**DOCUMENTAȚIE TEMA 4**

**FOOD DELIVERY MANAGEMENT SYSTEM**

**Ovreiu Auraș Dănuț**

**Grupa 30229**

**Profesor Laborator Assist Antal Marcel**

Contents

[1. Cerințe funcționale 3](#_Toc72397432)

[2. Obiective 3](#_Toc72397433)

[2.1. Obiectivul principal 3](#_Toc72397434)

[2.2. Obiective secundare 3](#_Toc72397435)

[3. Analiza problemei 4](#_Toc72397436)

[4. Proiectare 4](#_Toc72397437)

[4.1. Structuri de date 4](#_Toc72397438)

[4.2. Algoritmi 5](#_Toc72397439)

[4.3. Diagrame de clase 5](#_Toc72397440)

[5. Implementare 6](#_Toc72397441)

[5.1. Clase 6](#_Toc72397442)

[5.2. Metode 8](#_Toc72397443)

[6. Testare 9](#_Toc72397444)

[7. Concluzii și Dezvoltări Ulterioare 10](#_Toc72397445)

[8. Bibliografie 10](#_Toc72397446)

# 

# 1. Cerințe funcționale

La această temă se cere implementarea unui program care să simuleze un sistem de management al livrării alimentelor pentru o companie de catering, compusă din angajați, administrator și clienți. Accesul la aplicație trebuie să se facă prin nume și parolă pentru cele 3 categorii de persoane, iar în timp ce clienții și administratorul pot efectua diverse operații, precum adăugare, ștergere, modificare de produse, generare de rapoarte și facturi angajații trebuie să fie notificați de fiecare dată când un nouă comandă a fost efectuată de către client. Ca și suport pentru temă este dată o diagramă de clase.

# 2. Obiective

## 2.1. Obiectivul principal

Obiectivul principal al acestei teme este proiectarea și implementarea unei aplicații care să simuleze servirea unor clienți într-o companie de catering, meniul existent fiind modificat de administrator, iar angajații având rolul de a transmite mai departe comenzile efectuate de clienți.

## 2.2. Obiective secundare

|  |  |  |
| --- | --- | --- |
| **Obiectiv Secundar** | **Descriere** | **Capitol** |
| Descrierea cerinței | se prezintă sumar funcționalitatea proiectului | 3 |
| Alegerea structurilor de date | se aleg structuri de date care permit stocarea produselor și a comenzilor, precum și gestionarea acestora | 4 |
| Împărțirea pe clase | proiectul se împarte pe clase grupate în pachete astfel încât să se poată afișa rezultatele obținute în simulare pe interfața grafică, dar și în fișiere text și în consolă pentru afișarea mai multor detalii din cadrul simulării, totul petrecându-se prin apăsarea unor butoane din anumite ferestre | 4 |
| Dezvoltarea algoritmilor | se vor utiliza expresii lambda și stream-uri pentru o abordare mai elegantă a problemei, instrucțiunile devenind astfel mai scurte | 4 |
| Implementarea soluției | se va descrie fiecare clasă cu metode si câmpuri importante, precum și implementarea interfeței utilizator | 5 |
| Testare | se verifică funcționarea corectă a aplicației prin apăsarea diferitelor butoane din interfața grafică și vizualizarea rezultatelor obținute atât în aceasta, dar și în consolă și în fișiere text | 6 |

# 

# 3. Analiza problemei

În cadrul acestei teme sunt impuse o serie de specificații, după cum urmează:

* implementarea soluției conform diagramei de clase din cerință
* împărțirea liniilor din fișierul products.csv în 7 părți cu ajutorul stream-urilor, fiecare parte reprezentând un câmp al clasei *BaseProduct.*
* realizarea interfeței grafice atât pentru înregistrarea și logarea personalului și a clienților din companie, dar și pentru efectuarea operațiilor asupra acestora
* folosirea de stream-uri și expresii lambda pentru anumite operații
* modelarea soluției respectând anumite pattern-uri, precum Composite Design Pattern, Design by Contract și Observer Design Pattern
* salvarea informațiilor din clasa DeliveryService într-un fișier text prin serializare și încărcarea informațiilor salvate la pornirea aplicației

Modelarea problemei a fost facută în mare parte după diagrama de clase din prezentare. Proiectul a fost împărțit în 3 nivele de execuție, fiecare fiind reprezentat printr-un pachet:

* Business classes – conține clasele de bază, MenuItem, BaseProduct, CompositeProduct, DeliveryService, Order, IDeliveryServiceProcessing
* Presentation classes – conține clasele responsabile de afișarea interfeței grafice pe ecran și de vizualizarea modificării acesteia la apăsarea unor butoane. Aici se găsesc interfețele grafice pentru administrator, client și angajat, atât pentru logarea lor, dar și pentru vizualizarea operațiilor efectuate asupra acestora.
* Data classes – conține clasele FileWriter și Serializator

# 4. Proiectare

## 4.1. Structuri de date

Principalele structuri de date folosite sunt lista(List<MenuItem> list=new ArrayList<MenuItem>()) și hash mapul(HashMap<Order, List<MenuItem>> order=new HashMap<>()). Deși lista este destul de eficientă în cele mai multe situații, uneori este necesară folosirea unei colecții care să rețină chei și valori, cum ar fi hash mapul. În cazul de față, cheia este comanda de tipul Order, iar valoarea este lista de produse comandată, de tipul List<MenuItem>. Alte structuri de date folosite pentru a reține mai ușor produsele și comenzile au următoarea formă:

public BaseProduct(String title, double rating,int calories, int protein, int fat, int sodium, int price){  
 super(title, price);  
 this.rating=rating;  
 this.calories=calories;  
 this.protein=protein;  
 this.fat=fat;  
 this.sodium=sodium;  
}

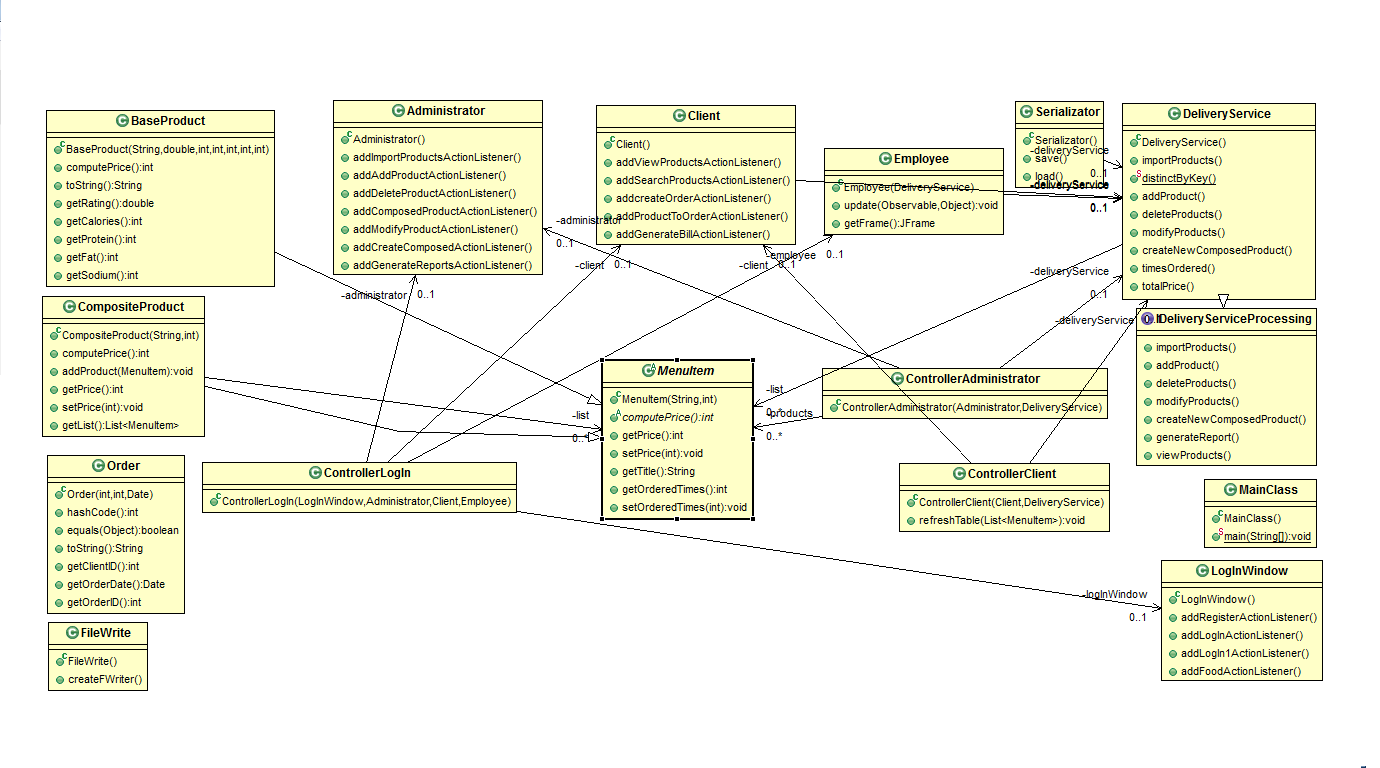
public Order(int orderID, int clientID, Date orderDate){  
 this.orderID=orderID;  
 this.clientID=clientID;  
 this.orderDate=orderDate;  
}

## 4.2. Algoritmi

La această temă au fost utilizați algoritmi de serializare și deserializare, precum și streamuri și expresii lambda care ușurează înțelegerea unor instrucțiuni mai stufoase și mai puțin eficiente.

## 4.3. Diagrame de clase

Unified Modeling Language sau UML pe scurt este un limbaj standard pentru descrierea de modele si specificații 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.



# 5. Implementare

## 5.1. Clase

***Clase din pachetul start***

***Clasa Start:***

Este clasa în care se realizează deschiderea interfeței grafice prin instanțierea celor 3 ferestre(administrator, client, angajat), dar și a ferestrei de login și se instanțiază clasa responsabilă de serializarea informațiilor din DeliveryService, apelându-se apoi metoda de load care afișează informațiile salvate din clasa DeliveryService în consolă la pornirea aplicației.

***Clase din pachetul business***

**Clasa MenuItem:**

Este clasa abstractă în care se definește structura de bază a unui produs, reprezentat de numele și prețul său. Tot în această clasă se definește metoda computePrice(), care calculează prețul tuturor produselor dintr-o anumită comandă și variabila orderedTimes, care reprezintă de câte ori a fost comandat un anumit produs.

**Clasa BaseProduct:**

Această clasă extinde clasa MenuItem și reprezintă un produs simplu caracterizat de mai multe variabile, cum ar fi ratingul sau numărul de calorii, proteine. Tot aici este definită și metoda toString(), pentru o afișare cât mai concisă a informațiilor despre produs atât în consolă, cât și în fișier.

**Clasa CompositeProduct:**

Această clasă moștenește de asemenea clasa MenuItem și singura diferență este existent unei colecții de tip listă în care sunt stocate unul sau mai multe produse, în această clasă construindu-se de fapt un produs compus din mai multe produse, având un titlu propriu și un preț care este suma prețurilor produselor din care este compus. Adăugarea produselor în listă se face prin metoda addProduct(MenuItem m), iar calcularea prețului prin metoda computePrice(), informațiile fiind manipulate prin getteri și setteri.

**Clasa Order:**

Reprezintă clasa responsabilă de crearea unei comenzi definte de ID-ul comenzii, al clientului care a făcut comanda și de data la care a fost realizată comanda. Metodele hashCode() și equals() sunt suprascrise în alte clase și au rolul de a evita inserarea unui element pe aceeași poziție. Metoda toString() are rolul de a realiza o afișare cât mai frumoasă a detaliilor comenzii, iar getterii sunt necesari pentru folosirea acestor detalii în alte clase.

**Clasa IDeliveryServiceProcessing:**

Este o interfață în care sunt enumerate principalele operații ce vor fi efectuate de clienți și respective de administrator. Pentru administrator principalele operații ce pot fi făcute sunt importul de produse, adăugarea, ștergerea, modificarea unui produs și generarea de rapoarte referitoare la comenzi și produse. Clientul poate vizualiza toate produsele din meniu, poate crea o comandă în care să existe unul sau mai multe produse și poate primi o factură în care apar produsele și prețul de la fiecare comandă.

La metodele la care se modifică produse sau comenzi există tag-uri de tipul @pre și @post. După tag-ul @pre se verifică niște condiții care trebuie îndeplinite pentru ca aplicația să nu crape în timpul execuției, iar după tag-ul @post ce ar trebui să obținem în urma apelării metodei.

**Clasa DeliveryService:**

Această clasă implementează interfața IDeliveryServiceProcessing, dar și Serializable, această clasă trebuind să fie serializată. Cele două câmpuri ce o defines sunt colecții de obiecte, prima fiind un hashmap în care se stochează comenzile, iar cealaltă o listă de produse ce există în meniu. Cele două colecții conținând obiecte de tipul Order și MenuItem, cele două clase trebuie să implementeze de asemenea interfața Serializable.

O altă metodă ce se regăsește în această clasă este isWellFormed(), în care se verifică niște condiții de bază, dar care ar trebui îndeplinite de fiecare dată pentru ca programul să funcționeze fără bug-uri și corect. Atât în această metodă, dar și în cele suprascrise din interfață sunt implementate tag-urile cu ajutorul instrucțiunii assert, care presupune ca toate condițiile să fie adevărate.

***Clase din pachetul data***

**Clasa FileWrite:**

Aici este inițializată clasa FileWriter și creat un fișier al cărui nume este primit ca parametru în constructor.

**Clasa Serializator:**

Această clasă realizează practic operația de serializare și este definite de obiectul care trebuie serializat, în acest caz un obiect de tipul DeliveryService, și numele fișierului în care vor fi salvate informațiile din obiect.

Există două metode în această clasă: load()(care citește informația din fișier) și save()(care scrie informația în fișier).

***Clase din pachetul presentation***

În acest pachet se regăsesc clase de view și controller. Clasele de view construiesc tot ce ține de interfața grafică, iar cele de tip controller modelează informația din view prin acționarea butoanelor, ferestrele fiind de asemenea afișate doar în funcție de controller.

## 5.2. Metode

***Metode utilizate în clasa DeliveryService***

O primă metodă care se remarcă este importProducts(), ce are rolul de a popula meniul cu produsele din fișierul products.csv. Aceasta se realizează prin streamuri și expresii lambda. Inițial se citesc toate liniile din fișier și se procesează sub formă de colecții de string-uri. Prima linie din fișier conținând numele coloanelor se sare peste aceasta prin apelarea metodei skip(.skip(1)), se mapează lista de string-uri, fiecare linie fiind separate în 7 părți după virgule(“,”), obținându-se acum o listă de array-uri de string-uri(array de 7 elemnte), apoi prin apelarea metodei de filtrare(filter()) se selectează doar linii distincte din fișier, lucru garantat de metoda *distinctByKey* din cadrul metodei filter. În continuare, se face conversia la listă prin apelarea metodei toList() și se se crează obiecte de tipul BaseProduct printr-o parcurgere iterativă a liniilor stocate în listă. Obiectele sunt adăugate într-o listă inițial goală, iar elementele din listă, precum și dimensiunea acesteia sunt afișate la finalul metodei.

Metodele de adăugare, ștergere și modifiare a produselor din meniu sunt relativ simple, astfel că la adăugare doar se inserează în lista de produse un nou produs de tipul MenuItem, la ștergere se șterg toate produsele care au proprietățile menționate în metoda de căutare(aici se apelează metoda de căutare după mai multe criterii), iar în metoda de modificare a prețului se apelează din nou metoda de căutare într-o buclă for-each, modificându-se doar prețul produselor care au proprietățile menționate în metoda de searchProduct.

Metoda de creare a unui produs compus e un pic diferită, aceast având ca parametri numele care va fi dat produsului și o listă de produse care vor fi incluse în acesta. Toate valorile numerice inclusiv prețul caracteristice produselor vor fi însumate, iar rezultatele vor reprezenta noile proprietăți ale produsului compus.

Metodele de generare de rapoarte sunt construite tot folosind streamuri și expresii lambda, realizându-se filtrări după anumite criterii menționate în cerință, iar informațiile obținute vor fi afișate într-un fișier text, folosind clasa FileWriter și apelând metoda write() a clasei FileWrite.

Metoda de vizualizare a produselor din meniu este realizată de client, primindu-se ca parametru id-ul său și afișându-se produsele din listă și dimensiunea acesteia.

Metoda de căutare filtrează rezultatul după toate câmpurile unui produs, în caz că nu se introduce nimic la un câmp afișându-se toate produsele, altfel doar cele care conțin informația specificată în câmpuri.

Metoda de creare a unei comenzi se realizează prin crearea unui instanțe a clasei Order și a unei liste de produse goală inițial, comanda și lista fiind adăugate în hashmap-ul order prin apelarea metodei put.

Adăugarea unui produs la o comandă se realizează pe baza numelui produsului și a ID-ului comenzii. Se parcurge hashmap-ul până se găsește prima cheie al cărei ID coincide cu cel primit ca parametru, urmând să se adauge produsul la lista de produse aflată la cheia găsită. La fiecare comandă creată angajatul este notificat prin design-ul de tip Observer, un mesaj fiind afișat în acel moment în fereastra corespunzătoare angajatului.

Metoda computePrice() calculează prețul unei comenzi prin parcurgerea iterativă a hashmap-ului până se ajunge la cheia dată ca parametru, moment în care se vor aduna la prețul comenzii toate prețurile produselor aflate la cheia respectivă.

În metoda generateBill() de generare a facturii unui client se vor afișa atât în consolă, cât și într-un fișier text produsele comandate de client, precum și prețul total al acestora.

***Metode utilizate în clasa Serializator***

Metoda save() va scrie într-un fișier text toate informațiile din clasa DeliveryService sub forma unei secvențe de bytes.

Metoda load() va citi din fișierul serializat și va afișa în consolă la deschiderea aplicației lista de produse din meniu, numărul de produse, comenzile efectuate și ID-ul fiecărui client.

***Metode utilizate în pachetul presentation***

În acest pachet în clasele de view sunt instanțiate și adăugate la interfața grafică elemente specific acesteia(textfield-uri, label-uri, butoane, tabele, panel-uri), iar în clasele de tip controller sunt implementate metodele cu listeneri pentru butoane, în care acestea fac diferite lucruri când sunt accesate.

O metodă care se deosebește este cea de refreshTable() din clasa ControllerClient, care modifică conținutul unui JTable după ce acesta a fost modificat la o apăsare de buton.

Altă metodă aparte este cea de update(Observable o, Object arg) din clasa Employee, în care se afișează un mesaj în fereastra Employee atunci când o comandă este creată. Mesajul, reprezentând în interfața grafică un JLabel, este colorat cu albastru și conține ID-ul comenzii, produsele din cadrul comenzii și ID-ul clientului care a făcut comanda respective.

# 6. Testare

Testarea în cadrul acestui proiect nu este una precisă și bine definită, verificarea funcționării aplicației putând fi văzută prin analizarea rezultatelor obținute în interfața grafică, în fișierele text și în consolă.

# 7. Concluzii și Dezvoltări Ulterioare

Această temă m-a ajutat să înțeleg lucrul cu streamuri și expresii lambda, cu care nu am mai lucrat până acum și m-a învățat noi pattern-uri, care fac codul să fie mai organizat și mai efficient. Ca și dezvoltări ulterioare, ar putea fi implementate mai multe operații pentru administrator și client, iar angajații ar putea fi puși să îndeplinească mai multe taskuri.

# 8. Bibliografie

1. Tema\_4.pdf

2. ASSIGNMENT\_4\_SUPPORT\_PRESENTATION.pdf