite retinerea cantitatii comandate la un order.

Structura relationala ne asigura ca programul nostru nu va putea face modificari nefiresti la baz de date. De exemplu, nu se va putea adauga un order pentru un client care nu exista, fiindca vom primi o eroare la foregn key. De asemenea, nu se va putea adauga un order pentru un produs care nu exista din acelasi motiv. Cu toate acestea, problema comandarii unui produs care nu este in canititate suficienta pe stoc va fi rezolvata in cod Java.

Se pune problema stergerii unui client din baza de date. MySql nu ne va lasa sa facem acest lucru daca inca exista orderuri pe numele acelui client. De aceea am gasi solutia ca orice operatie de delete sa fie preecedata de un delete in toate tabelele referite.

*“Architecture is kind of an overloaded term, so we should probably dig deeper into what the term really means in the context of layers. The main idea behind Layered Architecture is a separation of concerns – as we said already, we want to avoid mixing domain or database code with the UI stuff, etc. The actual idea of separating a project into layers suggests that this separation of concerns should be achieved by source code organization. This means that apart from some guidance to what concerns we should separate, the Layered Architecture tells us nothing else about the design and implementation of the project. This implies that we should complement it with some other architectural processes, such as some upfront design, daily design sessions, or even full-blown*[*Domain-Driven Design*](http://amzn.to/2lkEVn2)*. Whichever option we choose doesn’t matter, at least for the sake of layering, but we need to remember:****Layered Architecture gives us nothing apart from a guideline on how to organize the source code****.”*

Pentru impartirea pe pachete, mi-a venit natural sa folosesc o arhitectura de tip Layered, dupa cum urmeaza:

**Pachetul Connection**

Contine numai Clasa ConnectionFactory care se ocupa cu realizarea conxiunii cu baza de date.

**Pachetul DAO**

Conține clase care conțin interogări pentru baza de date. Se ocupa cu accessul la date. (DAO=data acces objects). Fiecare Obiect va acea clasa lui DAO specifica si toate vor mosteni din clasa generala, AbstractDAO. Tehnica reflectiei va fi discutata in continuare.

**Pachetul Model**

Contine clasele specifice fiecarui tabel din baza de date. Este important ca acestea sa aiba numele tabelului si doar atributele acestora, in ordine, pentru a usura lucrul, mai ales ca folosim tehnica reflectiei.

**Pachetul presentation**

Contine metode care realizeaza interfata cu utilizatorul. In cazul de fata, avem clasa FileReader care citeste,cate o linie, datele de intrare din fisier. Acestea sunt trimise la clasa Parser, care le traduce si apeleaza metodele DAO specifice. PDFGenerator se ocupa de generarea fisierelor Pdf de raport, cat si a celor de eroare, Si aceste metode sunt apelate tot din Parser.

Necesitatea inmplementarii de functionalitati de INSERT, SELECT si DELETE asupra fiecarei tabele sugereaza folosirea tehnicii Reflection, pentru a evita scrierea a foarte mult cod asemanator. Asadar, clasa AbstractDAO va contine toate metodele generice de constructie si executie a query-urilor, iar metodele vor fi accesate direct prin mostenire. (clasele ClientDAO, ProductDAO, etc. vor extinde clasa AbstractDAO si vor implementa doar metode specifice).

*The name reflection is used to describe code which is able to inspect other code in the same system (or itself).*

*For example, say you have an object of an unknown type in Java, and you would like to call a 'doSomething'* method on it if one *exists. Java's static typing system isn't really designed to support this unless the object conforms to a known interface, but using reflection, your code can look at the object and find out if it has a method called 'doSomething' and then call it if you want to.*

# Rezultate

Rezultatele sunt cele asteptate, atat pentru fisierul dat ca exemplu, cat si pe alte fisiere.

Asadar, comenzile din fisierul de intrare sunt parsate si executate secvential. Se realizeaza modificarile in baza de date in functie de comanda, rapoartele generate fiind la curent cu modificarile. La fiecare apel de raport se genereaza un fisier .pdf de tipul reportx.pdf, unde x este numarul de rand al raportului. In caz de eroare(de exemplu la solicitarea unui produs care nu este pe stoc in cantitatea dorita, se genereaza un fisier .pdf de eroare care precizeaza care este produsul care s-a dorit si faptul ca este „low on stock”.

De asemenea, am urmarit schimbarile in baza de date si ele se intampla ca atare. Programul nu ne lasa sa introducem date care nu respecta relatiile dintre tabele.

# Concluzii

In urma finalizarii acestui proiect consider ca am fixat unele dintre notiunile programarii orientate pe obiect, m-am familiarizat cu lucrul cu obiecte si pachete si am fost introdusa unor notiuni cu totul noi pentru mine, precum arhitectura layered,tehnica reflection, generarea unui fisier .jar si rularea lui.

Desi la crearea claselor si a metodelor nu am intalnit probleme, lucrul cu tehnica reflection Java mi-a dat de furca.. Din fericire am avut prezentarea powerpoint de pe site-ul domnului profesor pentru informatii necesare si indicii referitoare la crearea claselor, astefel ca am reusit sa rezolv treptat toate problemele si nelamuririle pe care le aveam initial.

Obiectul sisteme de operare s-a dovedit totusi de folos spre final, cand am generat fiiserul .jar si am modificat programul sa citeasca argumentele la apelul din linia de comanda.

Din pacate timpul nu mi-a permis sa adaug mai mult proiectului, insa m-am gandit la mai multe posibilitati de imbunatatire ulterioara a lui:

* folosirea unui regex pentru a citi datele de intrare;
* realizarea unei interfete grafice;

- tratarea posibilitatii in care un client poate avea mai multe orders

- tratarea explicita a cazurilor in care datele de intrare sunt incorecte si notificarea utilizatorului;

-tratarea mai multor exceptii, atat la procesarea comenzilor, cat si la introducerea datelor in tabel;

-imbunatatiri la interfata grafica: imagini, culori;

-adaugarea unei melodii pe fundal;

In concluzie, pot spune ca m-am distrat realizand acest proiect si sunt multumita cu rezultatul final. Sunt fericita ca am fixat niste notiuni importante de OOP, care ma vor ajuta la dezvoltarea in viitor a unor proiecte mai complexe. Tehnica reflectiei mi se pare foarte facila fiindca te scuteste de scrisul multor linii de cod! Cu siguranta voi gravita spre aceasta solutie in viitor.