DOCUMENTAȚIE

TEMA *3*

NUME STUDENT: Georgian Andreea

GRUPA: 30221

# CUPRINS

[1. Obiectivul temei 3](#_Toc95297885)

[2. Analiza problemei, modelare, scenarii, cazuri de utilizare 3](#_Toc95297886)

[3. Proiectare 3](#_Toc95297887)

[4. Implementare 3](#_Toc95297888)

[5. Rezultate 3](#_Toc95297889)

[6. Concluzii 3](#_Toc95297890)

[7. Bibliografie 3](#_Toc95297891)

1. **OBIECTIVUL TEMEI**

Obiectivul principal al acestei teme este proiectarea și implementarea unei aplicații, care are ca scop gestionarea comenzilor pentru un depozit. Aplicatia pune la dispozitia utilizatorilor o gama larga de functii, cum ar fi: adaugarea unor produse, editarea produselor existente, crearea comenzilor, stergerea produselor si multe altele. De asemena, aplicatia emite o facture la plasarea fiecarei comenzi.

Obiectivele secundare ale temei sunt:

* Analizarea problemei și identificarea cerințelor
* Proiectarea aplicației de plasare si gestionare a comenzilor
* Implementarea aplicației de plasare si gestionare a comenzilor
* Testarea aplicației de plasare si gestionare a comenzilor

1. **Analiza problemei, modelare, scenarii, cazuri de utilizare**

Comenzile online devin tot mai populare in ultimul timp, deoarece ele sunt mai eficiente. Gestionarea comenzilor unui depozit utilizand foi scrise de mana este dificila si necesita destul de mult timp, fapt care duce la intarzieri in coordonarea angajatilor si in livrarea comenzilor. Solutia ar fi o aplicatie de gestionare a comenzilor, a produselor si a clientilor.

Caz de utilizare : adaugarea unui produs

User principal : angajat

Scenariul de succes:

1. Angajatul selecteaza optiunea de a adauga un produs nou.
2. Aplicatia va afisa un formular in care detaliile produsului ar trebui inserate.
3. Angajatul insereaza numele produsului, pretul lui si cantitatea in care acesta este disponibil
4. Angajatul apasa butonul pentru adaugare.
5. Aplicatia salveaza datele produsului nou in baza de date si afiseaza un mesaj de confirmare.

Scenariu alternativ: Set de valori incorect

* Userul introduce o valoare negativ pentru stocul produsului
* Aplicația afișează un mesaj de eroare și îi solicita userului să introducă o valoare valida pentru stoc
* Scenariul revine la pasul 3

# Proiectare

Pentru stocarea datelor acestei aplicatii s-a folosit o baza de date, in care se gasesc toti clientii, toate produsele si comenzile efectuate. Astfel, clasele programului se impart in patru pachete :

Presentation Layer : contine clasele care definesc interfata utilizatorului

Business Layer : contine clasele care incapsuleaza logica aplicatiei ; Din acest pachet fac parte clasele de validare a datelor, cat si clasele care implementeaza query-urile care fac legatura cu baza de date

Data Access Layer : contine clasele care contin query-urile si conexiunile cu baza de date ; Din acest pachet fac parte clasele care creeaza query-urile care fac legatura cu baza de date la nivel general , urmand apoi sa se concretizeze pentru fiecare tabela a bazei de date

Model : contine clasele mapate la tabelele bazei de date

Implementarea interfeței grafice este făcută după modelul MVC (model-view-controller). Acesta este un model pentru proiectarea software-ului utilizat în mod obișnuit pentru a implementa interfețe cu utilizatorul, date și logica de control. Presupune o separare între logica software-ului și afișaj. Această “separare a preocupărilor” asigură o mai bună diviziune a lucrurilor și o întreținere în timp îmbunătățită. Multe interfețe se bazează pe acest șablon. Ideea este separarea programelor în Model, View (vedere, vizualizare) (creează afișajul, interacționând cu Modelul după nevoi), și Controller (răspunde la cererile utilizatorului, interacționând atât cu Vizualizarea cât și cu Modelul) după nevoi.

Unified Modeling Language (UML) joacă un rol important în dezvoltarea de software, dar și în pentru sistemele non-software din numeroase domenii, deoarece este o modalitate prin care puteți vizualiza comportamentul și structura unui sistem sau proces. UML ajută la prezentarea posibilelor erori din structurile aplicațiilor, comportamentele sistemelor și alte procese de afaceri.

UML a apărut pentru prima dată în anii 90, datorită lui Grady Booch, Ivar Jacobson și James Rumbaugh, trei ingineri de software care au dorit să dezvolte o modalitate mai puțin haotică de a reprezenta dezvoltarea de software din ce în ce mai complexă, separând, în același timp, metodologia de proces. În prezent, UML este în continuare notația preferată atât pentru dezvoltatori, cât și pentru manageri de proiect, deținători de companii, antreprenori din domeniul tehnologiei și specialiști din diferite domenii.

# Implementare

**CLASA ‘AbstractDataAccess’**

Aceasta este o clasa abstracta contine metodele care ajuta la conectarea la baza de date, cum ar fi : crearea unui obiect, modificarea unui obiect, stergerea unui obiect si gasirea unui anumit obiect. Clasa foloseste tehnici de reflexie pentru implementarea generala a comportamentului metodelor, astfel interogarile pentru accesarea bazei de date pentru un anumit obiect care ii corespunde unei tabele vor fi generate prin reflexie.

La partea de metode, aceasta clasa contine metode pentru fiecare functie pusa la dispozitie utilizatorului aplicatiei si care au legatura cu valorile stocate in baza de date.

O imagine care conține text

Descriere generată automatO imagine care conține text

Descriere generată automatUn exemplu de metoda care se gaseste in aceasta clasa este metoda pentru inserarea unui obiect, ilustrata mai jos :

De asemenea, clasa mai contine si metode pentru crearea query-urilor, cum ar fi metoda pentru crearea query-ului insert :

O imagine care conține text

Descriere generată automat Pe langa acestea, clasa contine o metoda care scrie intr-un tabel valorile elementelor care exista intr-o anumita tabela din baza de date.

**CLASA ‘ProductBL’**

Aceasta clasa incapsuleaza logica aplicatiei atunci cand vine vorba de produse. Contine un vector de validatori, care sunt necesari pentru a ne asigura ca datele introduse in baza de date sunt corecte, pentru a nu aparea erori ulterioare. De asemenea, mai contine o variabila de tipul „AbstractDataAccess” pentru a avea acces la toate metodele care preiau si introduc date in baza de date.

De exemplu, metoda „insertProduct” valideaza si insereaza, in cazul in care datele obiectului sunt corecte, un obiect de tip „Product” in baza de date, si returneaza cheia care a fost generata automat pentru acel produs nou :

O imagine care conține text

Descriere generată automat

**CLASA ‘ClientBL’**

Aceasta clasa incapsuleaza logica aplicatiei atunci cand vine vorba de clienti. Contine un vector de validatori, care sunt necesari pentru a ne asigura ca datele introduse in baza de date sunt corecte, pentru a nu aparea erori ulterioare. De asemenea, mai contine o variabila de tipul „AbstractDataAccess” pentru a avea acces la toate metodele care preiau si introduc date in baza de date.

De exemplu, metoda „updateClient” modifica valoarea unui camp, a carui nume este dat ca parametru, pentru un client, inserand in locul ei o noua valoare, de asemenea data ca parametru. Clientul pentru care se face modificarea este ales dupa id-ul sau, care este preluat din interfata grafica si dat ca parametru in functie.

O imagine care conține text

Descriere generată automat

**CLASA ‘PriceValidator’**

Aceasta clasa implementeaza o interfata numita „Validator”, in care se verifica daca datele introduse sunt corecte. Clasa „PriceValidator” este responsabila pentru validarea pretului unui produs, si ne asigura ca un produs cu un pret negativ nu poate fi inserat in baza de date.

O imagine care conține text

Descriere generată automatMetoda prin care clasa face acest lucru este metoda mostenita de la interfata pe care o implementeaza, adica metoda „Validate” :

**CLASA ‘EmailValidator’**

Aceasta clasa implementeaza o interfata numita „Validator”, in care se verifica daca datele introduse sunt corecte. Clasa „EmailValidator” este responsabila pentru validarea email-ului unui client, si ne asigura ca un client care are o adresa de email invalida nu poate fi inserat in baza de date.

O imagine care conține text

Descriere generată automatMetoda prin care clasa face acest lucru este metoda mostenita de la interfata pe care o implementeaza, adica metoda „Validate” :

Variabila „EMAIL\_PATTERN” este o variabila de tip final in care este stocat un pattern de adrese de email.

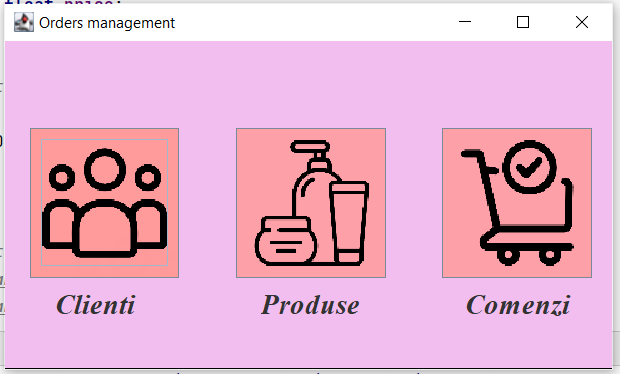
**CLASA ‘Orders’**

Aceasta clasa mapeaza tabela „Orders” din baza de date. Asta inseamna ca are exact aceleasi campuri ca si tabela corespunzatoare, adica trei variabile de tip int : „OrderID”, „ClientID” si „ProductID”, care fac legatura dintre comanda si clientulncare a comandat, si intre comanda si produsul care afost comandat. Pe langa acestea, clasa mai are o variabila de tip int „cantity”, in care se retine numarul de produse care au fost comandate, si respectiv o variabila de tip float „price”, in care se retine costul total al comenzii.

La partea de metode, aceasta clasa contine setter-ele si getter-ele specifice fiecarui camp, cat si o metoda „toString”, care returneaza reprezentarea sub forma de String a obiectului de tip comanda.

**CLASA ‘VIEW’**

Această clasă face parte din modelul MVC pentru interfața grafică. Practic, această clasă redă conținutul unui model. Specifică exact cum se prezintă utilizatorului datele din model. Daca modelul se schimbă, vederea trebuie să își actualizeze prezentarea după nevoi. Vizualizarea și modelul sunt obligatoriu independente, dar comunică între ele. În cazul aplicației de față, exista mai multe interfete care compun aplicatia. Clasa care va fi prezentata in cele ce urmeaza este clasa “FirstView”. Aceasta interfata este prima care apare atunci cand aplicatia este rulata.

Interfata contine trei butoane, unul pentru a intra in interfata pentru client, unul pentru a intra in interfata pentru produse, respective unul pentru a intra in interfata unde se fac comenzile. Daca, de exemplu, utilizatorul apasa pe butonul “Clienti”, se va deschide interfata prin care se gestioneaza clientii din baza de date.

Aici, user-ul poate sa vizualizeze in timp real lista clientilor din baza de date, deoarece se actualizeaza la fiecare actiune a user-ului, si ii se pun la dispozitie mai multe functionalitati : acesta poate sa adauge un client nou, sa modifice datele unui client deja existent, sau sa stearga un client din lista.

**O imagine care conține masă

Descriere generată automat**

**CLASA ‘MESSAGE’**

O imagine care conține text

Descriere generată automatAceastă clasă a fost creată pentru a afișa un mesaj de eroare în cazul în care utilizatorul introduce un set de date invalid, sau pentru a afisa un mesaj in cazul in care se creeaza o comanda noua sau se adauga un produs sau un client valid, pentru ca utilizatorul sa stie ca acea operatie s-a efectuat cu succes.

**CLASA ‘Start’**

Toate programele Java trebuie să aibă un punct de intrare, care este întotdeauna metoda main (). Ori de câte ori programul este apelat, acesta execută automat mai întâi metoda main ().

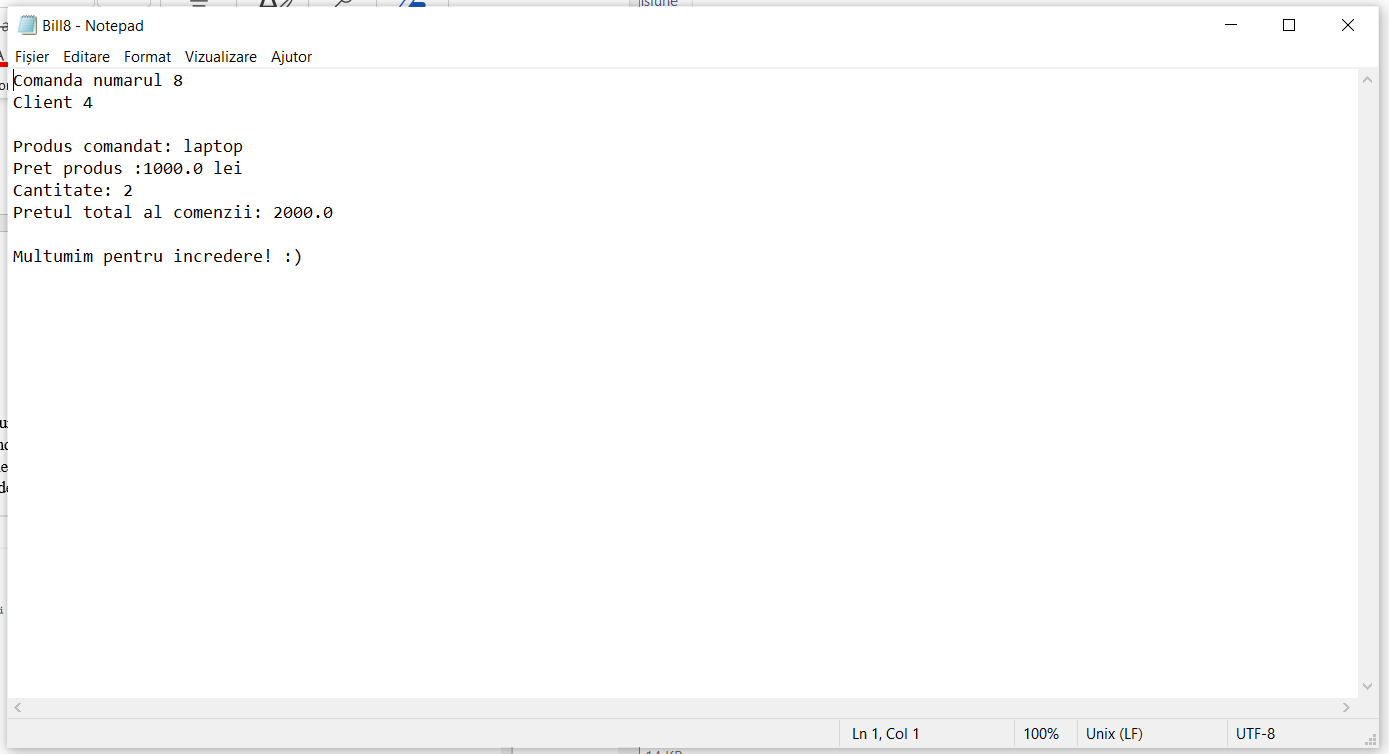
O imagine care conține text

Descriere generată automatPrincipala [metoda](https://www.greelane.com/link?to=method-signature-2034235&lang=ro&alt=https://www.thoughtco.com/method-signature-2034235&source=main-class-2034233) poate apărea în orice clasă care face parte dintr - o aplicație, dar în cazul în care cererea este un complex care conține mai multe fișiere, este comun pentru a crea o clasă separată doar pentru main (). Clasa principală poate avea orice nume, deși în mod obișnuit va fi numită doar „Start”, la fel cum s-a procedat și în proiectul de față.

# Rezultate

# 

Atunci cand un client face o comanda, se va genera automat o factura cu detaliile comenzii (numarul de comanda, id-ul clientului, numele, pretul si cantitatea produsului comandat si totalul pe care clientul trebuie sa il plateasca). De exemplu, data utilizatorul cu id-ul 4 isi comanda doua laptopuri, fiecare in valoare de 1000 de lei, atunci se va genera factura cu un total de 2000.0 lei.



# Concluzii

Acest aplicatie online pentru comandarea si gestionarea produselor din mediul unui depozit este foarte utila în ceea ce privește scopul în care a fost creat, oferind o rezolvare pentru clasicele carnetele pe care se scrie de mana, care necesita timp in plus si duc la intarzieri in coordonarea angajatilor si in livrarea comenzilor. Aceasta are o interfață simplă, prietenoasă, ușor de înțeles și de folosit de către utlizator.

În plus, această aplicație se respecta conceptele fundamentale ale programării orientate pe obiect. S-au folosit metode de încapsulare, clase și obiecte. Toate obiectele care prezintă caracteristici și comportamente similar au fost grupate intr-o singura clasă.

De asemenea, modul în care a fost creata aceasta aplicatie pentru comandare si gestionare online o face ușor de îmbunătățit pe viitor, fiind unele funcții care nu sunt oferite în prezent și care pot fi implementate ulterior. De exemplu, produsele impreuna cu numarul de exemplare droit pot fi adaugate intr-un cos temporar, astfel sa poata fi cumparate mai multe produse diferite pe aceeasi comanda.

In plus, userii aplicatiei ar putea sa poata vedea comenzile care s-au facut pana in acest moment, astfel ar fi mai usor pentru angajati si sa le poata sterge sau sa le modifice, in cazul in care clientul s-a razgandit asupra numarului de produse comandate sau chiar nu mai vor produsele deloc.

Pe langa asta, ar putea fi implementata o functie prin care userul sa poata cauta anumite produse, client, sau anumite comenzi, fie printr-un cuvant cheie, fie printr-un camp caracteristic acestora (id, pret, cantitate, adresa, email, varsta, data in care a fost creata comanda).

# Bibliografie

<https://www.baeldung.com/java-jdbc>

<http://www.mkyong.com/jdbc/how-to-connect-to-mysql-with-jdbc-driver-java/>

<https://dzone.com/articles/layers-standard-enterprise>

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

<https://www.baeldung.com/java-pdf-creation>

<https://www.baeldung.com/javadoc>

<https://dev.mysql.com/doc/workbench/en/wb-admin-export-import-management.html>