

Facultatea de Automatică și Calculatoare

Tehnici de programare fundamentale

Laborator – Tema 4

FOOD DELIVERY MANAGEMENT SYSTEM

Profesor: Dr. Cristina Pop

Student: Jiroveanu Ștefan

Grupa: 30229

# CUPRINS

[1. Obiectivul temei 3](#_Toc95297885)

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

[3. Proiectare 5](#_Toc95297887)

[4. Implementare 8](#_Toc95297888)

[5. Rezultate 17](#_Toc95297889)

[6. Concluzii 19](#_Toc95297890)

[7. Bibliografie 20](#_Toc95297891)

1. Obiectivul temei

Obiectivul temei este proiectarea și implementarea unui sistem de gestionare al livrării unor produse, în special mâncare, pentru o firmă de catering. Clientul care accesează aplicația va putea comanda produse din meniul companiei. Sistemul proiectat trebuie să aibă trei tipuri de utilizatori care se pot conecta utilizând un username și o parolă: administratorul, un angajat obișnuit și clientul.

**Administratorul** companiei poate executa următoarele operații:

* Importă setul inițial de produse care vor popula meniul dintr-un fișier de tip .csv
* Gestionează produsele din meniu: adaugă/ șterge/ modifică produse și creează noi produse compuse din mai multe produse
* Generează rapoarte despre comenzile depuse, după următoarele criterii:
  + Intervalul de timp în care au fost depuse comenzile: acest raport presupune afișarea comenzilor care au fost depuse într-un interval de timp dat de administrator
  + Produsele care au fost comandate mai mult decât un număr de ori dat
  + Lista clienților care au comandat de mai multe ori decât un număr dat și valoarea produselor comandate să fie mai mare decât o sumă furnizată de către administrator
  + Produsele care au fost comandate într-o anumită zi, însoțite de numărul care reprezintă de câte ori au fost comandate

**Clientul** care accesează aplicația poate executa operațiile:

* Să se înregistreze în sistem și să își utilizeze numele de utilizator și parola pentru a se conecta
* Poate vizualiza lista de produse din meniu
* Poate căuta produsele din meniu care îndeplinesc unele criterii precum: numele produsui conține un anumit cuvânt, după rating, după numărul de calorii, proteine, grăsimi, sau au un anumit preț
* Poate, de asemenea, crea o comandă care conține mai multe produse: pentru fiecare comandă se va genera o factură cu lista produselor, prețul acestora și prețul total al produselor

**Angajatul** companiei va fi notificat de fiecare dată când o nouă comandă a fost depusă de către un client, deci el va putea începe prepararea produselor comandate.

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

Această aplicație trebuie să îndeplinească toate cerințele necesare pentru ca mai întâi utilizatorii să se conecteze în aplicație și apoi, în funcție de identitatea utilizatorului ( administrator, client sau angajat), să poată realiza operațiile cerute.

Pentru îndeplinirea cerințelor trebuie utilizate design pattern-urile Observer Design Pattern( pentru a notifica angajații de fiecare dată când o nouă comandă a fost plasată) și Composite Design Pattern ( pentru definirea claselor BaseProduct, CompositeProduct care vor extinde MenuItem).

Design Pattern-ul *Observer* definește o relație de dependență 1..\* (one to many) între obiecte astfel încât când un obiect își schimbă starea, toți dependenții lui sunt notificați și actualizați automat. Folosirea acestui pattern implică existența unui obiect cu rolul de subiect, careia ii e asociată o listă de obiecte dependente, cu rolul de observatori, pe care le apelează automat de fiecare dată când se întâmplă o acțiune.

Design pattern-ul Composite este un design de tip structural ce permite compunerea obiectelor sub forma de arbori pentru a putea fi folosite ca si cum ar fi obiecte individuale. Astfel, in aceasta problema acest design pattern exte exemplificat prin utilizarea celor 3 clase : MainItem, BaseProduct și CompositeProduct, ultimele doua fiind subclase ale primei.

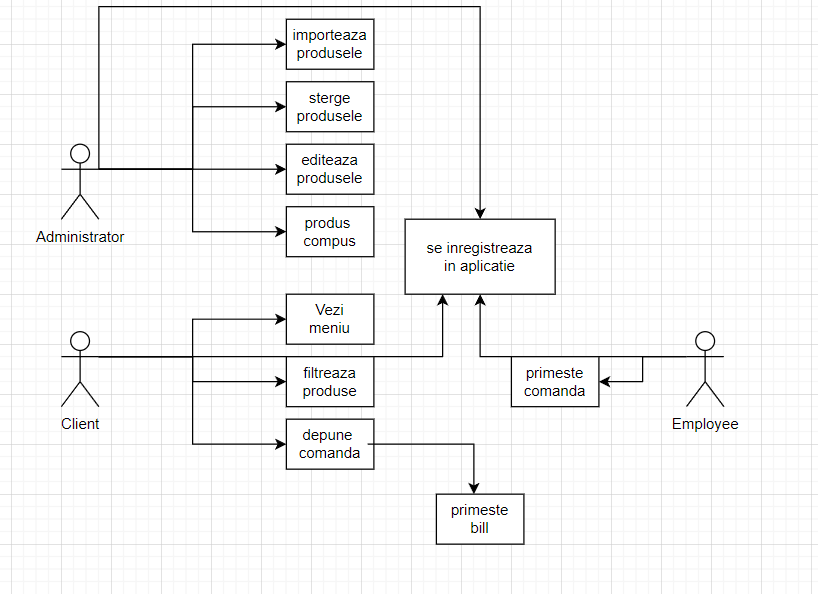
1. **Modelarea problemei**

Aplicația va avea o interfață care va oferi utilizatorului posibilitatea de a alege acțiunea pe care dorește să o realizeze. Va exista, astfel, o interfață principală care leagă utilizatorul de interfața corespunzătoare în funcție de rolul său. Fiecare are butoane și câmpuri de text pentru ca aplicația să primească solicitarea utilizatorului, apoi să rezolve și să returneze rezultatul.

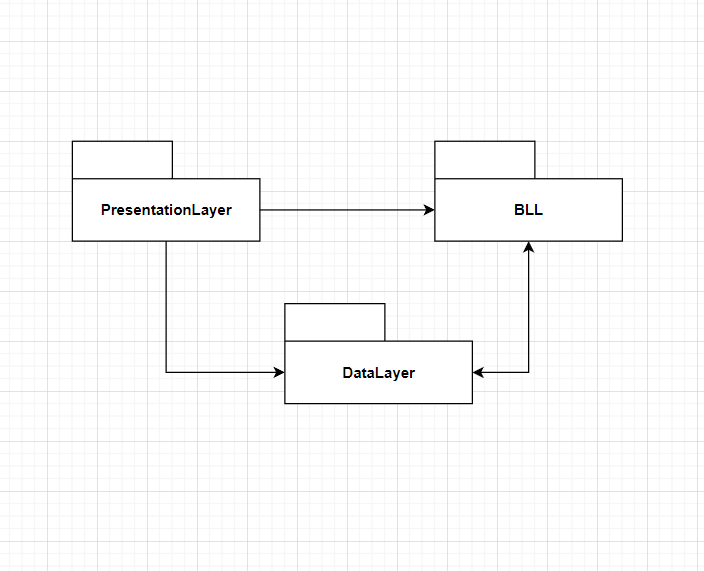
1. **Scenarii și cazuri de utilizare**

Scenariile și cazurile de utilizare sunt metode utilizate în analiza sistemelor pentru a identifica, clarifica și organiza cerințele de sistem. Un caz de utilizare este un set de posibile interacțiuni utilizator-aplicație. Deoarece cazurile de utilizare sunt de fapt pași pe care trebuie să-i parcurgă utilizatorul, încercăm să proiectăm o interfață cât mai ușor de utilizat.

**Diagrama UML – Use-case**



1. Proiectare



Proiectarea aplicației a fost realizată folosind un model arhitectural stratificat (Layered Architecture). Fiecare pachet are propriul scop și apelează funcții ale pachetelor de sub acesta.

**Presentation** - Conține clase care implementează interfața grafică a aplicației și interacțiunea dintre logica externă (ceea ce vede utilizatorul) și logica internă. Acest pachet are sase clase care implementează reprezentări grafice, iar rolul lor este de a prelua datele introduse de utilizator și de a returna rezultatul dorit. Există o interfață principală în care utilizatorii se logheaza in aplicație. In cazul in care utilizatorul nu are un cont, acesta poate deschide o noua fereastra pentru a-si creea un cont nou. Dupa introducerea creditentialelor, fiecare utilizator va fi trimis către interfața corespunzătoare, astfel încât există o fereastră pentru clienți, administratori și angajați, fiecare cu câmpurile necesare pentru a selecta o acțiune de rulat, a introduce datele de intrare și a returna rezultatul.

**BLL(business logic layer)** – conține clasele care încapsulează logica aplicației

**DataLayer** – este impartit intre doua pachete : dao, ce conține clasa *Serializator* ce are rolul de a serializa informațiile clasei *DeliveryService* memorandu-le intr-un fisier separat salvat in folder-ul resources al proiectului Maven, în acest mod, modificările aduse asupra datelor vor fi stocate. De asemenea, acest pachet contine si clasa FileWriter este folosita pentru generarea facturii pentru client și model ce contine clasa User.

Pachetele din interiorul clasei sunt :

**Pentru Presentation Layer** : avem **pachetul presentation** ce contine clasele AdminInterface, ClientInterface, EmployeeInterface, GenerateRaportInterface, LogInPage si SignUpPage prin intermediul carora sunt implementate interfatele corespunzatoare fiecarui tipe de utilizator. Un alt pachet ce apartine substratului de prezentare il constituie **pachetul controller** ce contine clasele controller folosite pentru implementarea logicii din spatele interfatelor sub forma unui design MVC. Acest pachet contine clasele AdminController, ClientController si ControllerCommonMethods, cea din urma continand metode comune folosite in ambele clase controller.

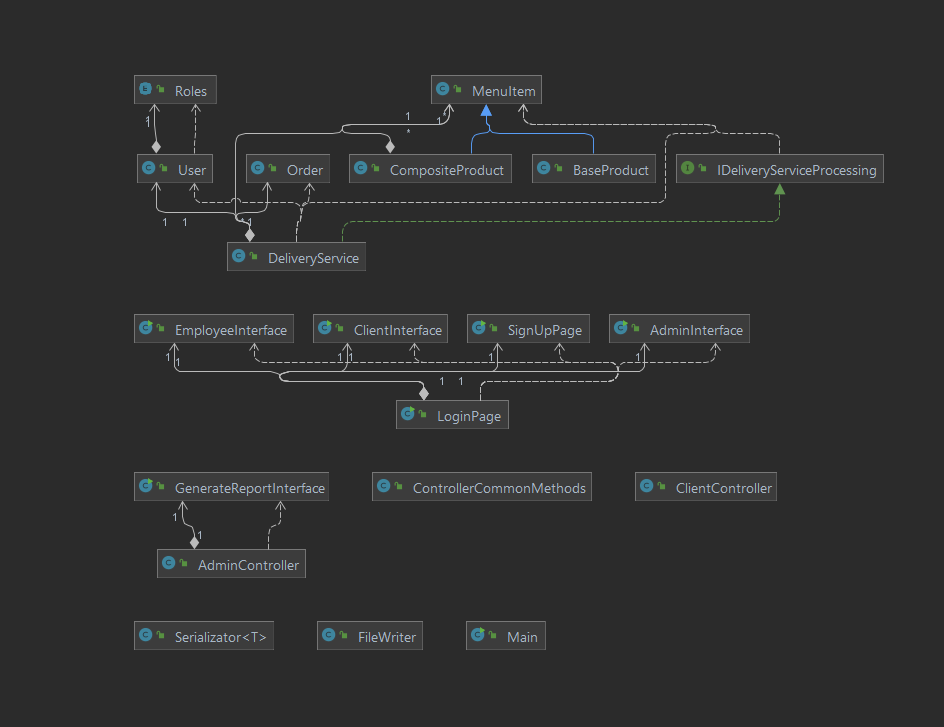
**Pentru DataLayer:** avem pachetul dao ce contine clasele FileWriter si Serializer. Clasa FileWriter are rolul de a intocmi bill-urile, iar clasa serializer are rolul de a serializa clasele astfel incat consistenta datelor de la rulari anterioare sa fie pastrata.

**Pentru BusinessLogicLayer:** avem pachetul BLL ce contine clasele care impreuna reprezinta logica din spatele aplicatiei. Clasa principala din acest pachet este DeliveryService, aceasta implementeaza interfata IDeliveryServiceProcessing ce are metode ce imita stilul unei interfate CRUD(Create, Remove, Update, Delete). Alte trei clase apartinand pachetului BLL sunt clasele ce intocmesc structura Composite, si anume MainItem, BaseProduct si CompositeProduct. Ultima clasa din interiorul acestui pachet o reprezinta POJO-ul Order folosit pentru memorarea comenzilor realizate de catre clienti.

Ultimul pachet din interiorul acestei aplicatiei este Role ce contine o enumeratie pentru rolul unui User :

Administrator, Client, Employee.

**Diagrama de clase UML:**

****

# Proiectare

* **Clase folosite, campuri si metode importante**

**Pachetul bll contine sase clase :**

**Clasa BaseProduct :**

* + - Exista doi constructori: unul ce primeste ca parametrii numele, pretul, rating-ul, numarul de calorii, proteinele, sodium-ul si grasimile, si unul care primeste aceleasi atribute, dar sub forma unor string-uri urmand sa le converteasca pe fiecare la tipul potrivit.
    - Metoda importanta din interiorul acestei clase este ReadFromCsv ce citeste obiecte de tipul BaseProduct si le mapeaza

**Clasa CompositeProuct :**

* + - Exista doi constructori: unul ce primeste ca parametrii numele, pretul, rating-ul, numarul de calorii, proteinele, sodium-ul si grasimile, si unul care care nu primeste niciun parametru.
    - Aceasta clasa are rolul de a incapsula mai multe obiecte de tipul BaseProduct pentru a crea un obiect Composite
    - De asemenea, precum clasa precedanta aceasta clasa extine clasa MenuItem.
    - Metodele importante din interiorul clasei sunt computePrice, ceruta de indrumator si compute ce calculeaza proprietatile CompositeProduct-ului.

**Interfata IDeliveryServiceProcessing :**

* + - Aceasta interfata incearca sa imite interfatele de tip CRUD(Create, Read, Update, Delete).
    - Metodele in interiorul acestei interfate sunt importProduct, deleteProduct, updateProduct, importAllItems, generateReportOne, generateReportTwo, generateReportThree, generateReportFour, ultimele 4 sunt menite pentru a generea cele 4 rapoarte cerute in cerinta aplicatiei.

**Clasa DeliveryService :**

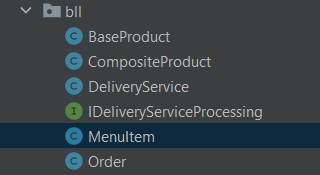
* + - Aceasta clasa este cea mai importanta clasa din interiorul aplicatiei
    - Field-urile importante ale acestei clase sunt currentUser, ce reprezinta user-ul ce acceaseaza aplicatia la un anumit moment dat in timp. BasicProducts este un hashset in interiorul caruia sunt memorate cele 14 mii de obiecte citite din fisierul products.csv. hashMap este un map ce are drept chei un obiect de tip Order, iar ca valorii memoreaza liste de MenuItems.
    - Aceasta clasa implementeaza metodele din interiorul interfatei IDeliveryServiceProcessing.
    - Metoda importProduct adauga la hashset un nou produs si returneaza noul hashet
    - Metoda deleteProduct sterge produsul din hashest si returneaza produsul
    - Metoda updateProduct updateaza produsul in hashset si returneaza noul hashset.
    - Metoda createNewOrder e o metoda apelata doar de utilizatorii de tip client, atunci cand acestia creeaza o noua comanda.
    - Metoda importAllItems are rolul de a importa in hashset toate itemele din products.csv
    - Metoda GenerateReportOne genereaza primul raport conform caruia trebuie salvate toate comenzile date aflate intre doua ore, indifrent de date.
    - Metoda GenerateReportTwo genereaza al doilea raport, conform caruia trebuie salvate toate produsele care au fost comandate de un numar mai mare de ori decat un parametru dat.
    - Metoda GenerateReportThree genereaza al treia raport, conform caruia trebuie salvati toti clienții care au comandat mai mult de un anumit număr de ori până acum și valoarea comenzii a fost mai mare decât o sumă specificată.
    - Metoda GenerateReportFour genereaza al patrulea raport, conform caruia trebuie salvate toate produsele comandate într-o zi specificată cu numărul de ori pe care le au comandat.

**Clasa MenuItem :**

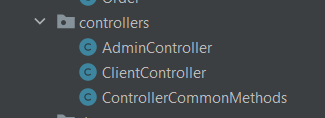
* + - Exista doi constructori: unul ce primeste ca parametrii numele, pretul, rating-ul, numarul de calorii, proteinele, sodium-ul si grasimile, si unul care primeste aceleasi atribute, dar sub forma unor string-uri urmand sa le converteasca pe fiecare la tipul potrivit.

**Clasa Order :**

* + - Este o clasa POJO



* **Pachetul Controllers contine trei clase :**

****

Aceste trei clase fac legatura intre logica externa si logica interna a programului.

* **Pachetul dao contine doua clase :**

**Clasa Serialization :**

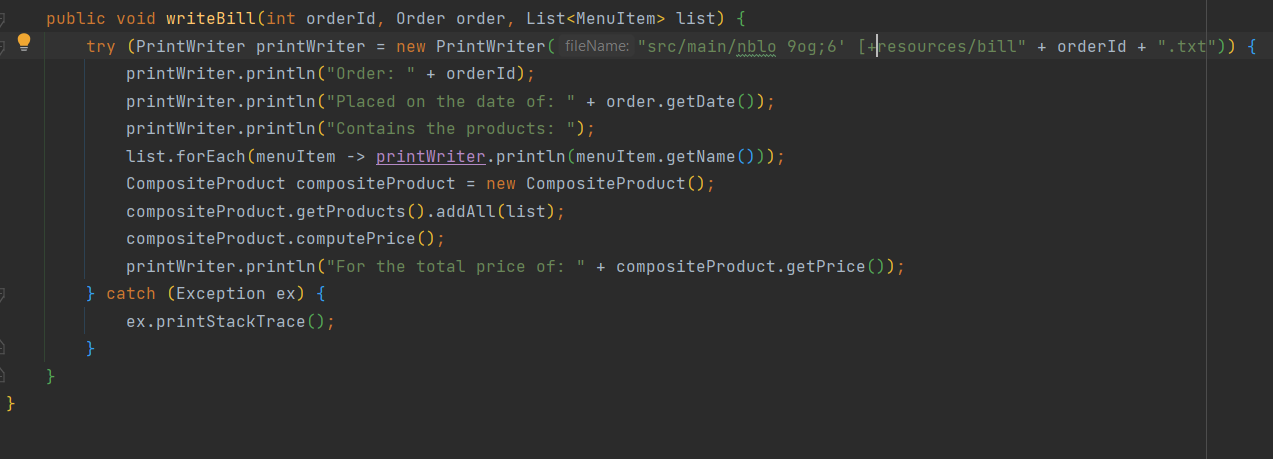
* + - Aceasta clasa are rolul de a serializa clasa User si clasa DeliveryService
    - Aceasta clasa a fost implementata folosind generice
    - Contine 4 metode : 2 de write si 2 de read.





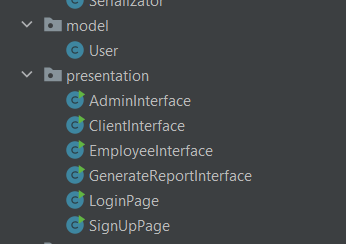
**Clasa FileWriter :**

* + - Aceasta clasa are rolul de a intocmi Bill-urile pentru comenzi.
    - Contine o singura metoda writeBill.



**Pachetul model contine clasa User**

**Pachetul Presentation contine implementarea interfatelor grafice.**

****

**Pachetul Role contine enum-ul Roles.**

****

# Rezultate

Conținutul comenzii plasate de client va fi afișat în fișierul bill.txt. Pentru fiecare comanda se va crea un nou fisier de tipul bill.txt Tot pentru clienti, in campul care afiseaza rezultatele corespunzatoare cererii de cautare vor fi afisate produsele din meniu. Rezultatele raportului vor fi scrise in fisiere separate intitulate Report1, Report2, Report3, Report4, fiecare dintre acestea urmand sa fie updatat atunci cand se genereaza un nou raport.

# Concluzii

Proiectul a fost o modalitate fantastică de a-mi reîmprospăta și chiar de a-mi extinde înțelegerea ideilor OOP pe care le stăpânisem anterior. Voi împărtăși câteva lucruri pe care le-am câștigat din experiența acestui proiect în secțiunile de mai jos. În primul rând, și poate cel mai important, este gestionarea adecvată a timpului, deoarece o organizare bună, la începutul timpului, poate îndeplini o misiune de asigurare într-o varietate de moduri. În al doilea rând, modelarea corectă a problemelor de la început ajută la rezolvarea lor mai rapidă și exclude implementarea anumitor funcționalități ineficiente.

Din punct de vedere informational am dobandit cunostinte despre mai multe tipuri de Design Pattern-uri, precum Composite Design Pattern si Observer Design Pattern ( folosit in implementarea interfatei pentru angajati). De asemenea am invatat importanta serializarii si deserializarii atunci cand programam o aplicatie si avem nevoie de datele unei rulari anterioare. Lucrul cu stream-uri si functii lambda m-a ajutat sa realizez cat de usor se poate scrie un cod mai curat, mai lizibil si mai usor de inteles, dar insa si mai rapid din punct de vedere al executiei. Am dobandit cunostinte si despre assert-uri, invariant, Design Contract si Colectii

# Bibliografie

* <https://dsrl.eu/courses>
* <https://stackoverflow.com/>
* <https://ro.wikipedia.org/>
* <https://www.jetbrains.com/help/idea/>
* <https://www.geeksforgeeks.org/>
* <https://www.youtube.com/watch?v=AEnePs2Evg0&list=PLrhzvIcii6GMQceffIgKCRK98yTm0oolm>(Single Responsibility Principle (SOLID) | A single reason to change — Code Walks 006)
* <https://www.diagrams.net/>
* <https://www.baeldung.com/java-try-with-resources>
* <https://www.geeksforgeeks.org/stream-distinct-java/>
* <https://refactoring.guru/design-patterns/composite>
* <https://dzone.com/articles/what-is-serialization-everything-about-java-serial>