**DOCUMENTAȚIE**

**Tema 4**

**Restaurant**

Nume student: Molnar Zsanett-Ingrid

Grupa: 10

# CUPRINS

[1. Obiectivul temei 3](file:///E:\Facultate\Anul%202\SEM%202\TP\PT2021-2022_Documentation_Template.doc#_Toc95297885)

[2. Analiza problemei, modelare, scenarii, cazuri de utilizare 3](file:///E:\Facultate\Anul%202\SEM%202\TP\PT2021-2022_Documentation_Template.doc#_Toc95297886)

[3. Proiectare 3](file:///E:\Facultate\Anul%202\SEM%202\TP\PT2021-2022_Documentation_Template.doc#_Toc95297887)

[4. Implementare 3](file:///E:\Facultate\Anul%202\SEM%202\TP\PT2021-2022_Documentation_Template.doc#_Toc95297888)

[5. Rezultate 3](file:///E:\Facultate\Anul%202\SEM%202\TP\PT2021-2022_Documentation_Template.doc#_Toc95297889)

[6. Concluzii 3](file:///E:\Facultate\Anul%202\SEM%202\TP\PT2021-2022_Documentation_Template.doc#_Toc95297890)

[7. Bibliografie 3](file:///E:\Facultate\Anul%202\SEM%202\TP\PT2021-2022_Documentation_Template.doc#_Toc95297891)

1. **Obiectivul temei:**

Cerința temei este de a crea și implementa o aplicație care sa reprezinte un restaurant. Aplicația are ca primă fereastră, o fereastră de log in, unde utilizatorul va introduce username-ul si parola. În funcție de rolul pe care îl are utilizatorul în fișierul administrator, client sau angajat. Dacă utilizatorul are rolul de administrator, acesta poate sa importeze produsele aflate in fisierul products.csv, poate sa adauge produse noi in meniu, să modifice produsele și sa le șteargă. De asemenea, acesta poate să creeze un produs nou care sa conțină mai multe produse mai mici, acest lucru se face prin apelarea metodelor composite produsct. În plus, administratorul poate sa accese 4 rapoarte legate de cum s-au desfășurat order-urile.

Dacă utilizatorul este client, acesta poate să vadă meniul cu toate produsele disponibile, dar doar după ce administratorul face importul la produse, apoi poate sa de-a search la produse in lista care pot indeplinii diferite criterii. Atunci cand alege produsele pe care le doreste, clientul le adauga în căruciorul de cumparaturi , vede pretul total, si depunde comanda. Aceasta comanda este treansmisa spre angajat, iar atunci cand utilizatorul se logeaza ca si angajat are acces la notificarea legata de comanda.

Obiectivele secundare ale proiectului:

* Analizarea problemei prin înțelegerea corectă a cerinței și modelarea de scenarii și cazuri de utilizare.
* Proiectarea unei soluții care să se plieze pe toate cazurile pe care le introduce utilizatorul.
* Implementarea soluției alese prin scrierea codului implicit și a interfeței utilizator.

Observarea rezultatelor finale și concluziile.

1. **Analiza problemei**

Aplicația a trebuit să îndeplinească mai multe cerințe funcționale, precum:

* Pentru administrator:
* Introducerea produselor din lista products.csv
* Introducerea produselor cu date de la tastatură
* Modificarea produselor deja existente in meniu
* Eliminarea produselor din meniu
* Generarea rapoartelor cerute cu datele introduse de către administrator
* Pentru client:
* Introducerea datelor care vor fi folosite pentru filtrarea produselor din meniu
* Filtrarea produselor din meniu
* Adăugarea produselor la comanda dorită
* Eliminarea produselor din comandă
* Finalizarea comenzii și crearea bonului fiscal ca urmare a achiziționării produselor
* Pentru angajat:
* Să primească o notificare atunci când un client a plasat o comandă
* Toți utilizatorii:
* Să închidă aplicația și să se salveze datele din ultima rulare a aplicației

1. Caz de utilizare: Introducerea produselor din lista products.csv

Actor principal: Utilizatorul(Administratorul)

Principalul scenariu de funcționare:

* Administratorul selectează un produs din tabelul cu produse
* Administratorul apasă pe butonul de adăugare a produsului
* Produsul este adăugat in lista din meniu

Secvența alternativă:

* Administratorul nu selectează niciun produs
* Administratorul apasă pe butonul de adăugarea a produsului
* Se va afișa un mesaj de eroare pentru ca nu a fost selectat niciun produs

1. Caz de utilizare: Introducerea produselor cu date de la tastatura

Actor principal: Utilizatorul(Administratorul)

Principalul scenariu de funcționare:

* Administratorul introduce datele despre produsul dorit
* Administratorul apasă pe butonul de a introduce datele
* Produsul este introdus in meniu

Secvența alternativă:

* Administratorul nu introduce toate datele necesare pentru introducerea produsului in meniu
* Administratorul apasă pe butonul de adăugare a produsului
* Se va afișa un mesaj de eroare care va specifica faptul ca toate câmpurile trebuie completate pentru a se realiza introducerea produsului in meniu

1. Caz de utilizare: Modificarea produselor din meniu

Actor principal: Utilizatorul(Administratorul)

Principalul scenariu de utilizare:

* Administratorul introduce datele care se vor schimba in produsul cu același nume
* Administratorul apasă pe butonul de actualizare a produsului din meniu
* Produsul cu numele respectiv din meniu este actualizat

Secvența alternativă:

* Administratorul introduce datele care se vor schimba in produsul cu numele dorit
* Administratorul apasă pe butonul de actualizare a produsului din meniu
* Produsul cu numele introdus nu se găsește în meniu și se va afișa un mesaj de eroare

1. Caz de utilizare: Eliminarea produsului din meniu

Actor principal: Utilizatorul(Administratorul)

Principalul scenariu de funcționare:

* Administratorul introduce numele produsului pe care urmează să îl șteargă din lista de meniu
* Administratorul apasă pe butonul de eliminare a produsului din meniu
* Produsul cu numele introdus este eliminat din meniu

Secvența alternativă de funcționare:

* Administratorul introduce numele produsului pe care dorește să îl șteargă din lista de meniu
* Administratorul apasă pe butonul de eliminare a produsului din meniu
* Produsul cu numele introdus nu se regăsește în meniu și se afișează un mesaj de eroare

1. Caz de utilizare: Generarea rapoartelor

Actor principal: Utilizatorul(Administratorul)

Principalul scenariu de funcționare:

* Administratorul introduce datele pentru a genera rapoartele dorite
* Administratorul apasă pe butonul care generează raportul dorit
* Se generează raportul

Secvența alternativă:

* Administratorul introduce datele pentru a genera rapoartele dorite
* Administratorul apasă pe butonul care generează raportul dorit
* Nu se generează raportul deoarece nu se poate crea fișierul

1. Caz de utilizare: Introducerea datelor utilizate pentru filtrarea produselor din meniu

Actor principal: Utilizatorul(Clientul)

Principalul scenariu de funcționare:

* Utilizatorul introduce datele care dorește să fie utilizate pentru filtrarea produselor

1. Caz de utilizare: Filtrarea produselor

Actor principal: Utilizatorul(Clientul)

Principalul scenariu de funcționare:

* Se introduc datele necesare pentru filtrare
* Utilizatorul apasă pe butonul pentru filtrarea butonului
* Se afișează produsele care respectă datele introduse

1. Caz de utilizare: Adăugarea produselor

Actor principal: Utilizatorul( Clientul)

Principalul scenariu de funcționare:

* Clientul selectează unul din produsele din meniu
* Clientul apasă pe butonul de adăugare a produsului
* Produsul este adăugat la lista din comandă

Secvența alternativă:

* Clientul nu selectează niciun produs
* Clientul apasă pe butonul de adăugare a produsului
* Nu se adaugă niciun produs și se va afișa un mesaj de eroare

1. Caz de utilizare: Finalizarea comenzii si achitarea acesteia

Actor principal: Utilizator(Client)

Principalul scenariu de funcționare:

* Clientul își face lista de produse pe care le va comanda
* Clientul apasă pe butonul de achitarea notei de plată
* Se finalizează comanda și se creează nota de plată într-un fișier de tip text

Secvența alternativă:

* Clientul nu are niciun produs in lista
* Clientul apasă pe butonul de plată
* Nu se realizează comanda din cauza lipsei de produse din lista sau dacă acesta introduce produse, nu se va crea fișierul pentru nota de plată, astfel afișându-se un mesaj de eroare corespunzător

1. Caz de utilizare: Notificarea angajatului

Actor principal: -

Principalul scenariu de funcționare:

* Clientul face o comandă
* Aplicația va trimite o notificare către angajat
* Angajatul primește notificarea și procesează comanda

1. Caz de utilizarea: Salvarea datelor între rulări

Actor principal: Utilizatorul

Principalul scenariu de funcționare:

* Utilizatorul închide aplicația
* Se salvează datele din această execuției prin intermediul serializării

Secvența alternativă:

* Utilizatorul închide aplicația

Aceasta se închide neașteptat și se pierd datele din acea rulare

Diagram

Description automatically generated

1. **Proiectare:**

Aplicatia este impartita in mai multe pachete. Pachetul bussines contine in primul rand clasa abstracta MenuItem, cele doua clase care o extind BaseProduct si CompositeProduct, Interfata DeliveryServiceProdeccing si clasa DeliveryService care contine toate metodele folosite in proiect, si clasa Order. Pachetul data contine clasa User care ia utilizatorii salvati in fisierul user.txt si clasa de serializare, respectiv calsele Administrator, Client, Employee, Serializer. Pachetul presentation contine toate interfetele: AdministratorGUI, ClientGUI, EmployeeGUI, CompositeGUI, MainGUI si Controllere: MainController, AdminController, ClientController, EmployeeController unde se afla toate listenerele la butoane.

Trecând la partea practică de rezolvare a problemei, și anume proiectarea în limbaj OOP, au fost urmăriți mai mulți pași, primul fiind crearea unei Diagrame de pachete urmărind modelul MVC (Model-View-Controller), dar si modelul Layers folosind Business Layer, Presentation Layer, Model Layer. Controllerul face legătura dintre clasele din Model și View prin intermediul butoanelor apăsate.

Intrând într-o analiză mai amănunțită a proiectului, am generat și diagrama UML. Aici se pot vedea în detaliu legăturile dintre clase, metodele folosite, precum și structura logică a întregului proiect.

Pentru a crea diagrama UML am folosit o extensie din Eclipse. Pe diagrama se observa legaturile dintre clase si modul in care a fost gandit acest proiect.

Diagram

Description automatically generated

1. **Implementare:**

Pentru implementarea acestui program a trebui sa implementez mai multe design patternuri precum Composite Design Pattern, Observer Design Pattern, Design by Contract method si o metoda well formed, o fereastra de log in care se verifica cine face operatiile, administratorul, angajatul sau clientul, pentru in fiecare in parte trebuiau implementate diferite operatii folosind streams sau un observer.

**Pachetul bussines**

In primul rand, am implementat o clasa abstracta numita MenuItem care contine toate fieldurile cerute pentru produse: title, raiting, calories, proteins, sodium, fats, price. Aceasta clasa este extinsa de catre BaseProduct, respectiv CompositeProduct. Apoi am si clasa Order care defineset pentru comenzi 3 atribute, data, id-ul orderului si numele clinetului care a facut comanda. In al doilea rand,am implementat si interfata IDeliveryServiceProcessing care contine toate metodele din clasa DeliveryService. Aceasta clasa, DeliveryService are principalele metode folosite pentru a face logica din spatele aplicatiei:

* Metoda **importProducts** este o metoda folosita pentru a imparti elementele primite dintr-un file in elemente de tipul BaseProduct, cu toate fieldurile din MenuItem. Aceasta clasa este implementata cu streams si expresii lambda.
* Metoda **addMenuItem** este functia care adauga in lista de produse un nou produs, clasa este chemata in interfata administratorului.
* Metoda **deleteMenuItem** este functia care sterge un item din lista de produse in functie de numele pe care adimistratorul il introduce. Aceasta metoda parcurge lista de produse si cand gaseste un produs ce este egal cu cel introdus de administrator ii da remove din lista.
* Metoda **modifyMenuItem** este folosita de catre administrator pentru a modifica un field al unui MenuItem in functie de valoarea unui alt field.
* Metoda **createOrder** este folosita pentru a crea o comanda si pentru a o adauga in lista de comenzi a DeliveryService-ului, respectiv in TreeMapul care contine comenzile impreuna cu fiecare elemente ale acestor comenzi
* Metoda **ComputePrice** este folosita pentru a genera suma dintre toate preturile produselor adaugate in cosul de cumparaturi
* Metodele **searchByTitle, searchByCalories, searchByRating, searchByProtein, searchByFat, searchBySodium, searchByPrice,** care folosesc streams si lambda expressions penrtu a filtra produsele din lista de produse in functie de ce introduce clientul
* Aici apar si cele patru metode destinate celor patru rapoarte de la administrator, toate sunt implementate folosind stream si indeplinesc urmatoarele: **report1** metoda care primeste doua inturi (ore) si care verifica daca sunt comenzi intre orele respective si pune rezultatul in fisier, **report2** primeste ca parametru un int si verifica ce produse au fost comandate de mai mult decat numarul dat, **report3** primeste ca parametru doua numere, unul pentru numarul de orderuri si al doilea pentru pretul minim si **report4** care primeste ca parametru o zi,o luna si un an sub forma de int si cauta produsele comandate in ziua respectiva si de cate ori au fost comandate.

**Pachetul data**

Pachetul Data contine clasele **Administrator, Client, Employee, FileWriter, Notificator,** **Serializer** si User. Clasa User contine field-urile role, name si password, impreuna cu constructorul si metodele de gettere si settere aferente, iar clasele Administrator, Client si Employee vor mosteni clasa User pentru ca fiecare rol sa fie clar. Clasa Notificator transmite informatiile despre comanda la Employee, scriind si citind din acelasi fisier informatiile. FileWriter este folosita in raporturi si va scrie o factura. Clasa Serializer se ocupă de serializarea și deserializarea datelor din DeliveryService, practic toate datele importante din aplicație, pentru a nu se pierde.

**Pachetul presentation**

Pachetul presentation contine toate interfetele si controller-ele, interfetele au fiecare un rol important si metode care sunt apelate in controlle penrtu a face controlul de la butoane, interfetele sunt: AdiministratorGUI, ClientGUI, EmployeeGUI, MainGUI.

* **Clasa AdministratorGUI** contine pe langa toate elementele specifice JFrame si getterele pentru aceastea, metodele care vor realiza ferestrele operatiilor principale, cum ar fi adaugarea unui produs nou, editarea unui produs, stergerea unui produs. etc
* **Clasa Client** nu are metode importante, doar un constructor
* Clasa **CompositeProduct** este folosita pentru a crea un produs compus, continand metodele **computeRating**, **computePrice**, **computeProtein**, **computeFat**, **computeSodium**, **computeCalories**.
* **MainGUI** este folosita pentru a verifica logarea utilizatorului si pentru a genera mesaje de eroare in cazul in carea logarea nu este corect facuta. La fel si pentru a ii oferii utilizatorului nou oportunitatea de a-si crea un cont nou si doar apasand butonul de Register.
* **Clasa Employee** este clasa definita pentru angajati, acesta primeste o notificare cu comanda de fiecare data cand se face o comanda prin implementarea unui observer.
* In **MainController** În această clasă sunt implementate funcționalitățile butoanelor din MainGUI. Ea implementează interfața ActionListener. Are o metodă importantă: void actionPerformed(ActionEvent e):metodă specifică interfeței ActionListener, ce trebuie implementată în mod obligatoriu. În această metodă se verifică cu câte un if, care buton s-a apăsat, și în funcție de butonul apăsat, se execută acțiunile necesare.
* In **AdminController** În această clasă sunt implementate funcționalitățile butoanelor din AdministratorGUI. Ea implementează interfața ActionListener. Are o metodă importantă: void actionPerformed(ActionEvent e):metodă specifică interfeței ActionListener, ce trebuie implementată în mod obligatoriu. În această metodă se verifică cu câte un if, care buton s-a apăsat, și în funcție de butonul apăsat, se execută acțiunile necesare.
* In **ClientController** În această clasă sunt implementate funcționalitățile butoanelor din ClientGUI. Ea implementează interfața ActionListener. Are o metodă importantă: void actionPerformed(ActionEvent e):metodă specifică interfeței ActionListener, ce trebuie implementată în mod obligatoriu. În această metodă se verifică cu câte un if, care buton s-a apăsat, și în funcție de butonul apăsat, se execută acțiunile necesare.
* In **EmployeeController** În această clasă sunt implementate funcționalitățile butoanelor din EmployeeGUI. Ea implementează interfața ActionListener. Are o metodă importantă: void actionPerformed(ActionEvent e):metodă specifică interfeței ActionListener, ce trebuie implementată în mod obligatoriu. În această metodă se verifică cu câte un if, care buton s-a apăsat, și în funcție de butonul apăsat, se execută acțiunile necesare.

1. **Rezultate**

În fișierul bill.txt va fi afișat conținutul unei comenzi plasate de către un client. Tot pentru client, în câmpul pentru afișarea rezultatelor corespunzătoare cerințelor de căutare, vor fi afișate produse din meniu. În câmpul de text din interfața administratorului se vor afișa rezultatele rapoartelor. În interfața pentru angajat la fiecare comandă depusă se vor afișa detaliile comenzii respective.

1. **Concluzii**

Proiectul a fost un bun exercițiu pentru a-mi reaminti conceptele POO învățate în trecut, și chiar aprofundarea acestora. Sunt câteva lucruri pe care le-am învățat din experiența acestui proiect pe care le voi prezenta în continuare. În primul rând, probabil cel mai important lucru este gestionarea corectă a timpului, pentru că o organizare bună, începerea din timp la rezolvarea unei sarcini asigură găsirea mai multor modalități de rezolvare. În al doilea rând, modelarea corectă a problemei de la început ajută la rezolvarea mai rapidă a acesteia și ne scapă de implementarea unor funcții care nu sunt utile. Din acest proiect am invatat sa aprofundez mai bine design pattern-urile din POO. O dezvoltare ulterioara a aplicatiei este adaugarea unei adrese de email pentru client si vizionarea cosului de produse cumparate .

1. **Bibliografie:**

Streams:

<https://www.baeldung.com/rxjava-vs-java-flow-api>

<https://www.baeldung.com/java-9-reactive-streams>

Observer:

<https://www.baeldung.com/java-observer-pattern>

File Writer:

<https://javarevisited.blogspot.com/2012/01/what-is-assertion-in-java-java.html>

<https://www.geeksforgeeks.org/java-swing-jtable/>

<https://docs.oracle.com/javase/7/docs/api/java/io/BufferedWriter.html>

Search:

<https://stackoverflow.com/questions/49462114/search-for-book-using-multiple-criteria-in-java>

Composite Design Pattern:

<https://docs.oracle.com/javase/7/docs/api/java/awt/MenuItem.html>