MINISTERUL EDUCAȚIEI ȘI CERCETĂRII ȘTIINȚIFICE

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE
CATEDRA CALCULATOARE

DOCUMENTAȚIE

TEMA *3*

GESTIONAREA COMENZILOR

CIOBAN FABIAN-REMUS
30223

CUPRINS

1. Obiectivul temei

Scopul acestei teme constă în implementarea unei aplicații cu interfață grafică pentru gestionarea comenzilor într-un depozit. Pentru stocarea datelor se va utiliza o bază de date MySQL, iar arhitectura folosită trebuie să fie "Layered".

Utilizatorul trebuie să poată efectua diverse operații pe date în fiecare fereastră:

- În fereastra "Clienti", utilizatorul trebuie să poată vizualiza toți clienții, să poată adăuga, modifica și șterge clienți.
- În fereastra "Produse", utilizatorul trebuie să poată vizualiza toate produsele, să poată adăuga, modifica și șterge produse.
- În fereastra "Comenzi", utilizatorul trebuie să poată vizualiza toate comenzile și să poată adăuga comenzi.

De asemenea, a fost necesară documentarea codului cu comentarii Javadoc și generarea unui fișier SQL dump care să conțină codul pentru crearea și popularea bazei de date.

2. Analiza problemei, modelare, scenarii, cazuri de utilizare

Analiza:

Pentru a putea realiza diferite seturi de operații, am dezvoltat trei ferestre separate, fiecare având scopul de a gestiona datele referitoare la clienți, produse și comenzi într-un depozit. Am implementat funcționalități de adăugare, modificare și ștergere a acestor date. Pentru stocarea informațiilor, am utilizat o bază de date MySQL, iar programul se conectează la aceasta prin intermediul unui nume de utilizator și a unei parole.

Modelare:

Structura de date "Clienti" reprezintă informațiile despre un client.
Structura de date "Produse" reprezintă informațiile despre un produs.
Structura de date "Comenzi" reprezintă informațiile despre o comandă.

Scenarii:

Funcționalitatea de modificare a comenzilor nu a fost implementată, deoarece nu are un scop clar în contextul dat.

În cazul în care se încearcă adăugarea unei comenzi care ar duce la un stoc negativ al unui produs, se va afișa o fereastră de eroare, iar comanda respectivă nu va fi efectuată.

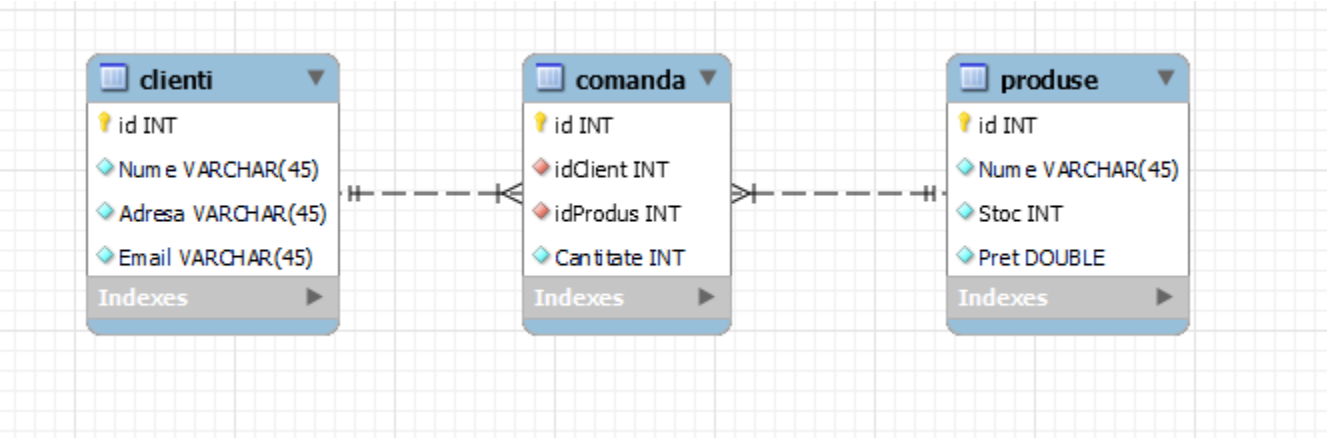
De asemenea, nu este posibilă ștergerea unui produs sau a unui client atunci când aceștia sunt implicați în cel puțin o comandă. Această restricție este impusă de utilizarea cheilor străine în cadrul bazei de date.

Utilizare:

User : => (Clienti) : Afișează , Adaugă , Modifică , Șterge
=> (Produse) : Afișează , Adaugă , Modifică , Șterge
=> (Comenzi) : Afișează , Adaugă

3. Proiectare

În MySql tabelele împreună cu legăturile arată așa :



Proiectul este ierarhizat pe baza arhitecturii „Layered”:

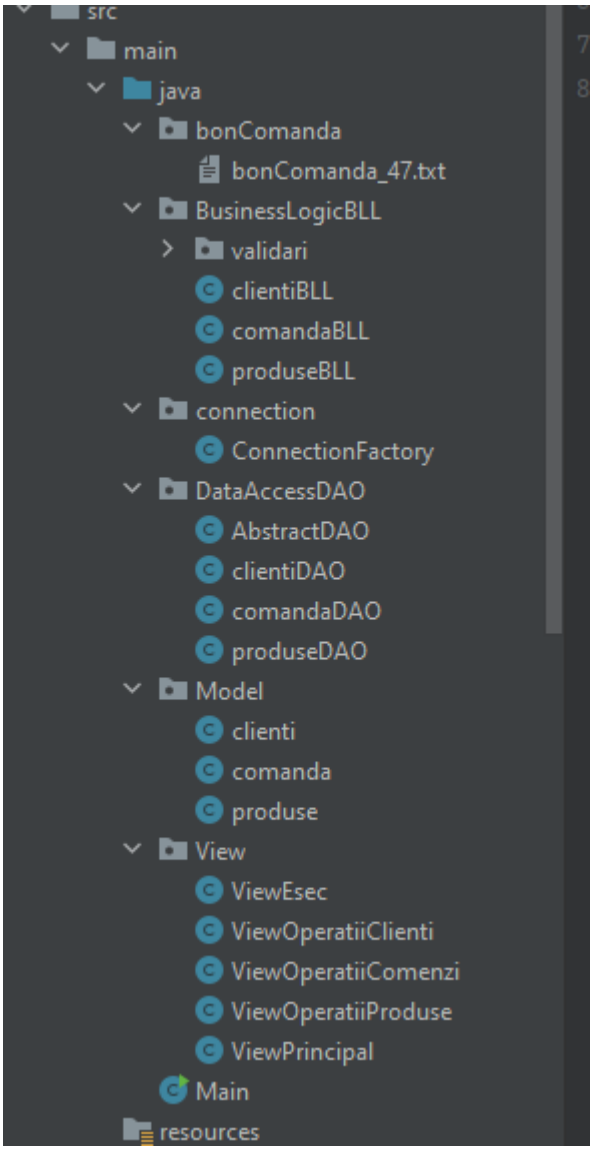
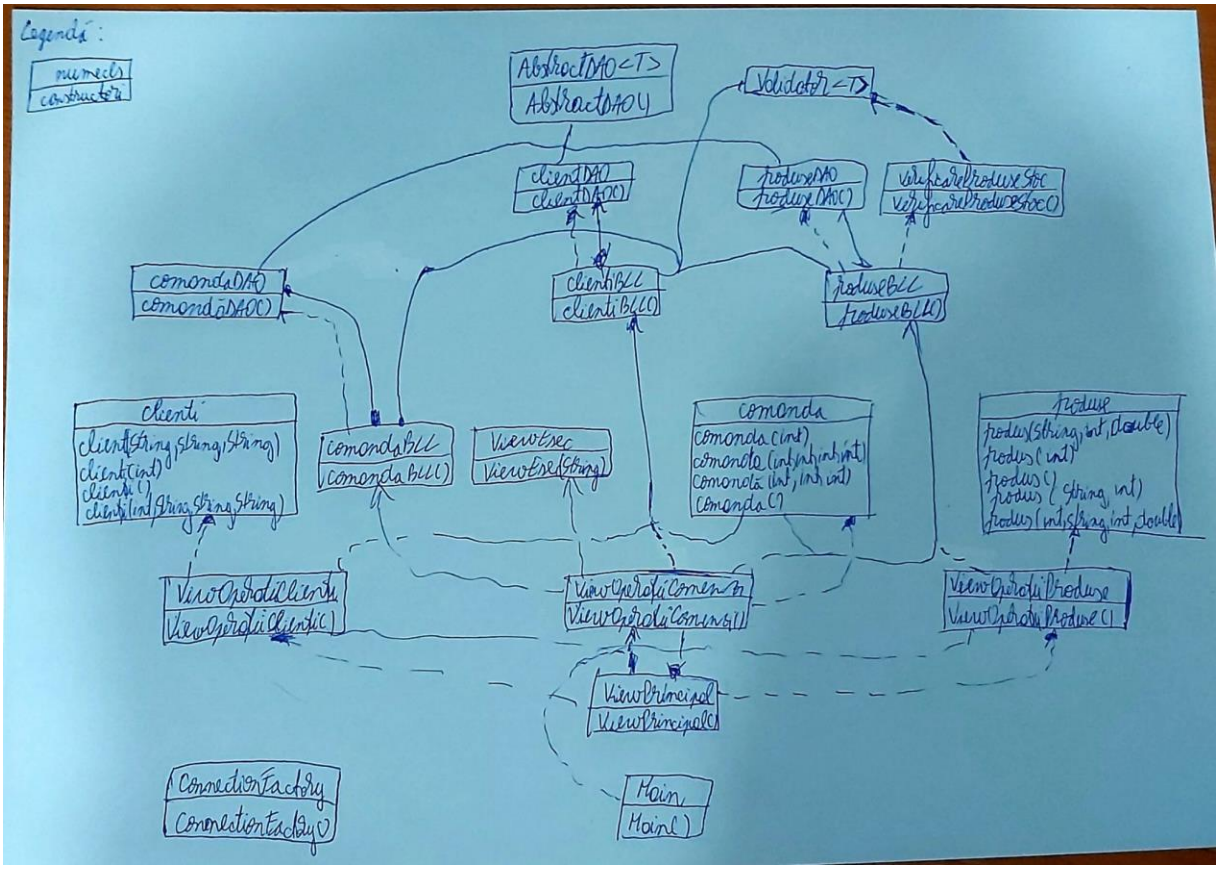
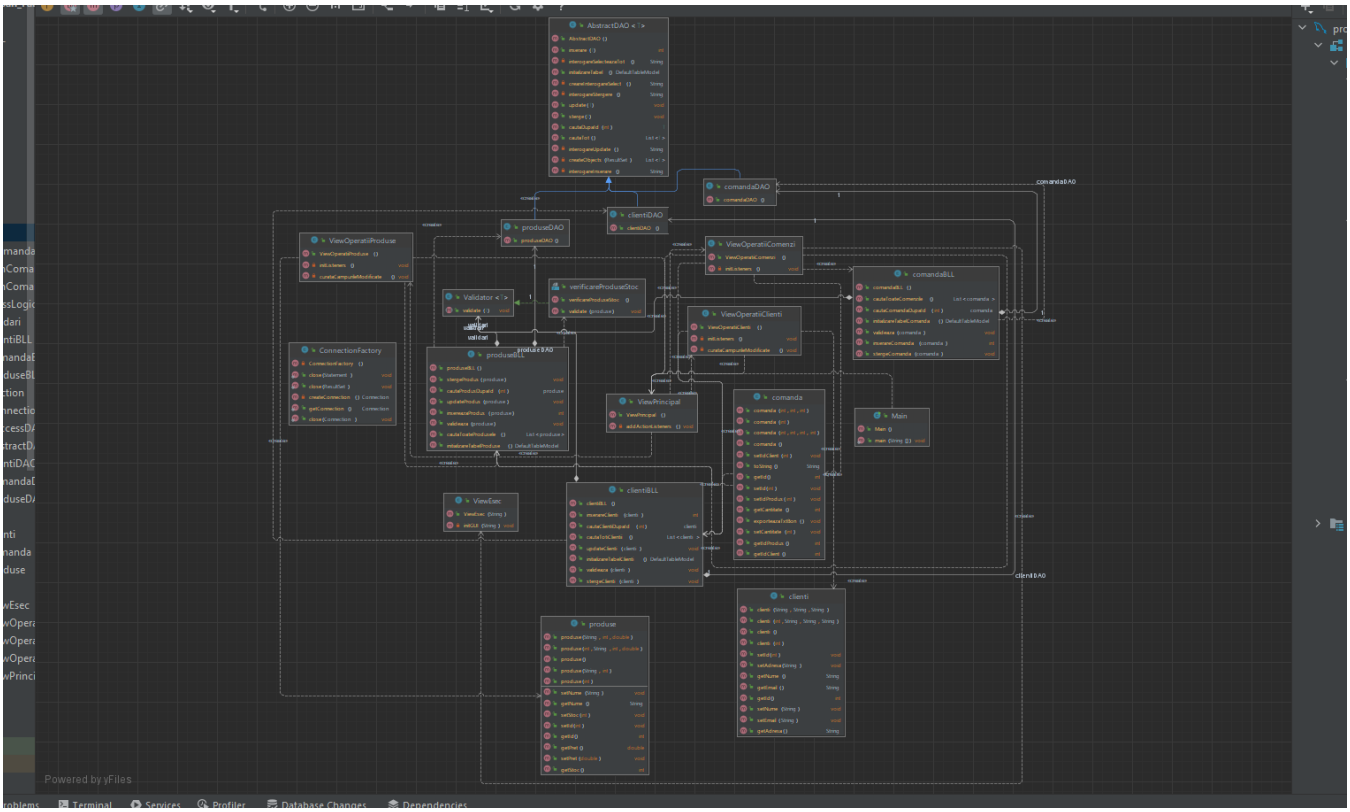


Diagrama UML

Unified Modeling Language sau UML pe scurt este un limbaj standard pentru descrierea de modele si specificatii pentru software. UML a fost la bază dezvoltat pentru reprezentarea complexității programelor orientate pe obiect, al căror fundament este structurarea programelor pe clase, și instanțele acestora (numite și obiecte). Cu toate acestea, datorită eficienței și clarității în reprezentarea unor elemente abstracte, UML este utilizat dincolo de domeniul IT.


Legenda:
numecls
constructori
AbstractDAO<T>
AbstractDAO()
Validator<T>
clientDAO
clientiDAO()
produseDAO
produseDAO()
verificareProduseStoc
verificareProduseStoc()
comandaDAO
comandaDAO()
clientiBLL
clientiBLL()
produseBLL
produseBLL()
clienti
comandaBLL
comandaBLL()
ViewSec
ViewSec(String)
comanda
produse
ViewOperatiiClienti
ViewOperatiiClienti()
ViewOperatiiComenzi
ViewOperatiiComenzi()
ViewOperatiiProduse
ViewOperatiiProduse()
ViewPrincipal
ViewPrincipal()
ConnectionFactory
ConnectionFactory()
Main
Main()

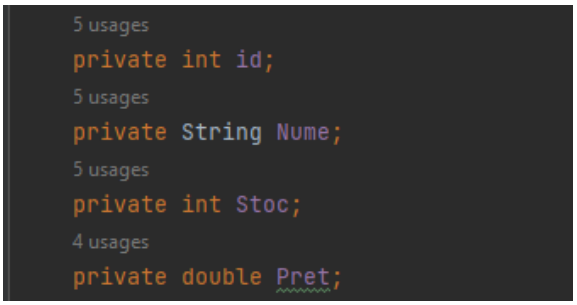

AbstractDAO<T>
comandaDAO
clientiDAO
produseDAO
ViewOperatiiProduse
verificareProduseStoc
Validator<T>
ViewOperatiiComenzi
comandaBLL
ViewOperatiiClienti
ConnectionFactory
produseBLL
ViewPrincipal
comanda
Main
ViewSec
clientiBLL
clienti
produse
Powered by yFiles
Terminal
Services
Profiler
Database Changes
Dependencies

4. Implementare

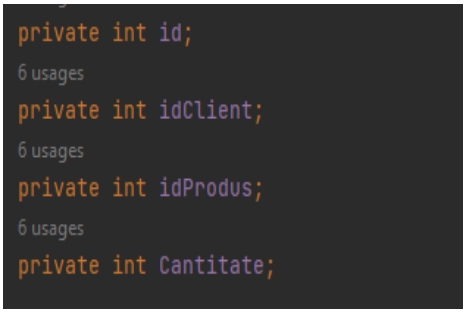
Clasa `**ViewPrincipal**` are rolul de a reprezenta fereastra principală a programului, care este deschisă automat în momentul rulării acestuia. Această fereastră este punctul central de interacțiune cu utilizatorul, oferindu-i posibilitatea de a naviga și de a selecta tipurile de operații pe care dorește să le efectueze în cadrul aplicației. Prin intermediul clasei `ViewPrincipal`, utilizatorul poate explora și accesa funcționalitățile referitoare la produse, clienți și comenzi. Aceasta reprezintă un hub de comandă și un panou de control intuitiv, unde utilizatorul poate alege cu ușurință tipul de operații pe care dorește să le desfășoare. Această abordare oferă utilizatorului flexibilitatea de a interacționa cu diferite aspecte ale programului în funcție de necesități și de contextul în care se află. De exemplu, utilizatorul poate alege să efectueze operații de gestionare a produselor atunci când este interesat de stocuri sau actualizări de prețuri, să acceseze operații legate de clienți pentru a gestiona informații despre clienții existenți sau să execute operații de gestionare a comenzilor pentru a procesa și a monitoriza comenzi noi sau în desfășurare. Prin furnizarea acestei ferestre principale și a posibilității de alegere a tipurilor de operații, clasa `ViewPrincipal` facilitează interacțiunea utilizatorului cu programul și contribuie la o experiență eficientă și personalizată în cadrul aplicației.

Clasa `**ViewOperatiiX**` reprezintă o fereastră specifică în cadrul aplicației noastre, care permite utilizatorului să efectueze diferite operații pe entitatea X. Această fereastră furnizează un mediu interactiv în care utilizatorul poate adăuga, modifica sau șterge înregistrări sau informații legate de X, oferindu-i astfel un nivel ridicat de control și flexibilitate asupra datelor respective. Prin intermediul clasei `ViewOperatiiX`, utilizatorul are posibilitatea de a interacționa cu entitatea X într-un mod intuitiv și eficient. Aceasta pune la dispoziție opțiuni clare și bine definite, cum ar fi adăugarea de noi înregistrări, modificarea celor existente sau eliminarea lor din sistem. Adăugarea presupune capacitatea utilizatorului de a introduce date relevante despre X într-un mod structurat și coerent, asigurând astfel păstrarea integrității și coerenței informațiilor. Modificarea permite utilizatorului să actualizeze înregistrări existente pentru a reflecta schimbări sau actualizări în ceea ce privește X. În cele din urmă, ștergerea oferă posibilitatea utilizatorului de a elimina înregistrări sau informații despre X care nu mai sunt relevante sau necesare. Clasa `ViewOperatiiX` are un rol esențial în a oferi utilizatorului acces și control asupra operațiilor pe X, contribuind astfel la eficiența și eficacitatea procesului de gestionare a acestor date. Această fereastră specifică reprezintă o componentă-cheie a interfeței aplicației noastre și permite utilizatorului să gestioneze cu ușurință entitatea X în conformitate cu cerințele și preferințele sale.

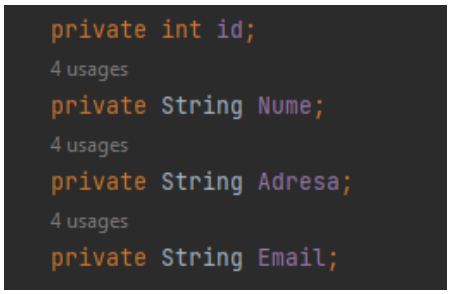
Clasa produse : Clasa care reprezinta produsul. Ea contine urmatoarele campuri:



Clasa comanda : Clasa care reprezinta comanda. Ea contine urmatoarele campuri:



Clasa clienti : Clasa care reprezinta clientul. Ea contine urmatoarele campuri:



Clasa `**AbstractDAO<T>**` reprezintă o clasă abstractă ce furnizează metode de acces la date pentru diferite entități în cadrul aplicației noastre. Această clasă servește drept bază pentru clasele `ClientiDAO`, `ComandaDAO` și `ProduseDAO`, care moștenesc funcționalitatea și extind comportamentul definit în clasa `AbstractDAO<T>`. Prin intermediul clasei `AbstractDAO<T>`, se pune la dispoziție un set de metode standardizate pentru interacțiunea cu baza de date în ceea ce privește entitatea specificată de tipul generic `T`. Aceste metode includ, de obicei, funcționalități de creare, citire, actualizare și ștergere (CRUD) pentru entitatea respectivă. De exemplu, `ClientiDAO`, `ComandaDAO` și `ProduseDAO` utilizează funcționalitățile moștenite din clasa `AbstractDAO<T>` pentru a efectua operații specifice entităților clienți, comenzi și produse, respectiv. Aceste clase extind clasa abstractă și adaugă logica specifică pentru fiecare entitate în parte. Prin utilizarea acestei abordări, se obține o structură modulară și ușor de întreținut pentru gestionarea datelor în aplicația noastră. Clasa abstractă

`AbstractDAO<T>` asigură o implementare centralizată a funcționalității de acces la date, în timp ce clasele concrete specializate precum `ClientiDAO`, `ComandaDAO` și `ProduseDAO` extind această funcționalitate și se concentrează pe particularitățile entităților lor specifice. Această arhitectură facilitează reutilizarea codului, reduce duplicarea și promovează coerența în gestionarea datelor în cadrul aplicației noastre. Prin intermediul clasei `AbstractDAO<T>` și a claselor concrete derivate, obținem un model flexibil și scalabil pentru gestionarea eficientă a datelor în diverse entități ale aplicației noastre.

Clasa `**ConnectionFactory**` reprezintă o componentă esențială în cadrul programului nostru Java, având responsabilitatea de a stabili și gestiona conexiunea între aplicație și baza de date. Această clasă are un rol crucial în asigurarea unei comunicări eficiente și securizate între programul Java și baza de date. Prin intermediul clasei `ConnectionFactory`, se realizează toate operațiile necesare pentru a stabili și menține o conexiune activă cu baza de date. Printre responsabilitățile `ConnectionFactory` se numără: Furnizarea parametrilor de configurare pentru conexiunea la baza de date, precum adresa IP, portul, numele bazei de date, utilizatorul și parola.; Crearea și gestionarea obiectelor de conexiune la baza de date, utilizând API-ul specific pentru manipularea conexiunilor în Java ; Validarea și verificarea stării conexiunii, pentru a se asigura că aceasta este întotdeauna disponibilă și funcțională.; Tratarea și gestionarea erorilor și excepțiilor care pot apărea în timpul comunicării cu baza de date, oferind un flux de lucru sigur și robust. Clasa `ConnectionFactory` este concepută pentru a izola detaliile tehnice legate de conectarea la baza de date, oferind astfel o abstractizare și modularitate mai mare în ceea ce privește operațiile de bază legate de accesul la date. Aceasta permite programului nostru să se concentreze pe logica aplicației și să beneficieze de o gestionare eficientă și centralizată a conexiunii la baza de date. În concluzie, clasa `ConnectionFactory` reprezintă o componentă crucială în arhitectura aplicației noastre Java, asigurând o conexiune sigură și eficientă între program și baza de date, contribuind astfel la funcționalitatea și performanța globală a aplicației noastre.

Clasa '**XBLL'** reprezintă o componentă esențială în cadrul aplicației noastre, având rolul de a gestiona logica de afaceri asociată accesului la date pentru entitatea X. Această clasă reprezintă un strat intermediar între componenta de acces la date și celelalte module ale aplicației care utilizează informațiile despre entitatea X. Prin intermediul clasei XBLL, se implementează și se organizează logica de afaceri specifică, asigurând coerența și corectitudinea operațiilor efectuate asupra datelor despre entitatea X. Principalele responsabilități ale clasei XBLL include : Validarea și verificarea datelor de intrare înainte de a fi procesate și salvate în baza de date. Implementarea regulilor și restricțiilor de afaceri asociate entității X, asigurând conformitatea cu cerințele și politicile specifice. Definirea și implementarea operațiilor specifice asupra datelor despre entitatea X, precum adăugare, modificare, ștergere și interogare. Manipularea și

gestionarea excepțiilor și erorilor care pot apărea în timpul procesării datelor, oferind un flux de lucru sigur și controlat. Prin intermediul clasei XBLL, se asigură separarea clară între logica de afaceri și detaliile specifice de acces la date. Această abordare permite flexibilitate în gestionarea și adaptarea logicii de afaceri în funcție de cerințele aplicației, fără a fi nevoie de modificări în mod direct în componenta de acces la date.

Clasa `**VerificareProduseStoc**` reprezintă o componentă importantă în cadrul aplicației noastre și are rolul de a valida stocul produselor. Această clasă implementează interfața `Validator<Produse>`, ceea ce înseamnă că definește și implementează o metodă specifică pentru a verifica dacă un produs are un stoc pozitiv. Prin intermediul clasei `VerificareProduseStoc`, se aplică logica necesară pentru a asigura că stocul fiecărui produs este într-o stare validă. Această clasă se concentrează exclusiv pe verificarea stocului și nu conține alte funcționalități suplimentare. Metoda implementată în clasa `VerificareProduseStoc` analizează stocul unui produs specific și determină dacă acesta are o cantitate pozitivă. În cazul în care stocul produsului este mai mare sau egal cu zero, verificarea este considerată cu succes și metoda returnează rezultatul corespunzător. Utilizarea clasei `VerificareProduseStoc` permite aplicației noastre să se asigure că stocul produselor este întotdeauna într-o stare validă, conform regulilor și restricțiilor stabilite. Aceasta adaugă un nivel suplimentar de validare și integritate a datelor, contribuind la funcționarea corectă și coerentă a sistemului în ansamblu.

5. Rezultate

Programul a fost implementat cu succes și a fost testat în mod riguros pentru a verifica funcționarea corectă. Pe parcursul testelor, nu au fost detectate sau raportate erori sau excepții care să nu fie în concordanță cu comportamentul normal al programului.

Această rezultată este extrem de satisfăcătoare, deoarece demonstrează că programul este fiabil și că este capabil să se descurce cu diferite situații și date de intrare fără să genereze erori sau comportamente neașteptate. Absența excepțiilor neobișnuite reprezintă un indicator al stabilității și robusteții programului, ceea ce înseamnă că utilizatorii vor beneficia de o experiență fluentă și fără întreruperi în timpul utilizării acestuia.

6. Concluzii

Prin intermediul acestui proiect, am avut ocazia să dezvolt și să aprofundez cunoștințele mele în lucrul cu JDBC (Java Database Connectivity). Prin intermediul JDBC, am putut interacționa cu baze de date relaționale și am învățat cum să efectuez diverse operații, precum inserarea, actualizarea sau ștergerea datelor într-o bază de date

Pe lângă lucrul cu JDBC, am avut o altă oportunitate valoroasă de învățare prin explorarea conceptului de reflexie în Java. Reflexia reprezintă capacitatea unui program de a examina și de a modifica structura, comportamentul și caracteristicile unei clase la timpul de execuție. Am înțeles cum să accesez și să manipulez dinamic clase, metode, atribute și constructori utilizând mecanismele oferite de reflexie.

Această experiență mi-a adus beneficii semnificative în dezvoltarea mea ca programator, deoarece m-am familiarizat cu instrumente și tehnici avansate care îmi permit să lucrez cu baze de date și să explorez și să modific dinamic structura și comportamentul claselor. Aceste cunoștințe îmi oferă posibilitatea de a crea aplicații mai flexibile și mai adaptabile la schimbări, îmbunătățind astfel calitatea și eficiența codului pe care îl scriu.

7. Bibliografie

1.YouTube
2.StackOverflow
3.W3School
4.Oracle