

Tehnici Fundamentale de Programare

Tema 3

Manager de comenzi

Proiect realizat de:

Moldovan Andrei

Grupa 30227

Profesor îndrumător: Dan Mitrea

Cuprins

[1. Obiectivul temei 2](#_Toc72919408)

[2. Analiza problemei, modelare, scenarii, cazuri de utilizare 2](#_Toc72919409)

[3. Proiectare 4](#_Toc72919410)

[4. Implementarea 5](#_Toc72919411)

[5. Rezultate 9](#_Toc72919412)

[6. Concluzii 9](#_Toc72919413)

[7. Bibliografie 9](#_Toc72919414)

# Obiectivul temei

Principalul obiectiv al acestei teme a fost de a implementa un manager de comenzi care a trebuit sa conțină o interfață grafică prin care utilizatorul să poată interacționa cu baza de date. Acțiunile pe care utilizatorul le putea realiza erau adăugarea, ștergerea, actualizarea și afișarea datelor din tabelele Clienți, Produse și Comenzi.

Obiective secundate:

* Să se utilizeze tehnici de reflexie pentru a crea tabelele necesare afișării, cât și crearea unor liste care vor fi utilizate pentru procesarea datelor
* Proiectul să fie structurat pe nivele, proiectul având cele puțin 4 pachete
* Să se genereze automat un bon după realizarea fiecărei comenzi

# Analiza problemei, modelare, scenarii, cazuri de utilizare

Diagram

Description automatically generatedAvem mai multe cerințe funcționale care necesită să fie tratate, precum:

* Introducerea datelor de către utilizator
* Posibilitatea de a alege în ce tabel să se realizeze operațiile
* Adăugarea unei noi linii în tabelul selectat
* Ștergerea unui linii după id-ul său în tabelul selectat
* Actualizarea datelor unei linii cu id-ul dorit din tabelul selectat
* Vizualizarea tuturor datelor din tabelul selectat
* Crearea bonusului fiscal rezultat în urma operației de adăugare sau actualizare a datelor din tabelul comenzilor

1. Caz de utilizare: Introducerea datelor

Actor principal: Utilizatorul

Principalul scenariu de funcționare:

* Utilizatorul introduce datele cu care dorește să se realizeze operația pe tabelul selectat
* Utilizatorul apasă pe operația dorită
* Aplicația verifică datele introduse
* Se realizează operația

Secvența alternativă:

* Utilizatorul introduce datele
* Se verifică datele, dar nu trec de validatori
* Se afișează mesajul de eroare corespunzător și utilizatorul va trebui sa reintroducă datele

1. Caz de utilizare: Adăugarea datelor

Actor principal: Utilizatorul

Principalul scenariu de funcționare:

* Datele introduse sunt corecte, iar utilizatorul apasă pe butonul de adăugare
* Aplicația se va conecta la baza de date
* Aplicația va adăuga datele în tabelul selectat

Secvența alternativă:

* Datele introduse sunt corecte, iar utilizatorul apasă pe butonul de adăugare
* Aplicația nu reușește să se conecteze la baza de date / Interogarea dorită nu se poate realiza din cauza constrângerilor
* Se afișează mesajul de eroare corespunzător

1. Caz de utilizare: Ștergerea datelor

Actor principal: Utilizatorul

Principalul scenariu de funcționare:

* Datele introduse sunt corecte, iar utilizatorul apasă pe butonul de șterge
* Aplicația se va conecta la baza de date
* Aplicația va șterge datele în tabelul selectat

Secvența alternativă:

* Datele introduse sunt corecte, iar utilizatorul apasă pe butonul de șterge
* Aplicația nu reușește să se conecteze la baza de date / Interogarea dorită nu se poate realiza din cauza constrângerilor
* Se afișează mesajul de eroare corespunzător

1. Caz de utilizare: Actualizarea datelor

Actor principal: Utilizatorul

Principalul scenariu de funcționare:

* Datele introduse sunt corecte, iar utilizatorul apasă pe butonul de actualizare
* Aplicația se va conecta la baza de date
* Aplicația va actualiza datele în tabelul selectat

Secvența alternativă:

* Datele introduse sunt corecte, iar utilizatorul apasă pe butonul de actualizare
* Aplicația nu reușește să se conecteze la baza de date / Interogarea dorită nu se poate realiza din cauza constrângerilor
* Se afișează mesajul de eroare corespunzător

1. Caz de utilizare: Vizualizarea datelor

Actor principal: Utilizatorul

Secvența principală de funcționare:

* Utilizatorul apasă pe butonul de afișare a datelor
* Aplicația se va conecta la baza de date
* Aplicația va extrage tabelul de date necesar afișării tabelului selectat
* Se deschide o nouă fereastră cu tabelul

Secvența alternativă:

* Utilizatorul apasă pe butonul de afișare a datelor
* Aplicația nu reușește să se conecteze la baza de date
* Se afișează mesajul de eroare corespunzător

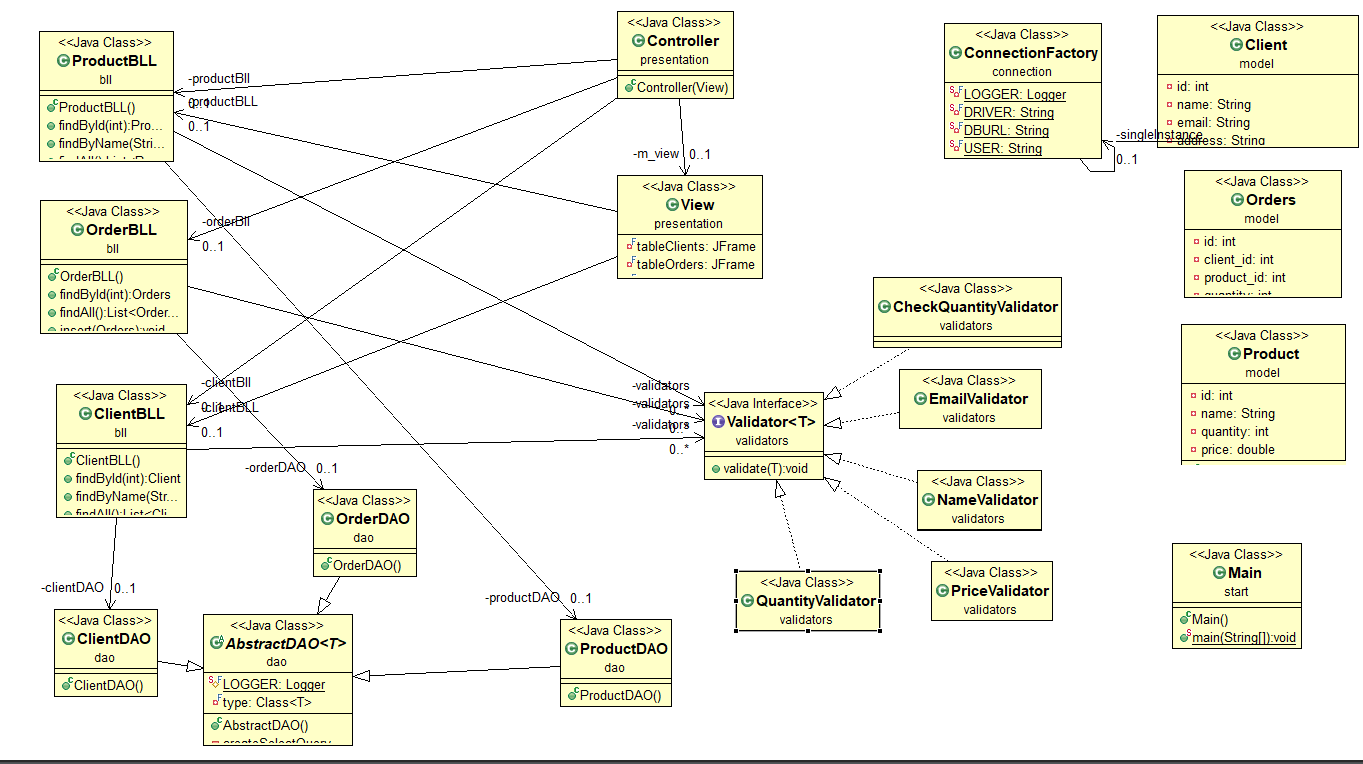
1. Caz de utilizare: Crearea bonului fiscal

Actor principal: Utilizatorul

Secvența principală de funcționare:

* Utilizatorul a realizat operația de adăugare / actualizare
* Aplicația realizează cu succes operația
* Se realizează o chitanță cu detaliile despre comanda făcută

# Proiectare

Diagrama UML:

Soluția pe care am utilizat-o în rezolvarea problemei propuse la tema curentă conține implementarea a 20 de clase diferite împărțite în 6 pachete diferite pentru a respecta arhitectura structurată pe nivele. Astfel pentru a respecta structura dorită au fost necesare 6 pachete.

Primul pachet este împărțit mai conține încă un pachet unde se regăsesc validatorii, clase care sunt necesare pentru verificarea datelor introduse de către utilizator, iar clasele de BLL vor conține metode prin care se vor realiza verificările datelor și introducerea lor în baza de date.

Al doilea conține clasa pentru conectarea aplicației la baza de date.

Al treilea pachet conține partea crearea, implementarea și executarea interogărilor pe baza de date, iar mai apoi extinderea clasei abstracte pentru fiecare tabel dorit.

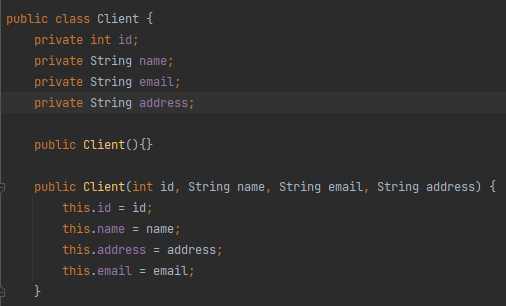
Al patrulea pachet conține toate clasele care sunt necesare pentru tipurile de date cu care vom lucra și încă o clasă care va genera bonul fiscal.

Al cincilea pachet este cel care va conține clasele care vor implementa interfața grafică pe care utilizatorul o va folosi pentru a folosi aplicația.

Ultimul pachet conține doar clasa Main de unde se va începe rularea programului.

În materie de algoritmi proiectul realizat nu are foarte multe de oferit, complexitatea proiectului fiind realizată în crearea claselor care să utilizeze reflecția pentru generalizarea interogărilor care se vor realiza pe baza de date. Acest subiect se va găsi mai jos la implementarea clasei abstracte din pachetul DAO.

# Implementarea

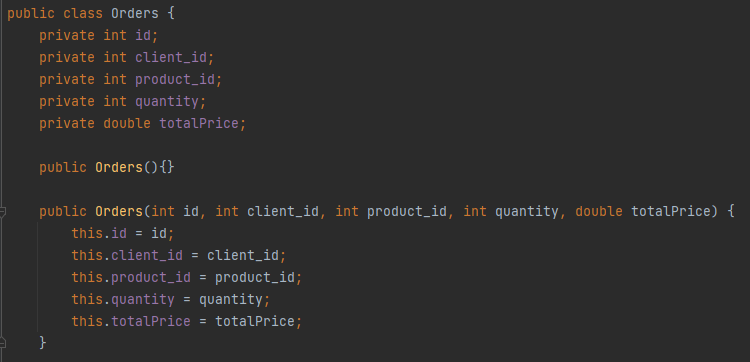
1. Clasa Client

Este clasa prin care se vor crea noi clienți sau pentru a extrage datele despre clienți din baza de date.

Fiecare client are câte un id unic , numele acestuia, un email care va fi necesar să respecte un anumit format și să fie și unic, iar mai apoi și o adresă unde se va livra comanda făcută.

Sunt prezenți doi constructori, unul gol care va fi utilizat pentru generarea clienților care se for afișa din datele din tabelul Clienți aflat în baza de date, iar al doilea va fi utilizat pentru a genera clienți din datele introduse de către utilizator.

În rest, sunt prezente metode de tip getter și setter care să obțină datele din obiectul Clienți sau să le seteze, dar se mai găsește și o clasă necesară pentru afișarea datelor într-un mod mai ușor de citit de către utilizator.

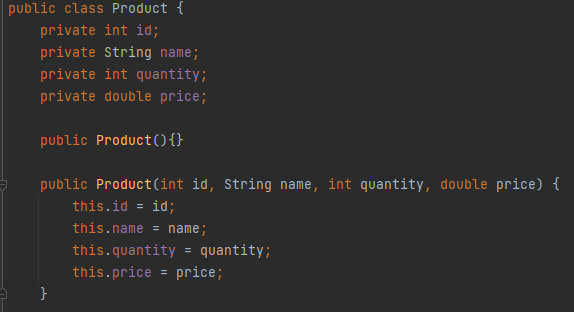
1. Clasa Orders

Este clasa prin care se vor crea noi obiecte de tipul comenzilor, aceasta fiind clasa principală care va fi utilizată cel mai des în proiect.

Fiecare comandă are câte un id unic, id-ul clientului care a făcut comanda, id-ul produsului care a fost comandat, cantitatea care a fost comandată și prețul total al comenzii.

Ca și la clasa precedentă sunt prezenți 2 constructori, unul gol care este utilizat la partea de reflecție din program și unul în care se primesc toți parametrii și se construiește obiectul cu datele respective.

Metodele prezente în această clasă sunt doar de tip getter și setter.



1. Clasa Product

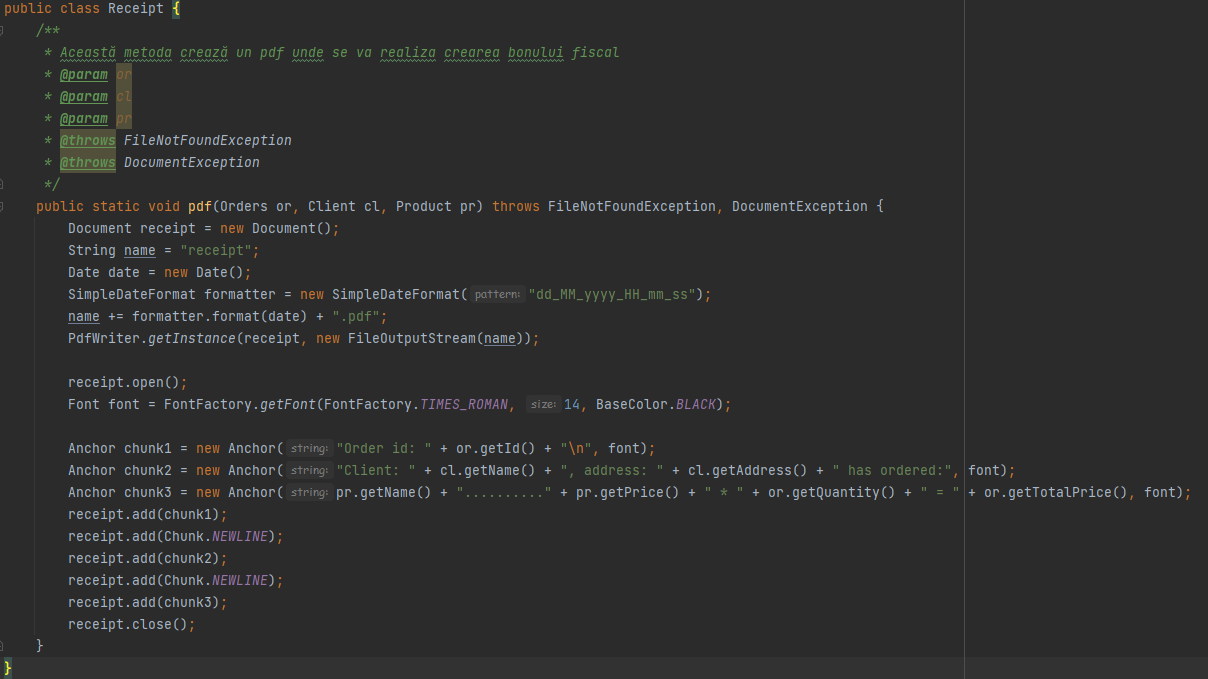
Este clasa utilizată pentru a se crea obiecte noi de tipul produs care ulterior vor fi utilizate în interogări sau pentru afișare.

Toate produsele au câte un id unic, un nume, cantitatea totală de produse care se regăsește în stoc, cât și prețul acestuia per bucată.

Și aici se poate observa prezența a doi constructori, primul fiind utilizat la partea de reflecție, iar cel de-al doilea pentru a crea obiecte noi de tipul Product care să conțină datele dorite.

Metodele prezente în această clasa sunt doar de tipul getter și setter.

1. Clasa Receipt

Aceasta este ultima clasă din pachetul Model. Aceasta este responsabilă de crearea bonului fiscal după fiecare comandă.

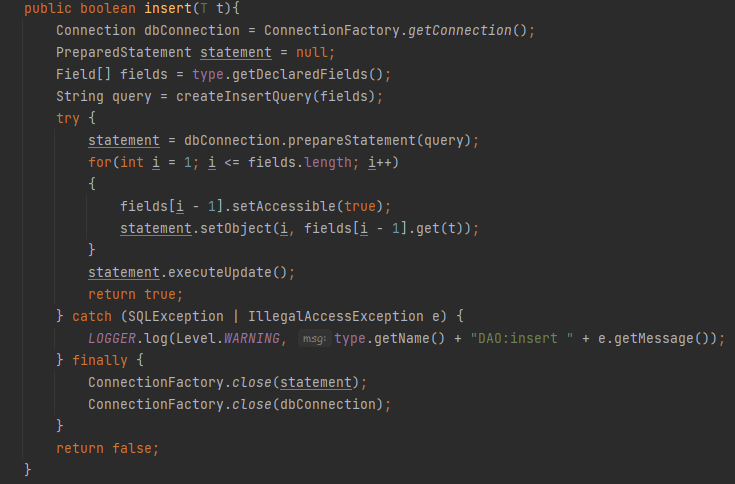
Această metodă îmi generează la fiecare comandă câte un pdf unic. Numele acestui pdf va conține „receipt” și va fi urmat de ora și data la care a fost creată comanda. Afișez datele despre clientul care a făcut comanda, apoi despre produsul pe care la comandat. Alături este cantitatea și prețul total plătit de către client.

La final adaug toate aceste date în fișier, iar mai apoi în închid.

1. Clasa AbstractDAO

Aceasta este clasa care va realiza toate interogările pe baza de date, aici realizându-se reflecția.

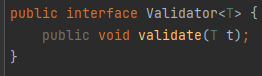
Primele 5 metode sunt utilizate pentru crearea interogărilor care utilizează tehnica reflecției pentru a le genera în funcție de tabelul în care se dorește să se realizeze operația. Cea mai complexă metodă prezentă aici este cea de actualizare, unde se preiau numele coloanelor din tabelul selectat și se pun în ordine pentru a se genera o interogare corect definită pentru fiecare tabel.

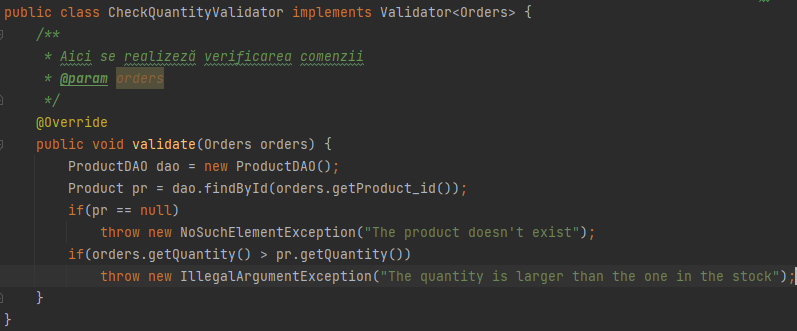
După acestea se regăsesc metodele care realizează interogările pe baza de date. Toate incep prin inițializarea conexiunii la baza de date, a interogării, a rezultatului, cât și crearea interogării prin metodele menționate mai sus. Mai apoi, într-o clauză try/ catch sunt prezente conexiunea la baza de date, crearea interogării prin String-ul realizat anterior, iar mai apoi setarea acestora cu datele dorite. Odată ce s-a trecut prin toți acești pași cu succes se va realiza interogarea pe baza de date. În caz că pe parcursul execuției acestei metode va apărea vreo excepție aceasta va fi tratată în clauza catch, iar în finally se realizează încheierea și închiderea conexiunilor la baza de date.  
 Ultima metodă, dar și cea mai importantă din această clasă este cea care realizează o listă cu toate rezultatele din interogarea făcută. Se inițializează lista, iar apoi se obțin constructoriii din clasa care se utilizează pentru realizarea interogării. Se parcurg metodele pentru realizarea unui obiect ded tipul respectiv, iar mai apoi se va adăuga în lista rezultatelor.

1. Clasele ClientDAO, ProductDAO, OrderDAO

Aceste clase reprezintă implementarea clasei AbstractDAO pentru tabelul pe care se va realiza interogarea, ele fiind goale, deoarece metodele sunt deja implementate prin reflecție în clasă pe care o extind. Spre exemplu, pentru tabelul clienți se va utiliza implementarea clasei ClientDAO.

1. Interfața Validator și validatorii

Aceasta este interfața care se va utiliza pentru a implementa toți validatorii necesari pentru a verifica datele introduse de către utilizator.

* Validator pentru cantitatea introdusă la o comandă

Aici iau id-ul produsului pe care clientul dorește să îl comande, verific dacă acesta există, iar mai apoi dacă stocul îi permite clientului să achiziționeze cantitatea introdusă.

* Validator pentru email

Verifică daca email-ul clientului este unul care să respecte formatul

* Validator pentru preț

Verifică dacă prețul introdus este unul valid, adică nu este negativ.

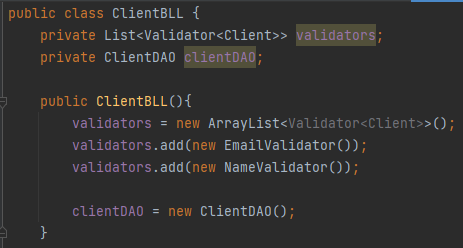
* Validator pentru cantitatea din stoc

Verifică dacă stocul introdus pentru produsul care urmează să fie depozitat este unul valid, adică nu este negativ.

* Validator pentru nume

Cum numele clienților nu poate să conțină numere, utlizez acest validator pentru a verifica dacă utilizatorul aintrodus datele corect.

1. Clasele ClientBLL/ OrderBLL / ProductBLL

Toate aceste clase sunt necesare pentru a verifica dacă datele pe care utilizatorul le-a introdus în aplicație sunt valide.

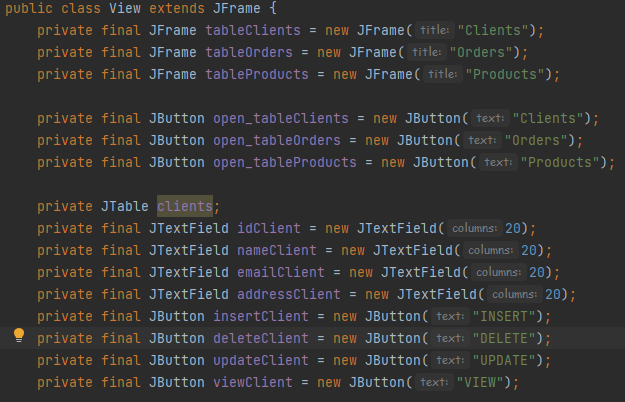
Fiecare are câte o listă de validatori asociați tipurilor de date utilizate in clasa respectivă. iar mai apoi se instanțiază câte o clasă DAO care va realiza interogările.

Aici se regăsesc metode pentru realizarea interogărilor pe datele corespunzătoare clasei, dar care mai întai sunt verificate de validatori, pentru a nu introduce date incorecte în baza de date.

1. Clasa ConnectionFactory

Este clasa care realizează conexiunea la baza de date, practic este care face legătura între aplicația noastră și tabelele pe care utilizatorul va realiza interogările.

1. Clasa View

Este clasa care reprezintă interfața grafică cu care utilizatorul va interacționa pentru a realiza operațiile dorite pe baza de date. Aplicația se deschide în meniul principal care este format din trei butoane. Fiecare din aceste butoane va deshide încă o fereastră nouă corespunzătoare tabelului pe care se va lucra.

1. Clasa Controller

Este clasa care conține implementările pentru butoanele care din aplicație.

1. Clasa Main

Este clasa de unde pornește execuția programului. Se creează view-ul și controller și se continuă execuția.

# Rezultate

Toate testele pe care le-am realizat până acum au mers așa cum mă așteptam. Atunci când email-ul sau numele la clienți nu erau corespunzătoare, se afișa un mesaj de eroare, la fel se întâmplă și pentru celelalte clase, iar pentru datele introduse corect, interogările se realizează în mod corect.

# Concluzii

Cerința mi s-a părut puțin vagă, dar suportul de cod, cât și îndrumările profesorului au fost necesare pentru terminarea temei cu nota maximă, întrucât am învățat o nouă tehnică de a scrie cod pentru cod orientat pe obiect. Reflecția, cât și conectarea și utilizarea bazei de date într-o aplicație sunt niște abilități necesare în crearea aplicațiilor, deci au fost un pas necesar în pentru a trece la următorul pas în scrierea programelor mele.

# Bibliografie

* <https://gitlab.com/utcn_dsrl/pt-layered-architecture/-/tree/master/>
* <https://gitlab.com/utcn_dsrl/pt-reflection-example/-/tree/master>
* <https://www.baeldung.com/java-pdf-creation>