Manager de depozit

Facultatea de Automatică și Calculatoare

Student: *Petrea Irina-Alexandra*

An 2, grupa 30226

. . . . . . . . . . . . . .

Cuprins:

1. **Obiectivul temei**
   1. **Obiectivul principal**
   2. **Obiective secundare**
2. **Analiza problemei**
3. **Proiectarea soluției**
4. **Implementare**
5. **Rezultate**
6. **Concluzii**
7. **Bibliografie**

. . . . . . . . . . . .. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

1. Obiectivul temei

**Obiectivul principal** al temei este cel de a realiza o aplicație desktop pentru gestionarea unui depozit. Această aplicație lucrează cu o bază de date și se permite adăugarea de clienți noi, produse noi și comenzi noi, în limita stocului de produse disponibil. Aplicația este proiectată conform paradigmelor programării orientate pe obiecte folosind limbajul de programare Java. Pentru organizarea pe pachete și clase s-a folosit modelul arhitectural **Layers**. Punem în evidență mai multe obiective secundare, părți constituente ale obiectivului principal sus-menționat:

* **Divizarea responsabilităților pe clase și metode**, ca bună practică în proiectarea aplicației, prin structurarea pe pachete de clase conform modelului arhitectural **Layers**, pentru a facilita înțelegerea programului și dezvoltarea ulterioară – va fi detaliată în capitolele 3 și 4
* **Realizarea diagramelor UML** (diagrame de clase, de pachete, etc), pentru a facilita proiectarea ulterioară și transpunerea funcționalităților în cod
* **Realizarea conexiunii cu baza de date** gestionată de serverul MySQL
* **Lucrul cu fișiere și realizarea unui executabil .jar**
* **Generarea de fișiere pdf**
* **Realizarea operațiilor asupra bazei de date prin intermediul tehnicii de reflection în Java**

1. Analiza problemei

Se vor prezenta funcționalitățile oferite de aplicație, precum și detalierea cazurilor de utilizare a acesteia.

**Aplicația** gestionează procesarea de comenzi ale clienților la un depozit după cum urmează:

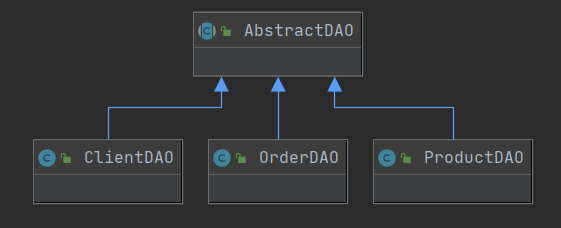
* Aplicația interacționează cu o bază de date conținând 3 tabele:
  + Tabela Client – va conține date despre clienți – un client este descris prin numele său (unic în tabel) și adresa sa.
  + Tabela Product – va conține date despre produsele aflate în stoc la depozit – un produs este descris prin numele său (unic în tabel), cantitatea de produs prezentă în stoc la momentul curent și prețul pe bucată al acelui produs
  + Tabela WarehouseOrder – va conține date despre comenzile efectuate la depozit – o comandă este caracterizată de numele clientului care a efectuat-o, numele produsului comandat, cantitatea din acel produs care a fost comandată și totalul comenzii. Este important de menționat că folosind o singură tabelă pentru gestionarea comenzilor, fiecare comandă va conține doar un singur produs, iar factura se va genera, similar, pentru un singur produs la un moment dat.
* Se pot adăuga și șterge clienți; la ștergerea unui client, se vor șterge și toate comenzile efectuate de către acel client
* Se pot adăuga și șterge produse; la încercarea de adăugare a unui produs care există deja în tabelă, acesta nu se va adăuga ca produs nou, ci se va actualiza cantitatea din produsul deja existent ca suma dintre cantitatea de produs existentă deja și cantitatea din produsul ce se încearcă a se adăuga.
* Se pot adăuga comenzi: comanda se adaugă în tabela corespunzătoare doar dacă există produsul dorit în stoc și cantitatea de produse din stoc este suficientă, i.e. cantitatea de produse ce se dorește a fi comandată este mai mică sau egală cu cantitatea de produs existentă pe stoc. La finalizarea unei comenzi, se generează o factură sub formă de fișier pdf în folderul aplicației, factură ce conține date referitoare la comandă: numele clientului, numele produsului, cantitatea de produs comandată, totalul comenzii (care se calculează înmulțind prețul per bucată al produsului cu cantitatea de produs comandată). Dacă nu există suficiente produse pe stoc, se va genera un alt fișier pdf în care se va specifica acest lucru.
* Se pot genera rapoarte pentru clienți, produse și comenzi, ce constituie fișiere pdf cu date despre toate rândurile din tabelele Client, Product și WarehouseOrder din baza de date, în formă tabulară.

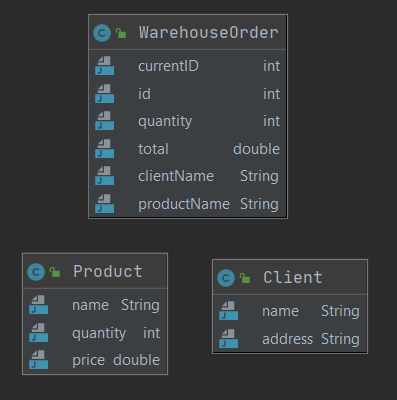
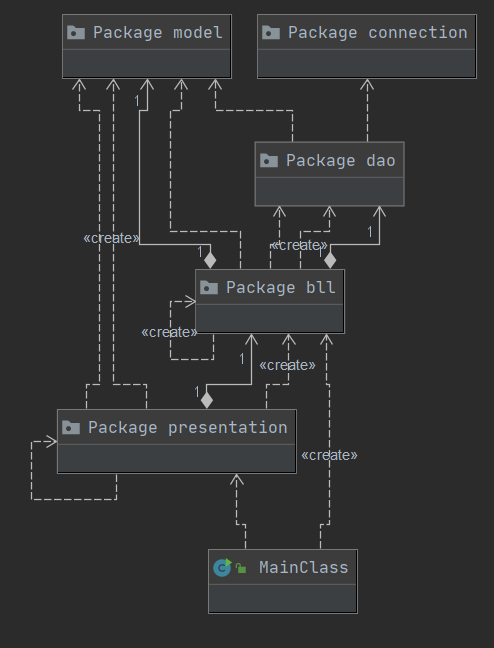
Interacțiunea cu utilizatorul se realizează astfel: această aplicație nu prezintă interfață grafică, ea lucrând cu un argument dat la execuție, ce reprezintă un fișier text care conține pe fiecare linie câte o comandă, după cum urmează:

* Insert client: Nume, Adresă – va determina inserarea în tabela de clienți a unui nou client cu datele specificate, dacă nu există deja un alt client cu același nume
* Insert product: Nume, Cantitate, Preț – va determina inserarea în tabela de produse a unui nou produs având numele, cantitatea și prețul specificate; dacă există deja un produs cu același nume în tabela de produse, atunci se va actualiza cantitatea produsului deja existent la suma cantităților produsului existent și a produsului nou adăugat – pe scurt, se va actualiza stocul disponibil.
* Delete client: Nume, Adresă – va determina ștergerea din tabela de clienți a clientului cu datele specificate. De asemenea, se vor șterge și toate comenzile clientului respectiv.
* Delete product: Nume – va determina ștergerea din tabela de produse a produslui cu numele specificat
* Order: Nume, Produs, Cantitate – va determina adăugarea unei noi comenzi în tabela de comenzi, numai dacă stocurile permit efectuarea comenzii (dacă în tabela Produs există o cantitate de produse mai mare sau egală cu cantitatea cerută în comandă prin câmpul Cantitate
* Report client – va genera un raport conținând toate datele despre toți clienții, în formă tabulară într-un fișier pdf
* Report product – va genera un raport conținând toate datele despre toate produsele, în formă tabulară, într-un fișier pdf
* Report order – va genera un raport conținând toate datele despre toate comenzile, în formă tabilară, într-un fișier pdf.

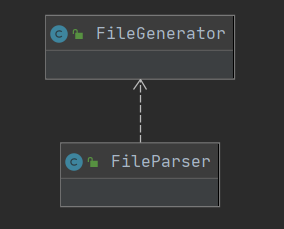
1. Proiectarea soluției

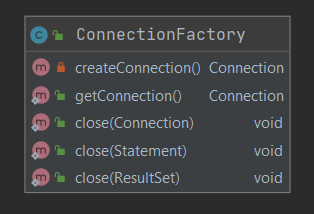
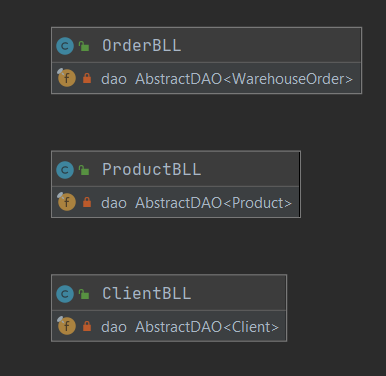
Se vor prezenta diagramele de pachete și de clase ce pun în evidență aspectele de proiectare OOP ale aplicației, precum și justificări legate de alegerile făcute în contextul proiectării.

 Pachetul Model



Pachetul main – diagrama de pachete Pachetul DAO





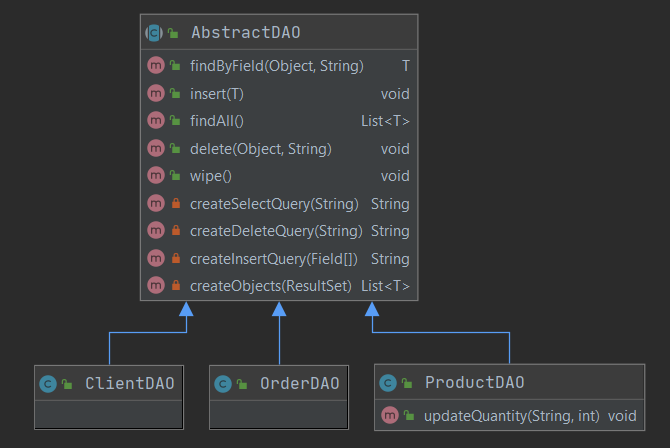
Pachetul Connection Pachetul BLL Pachetul Presentation

Diagramele de pachete de mai sus înfățișează structura pachetelor importante din implementarea aplicației. Această aplicație este structurată conform modelului arhitectural Layers, ce presupune împărțirea responsabilităților pe mai multe nivele, lucru care facilitează înțelegerea și dezvoltarea ulterioară a aplicației. Fiecare pachet este descris împreună cu toate clasele și responsabilitățile acestora după cum urmează:

* **Pachetul Connection** – conține doar clasa ConnectionFactory, care realizează conexiunea la baza de date folosind o singură instanță de obiect Connection, prin implementarea modelului arhitectural Singleton. De asemnea, sunt puse la dispoziție metode de obținere și închidere a conexiunii,precum și de închidere a unui PreparedStatement și un ResultSet.
* **Pachetul Model** – conține clasele ce modelează obiectele principale folosite în această aplicație: Client, Model, WarehouseOrder. Aceste corespund tabelelor din baza de date: au aceleași câmpuri, aceleași tipuri de date și în aceeași ordine. Conțin doar metode accesoare și mutatoare.
* **Pachetul DAO –** conține clasele folosite pentru interogarea bazei de date și modificarea datelor din aceasta. În acest pachet se folosește tehnica reflexiei, așa încât introducerea, ștergerea și interogarea în baza de date se fac folosind aceleași metode, indiferent de tipul obiectului instanță al unei clase din pachetul Model. Operațiile care sunt specifice fiecărei clase Model sunt implementate prin metode în afara clasei abstracte AbstractDAO (aici se folosește reflexia).
* **Pachetul BLL –** conține clase folosite pentru implementarea logicii de business a aplicației, pentru fiecare dintre clasele pachetului Model.
* **Pachetul Presentation –** conține clase destinate interacțiunii cu utilizatorul – un parser pentru fișierul de comenzi și un generator de fișiere pdf, în cazul comenzilor Order și Report client, product sau order.
* **Pachetul default –** conține clasa MainClass, cu metoda Main, de unde începe execuția aplicației.

1. Implementare

Se va analiza fiecare clasă importantă în parte, argumentând alegerile de implementare acolo unde este cazul.



* + - 1. **Pachetul DAO**
         * **Clasa AbstractDAO** – este clasa ce implementează metodele CRUD – create, read, update, delete pentru obiectele instanță ale claselor din pachetul Model. Aici se folosește tehnica de reflexie în Java pentru a nu fi nevoie să scriem mai multe metode similare particularizate pentru fiecare clasă ce modelează obiectele aplicației. În acest sens, a fost definit un tip generic asociat clasei AbstractDAO, care este o clasă abstractă. Acest tip se va obține în mod dinamic la runtime, în funcție de nevoie. Metoda **findByField** efectuează o interogare a bazei de date de forma ”SELECT \* FROM <corresponding table> WHERE fieldname = field AND deleted = 0” unde tabelul corespunzător este determinat în mod dinamic (pentru că numele acestuia corespunde cu numele clasei din pachetul model corespunzătoare obiectului DAO creat), iar fieldname și field sunt argumente ale funcției – fieldname este un String, iar field este un Object. Așadar, se realizează o interogare ce returnează doate coloanele primei linii din rezultat, filtrarea făcându-se după un singur câmp. Similar funcționează și metoda **delete**, care realizează ștergerea unuia sau mai multor elemente, filtrarea făcându-se după un singur câmp, cel specificat drept argument. Este important de menționat că ștergerea unui element se face setând un anumit câmp deleted la valoarea 1. Metoda **findAll** realizează o interogare de tipul ”SELECT \* FROM <corresponding table> WHERE deleted = 0” și returnează o listă cu obiectele obținute în urma acestei interogări; această interogare ignoră elementele marcate pentru ștergere prin coloana deleted. Metoda **wipe** se ocupă cu ștergerea definitivă a tuturor obiectelor marcate pentru ștergere prin coloana deleted.
         * **Clasa ClientDAO** – poate implementa operațiile cu baza de date specifice clienților.
         * **Clasa ProductDAO** – poate implementa operațiile cu baza de date specifice produselor. În această aplicație, s-a ales implementarea actualizării stocului ca funcție particulară în ProductDAO, pentru ușurință și conveniență. Atunci când se încearcă inserarea unui produs deja existent, acesta nu se inserează, în schimb, se actualizează stocul produsului deja existent.
         * **Clasa OrderDAO –** poate implementa operațiile cu baza de date specifice comenzilor.
      2. **Pachetul BLL**
         * **Clasa ClientBLL –** implementează logica de business pentru clienți, apelând metodele specifice pe un obiect din clasa DAO ce operează pe obiecte din clasa Client. Regăsim metode de adăugare, ștergere(marcare), ștergere obiecte marcate, găsire client (find), precum și metoda ce returnează o listă cu toate obiectele de tipul Client din baza de date: reportClient.
         * **Clasa ProductBLL -** implementează logica de business pentru produse, apelând metodele specifice pe un obiect din clasa DAO ce operează pe obiecte din clasa Product. Regăsim metode de adăugare, ștergere(marcare), ștergere obiecte marcate, găsire client (find), actualizarea stocului, precum și metoda ce returnează o listă cu toate obiectele de tipul Product din baza de date: reportProduct.
         * **Clasa OrderBLL** - implementează logica de business pentru comenzi, apelând metodele specifice pe un obiect din clasa DAO ce operează pe obiecte din clasa WarehouseOrder. Regăsim metode de adăugare, ștergere(marcare), ștergere obiecte marcate, găsire client (find), ștergerea comenzilor unui anumi client, precum și metoda ce returnează o listă cu toate obiectele de tipul WarehouseOrder din baza de date: reportOrder.
      3. **Pachetul Presentation**
         * **Clasa FileGenerator –** este clasa ce furnizează metode pentru generarea de fișiere PDF corespunzătoare cerințelor aplicației. S-a folosit librăria iTextpdf pentru generarea acestor fișiere. Fișierele de tip Report se generează în formă tabulară.
         * **Clasa FileParser –** este clasa ce furnizează metode pentru interpretarea fișierului de comenzi dat ca parametru la execuția programului. Se folosește de structura sintactică a fiecărei comenzi pentru a separa cuvintele cheie folosind expresii regulate. Tot această clasă apelează metodele corespunzătoare din clasele pachetului de business logic, precum și metodele de generare a fișierelor pdf din clasa FileGenerator.
      4. **Pachetul default**
         * **Clasa MainClass**

1. Concluzii

În urma realizării acestei teme, am dobândit cunoștințe cu privire la utilizarea modelelor ahitecturale Layers și Singleton, folosirea tehnicii de reflexie în Java pentru a scrie metode CRUD ce interacționează cu baza de date, precum și generarea de fișiere pdf și javadoc. De asemenea, am aprofundat cunoștințe cu privire la citirea și scrierea din fișier, realizarea unui executabil .jar care să fie rulat cu argumente în linia de comandă, realizarea diagramelor UML ca ajutor în proiectarea aplicației, precum și realizarea unei documentații cuprinzătoare.

1. Bibliografie

[1] <http://coned.utcluj.ro/~salomie/PT_Lic/4_Lab/Assignment_3/Assignment_3_Indications.pdf>

[2] <https://en.wikipedia.org/wiki/Singleton_pattern>

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

[4] <https://www.jetbrains.com/help/>

[5] <https://www.udemy.com/>

[6] <https://www.geeksforgeeks.org/>

[7] <https://www.w3schools.com/>

[8] <https://stackoverflow.com/>

. . . . . . . . . . . . . . . . . . . . . . . .. . . . . . . . . . . . . . . . . . . . . . .