



# Sistem de management al comenzilor dintr-un depozit

Student: Florea Gabriel-Alin

Grupa: 30226

Profesor: Pop Cristina

Profesor laborator: Moldovan Dorin

## Continut documentatie

1. Obiectivul temei
2. Analiza problemei puse in discutie
3. Detalii de proiectare
4. Implementare
5. Testare
6. Rezultate
7. Concluzii
8. Bibliografie

## 1. Obiectivul temei

Tema curenta pune in discutie realizarea unei aplicatii pentru procesarea comenzilor pe care clientii le dau la un depozit. Pentru retinerea datelor privitoare la clienti, produse si comenzi se vor folosi baze de date relationale. Interactiunea cu utilizatorul se va face prin intermediul unei interfete grafice.

## 2. Analiza problemei puse in discutie

### a. Asumptii

In cadrul aplicatiei curente se presupune in primul rand ca toate datele furnizate sunt corecte, neefectuandu-se verificari in acest sens. La efectuarea operatiilor asupra tabelelor, adaugare, editare, stergere, adaugare de noi comenzi, etc. se va presupune ca se respecta cerintele bazei de date relationale privind existenta si unicitatea cheii primare. Datele introduse se considera corecte, aplicatia de fata avand ca scop gestionarea unui depozit in care continutul acestuia, cat si a comenzilor sunt stocate corect. Odata cu adaugarea unei comenzi, stocul produsului comandat se va decrementa cu numarul de entitati necesare, iar produsul va fi adaugat la tabelul de comenzi, cu scopul de a ramane in istoricul comenzilor efectuate la respectivul depozit, istoric ce va putea fi consultat in orice moment. Tiparirea facturii se va face pentru fiecare comanda, nou adaugata. Pentru comenzile anterioare din istoric nu se va tipari o factura. Indiferent de pretul produsului, in aceasta aplicatie se considera faptul ca cumparatorul detine suma suficienta de bani pentru achitarea acestuia.

### b. Modelare

In ceea ce priveste modelarea cerintei puse in discutie, se va folosi o baza de date relationala, cu ajutorul mediului "MySQL Workbench", care va fi folosita dintr-o aplicatie Java. Ca si structura de baza, lucrandu-se cu baze de date, este tabelul de tip JTable. Completarea acestor tabele de tip JTable se face prin intermediul tehnicilor de reflection, care se dovedesc a fi deosebit de utile in acest context. Aplicatia de fata va reprezenta interfata utilizatorului cu o baza de date relationala, pentru un depozit, permitand operatii de adaugare, editare, stergere si, de asemenea, adaugarea unor comenzi in depozit.

### c. Utilizare

La rularea programului, se va deschide o fereastră, ce va conține doar un meniu în partea superioară. Utilizatorul poate naviga prin meniu, și poate alege ce dorește să facă, putând acționa atât asupra tabelelor prin vizualizare, adăugare, editare și ștergere de elemente, cât și putând adăuga noi comenzi, care vor putea sau nu fi satisfăcute, iar în caz afirmativ, putând fi observate în istoricul comenzilor.

Pentru adăugarea unui rând într-unul din tabele, se completează câmpurile corespunzătoare, și se apasă confirm.

Pentru editarea unui rând dintr-un tabel, se introduce id-ul rândului ce se dorește a fi editat (id-ul în sine nu poate fi editat), se apasă confirm din dreptul id-ului, iar apoi se pune valoarea câmpurilor ce se doresc a fi editate și se apasă butonul de confirmare corespunzător lor.

Pentru a șterge un rând dintr-un tabel, se introduce id-ul rândului ce se dorește a fi șters și se apasă pe butonul de confirmare.

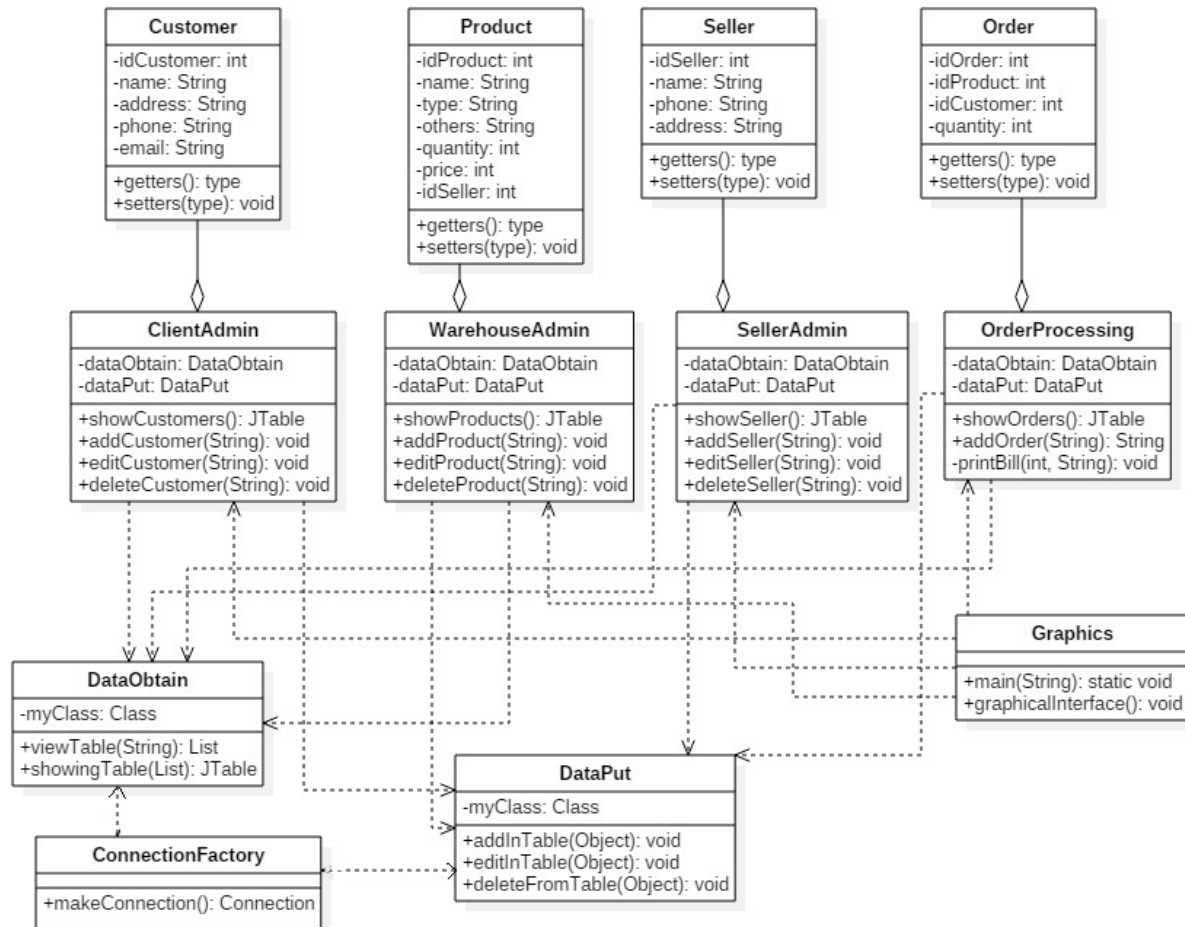
În utilizarea aplicației se consideră faptul că utilizatorul va introduce date corecte, pe baza cărora se poate opera cu baza de date și se pot efectua operațiile dorite.

## 3. Detalii de proiectare

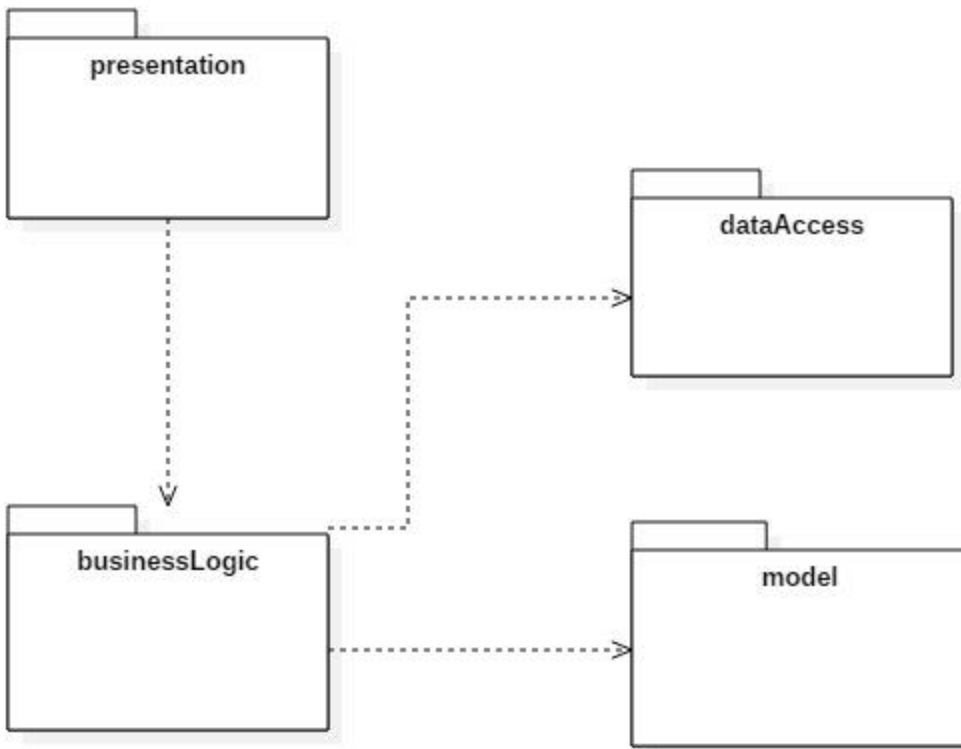
### a. Decizii de proiectare

Din punct de vedere al deciziilor de proiectare luate, proiectul de față se constituie din 4 pachete de clase, care formează o arhitectură pe 3 nivele a aplicației, fapt ce constituie omogenitate și claritate acestuia.

## b. Diagrame UML



Aceasta diagrama UML prezinta clasele din care este compus proiectul acesta, surprinzand campurile si metodele folosite in fiecare din clase. De asemenea din aceasta diagrama se poate observa modul in care clasele proiectului sunt legate intre ele. Print atenta si riguroasa observare a acestor diagrame de clase se poate observa modul in care aplicatia aceasta este structurata si dezvoltata.



Aceasta diagrama de pachete, prezentata mai sus, evidentiaza modul in care pachetele care compun acest proiect sunt interconectate, si respectiv relatiile de legatura ce au loc intre ele.

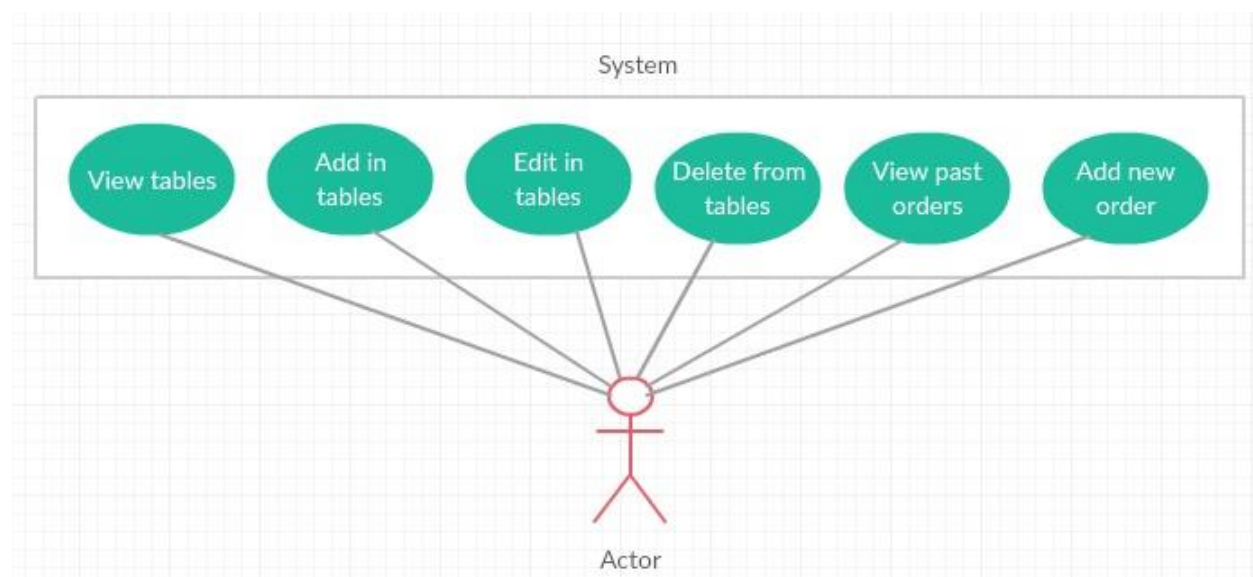


Diagrama cazurilor de utilizare, permite observarea rapida a posibilitatilor de interactiune pe care utilizatorul le are cu programul. Aceasta diagrama faciliteaza intelegerea modului in care programul se poate utiliza.

### c. Structuri de date utilizate

In ceea ce priveste structurile de date utilizate in cadrul proiectului, acestea au fost destul de clasice, iesind in evidenta doar utilizarea unor structuri de tip List<>, respectiv Vector<>. De asemenea s-a folosit Clasa Object si instante ale acesteia, alaturi de tehnici de reflection, pentru a permite ca anumite operatii pe tabele sa poate fi generalizate, fara a mai fi nevoie de a le implementa separate pentru fiecare din tabelele aplicatiei. Pe langa toate acestea s-au mai folosit instante ale diverselor clase care faciliteaza accesul la baze de date sau efectuarea operatiilor cu acestea.

### d. Proiectarea claselor

Daca e sa analizam mai amanuntit cele 12 clase ale proiectului curent, atunci se poate spune:

- Clasa Graphics prezinta doua metode: metoda main necesara oricarei aplicatii Java, respectiv o metoda graphicalInterface, in care este efectuata partea de grafica a aplicatiei cu ajutorul Java Swing.

- Clasa Customer modeleaza o astfel de entitate, avand campurile pe care le detine si tabelul cu acelasi nume: idCustomer, name, address, phone, email. Aceasta clasa contine doar getteri si setteri.

- Clasa Product modeleaza o astfel de entitate, avand campurile pe care le detine si tabelul cu acelasi nume: idProduct, name, type, others, quantity\_in\_stock, price\_per\_unit, idSeller. Aceasta clasa contine doar getteri si setteri, care pot fi folositi pentru a facilita accesul la date.

- Clasa Seller modeleaza o astfel de entitate, avand campurile pe care le detine si tabelul cu acelasi nume: idSeller, name, phone, address. Aceasta clasa contine doar getteri si setter, care faciliteaza accesul la date.

-Clasa Order modeleaza o astfel de entitate, avand campurile pe care le detine si tabelul cu acelasi nume: idOrder, idProduct, idCustomer, quantity. Aceasta clasa contine doar getteri si setteri, cu ajutorul carora pot fi manipulate campurile dorite.

-Clasa ConnectionFactory prezinta o singura metoda makeConnection, care are scopul de a realiza conexiunea propriu-zisa la baza de date.

-Clasa DataObtain prezinta ca si camp un obiect de tipul Class, iar ca si metode o metoda viewTable, care returneaza o lista cu continutul tabelului bazei de date, respectiv o metoda showingTable, care returneaza un JTable, pe care il formeaza dintr-o lista primita ca parametru. Aceste metode actioneaza pe tipul Object, utilizand tehnici de reflection.

-Clasa DataPut prezinta ca si camp un obiect de tipul Class, iar ca si metode, prezinta 3 metode care realizeaza diverse operatii pe tabelele bazei de date: addInTable, editInTable, deleteFromTable. Toate aceste metode actioneaza pe tipul Object, utilizand tehnici de reflection.

-Clasa ClientAdmin prezinta ca si campuri 2 obiecte, unul de tip DataObtain si altul de tipul DataPut, iar ca si metode, prezinta toate operatiile posibile pe datele tabelului Customer: showCustomers, addCustomer, editCustomer, deleteCustomer. Instantierea acestei clase se face din interfata grafica.

-Clasa SellerAdmin prezinta ca si campuri 2 obiecte, unul de tip DataObtain si altul de tipul DataPut, iar ca si metode, prezinta toate operatiile posibile pe datele tabelului Seller: showSellers, addSeller, editSeller, deleteSeller. Aceasta clasa este instantiata direct din ActionListener-ii interfetei grafice.

-Clasa WarehouseAdmin prezinta ca si campuri 2 obiecte, unul de tip DataObtain si altul de tipul DataPut, iar ca si metode, prezinta toate operatiile posibile pe datele tabelului Product: showProducts, addProduct, editProduct, deleteProduct. Interfata grafica instantiaza aceasta clasa pentru a manipula obiecte de tip Product.

-Clasa OrderProcessing prezinta ca si campuri 2 obiecte, unul de tip DataObtain si altul de tipul DataPut, iar ca si metode, prezinta o metoda de afisare a tuturor comenzilor trecute: showOrders, o metoda de adaugare a unei noi comenzi: addOrder, respectiv o metoda de tiparire a unei facturi.



## e. Pachete utilizate in cadrul proiectului

In ceea ce priveste pachetele utilizate in acest proiect, acestea sunt in numar de 4 si respecta o arhitectura stratificata pe 3 nivele, respectiv paradigma Model-View-Controller. Pachetul presentation prezinta o singura clasa ce realizeaza interfata grafica a aplicatiei. Pachetul model prezinta principalele structuri care modeleaza proiectul curent, si anume: Customer, Product, Order, Seller. Pachetul dataAccess prezinta clasele care realizeaza conexiunea la baza de date, respectiv contin interogările necesare pentru efectuarea operatiilor dorite de catre utilizator, si anume: ConnectionFactory, DataObtain, DataPut. Pachetul businessLogic contine clasele care realizeaza partea de business a aplicatiei, si anume: ClientAdmin, SellerAdmin, OrderProcessing, WarehouseAdmin.

## f. Interfata grafica

Interfata grafica a acestui sistem de management al comenzilor de la un depozit este un mediator intre utilizator si partea de cod a programului. Interfata grafica a acestui proiect este realizata cu Java Swing, intr-un mod relativ simplu, dar totusi deosebit de intuitiv si de usor de utilizat pentru utilizatorul aplicatiei.

Fereastra care se deschide la rularea aplicatie poarta numele de Order Management. Fereastra prezinta in partea superioara un meniu simplu din care se poate alege operatia dorita de utilizator. La apasarea butoanelor ce tin de vizualizarea tabelelor, se afiseaza tabelul solicitat de utilizator. Cand se deschide fereastra de introducere a unui nou obiect in tabel, utilizatorul trebuie sa completeze toate campurile cu valorile solicitate, de tipul solicitat, iar apoi sa apese butonul de confirmare. In momentul cand utilizatorul doreste sa editeze un obiect, va trebui mai intai sa introduca id-ul acestuia si sa apese butonul de confirmare din dreptul id-ului, iar apoi poate sa introduca campul, sau campurile pe care doreste sa le editeze, urmand apoi sa apese butonul de confirmare pentru fiecare din campurile dorite a fi editate. La stergerea unui rand dintr-un tabel, utilizatorul nu trebuie decat sa introduca id-ul randului respectiv si sa apese pe butonul de confirmare. La introducerea unei comenzi, modul de abordare este asemanator cu introducerea unui nou rand in tabel, cu mentiunea, ca acesta va fi sau nu adaugat, in functie de posibilitatea satisfacerii de catre depozit a respectivei comenzi.

## 4. Implementare

În ansamblul procesului de proiectare, pe lângă luarea deciziilor importante de proiectare, alături de realizarea diagramelor UML de clase, de pachete și de cazuri de utilizare, încă un pas deosebit de important este scrierea codului efectiv al aplicației.

Ideea de bază care a stat la realizarea acestei aplicații a fost realizarea câte unei clase pentru fiecare tabel al bazei de date. Aceste clase, reprezentând câte un obiect de tipul celor descrise de tabelele bazei de date, se supun unor clase care realizează partea de business a aplicației.

Utilizând tehnici de reflection, s-au creat două clase care efectuează anumite operații pe orice tabel al bazei de date (operand pe clasa `Object`).

Conectarea la baza de date este realizată de o clasă dedicată acestui lucru, clasă care este instantiată din clasele care efectuează operațiile generice pe baza de date.

Partea grafică a aplicației este realizată într-o clasă special dedicată acestui lucru, clasă din care sunt instantiate obiecte ale claselor de business a aplicației.

## 5. Testare

Această aplicație, fiind un instrument de interacțiune cu o bază de date, nu se pretează pentru o testare unitară, sau vreun altfel de testare în care se așteaptă un rezultat constant, deoarece o bază de date este mereu dinamică, cu intrări și ieșiri.

Totusi, această aplicație se poate testa, și a fost testată, observând comportamentul aplicației pentru fiecare caz posibil de utilizare, observând corectitudinea introducerii, editării, ștergerii și vizualizării tabelelor, cât și observând corectitudinea introducerii noilor comenzi, alături de decrementarea stocului disponibil și de tipărirea facturii, sau în cazul în care comanda nu poate fi onorată din cauza neexistenței în stoc a cantității de marfă solicitate, observând mesajul de under-stock afișat.

Bineînțeles că, testarea programului, cât și utilizarea acestuia trebuie făcută cu respectarea condițiilor minime amintite în secțiunea de asumptii, mai ales cele

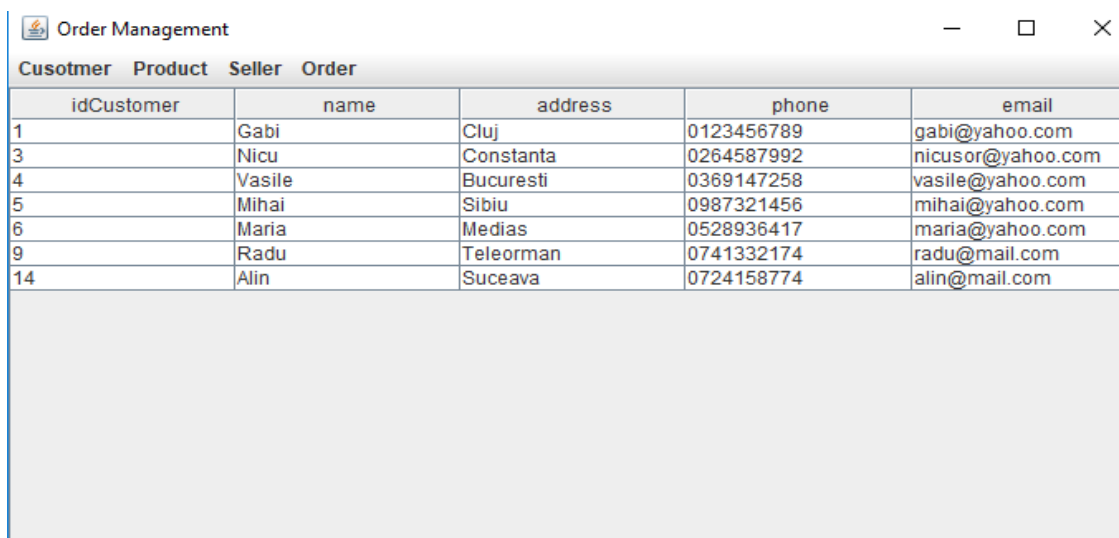
referitoare la corectitudinea datelor introduse. In cazul nerespectarii acestor conditii minime, programul va arunca exceptii potrivite fiecarei situatii.

## 6. Rezultate

In functie de cazul de utilizare ales, respectiv de ceea ce apasa utilizatorul, rezultate sunt diferite. In cazul in care se doreste vizualizarea tabelului, in fereastra curenta, se va afisa tabelul solicitat. In cazul in care utilizatorul doreste efectuarea unei operatii de adaugare, editare sau stergere din tabele, el va trebui sa introduca datele solicitate, respectiv sa apase butoanele solicitate, dar pe fereastra curenta nu se va afisa nimic. Pentru a vizualiza rezultatul operatiei efectuate, utilizatorul trebuie sa mearga din nou pe vizualizarea tabelului. La introducerea unei noi comenzi, in cazul in care aceasta poate fi onorata, utilizatorul, poate observa comanda sa in comenzi trecute, stocul depozitului decrementat cu cantitatea cumparata de el, respectiv, factura tiparita in format text, care prezinta mai multe lucruri, printre care si totalul de plata pentru comanda adaugata. In cazul in care comanda nu poate fi satisfacuta, in ecranul curent, chiar in apropierea butonului de confirmare a comenzii se va afisa un mesaj de under-stock, iar comanda respectiva nu va fi luata in considerare.

In continuare se pot vedea cateva poze semnificative pentru modul de functionare a aplicatie.

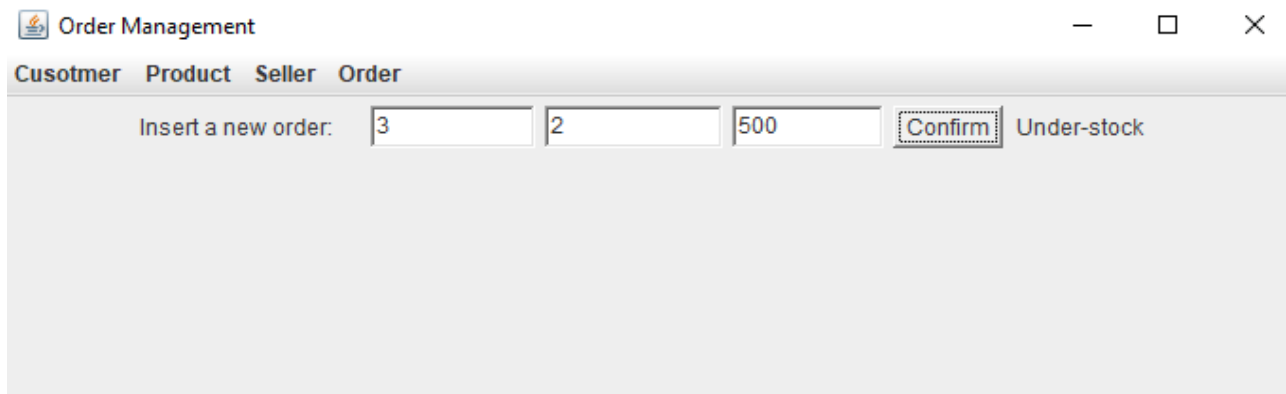
- Afisarea unui tabel



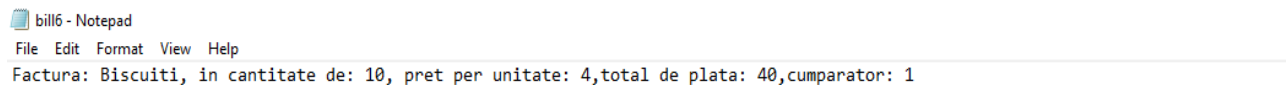
The screenshot shows a window titled "Order Management" with a standard Windows interface (minimize, maximize, close buttons). Below the title bar, there are four tabs: "Cusotmer", "Product", "Seller", and "Order". The "Cusotmer" tab is selected, displaying a table with the following columns: "idCustomer", "name", "address", "phone", and "email". The table contains seven rows of data, with the last row having an "idCustomer" of 14. Below the table, there is a large, empty rectangular area, likely a placeholder for additional information or a detailed view of the selected customer.

idCustomer	name	address	phone	email
1	Gabi	Cluj	0123456789	gabi@yahoo.com
3	Nicu	Constanta	0264587992	nicusor@yahoo.com
4	Vasile	Bucuresti	0369147258	vasile@yahoo.com
5	Mihai	Sibiu	0987321456	mihai@yahoo.com
6	Maria	Medias	0528936417	maria@yahoo.com
9	Radu	Teleorman	0741332174	radu@mail.com
14	Alin	Suceava	0724158774	alin@mail.com

- Afisarea unui mesaj de under-stock, pentru o comanda ce nu poate fi satisfacuta



- O factura tiparita pentru o comanda ce a fost satisfacuta de catre depozit



## 7. Concluzii

Din punctul meu de vedere, acest proiect a fost unul extraordinar de util si important pentru aprofundarea si intelegerea unor notiuni si concepte deosebit de folositoare in programarea orientata pe obiecte in general si in Java in particular. Acest proiect mi-a oferit oportunitatea de a intelege si de a lucra pentru prima data cu tehnici de reflection si cu clasa Object, in mod explicit. Metodele, cat si modul in care se opereaza cu instante ale clasei Object, invocarea de metode, si altele ce tin de tehnicile de reflection au impus un studiu intens, dar totodata m-au facut sa inteleg cu adevarat necesitatea si utilitatea programarii orientate pe obiecte, reusind sa vad faptul ca, in acest tip de programare, chiar se poate lucra cu obiecte. O alta importanta provocare a fost lucrul cu baze de date in aplicatie Java. Desi mai lucrasem cu asa ceva, totusi, acest lucru imbinat cu tehnicile de reflection a constituit o adevarata provocare pentru mine. Organizarea pe nivele a aplicatiei, a reprezentat un impediment la inceput, dar ulterior, am observat ca acest lucru este foarte benefic, si chiar faciliteaza

organizarea si scrierea codului aplicatie. Utilizarea Javadoc a fost un alt lucru important in cadrul acestui proiect, lucru care m-a facut sa inteleg necesitatea documentarii claselor unei aplicatii si cat de clar poate fi facut acest lucru daca se lucreaza corespunzator.

Cu siguranta ca, acest proiect, nu este o forma desavarsita a ceea ce ar putea fi o astfel de aplicatie, dar reprezinta, cu siguranta, o ustensila practica si intuitiva de manipulare a unei baze de date pentru un depozit, chiar si de catre o persoana fara prea multa experienta in sisteme de calcul. Printre posibilele dezvoltari ulterioare ale acestui proiect se numara: o interfata grafica mai placuta si mai usor de utilizat, posibilitatea adaugarii tabelelor si efectuarii operatiilor pe acestea, nu doar a randurilor in tabele, posibilitatea intocmirii unor grafice statistice, care sa reflecte progresul economic al depozitului, si lista ar putea continua.

## 8. Bibliografie

<http://users.utcluj.ro/~igiosan/>

<https://stackoverflow.com/>

<http://staruml.io/>

<https://creatly.com/diagram-type/use-case>

<http://www.tutorialspoint.com/listtutorial/How-to-call-method-using-Reflection-in-Java/4090>

[https://www.tutorialspoint.com/java/java\\_documentation.htm](https://www.tutorialspoint.com/java/java_documentation.htm)

<http://www.baeldung.com/java-write-to-file>

<http://tutorials.jenkov.com/java-reflection/index.html>