****

**DOCUMENTAȚIE TEMA 3**

**ORDER MANAGEMENT**

**Trifu Diana-Maria**

**Grupa 30223**

**Profesor Laborator: Dorin Moldovan**

Cuprins:

**1. Cerințe funcționale …………………………………………………………………….……. 3**

**2. Obiectivul temei ………………………………………………………………………..……. ..3**

**2.1 Obiective principale …………………………………………………...……...…...….…3**

**2.2 Obiective secundare ………….……………………………………………………....… 4**

**3. Analiza problemei ……………………………………………….………………………....… 5**

**3.1 Scenarii …………………………………………………………………………..…..…6**

**3.2 Modelare...………………………………………………………………………..….....6**

**4. Proiectarea ……………….………………………………………………………….………….. 7**

**4.1 Structuri de date folosite ………………………………………….…………..........…7**

**4.2 Diagrama de clase....………….………………………………………………………….7**

**5.Implementare …….…………………………………………………………...……………….....8**

**6. Rezultate …………..………………………………………………………...………………….....10**

**7.Conluzii ……………………………………………………………………...……………..………12**

**8.Bibliografie …………………………………………………...………………………...………....12**

1. **Cerințe funcționale**

**Propuneți, proiectați și implementați o aplicație pentru procesarea comenzilor unor clienți pentru un depozit. Se folosește o bază de date relațională pentru stocarea produselor, a clinenților și a comenzilor. În plus, aplicația trebuie structurată în mai multe pachete, ținând cont de arhitectura stratificată prezentată în materialul support(Assignment\_3\_Indications.pdf) și ar trebui să se utilizeze(minim) următoarea împărțire a claselor: Clasele model(modelele aplicației), Clasele cu logica aplicației, Clasele de prezentare(datele de intrare/ieșire pentru utilizator) și Clasele ce asigura accesul la baza de date.**

**Aplicația trebuie: să permită procesarea comenzilor dintr-un fișier text dat ca argument, să execute operațiile cerute, să salveze datele în baza de date și să genereze raporturi în format pdf. Alte clase și pachete pot fi adăugate pentru a putea implementa funcționalitatea completă a aplicației.**

**Considerații de implementare:**

* **Folosirea limbajului de programare Java**
* **Implementarea claselor cu maxim 300 de linii de cod**
* **Implementarea metodelor cu maxim 30 de linii de cod**
* **Folosirea javadoc pentru documentarea claselor și generarea fișierelor JavaDoc corespunzătoare**
* **Folosirea bazei de date relațională pentru stocare(minim 3 tabele: produs, client, comandă)**
* **Pentru comanda ORDER: se va genera o factură în format pdf, stocul produsului respective va fi decrementat, iar în cazul în care stocul este insufficient se va genera un document pdf în care va fi scris un mesaj pentru informarea clientului asupra acestui fapt**
* **Raporturile vor fi generate sub forma unor documente pdf și vor conține datele din tabelul corespunzător afișate într-un tabel**

1. **Obiectivul temei**
   1. **Obiectivele principale**

**Obiectivul principal al proiectului este de a creea o aplicație care să pună la dispoziția utilizatorului un sistem de management al unui deposit de fructe. Această aplicație va permite utilizatorului să introducă prin intermediul produse și client, dar va oferi totodată și posibilitatea de a înregistra și procesa comenzi. În cazul unei comenzi valide, aplicația va genera și o factură pentru clientul respectiv, care va conține informații despre comandă și care la finalul ei va prezenta și suma totală pe care acesta o are de achitat. În cazul unei comenzi invalide(nu există suficiente produse în stocul depozitului), se va genera un document pdf prin care clientul va fi înștiințat asupra acestui lucru. În orice moment utilizatorul are posibilitatea de a verifica baza de date existent cu ajutorul comenzii ” Report name\_of\_the\_table”, care va genera un document pdf care va conține tabelul respectiv.**

* 1. **Obiectivele secundare**

**Obiectivele secundare ale temei reprezintă pașii careau fost urmați pentru atingerea obiectivului final.**

1. **JDBC**

**JDBC (Java Database Connectivity) este o interfaţă standard de programare (API- Application Programming Interface) a accesului la baze de date fiind constituită dintr-un set de clase și interfeţe scrise în Java, ce oferă mecanisme pentru dezvoltarea aplicaţiilor Java ce au nevoie de persistarea datelor în baze de date . Folosind facilitățile oferite de JDBC este usor să transmitem secvențe SQL către baze de date relaționale (MYSQL ,MS-SQL,Oracle) . Pachetele Java care oferă suport pentru lucrul cu baze de date sunt incluse în java.sql , acesta reprezintă nucleul tehnologiei JDBC oferind posibilitatea lucrului cu instrucțiuni SQL și procesarea rezultatelor obținute în urma interogărilor. Complementar, javax.sql oferă capabilități diverse,evoluate pe parte de server, cum ar fi tranzacții distribuite și conexiuni multiple, în condițiile în care într-o arhitectură client-server baza de date se poate afla pe aceeași mașină sau pe o altă mașină cu care clientul ce solicită acces este conectat dintr-un Intranet sau Internet. Utilizând API-ul JDBC, aplicațiile pot fi programate în mod independent de sistemul de gestiune a bazei de date (SGBD), respectiv , de baza de date relațională, iar apelurile SQL către baza de date vor fi preluate și transformate de către diverse drivere specifice pentru fiecare SGBD,apoi trimise, bazei de date. Specificația JDBC pune la dispoziția producătorilor de drivere reguli de construire a driverelor, astfel, în timp, atât specificația cât și driverele au cunoscut o evoluție continuă, sub aspectul funcționalităților oferite dezvoltatorului de aplicații. Pentru a putea dezvolta în mod corect și optim aplicația client Java, este necesară verificarea caracteristicilor/funcționalităților oferite de driverul ales.**

1. **Accesul la baza de date**

**Accesarea unei baze de date folosind mecanismul oferit de JDBC este simplă şi implică :**

**1. Obţinerea unui obiect de tip Connection ce încapsulează conexiunea la baza de date (în acest pas se realizează conexiunea la baza de date).**

**2. Obţinerea unui obiect Statement dintr-un obiect de tip Connection. Acest obiect este folosit pentru a transmite spre execuţie comenzi SQL către baza de date. Prin intermediul acestui obiect sunt efectuate operaţii de interogare şi modificare a bazei de date.**

**3. Obţinerea unui obiect ResultSet dintr-un obiect Statement. Obiectul ResultSet încapsulează rezultatele operaţiilor de interogare a bazei de date.**

**Procesul de conectare la o bază de date implică înregistrarea unui driver corespunzător și realizarea unei conexiuni cu baza de date. O conexiune (sesiune) la o bază de date reprezintă un context prin care sunt trimise secvențe SQL către baza de date și sunt primite rezultate la nivel de aplicație client. API-ul JDBC furnizează acces orientat obiect la bazele de date, prin definirea de clase și interfețe ce 2 modelează concepte abstracte specifice tehnologiei, semnificația celor mai importante concepte JDBC fiind următoarea:**

**Driver - clasa Driver și Driver Manager**

**Conexiunea la baza de date – clasa Connection**

**Interogări SQL - clasele Statement, Prepared Statement, Callable Statement**

**Mulțimi rezultat al execuției - clasa Result Set**

**Specificarea accesului la o bază de date creată și găzduită de un SGBD , dintr-o aplicație client Java presupune mai multe etape, ce trebuie parcurse într-o secvențiere strictă și anume: înregistrarea unui driver ce va fi utilizat pentru translatarea către baza de date a frazelor SQL , respectiv stabilirea conexiunii cu baza de date , corect specificată prin intermediul unui string de conexiune specific fiecărei baze de date. Aceste etape vor fi descrise detaliat în continuare.**

1. **Tehnici de reflexie**

**Reflexia este un mecanism pus la dispoziție de un limbaj de programare prin care un program scris țn respectivul limbaj poate afla informații la runtime despre entitățile sale componente și prin care își poate modifica comportamentul. Reflexia este comună în special în rândul limbajelor interpretate, precum python, ruby, php, javascript, Lisp, Scheme, dar și în rândul celor compilate precum Java sau C#. Așa cum s-a menționat reflexia poate fi folosită atât pentru a obține informații despre obiectele din program la runtime cât și pentru a modifica comportamentul programului. Spre exemplu, datele pe care le putem obține în cazul limbajelor orientate pe obiect reprezintă informații despre structura clasei unui obiect.**

1. **Arhitectură structurată**

****

**Aplicația trebuie împărțită în 4 straturi diferite precum în figura din dreapta. Primul strat va conține pachetele în care vor fi clasele ce vor asigura interacțiunea cu utilizatorul. În cel de-al doilea strat se vor încadra pachetele a căror clase încapsulează logica aplicației. În unele cazuri cel de-al treilea strat este combinat împreună cu cel de-al doilea în unul singur, care poartă denumirea de Business Layer, în special atunci când persistența logicii (ex. SQL) este încorporată în componentele logicii aplicației. Cel de-al patrulea strat conține pachetele a căror clase fac posibilă realizarea conexiunii la baza de date relațională și care permit interacțiunea cu tabelele acesteia.**

1. **Analiza Problemei**

**Programarea orientate pe obiect oferă posibilitatea începerii dezvoltării unui proiect fără nevoia efectivă de a cunoaște de la început toate amănuntele funcționării efective a aplicației care urmează să fie dezvoltată. Această strategie folosită este cunoscută sub numele de “Top Down.” O abordare de acest tip este esențială într-un proiect de acest tip, deoarece îți oferă o perspectivă mult mai bună asupra sub-sistemelor care alcătuiesc întregul. O astfel de metodă oferă posibilitatea proiectantului să își faca o imagine de ansamblu asupra sistemului, fără a fi nevoit să detalize fiecare nivel al subsistemelor.**

* 1. **Scenarii**

**Descrierea funcționării aplicației și a modului de introducere a datelor în fișierul text de intrare:**

**Exemplu de date de intrare pentru fișierul text:**

**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, 20**

**Report product**

**Order: Luca George, apple, 5**

**Order: Luca George, lemon, 5**

**Order: Sandu Vasile, apple, 100**

**Report client**

**Report order**

**Report product**

**Introducerea sub o altă forma a datelor de intrare vor duce la eșecul simulării, astfel încât citirea datelor de intrare din fișier nu va mai fi realizată corespunzător.**

**Posibilele scenarii de eșec care pot apărea: introducerea sub o formă greșită a datelor în fișier, numele comenzilor sun greșite sau modificate), datele pentru o anumită comadă sunt insuficiente(de exemplu, pentru introducerea unui nou produs nu este specificată cantitatea). Execuția programului se va termina dup ace toate comenzile din fișierul text de intrare au fost executate. Datele de ieșire, disponibile utilizatorului, vor fi reprezentate de câteva documente pdf: raporte ale tabelelor din baza de date, facturi ale clienților în funcție de comenzile acestora sau documente de înștiințare a clienților asupra stocului insuficient pentru realizarea comenzilor, dacă este cazul.**

**3.2 Modelare**

**Utilizitarolu va putea accesa aplicația fie prin intermediul Eclipse, fie prin intermediul fișierului cu extensia .jar, care ofera o compilare imediată a programului printr-un simplu click. Pentru a modifica datele de intrare se va accesa fișierul text dat ca parametru în linia de comandă/cod. După o scurtă perioada de timp, comenzile din fișierul de intrare vor fi executate, iar documentele pdf generate în urma comenzilor vor fi disponibile.**

1. **Proiectare**

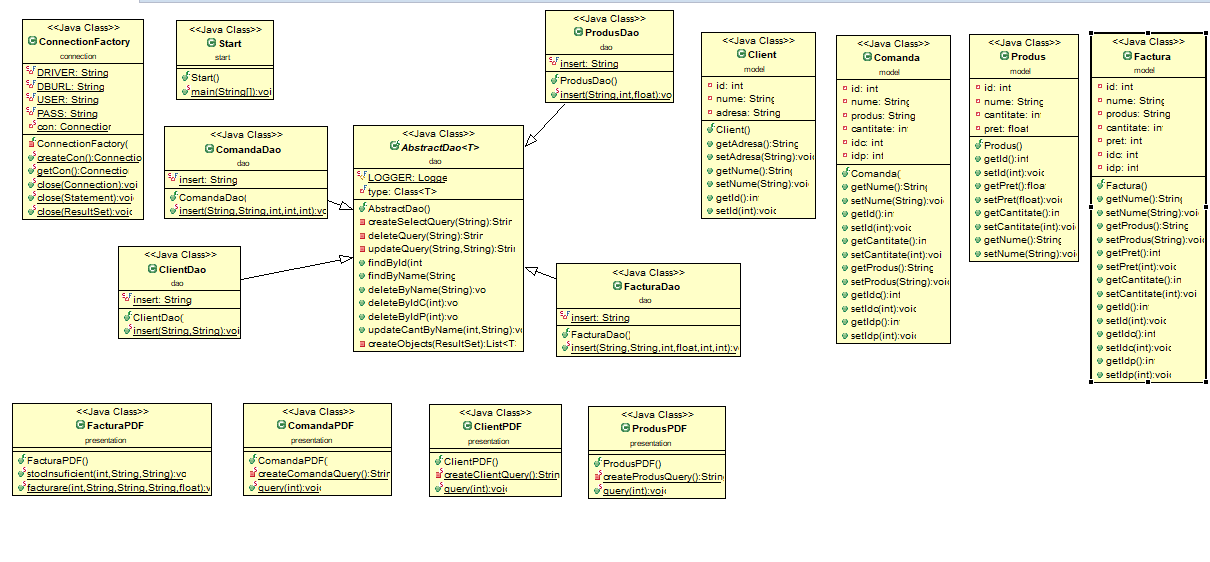
**4.1. Structuri de date și clasele folosite**

**Ca și clase principale folosite clasele din pachetele: DAO, CONNECTION și START. Clasa ConnectionFactory, din pachetul CONNECTION, este cea care face posibila conexiunea cu baza de date relațională. Clasa Start, din pachetul START, reprezintă logica propriu-zisă a aplicației. În interiorul acestei clase se află metoda main care reunește toate celelate clase create pentru a întregi funcționalitatea aplicației. În această metodă main are loc citirea din fișier și apelarea metodelor necesare pentru executarea comenzilor preluate din fișierul de intrare. Clasele din pachetul DAO sunt: AbstractDao, ProdusDao, FacturaDao, ComandaDao și ClientDao. Clasa AbstractDao conține metodele care definesc operațiile comune de pentru accesarea tabelelor din baza de date(Interogare, Ștergere, Modificare). Aceste operații sunt definite pe un tip genric <T>, unde T poate fi orice clasa din pachetul MODEL, care reprezint tabelele din baza de date, având exact același nume și aceleași variabile instanță, dar și aceleași tipuri de date precum câmpurile tabelelor. Celelate 4 clase din acest pachet moștenesc clasa AbstractDao, deci implicit metodele acesteia. În fiecare dintre celelalte clase există câte o metodă particulară pentru fiecare dintre cele 4 tabele corespunzătoare din baza de date relațională, și anume: insert. Fiecare dintre aceste 4 metode reprezintă oprația de adăugare a unei noi înregistrări în fiecare dintre cele 4 tabele: Client, Comanda, Produs și Factura.**

**Ca și structuri de date folosite am ales ArrayList-urile. ArrayListurile reprizintă o structură de date ușor de utilizat datorită numeroaselor metode care sunt deja gata implementate, ceea ce oferă o mobilitate foarte mare și ușurință în lucrul cu acestea.**

**4.2. Diagrama de clase**

**Unified Modeling Language (prescurtat UML) este un limbaj standard pentru descrierea de modele și specificații software. Diagrama de clase UML este folosită pentru reprezentarea vizuală a claselor și a interdependențelor, taxionomiei și a relațiilor de multiplicitate dintre ele. Diagramele de clasă sunt folosite și pentru reprezentarea concretă a unor instanțe de clasă, așadar obiecte și a legăturilor concrete dintre acestea.**

****

1. **Implementare**

**Clasa ConnectionFactory – această clasă face parte din pachetul ‘Connection’ și este clasa care face posibilă interacțiunea aplicației cu baza de date relațională, deoarece metodele din cadrul acesteia asigură conexiunea. Conexiunea la baza de date este plasată într-un obiect de tip Singleton. Celelalte metode ale clasei sunte pentru crearea conexiunii, pentru obținerea unei conexiuni active și pentru închiderea conexiunii, a unui Statement sau a unui ResultSet.**

**Clasa AbstractDao – această clasă face parte din pachetul ‘Dao’ și este clasa în care sunt definite prin intermediul unor metode operațiile comune care permit accesul la baza de date. Aceste metode sunt definite pe tipul specific generic <T>, unde T poate fi orice clasa din pachetul MODEL, care reprezint tabelele din baza de date, având exact același nume și aceleași variabile instanță, dar și aceleași tipuri de date precum câmpurile tabelelor. Metodele aestei clase reprezintă diverse: interogări ce pot fi efectuate pe mai multe dintre tabele(findById, findByName), ștergeri(deleteByIdP, deleteByIdC, deleteByName) și actualizări (updateQuery). Metodele *createSelectQuery, deleteQuery și updateQuery* contribuie la construirea comenzii în limbaj sql, în funcție de parametrul rimit(de obicei condiția de pe clauza WHERE). O altă metodă importantă a acestei clase este metoda *createObjects* care preia rezultatul unei interogări din tabel și creează un obiect de tipul respectiv cu câmpurile obțiunute în urma interogării executate.**

**Clasa ClientDao – această clasă face parte din pachetul ‘Dao’ și moștenește o altă clasă din cadrul aceluiași pachet: *AbstractDao.* Obiectele de acest timp vor putea beneficia de toate metodele moștenite de la clasa *AbstractDao* pentru a putea accesa tabela *Client* din baza de date. În această clasă este implementată o singură metodă *insert* pentru a face posibila adăugarea de noi înregistrări în această tabelă. Parametrii funcției sunt: numele clientului și adresa acestuia, iar coloana id din baza de date se va auto-completa.**

**Clasa ComandaDao – această clasă face parte din pachetul ‘Dao’ și moștenește o altă clasă din cadrul aceluiași pachet: *AbstractDao.* Obiectele de acest timp vor putea beneficia de toate metodele moștenite de la clasa *AbstractDao* pentru a putea accesa tabela *Comanda*  din baza de date. În această clasă este implementată o singură metodă *insert* pentru a face posibila adăugarea de noi înregistrări în această tabelă. Parametrii funcției sunt: numele clientului , numele produsului comandat, cantitatea de produs comandată, id-ul clientului care a efctuat comanda și id-ul produsului comandat, iar coloana id din baza de date se va auto-completa.**

**Clasa FacturaDao – această clasă face parte din pachetul ‘Dao’ și moștenește o altă clasă din cadrul aceluiași pachet: *AbstractDao.* Obiectele de acest timp vor putea beneficia de toate metodele moștenite de la clasa *AbstractDao* pentru a putea accesa tabela *Factura*  din baza de date. În această clasă este implementată o singură metodă *insert* pentru a face posibila adăugarea de noi înregistrări în această tabelă. Parametrii funcției sunt: numele clientului , numele produsului comandat, cantitatea de produs comandată, prețul total, id-ul clientului care a efctuat comanda și id-ul produsului comandat, iar coloana id din baza de date se va auto-completa.**

**Clasa ProdusDao – această clasă face parte din pachetul ‘Dao’ și moștenește o altă clasă din cadrul aceluiași pachet: *AbstractDao.* Obiectele de acest timp vor putea beneficia de toate metodele moștenite de la clasa *AbstractDao* pentru a putea accesa tabela *Produs*  din baza de date. În această clasă este implementată o singură metodă *insert* pentru a face posibila adăugarea de noi înregistrări în această tabelă. Parametrii funcției sunt: numele produsului, cantitatea de produs introdusă în stoc, prețul unei bucăți, iar coloana id din baza de date se va auto-completa.**

**Clasa Client – această clasă face parte din pachetul ‘Model’. Această clasă a fost create special pentru a face posibilă extragerea datelor din baza de date. Această clasă are exact același nume precum tabelul căreia îi corespunde, iar câmpurile au exact același tip și același nume cu coloanele tabelului. Metodele acestei clase constau în: gettere și settere.**

**Clasa Comanda – această clasă face parte din pachetul ‘Model’. Această clasă a fost creată special pentru a face posibilă extragerea datelor din baza de date. Această clasă are exact același nume precum tabelul căreia îi corespunde, iar câmpurile au exact același tip și același nume cu coloanele tabelului. Metodele acestei clase constau în: gettere și settere.**

**Clasa Factura – această clasă face parte din pachetul ‘Model’. Această clasă a fost create special pentru a face posibilă extragerea datelor din baza de date. Această clasă are exact același nume precum tabelul căreia îi corespunde, iar câmpurile au exact același tip și același nume cu coloanele tabelului. Metodele acestei clase constau în: gettere și settere.**

**Clasa Produs – această clasă face parte din pachetul ‘Model’. Această clasă a fost create special pentru a face posibilă extragerea datelor din baza de date. Această clasă are exact același nume precum tabelul căreia îi corespunde, iar câmpurile au exact același tip și același nume cu coloanele tabelului. Metodele acestei clase constau în: gettere și settere.**

**Clasa ClientPDF – această clasă face parte din pachetul ‘Presentation’. În acestă clasă există 2 metode care fac posibilă generarea unui raport a tabelei *Client* din baza de date: *createClientQuery (*folosită pentru creearea comenzii de interogare a tabelei*) și query(*pentru parcurgerea rezultatului interogării și scrierea datelor într-un nou tabel, într-un document pdf*).***

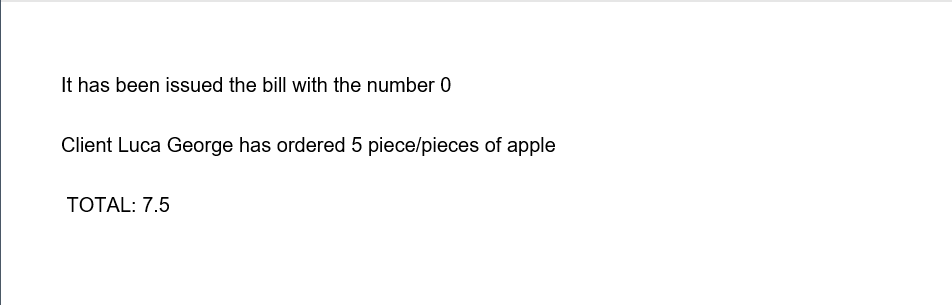
**Clasa ComandaPDF – această clasă face parte din pachetul ‘Presentation’. În acestă clasă există 2 metode care fac posibilă generarea unui raport a tabelei C*omanda* din baza de date: *createComandaQuery (*folosită pentru creearea comenzii de interogare a tabelei*) și query(*pentru parcurgerea rezultatului interogării și scrierea datelor într-un nou tabel, într-un document pdf*).***

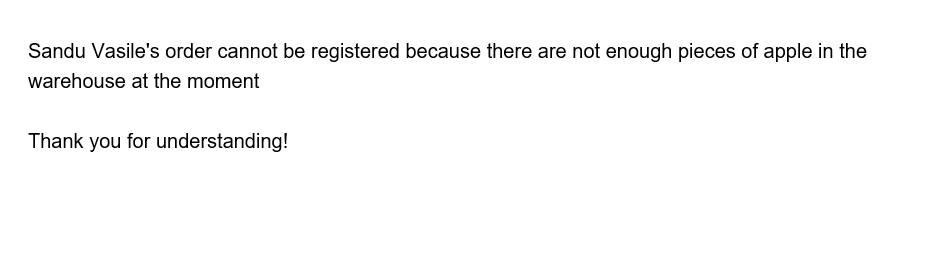
**Clasa ProdusPDF – această clasă face parte din pachetul ‘Presentation’. În acestă clasă există 2 metode care fac posibilă generarea unui raport a tabelei *Produs* din baza de date: *createProdusQuery (*folosită pentru creearea comenzii de interogare a tabelei*) și query(*pentru parcurgerea rezultatului interogării și scrierea datelor într-un nou tabel, într-un document pdf*).***

**Clasa FacturaPDF – această clasă face parte din pachetul ‘Presentation’. În acestă clasă există 2 metode: *facturare și stocInsuficient.* Metoda *facturare* face posibilă generarea unui document pdf, care va reprezenta factura corespunzătoare comenzii effectuate de client și va conține detalii despre comandă și prețul total al comenzii. Cea de-a doua metodă, *stocInsuficient,* va face posibilă generarea unui document pdf prin care clientul va fi înștiințat în cazul în care comanda lui nu poate fi procesată, deoarece cantitatea cerută nu este disponibilă la cael moment în stocul depozitului.**

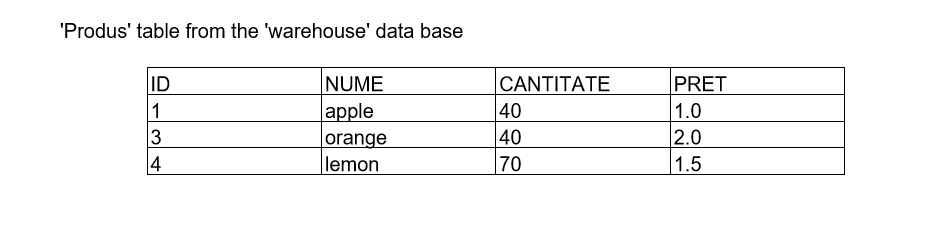
**Clasa Start - această clasă face parte din pachetul ‘Start’. Această clasă conține o singură metodă: main. Aceasta este clasa care reunește toate celelalte clase, formând logica aplicației. Această meodă primește ca și argument un fișier text. Se citesc datele din acest fișier, care sunt reprezentate de comenzile ce trebuie efectuate asupra bazei de date, apoi se identifică comanda și se apelează metodele corespunzătoare pentru efctuarea instrucțiunii respective. La sfarșitul simuării aplicaiei utilizatorul va avea la dispoziție documentele pdf generate în urma executării comenzilor din fișierul text. În cazul unei comenzi de ștergere a unui client sau a unui produs din stoc, se vor șterge și comenzile corespunzătoare clientului/produsului respectiv și, implicit, si facturile.**

1. **Rezultate**

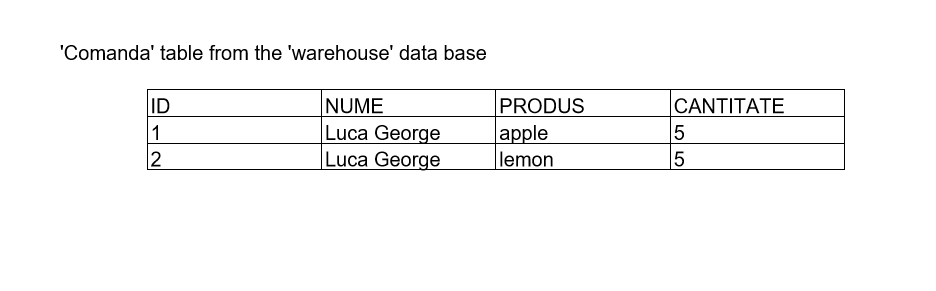
**-Exemplu factură**

**-Exemplu înștiințare pentru stoc insuficient**

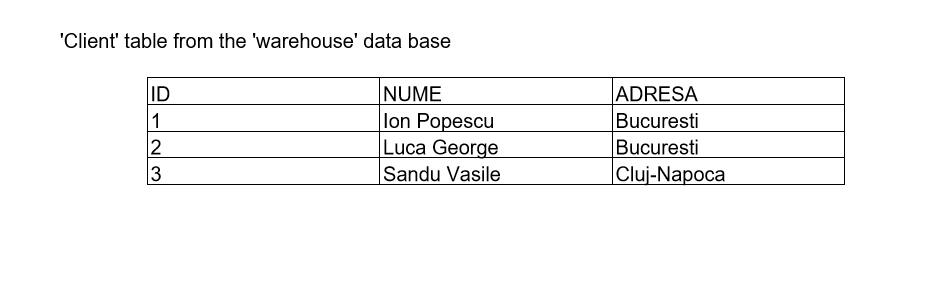
**-Exemplu raport tabel ‘produs’**

****

**-Exemplu raport tabel ’comanda’**

****

**-Exemplu raport tabel ’client’**

****

# Concluzii

**În concluzie realizarea acestui proiect a fost foarte utilă , amintind de o mare parte din principalele concepte ale programării orientate pe obiect învățate semestrul trecut. Datorită modului în care a fost gândită și proiectată aplicația, dezvoltările ulterioare pot fi foarte ușor implementate în orice moment și nu doar de către cel ce a facut-o, ci și de către alți programatori care ar dori să aducă îmbunătățiri ulterioare**

**aplicației.**

**O dezvoltare ulterioară care ar putea fi făcută pentru ca aplicația să fie mai ușor de utilizat ar putea fi, spre exemeplu, o interfață grafică care să permit utilizatorului o interacțiune mai mare cu aplicația. În plus, acesta ar fi mult mai practică și mai utilă, dacă avem în vedere un utilizator neexperimentat. În ceea ce privește datele de intrare acestea ar fi mult mai ușor de introdus prin intermediul unei interfațe decât prin modul prin care se realizează în acest moment: citirea dintr-un fișier text de intrare. De asemenea, ar putea fi înbunătățită și baza de date relațională prin adăugarea mai multor tabele și relații între acestea, pentru a permite utilizatorului să efectueze comenzi mult mai complexe în cadrul aplicației, cum ar fi: împărțirea cliențiolor în 2 categorii pentru diferențierea celor fideli de restul; clienților fideli li s-ar putea oferi după un anumit număr de comenzi o anumită reducere pentru o comandă viitoare, iar clienții a căror comandă nu a putut fi procesată din cauza stocului insuficient li s-ar putea trimite o nouă înștiințare atunci când produsul va reapărea în stoc. Totodată și codul ar putea fi îmbunătățit prin scrierea sa mai eficientă, oferind aplicației o mai mare viteză și un timp de execuție mult mai bun în cazul datelor de intrare mult mai mare.**

# Bibliografie

* [**https://www.baeldung.com/javadoc**](https://www.baeldung.com/javadoc)
* **https://www.baeldung.com/java-pdf-creation**
* [**https://www.baeldung.com/java-copy-on-write-arraylist**](https://www.baeldung.com/java-copy-on-write-arraylist)
* [**https://www.oreilly.com/library/view/software-architecture-patterns/9781491971437/ch01.html**](https://www.oreilly.com/library/view/software-architecture-patterns/9781491971437/ch01.html)
* [**http://coned.utcluj.ro/~salomie/PT\_Lic/4\_Lab/Assignment\_3/Assignment\_3\_Indications.pdf**](http://coned.utcluj.ro/~salomie/PT_Lic/4_Lab/Assignment_3/Assignment_3_Indications.pdf)
* [**https://www.w3schools.com/java/java\_files\_read.asp**](https://www.w3schools.com/java/java_files_read.asp)
* [**https://ftp.utcluj.ro/pub/users/civan/IBD/2\_LABORATOR/09\_JDBC/IBD\_Lab9\_jdbc.pdf**](https://ftp.utcluj.ro/pub/users/civan/IBD/2_LABORATOR/09_JDBC/IBD_Lab9_jdbc.pdf)
* **Cursurile și laboratoarele de “Programare orientate pe obiect” și “Tehnici de Programare”**