**DOCUMENTATIE TEMA 3**

**ORDER MANAGEMENT**

**Nume prenume: Dragomir Mihai-Robert**

**Grupa: 302210**

**Profesor Laborator Assist Mitrea Dan**

Contents

[1. Cerinte Functionale 3](#_Toc476131445)

[2. Obiective 3](#_Toc476131446)

[2.1. Obiectiv Principal: 3](#_Toc476131447)

[2.2. Obective Secundare: 4](#_Toc476131448)

[3. Analiza Problemei 4](#_Toc476131449)

[4. Proiectare 6](#_Toc476131450)

[4.1. Structuri de date 6](#_Toc476131451)

[4.2. Diagrama de clase 6](#_Toc476131452)

[4.3. Algoritmi 6](#_Toc476131453)

[5. Implementare 6](#_Toc476131454)

[6. Concluzii si Dezvoltari Ulterioare 8](#_Toc476131456)

[7. Bibliografie 8](#_Toc476131457)

# Cerinte Functionale

Dezvoltati o aplicatie de gestionare a comenzilor depuse de clienti in cadrul unei instante virtuale de tip depozit, care sa permita urmatoarele functionalitati:

* Permite utilizatorului sa opteze pentru instanta asupra careia sa se rasfranga operatiile, in cazul concret utilizatorul poate opta daca urmeaza sa realizeze operatii asupra instantelor de client, comanda sau produs.
* Permite utilizatorului sa opteze pentru una dintre operatiile “adauga client nou”, “editeaza un client existent”, “sterge un client existent”, “vizualizeaza toti clientii din baza de date”, in ceea ce priveste operatiile ce se rasfrang asupra clientilor.
* Permite utilizatorului sa opteze pentru una dintre operatiile “adauga produs nou”, “editeaza un produs existent”, “sterge un produs existent”, “vizualizeaza toate produsele din baza de date”, in ceea ce priveste operatiile ce se rasfrang asupra produselor.
* Permite utilizatorului sa adauge o comanda inserand in baza de date id-ul clientului pentru care se doreste adaugarea comenzii, id-ul produsului pentru se doreste a fi entitatea tranzactionata in cadrul comenzii, respective cantitatea de produs ce urmeaza a fi comandata.
* Permite utilizatorului sa vizualizezii in cadrul unei facturi in format fisier txt detaliile relevante legate de comanda plasata, in speta, id-ul produsului comandat, insotit de numele acestuia, id-ul clientului care a depus comanda, insotit de numele acestuia, cantitatea de produs tranzactionata in cadrul comenzii, respective pretul total pe care il presupune realizarea comenzii.
* Atentioneaza utilizatorul in cazul in care comanda simulate de acesta presupune tranzactionarea unei cantitati de produs mai mare decat stocul curent de care dispune instanta virtuala a depozitului, caz in care comanda nu e procesata.
* In cazul in care o comanda este procesata cu succes, aplicatia pastreaza integritatea datelor inregistrate la nivelul magazinului, scazand din stocul disponibil pentru produsul care a fost tranzactionat in cadrul comenzii, cantitatea de produs tranzactionata.
* Atentioneaza utilizatorul in cazul in care acesta doreste sa faca o operatie de natura sa altereze integritatea datelor din cadrul bazei de date ale magazinului, neprocesand operatia respectiva pana in momentul in care operatia este valida din punct de vedere logic si viabila din punct de vedere al uzitarii sale.

# Obiective

## Obiectiv Principal:

Propuneti, proiectati si implementati un sistem de gestiune a comenzilor in cadrul unui magazin, care dispune de o interfata grafica, prin intermediul careia utilizatorul poate adauga, sterge, actualiza sau vizualiza intrgral clienti sau produse, respectiv poate plasa comenzi in baza produselor si clientilor existenti in baza de date si poate vizualiza o factura detaliata rezultata in urma plasarii cu succes a unei comenzi.

## Obective Secundare:

|  |  |  |
| --- | --- | --- |
| **Obiectiv Secundar** | **Descriere** | **Capitol** |
| Dezvoltarea de use case-uri si scenario | Prezinta o reprezentare grafica a modului in care utilizatorul interactioneaza cu interfata grafica a sistemului de gestionare a magazinului virtual, urmarind anumite spete si scenarii de functionare si utilizare. | 3 |
| Alegerea structurilor de date | Ofera explicatii cu privire la optiunea pentru anumite structuri de date in implementarea solutiei. | 4 |
| Impartirea pe clase | Urmareste din puct de vedere schematic modul de interactiune si asamblare al claselor in vederea functionarii corecte a sistemului de gestiune a comenzilor. | 4 |
| Dezvoltarea algoritmilor | Aduce adnotari imperios necesare asupra modului in care sunt traduse in practica, sub forma algoritmica, anumite metode auxiliare sau fundamentale in implementarea finala. | 4 |
| Implementarea solutiei | Realizeaza o introspectie asupra functionalitatii si a necesitatii fiecarei clase constituente. | 5 |

# Analiza Problemei

Diagrama de use case generala, ce ofera o vedere minimala a modului in care utilizatorul interactioneaza la un prim contact cu fereasta principala a interfetei grafice:

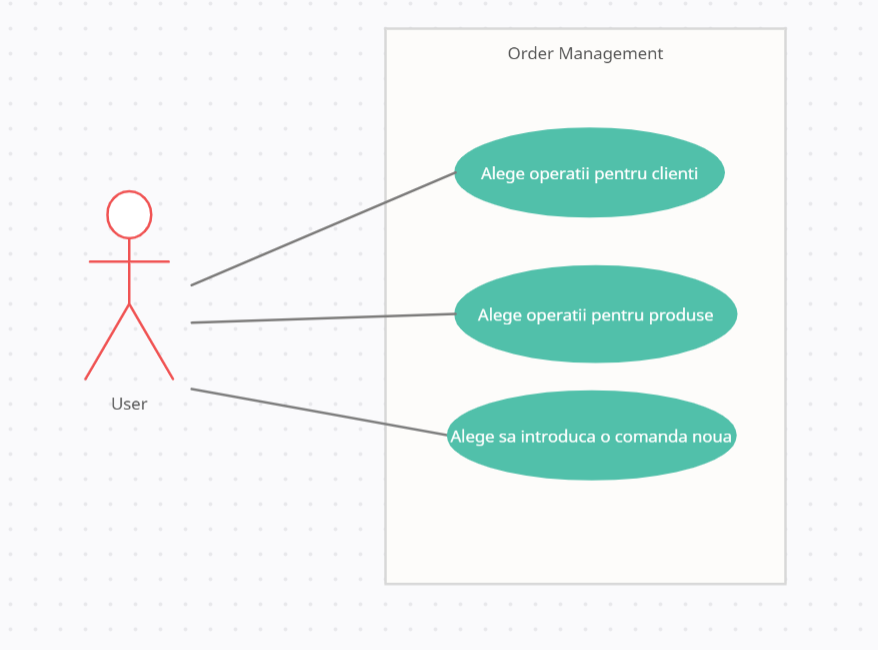
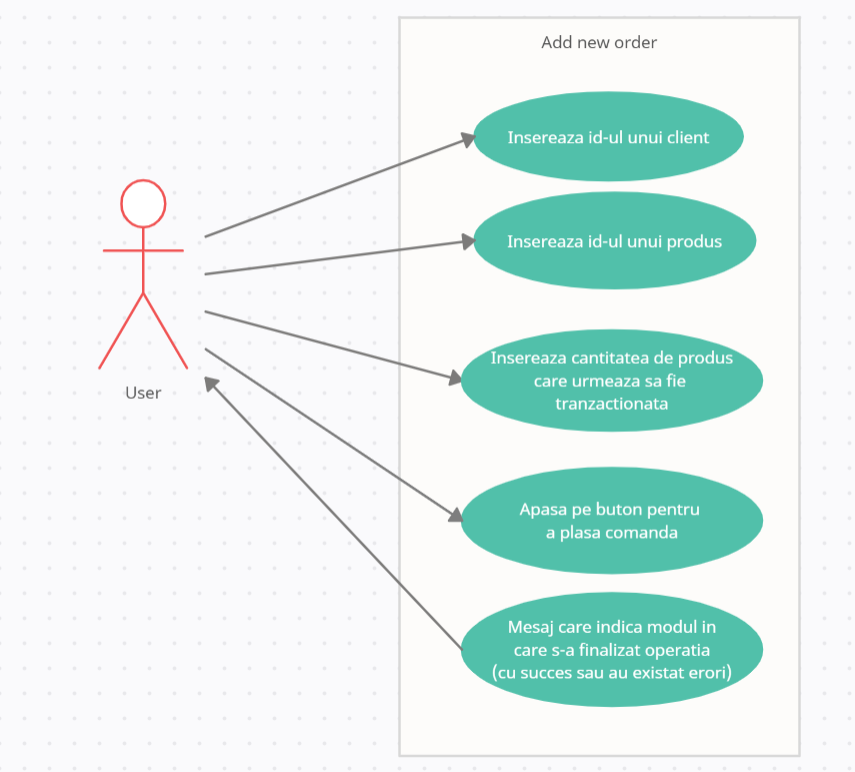


Diagrama use case aferenta operatiei de plasare comanda:



Use Case: plaseaza comanda

Actor principal: user

Principalul scenariu

1. Utilizatorul insereaza id-ul unui client in interfata grafica;

2. Utilizatorul insereaza id-ul unui produs in interfata grafica;

3. Utilizatorul insereaza cantitatea de produs ce urmeaza sa fie tranzactionata;

4. Utilizatorul apasa butonul de “Add” pentru a plasa comanda;

5. Sistemul de gestiune adauga comanda in baza de date si afiseaza un mesaj care

Semnaleaza ca operatia a fost realizata cu succes.

Scenariu alternativ

Datele introduse nu resepecta formatul: id-ul clientului, id-ul produsului sau cantitatea tranzactionata nu

Sunt numere naturale pozitive.

-

Utilizatorul introduce un id de client sau un id de produs care nu exista in baza de date.

-

Utilizatorul introduce o cantitate de produs ce se doreste a fi tranzactionata care depaseste stocul disponibil al produsului respectiv.

-

Interfata grafica afiseaza un mesaj care semnaleaza motivul erorii, comanda nu este procesata si se realizeaza redirectionarea catre primul pas.

# Proiectare

## Structuri de date

Un element fundamental in implementarea solutiei finale il constituie generalizarea realizata prin intermediul tehnicilor de Reflection. Astfel, abordarea majoritara tinde spre utilizare predominanta a obiectelor abordate dintr-o perspectiva generala, prin uzitarea tipului primar si maleabil de Object, in detrimentul unui obiect, instanta a unei clase mai specifice. Structura de date principala care opereaza pe Object-uri este ArrayList-ul, ca urmare a posibilitatii de acces secvential asupra elementelor si alocarii dinamice a datelor.

## Diagrama de clase

Diagrama UML din randurile ce urmeaza descrie modul in care clasele utilizate in implementarea sistemului de gestiune a instantei virtuale de magazin interactioneaza intre ele.

O imagine care conține text, electronice

Descriere generată automat

## Algoritmi

Mecanismul de reflectie permite ca informatiile despre obiecte sa fie complet determinate in faza de executie, fara nicio interventie in faza de compilare. Prin intermediul acestui mecanism programul primeste capacitatea de introspective, de a se examina pe sine, de a incarca o clasa despre care nu se stie nimic, de a-i afla variabilele, metodele si constructorii si apoi de a lucre cu ele. Principala utilizare a tehnicilor de reflectie in cadrul proiectului urmarese generalizarea sub deviza unei clase generice T in clasa AbstractDAO a metodelor de adaugare, stergere dupa id, cautare dupa id, cautare integrala, actualizare sau determinare campuri si headere pentru orice tip de tabela ori clasa, eliminand codul duplicat si potentand caracterul comprehenisv.

# Implementare

Basorelieful architectural in implementarea sistemului de gestiune a comenzilor, reflectat fizic la nivel de cod in cadrul pachetelor propune o arhitectura stratificata de tip 4-tier, unde fiecare strat are un rol bine definit, si beneficiaza de izolare, modificarile operate la un anumit strat neavand impact asupra componentelor din alt strat. In ceea ce priveste pachetele utilizate la nivelul acestui proiect, prin intermediul carora stratificarea devine mai palpabila, in randurile ulterioare urmeaza o prezentare detaliata a acestora

**Pachetul connection** este de sine statator, are mai curand un caracter auxiliar pentru stratul de access al datelor, dar rolul sau este essential. In cadrul acestuia regasim clasa ConnectionFactory, prin intermediul ei se realizeaza conexiunea la baza de date, pe baza datelor utilizatorului. Totodata clasa ConnectionFactory se ocupa si cu deconectarea de la baza de date, devenind imperioasa prin legatura unilaterala realizata intre mediul de lucru java si SQL.

**Pachetul dao** reprezinta stratul de access al datelor. La nivelul sau se realizeaza interactiunea cruda, primara intre baza de date si java. Clasa AbstractDao urmareste prin intermediul unei clase generice T, generalizarea metodelor de access si de modificare a datelor din baza de date. Sunt definite metode de creare de interogari care sa acopere toate operatiile ce urmeaza a fi efectuate asupra tabelelor, prin modelarea de String-uri, in conformitate cu sintaxa SQL. Pe baza fiecareia dintre interogarile create sunt definite metode care le executa, prin conectarea la baza de date si opereaza modificarile dezirabile asupra tabelei. Astfel pentru adaugarea, stergerea, actualizarea, cautarea dupa ID, sau vizualizarea integrala a datelor este abordeata o perspectiva generala de executie a operatiilor solicitate, in baza tehnicilor de reflective si a capacitatii de introspectie. Clasa T, devine astfel un alter ego al claselor din model, care, prin genericitate, se metamorfozeaza contextual in oricare dintre acestea. Intrucat primary key-ul tabelelor este reprezentat de id, caruia i-a fost adaugat la crearea acestora capacitatea de auto-incrementare, urmarim insertia unui obiect nou in tabele in absenta unui id, care am putea spune ca se “auto-determina”. Astfel metoda getPrimaryKey() urmareste determinarea cheii primare a tabelului si isi demonstreaza utilitatea in cadrul metodei de getFields() care stabileste campurile din tabela pe care se opereaza diferitele actiuni, prin excluderea campului ce constituie un primary key rezultat prin apelul metodei de getPrimaryKey().

De asemenea, in vederea inserarii viitoare la niel de view a datelor intr-un JTable, la nivelul clasei AbstractDAO exista 2 metode care urmaresc, sa extraga la nivel generic, in baza tehnicii de refelctie, elementele din baza de date (getTableCells()), respectiv header-ul coloanelor(getFieldNames()).

Clasele ClientDAO, ClientOrderDAO, OrderItemDAO(), ProductDAO() extind clasa AbstractDAO() in vederea stabilirii la nivelul stratului de bussines logic a unei concretizari pentru realizarea la nivel de clase concret, a operatilor generic definite.

**Pachetul bll**, reprzinta stratul de “business logic” sau de logica a aplicatiei. Acesta preia datele din interfata grafica si le transmite mai departe stratului de access al datelor, jucand la nivel ideatic rol de pachet invelitor al dao. Aprioric transmiterii datelor stratului dao, acestea sunt validate pentru a nu corupe datele din baza de date. Subpachetul validators prezinta o interfata Validate care este implementata de catre toate celelalte clase de validare. Clasa StockValidator opereaza pe produse si urmareste ca stocul sa aiba o valoare mai mare sau egala cu 0. Clasa PriceValidator opereaza de asemenea, tot pe produse urmarind ca pretul sa aiba o valoare pozitiva strict mai mare decat 0. Clasa EmailValidator opereaza pe client si urmareste in baza unui pattern si uzand de regex ca adresa de email a unui client sa respecte calapodul predefinit. Clasa QuantityValidator opereaza pe comenzi si urmareste pe de o parte ca valoarea canitatii de produs care urmeaza sa fie tranzactionat, sa fie strict pozitiva, iar, pe de alta parte, ca stocul disponibil din produsul cerut sa poata sustine implinirea comenzii.

In cadrul pachetului principal, se regasesc urmatoarele clase:

ClientBLL – instantiaza un obiect de tip ClientDAO si o lista de validatori pentru a verifica pastrarea integritatatii datelor in cazul inserarii si actualizarii. In ceea ce priveste stergerea si actualizarea, se verifica aprioric existenta unui client cu id-ul respectiv, prin cautarea dupa id. In cazul unor neconformitati sunt aruncate exceptii, fie in cadrul metodei, fie de catre validatori, prinse la nivel de controller.

ProductBLL – instantiaza un obiect de tip ProductDAO si o lista de validatori pentru a verifica pastrarea integritatatii datelor in cazul inserarii si actualizarii. In ceea ce priveste stergerea si actualizarea, se verifica aprioric existenta unui produs cu id-ul respectiv, prin cautarea dupa id. In cazul unor neconformitati sunt aruncate exceptii, fie in cadrul metodei, fie de catre validatori, prinse la nivel de controller.

OrderBLL – instantiaza un obiect de tip ClientOrderDAO si unul de tip OrderItemDAO, intrucat urmarim inserarea in doua tabele si are de asemenea, ca atribut si o lista cu validatori. Inainte de inserare se verifica, prin cautarea dupa id, existenta produsului si a clientului in baza de date.In client order se va insera id-ul clientului caruia ii va fi atribuit un id, iar pe baza acestui id, a id-ului produsului si a cantitatii dorite, se realizaza si insertia in OrderItem pentru a nu viola proprietatea de cheie straina. Produsului cu id-ul indicat, adaugat la comanda ii este actualizat si stocul, scazand din valoarea curenta, cantitatea de produs comandata. In cazul unor neconformitati sunt aruncate exceptii, fie in cadrul metodei, fie de catre validatori, prinse la nivel de controller.

**Pachetul model** contine cele 4 clase descries in limbaj java care isi au echivalentul in baza de date: Client, Product, ClientOrder, OrderItem. Fiecare dintre acestea are metode de get si set pentru fiecare atribut si o serie de constructori supraincarcati pentru a face viabila tehnica de reflectie aplicata la nivelul pachetului dao in clasa AbstractDAO.

**Pachetul presentation** se scindeaza in 2 subpachete: pachetul view si controller. Pachetul view contine o serie de clase care impreuna formeaza interfata grafica cu care interactioneaza utilizatrul. Pachetul controller face legatura intr view si bll, adaugand listener pe fiecare dintre butoane si prelevand informatia din textField-urile din view-uri. Acesta ghideaza pe baza informatiilor colectate modul de actiune ulterioara al interfetei grafice si al stratului de business logic.

**Pachetul start** contine clasa Main de unde porneste executia programului.

# Concluzii si Dezvoltari Ulterioare

Finalmente, proiectul supus spre implementare isi dovedeste utilitatea prin euristica sa bazala, anume modul de interactiune intre mediul de lucru Java si un sistem de manipulare de baze de date. Totodata, generalizarea inerenta adusa de utilizarea tehnicilor de reflection si evitarea codului duplicat constituie un precedent valoros in vederea implementarii unor proiecte ulterioare.

Intre dezvoltarile ulterioare ce ar putea fi aduse sistemului de gestiune, regasesc de o importanta preeminenta posibilitatea adaugarii in cadrul aceleiasi comenzi de produse multiple pentru un client, adaugarea de metode care sa permita stergerea sau actualizarea unor comenzi preexistente, dar si posibilitatea vizualizarii intr-un mod comprehensiv a modului de stocare al comenzilor. Bineinteles, toate aceste posibile dezvoltari ar trebui realizate prin actualizarea interfetei grafice si aprioric asigurarii prezervarii integritatii datelor.

# Bibliografie

1. <https://www.baeldung.com/java-jdbc>
2. <https://mkyong.com/jdbc/how-to-connect-to-mysql-with-jdbc-driver-java/>
3. <https://dzone.com/articles/layers-standard-enterprise>
4. <http://tutorials.jenkov.com/java-reflection/index.html>
5. <https://www.baeldung.com/javadoc>
6. <https://dev.mysql.com/doc/workbench/en/wb-admin-export-import-management.html>
7. <https://www.w3schools.com/java/java_files_create.asp>
8. <https://stackoverflow.com/questions/17711324/database-structure-for-customer-table-having-many-orders-per-customer-and-many/17711375>
9. <https://docs.oracle.com/javase/tutorial/uiswing/components/table.html>