DOCUMENTATIE

TEMA *3*

NUME STUDENT: Ardelean Mitică-Mario
GRUPA: 30224

CUPRINS

1. Obiectivul temei

Obiectivul temei este de a proiecta și implementa o aplicație care are ca scop gestionarea comenzilor unor clienți pentru un depozit.
Obiectivele secundare al acestei teme sunt următoarele:

- Analiza problemei și identificarea cerințelor (Capitolul 2)
- Proiectarea aplicației de gestionare a comenzilor (Capitolul 3)
- Implementarea aplicației de gestionare a comenzilor (Capitolul 4)
- Testarea aplicației de gestionare a comenzilor (Capitolul 5)

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

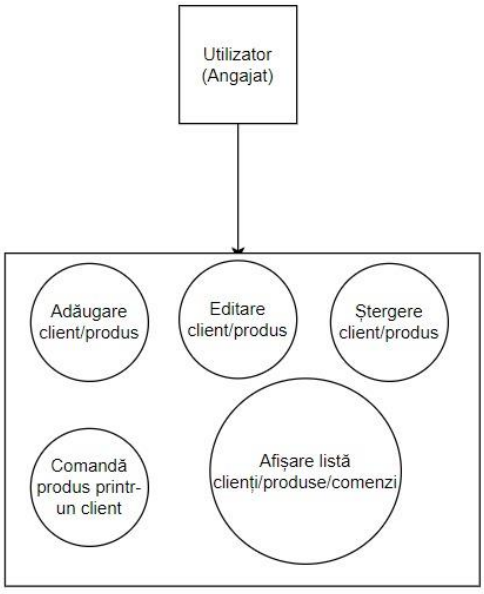
Cerințele funcționale sunt următoarele:

- Aplicația de gestionare a comenzilor ar trebui să permită utilizatorilor să adauge un nou client/produs
- Aplicația de gestionare a comenzilor ar trebui să permită utilizatorilor să editeze un client/produs
- Aplicația de gestionare a comenzilor ar trebui să permită utilizatorilor să șteargă un client/produs
- Aplicația de gestionare a comenzilor ar trebui să permită utilizatorilor să comande un produs prin intermediul unui client deja înregistrat
- Aplicația de gestionare a comenzilor ar trebui să afișeze lista de clienți/produse/comenzi în timp real

Cerințele non-funcționale sunt următoarele:

- Aplicația de gestionare a comenzilor trebuie să fie intuitivă și ușor de utilizat de către utilizator
- Aplicația de gestionare a comenzilor trebuie să afișeze rezultatele într-un timp rezonabil

Diagrama de use-case:



Descrierile de use-case:

- Adăugare client/produs:
  - Actor primar: utilizatorul(angajatul)
  - Principalul scenariu de succes:

1) Utilizatorul selectează opțiunea de adăugare client/produs
2) Aplicația va afișa o interfață unde se vor introduce datele
3) Utilizatorul introduce datele necesare pentru adăugarea unui client/produs și apasă pe butonul 'Save'
4) Aplicația validează datele și, dacă acestea sunt corecte, adaugă clientul/produsul

- Secvență alternativă -> Valori invalide ai parametrilor:
  1) Utilizatorul introduce valori invalide pentru parametrii de adăugare client/produs
  2) Aplicația afișează un mesaj de eroare și solicită utilizatorului să introducă valori valide
  3) Scenariul revine la pasul 1

➢ Editare client/produs:
- Actor primar: utilizatorul(angajatul)
- Principalul scenariu de succes:
  1) Utilizatorul selectează opțiunea de editare client/produs
  2) Aplicația va afișa o interfață unde se vor introduce datele
  3) Utilizatorul introduce datele necesare pentru editarea unui client/produs și apasă pe butonul 'Save'
  4) Aplicația validează datele și, dacă acestea sunt corecte, editează clientul/produsul
- Secvență alternativă -> Valori invalide ai parametrilor:
  1) Utilizatorul introduce valori invalide pentru parametrii de editare client/produs
  2) Aplicația afișează un mesaj de eroare și solicită utilizatorului să introducă valori valide
  3) Scenariul revine la pasul 1

➢ Ștergere client/produs:
- Actor primar: utilizatorul(angajatul)
- Principalul scenariu de succes:
  1) Utilizatorul selectează opțiunea de ștergere client/produs
  2) Aplicația va afișa o interfață unde se va trece id-ul clientului/produsului ce urmează a fi șters
  3) Utilizatorul introduce id-ul pentru ștergerea unui client/produs și apasă pe butonul de ștergere
  4) Aplicația validează datele și, dacă acestea sunt corecte, șterge clientul/produsul
- Secvență alternativă -> Valori invalide ai parametrilor:
  1) Utilizatorul introduce un id invalid pentru parametrii de ștergere client/produs
  2) Aplicația afișează un mesaj de eroare și solicită utilizatorului să introducă un id valid
  3) Scenariul revine la pasul 1

➢ Comandă produs printr-un client:
- Actor primar: utilizatorul(angajatul)
- Principalul scenariu de succes:
  1) Utilizatorul selectează un client din cei valabili
  2) Utilizatorul selectează un produs din cele valabile
  3) Utilizatorul introduce o cantitate
  4) Aplicația validează datele și, dacă acestea sunt corecte, realizează comanda
- Secvență alternativă -> Valori invalide ai parametrilor:
  1) Utilizatorul introduce o cantitate care este mai mare decât stocul disponibil pentru acel produs
  2) Aplicația afișează un mesaj de eroare și solicită utilizatorului să introducă o cantitate validă
  3) Scenariul revine la pasul 1

➢ Afișare listă clienți/produse/comenzi:
- Actor primar: utilizatorul(angajatul(
- Principalul scenariu de succes:
  1) Utilizatorul apasă pe butonul de vizualizare a clienților/produselor/comenzilor
  2) Aplicația înregistrează apăsarea butonului și afișază lista dorită
- Secvență alternativă -> N/A

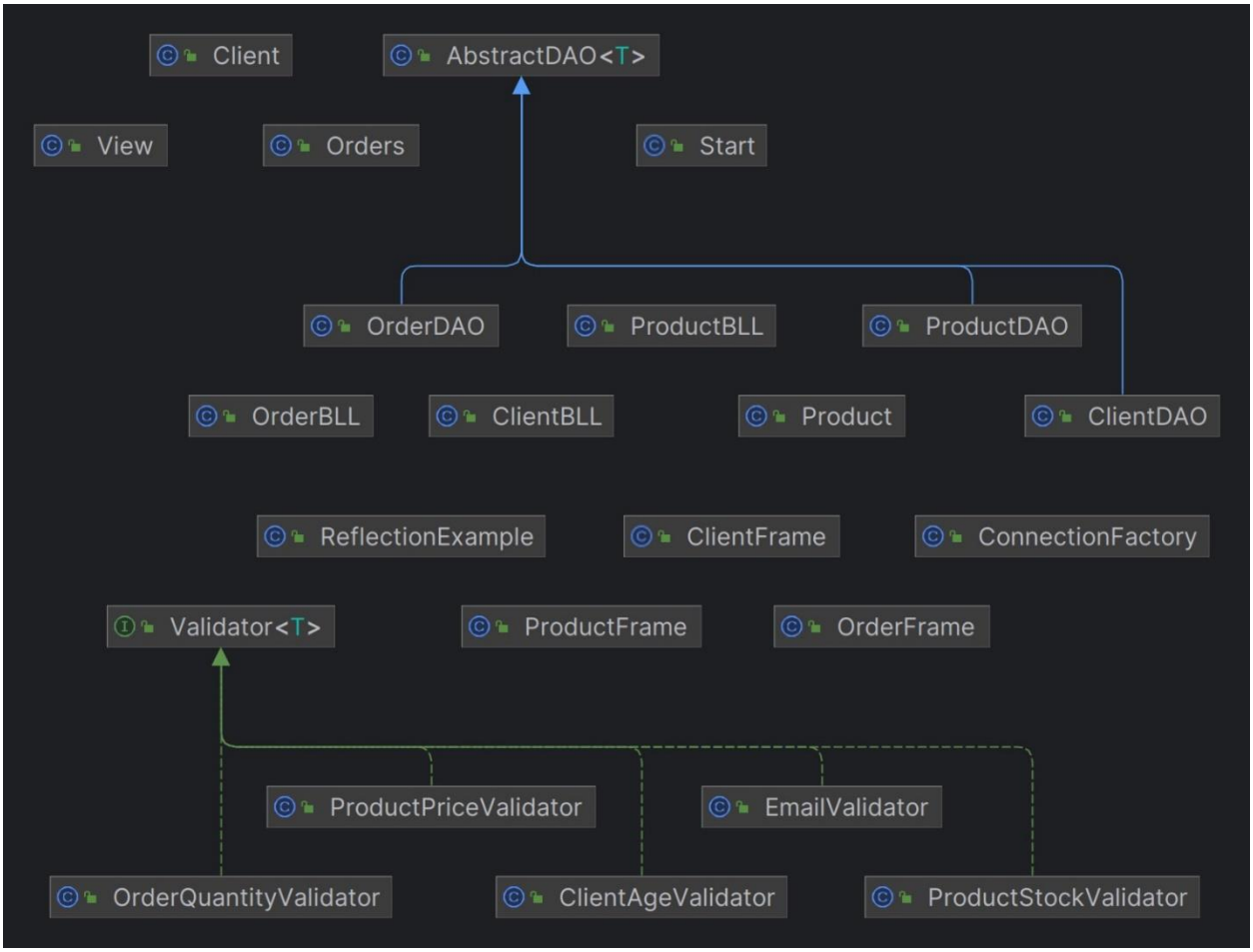
3. Proiectare

1) Proiectarea OOP

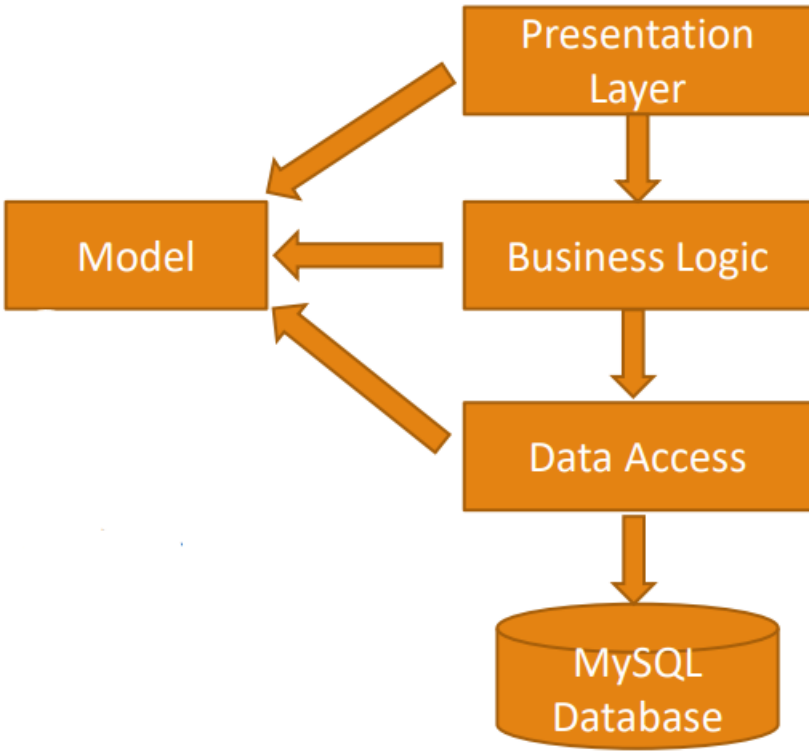
În proiectarea aplicației noastre de simulare folosind OOP, am identificat principalele entități și concepte din domeniul problemei noastre. Acestea includ clienții, produsele și comenzile, fiecare cu atributele sale specifice. Am organizat aceste entități în clase corespunzătoare pentru a gestiona funcționalitățile și datele asociate.

2) Diagramele UML de clase și de pachete

Clase:



Pachete:



3) Structurile de date folosite
Am folosit structurile de date obișnuite pentru o aplicație pe baze de date, dar am mai folosit și structurile de date de tip Field, Constructor, StringBuilder, Logger, Connection, PreparedStatement și ResultSet.
De asemenea, am folosit și JTable, fiind prima structură de tip tabel pe care am folosit-o în interfețele grafice.

4) Interfețe definite:
Interfața 'View': 3 butoane, legătură către celelalte 3 interfețe
Interfața 'ClientFrame': 4 butoane, fiecare cu funcționalitățile ei specifice
Interfața 'ProductFrame': 4 butoane, fiecare cu funcționalitățile ei specifice
Interfața 'OrderFrame': 2 butoane, fiecare cu funcționalitățile ei specifice
5) Algoritmi folosiți:
Algoritmi pentru operații pe baze de date: Reflection. Acesta e folosit pentru a obține informații despre clase și câmpuri la timpul de execuție.

4. Implementare

Voi explica, pe scurt, fiecare clasă în parte, iar cele asemănătoare vor merge împreună în descriere.

a) Clasa `Start`:

Câmpuri: -
Metode importante:
1. `main': pornește interfața de start

b) Clasa `View`

Câmpuri: -

Metode importante:

- Metodele de pornire a interfețelor grafice asociate(Client, Product și Order)

c) Clasele ‘ClientFrame’, ‘ProductFrame’ și ‘OrderFrame’

Câmpuri:

- Fiecare clasă are câmpurile ei asociate, pentru a identifica obiectul din baza de date. Fiecare in ele au obligatoriu un id
- JTable pentru gestionarea obiectelor într-un tabel cu fiecare câmp asociat unei coloane

Metode importante:

1. insertClient/Product: inserează un client/produs în baza de date
2. editClient/Product: editează un client/produs în baza de date
3. deleteClient/Product: șterge un client/produs în baza de date
4. findAll: găsește toți clienții/produsele/comenzile din baza de date
5. findStudent/ProductByID: găsește un client/produs din baza de date pe baza id-ului său
6. viewListOfFields: afișază într-un JTable fiecare client/produs/comandă din baza de date

d) Clasele ‘Client’, ’Product’ și ’Orders’

Câmpuri:

- Fiecare clasă are câmpurile ei asociate, pentru a identifica obiectul din baza de date. Fiecare in ele au obligatoriu un id

Metode importante:

- Getters și setters, plus toString

e) Clasele ‘AbstractDAO’, ‘ClientDAO’, ‘ProductDAO’ și ‘OrderDAO’
Câmpuri:

1. Logger: pentru a lua informații despre obiectul respectiv
2. Type: tipul de clasă al obiectului

Metode importante:

- Asemănătoare cu cele de la interfețe(insert, update, delete, findAll, findByID, viewListOfFields), utilizate cu ajutorul reflexiei
- Subfuncții de creare a obiectelor

f) Clasa ’ConectionFactory’:
Câmpuri:

1. Logger
2. 4 string-uri(DRIVER, DBURL, USER, PASS)

Metode importante:

- createConnection(): creează conexiunea dintre serverul bazei de date din MySQL cu Java

g) Clasele ‘ClientBLL’, ‘ProductBLL’ și ‘OrderBLL’:
Câmpuri:

1. List<Validator<T>>: lista de validări pentru un obiect T anume
2. Client/Product/OrderDAO – vezi punctul e)

Metode importante:

- Aceleași metode cu cele din punctul e), doar că validate

h) Clasele ‘ClientAgeValidator’, ‘EmailValidator’, ‘OrderQuantityValidator’, ‘ProductPriceValidator’ și ‘ProductStockValidator’

Câmpuri: -

Metode importante:

- Validate(T t): validează obiectul dat ca parametru(T)

5. Rezultate

Pentru oricare set de parametrii introduși în aplicația de gestionare a comenzilor, output-ul se va afișa atât în JTable, cât și în baza de date folositâ.

De asemenea, avem și un fișier dump, acesta conținânt codul SQL cu toate modificările aduse tabelelor din baza de date.

6. Concluzii

În concluzie, această temă a fost una destul de interesantă, prin prisma faptului că am creat, de la 0, o aplicație care gestionează fieclienți, fie produse sau comenzi. De asemenea, utilizarea unei interfețe grafice pentru inserarea argumentelor necesare aplicației, respectiv utilizarea reflexiei, a fost un lucru interesant de implementat.

Din această temă am învățat cum se poate gestiona comenzile unor clienți pentru un depozit prin intermediul unei baze de date. De asemenea, am învățat cum să utilizez Reflexia, ceea ce o să mă ajute foarte mult în viitorul apropiat.

Niște posibilități de dezvoltări ulterioare ar putea fi:

- Înfrumusețarea interfeței grafice(adăugare culori, font mai frumos, etc.)
- Implementarea unei interfețe care afișază, în timp real, evenimentele(nu după un refresh)
- Alte funcționalități ale claselor în interfață

7. Bibliografie

1. *https://dsrl.eu/courses/pt/materials/PT2024_A3_S1.pdf*
2. *https://dsrl.eu/courses/pt/materials/PT2024_A3_S2.pdf*
3. *https://www.geeksforgeeks.org/reflection-in-java/*