Logo, company name

Description automatically generated

**Documentatie tema 4**

**Nume student: Maracine Stefania Maria**

**Profesor coordonator: Rancea Alexandru**

**Grupa: 30223**

# CUPRINS

[1. Obiectivul temei **.**](#_Toc95297885)

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

[3. Proiectare **.**](#_Toc95297887)

[4. Implementare **.**](#_Toc95297888)

[5. Rezultate **.**](#_Toc95297889)

[6. Concluzii **.**](#_Toc95297890)

[7. Bibliografie **.**](#_Toc95297891)

1. **Obiectivul temei**

Obiectivul acestei teme este proiectarea si implementarea unei aplicatii de catering pentru o companie ce detine o gama larga de produse.

Obiectivele secundare ale acestui proiect sunt urmatoarele:

* Analiza problemei si identificarea cerintelor;
* Proiectarea unei aplicatii de gestiune a comenziloe de mancare;
* Implementarea aplicatiei de gestiune a comenzilor de mancare;
* Testarea aplicatiei de gestiune a comenzilor de mancare;

# Analiza problemei, modelare, scenarii, cazuri de utilizare

Problema este cea mentionata mai sus, implementarea unei aplicatii care poate fii utilizata de clienti si gestionata de administrator.

Fiecare utilizator al aplicatie are anumite actiuni pe care le poate face prin intermediul interfetei grafice.

Asa dar, administratorul are urmatoarele atribute:

* Aplicatia de gestionare ar trebui sa permita administratorului sa poata sa se logheze in aplicatie folosind usernameul si parola corespunzatoare.
* Aplicatia de gestionare ar trebui sa permita administratorului sa poata sa populeaze tabela initiala de produse prin importarea setului initial de produse
* Aplicatia de gestionare ar trebui sa permita administratorului sa poata sa adauge noi produse
* Aplicatia de gestionare ar trebui sa permita administratorului sa poata sa stearga produse din meniu
* Aplicatia de gestionare ar trebui sa permita administratorului sa poata sa modifice produse deja existente
* Aplicatia de gestionare ar trebui sa permita administratorului sa poata sa aleaga anumite produse de baza din meniu si sa alcatuiasca un produs compus din produsele selectate
* Aplicatia de gestionare ar trebui sa permita administratorului sa poata sa genereze 4 tipuri de rapoarte ce vor fii posibil de fizualizat prin intermediul unor fisiere de tip .txt

De asemenea, clientul are urmatoarele atribute:

* Aplicatia de gestionare ar trebui sa permita clientului sa poata sa se logheze in aplicatie folosind usernameul si parola corespunzatoare.
* Aplicatia de gestionare ar trebui sa permita clientului sa poata sa vizualizeze toate produsele din meniul important de admin
* Aplicatia de gestionare ar trebui sa permita clientului sa poata sa comande unul sau mai multe produse din lista importata de admin
* Aplicatia de gestionare ar trebui sa permita clientului sa poata sa descarce facture comenzii plasate in format .txt
* Pentru clientii noi, aplicatia de gestionare ar trebui sa le permita acestora sa poata sa se inregistreze cu un user si o parola cu care se pot loga mai tarziu in aplicatie

# Diagram Description automatically generatedProiectare

In acest capitol este prezentata diagrama UML, clasele existente, pachetele folosite si structurile de date folosite.

Pachete folosite sunt urmatorele in numar de trei: pachetul BusinessLayer, pachetul Interface si pachetul Serialization.

Pachetul BusinessLayer contine 6 clase: BaseProduct, CompositeProduct, DeliveryService, MenuItem, Order, Client. De asemenea acest pachet contine si o interfata IDeliveryServiceProcessing.

Pachetul Interface are in componenta un numar de 4 clase care au rolul de a defini interfata grafica cu utilizatorul. Clasele din componenta sunt: AdministratorForm, ClientForm, LogInForm, ReportsForm. De asemenea acest pachet contine si o clasa ce are metoda main: Test si de unde este apelata initial interfata grafica.

Pachetul Serialization are in componenta o singura clasa si anume clasa Serializator.

# Implementare*.*

In aceasta parte vom descrie fiecare clasa in parte si funtionalitatile acesteia, atributele pe care le are si metodele pe care le-am implementat.

**Clasa BaseProduct:**

Aceasta clasa extinde clasa MenuItem si implementeaza Serializable.

In aceasta clasa gasim cele 7 atribute specific fiecarui produs din meniul nostru: String title, double rating, int calories, int protein, int fat, int sodium, int price.

De asemenea am implementat un constructor cu toti parametrii mentionati mai sus: **public BaseProduct**( String title, double rating, int calories, int protein, int fat, int sodium, int price ) ce initializeaza toti parametrii, dar avem si un constructor gol.

Ca si metode implementate avem metoda **public String toString()** pentru a putea afisa mai usor informatiile din aceasta clasa.

De asemenea avem nevoie de metode de tip getter si setter pentru fiecare atribut in parte:

**public String getTitle()** – esteun getter pentru atributul title ce o sa returneze atributul title deoarece este de tip private.

**public void setTitle(String title)** – este un setter pentru atributul title ce o seteze atributul title cu variabila title data ca parametru, deoarece este de tip private

**public double getRating()** – este un getter pentru atributul rating ce o sa returneze atributul rating deoarece este de tip private.

**Public void setRating(double rating)** – este un setter pentru atributul rating ce o sa seteze acest atribut cu variabila rating data ca parametru, deoarece este de tip private

**Public int getCalories()** – este un getter pentru atributul calories ce o sa returneze atributul calories deoarece este de tip private.

**Public void setCalories(int calories)** – este un setter pentru atributul calories ce o sa seteze acest atribut cu variabila calories data ca parametru deoarece este de tip private.

**Public int getProtein()** – este un getter pentru atributul protein ce o sa returneze atributul protein deoarece este de tip private

**Public void setProtein(int protein)** – este un setter pentru atributul protein ce o sa seteze acest atribut cu variabila protein data ca parametru deoarece este de tip private.

**Public int getFat()** – este un getter pentru fat ce o sa returneze atributul fat deoarece este de tip private.

**Public void setFat(int fat** ) – este un setter pentru atributul fat ce o sa seteze acest atribut cu variabila fat data ca parametru deoarece este de tip private

**Public int getSodium()** – este un getter pentru atributul sodium ce o sa returneze atributul sodium pentru ca este de tip private

**Public void setSodium(int sodium)** – este un setter pentru atributul sodium ce o sa seteze acest atribut cu variabila sodium data ca parametru deoarece este de tip private.

**Public int getPrice()** – este un getter pentru atributul price ce o sa returneze atributul price pentru ca este de tip private.

**Public void setPrice(int price)** – este un setter pentru atributul price ce o sa seteze atributul price cu variabila price data ca parametru deoarece este de tip private.

**Clasa CompositeProduct:** aceasta clasa extinde MenuItem si implementeaza Serializable.

In aceasta clasa gasim cele 7 atribute specific fiecarui produs din meniul nostru: String title, double rating, int calories, int protein, int fat, int sodium, int price.

Are de asemenea atat un constructor ce seteaza cele 7 atribute de mai sus, un gonstructog gol, cat si gettere si settere pentru toate atributele, asemanator cu ce s-a realizat in clasa BaseProduct, dar si metoda toString pentru o afisare mai frumoasa.

**Clasa MenuItem:** aceasta este clasa mama pentru produsele noastra si extinde Serializable. In aceasta clasa care este de tip abstract gasim urmatoarele metode: public abstract String getTitle(); public abstract double getRating(); public abstract int getCalories(); public abstract int getProteins(); public abstract int getFats(); public abstract int getSodium(); public abstract int getPrice(); dar si metoda toString;

**Clasa Client**: aceasta clasa implemeneaza Serializable. Aceasta continue datele necesare pentru un client. Are ca parametrii private String name; private String parola; si private int id.

De asemenea are un constructor gol, un contructor care seteaza datele pentru fiecare atribut si metode de tip getter si setter pentru a putea ultiliza informatiile si din alte clase.

**Clasa Order**: aceasta clasa implemeneaza Serializable. Aceasta continue datele necesare pentru un client. Are ca parametrii private int orderID; private int clientID; si private Date orderDate = new Date(). De asemenea are un constructor gol, un contructor care seteaza datele pentru fiecare atribut si metode de tip getter si setter pentru a putea ultiliza informatiile si din alte clase.

**Clasa Administrator:** aceasta clasa implemeneaza Serializable. Aceasta continue datele necesare pentru un client. Are ca parametrii private String username; si private String password. De asemenea are un constructor gol, un contructor care seteaza datele pentru fiecare atribut si metode de tip getter si setter pentru a putea ultiliza informatiile si din alte clase. Avem la final si metoda toString.

**Clasa Serializator**: acesta clasa implementeaza serializable si este clasa in care am implementat toate metodele de serializare a datelor utilizate in proiectul nostru.

Incepem cu serializarea datelor de logare pentru admin. Avem doua metode pentru acest lucru: public void serializeAdmin(Administrator admin) care ia datele de logare salvate in obiectul de tip admin pe care il primeste ca parametru si le serializeaza intr-un fisier de tip .ser si metoda public Administrator deserializeAdmin() care deschide fisierul fisierul AdminInfo.ser si deserializeza informatiile de logare ale unui admin, salvandu-le intr-ul obiect de tip admin.

Urmeaza serializarea datelor de logare pentru client care urmeaza aceeasi metoda. Avem doua metode pentru acest lucru: public void serializareClient(Client client) care ia datele de logare salvate in obiectul de tip client pe care il primeste ca parametru si le serializeaza intr-un fisier de tip .txt si metoda public List<Client> deserializareClient() care deschide fisierul fisierul DateLogareClienti.txt si deserializeza informatiile de logare ale unui client, salvandu-le intr-ul obiect de tip client.

Urmatorul lucru pe care il serializam este meniul de produse.Dupa citirea initiala din fisierul .csv si adaugarea produselor intr-o listata de menu items vom serialize intreg meniul intr-un fisier MenuItem.txt pe care il vom serialize si deserializa de fiecare data cand vom avea nevoie si inainte si dupa fiecare uptade pe care il facem la lista noastra de produse. Pentru acest lucru folosim metodele public void writeObject (List<MenuItem> list) si public List <MenuItem> readObject().

Ultima serializare este serializarea pentru Maps realizata cu ajutorul metodelor public void serializareMap (Map<Order, List <MenuItem>> map) si public Map <Order, List <MenuItem>> deserializareMap().

**Clasa DeliveryService:** aceasta clasa implementeaza Serializable. Este cea mai importanta clasa deoarece continue majoritatea metodelor pentru a pune la punct aplicatia noastra si pentru a o face sa functioneze. Prima metode este cea de importare a produselor din fisierul initial. public List<MenuItem> importProducts() deschide fisierul products.csv si pune toate produsele intr-o lista de menu items. Informatiile sunt luate linie cu linie iar fiecare atribut al produsului este despatit dupa fiecare virgula. Dupa ce lista a fost populate o serializam.

Metoda public TableModel tableModel() populeaza tabela noastra de unde pot fii vizualizate toate produsele din meniu. Prima data ne luam un obiect de tip default table model si cu ajutorul unui vector de stringuri predefinit de noi setam numele coloanelor, dupa care deserializam lista de produse si incepem sa populam tabela rand cu rand.

Metoda public void addItem(BaseProduct bs) primeste ca parametru un base product. Incepem prin a deserializa lista de produse dupa care adaugam in lista noastra produsul pe care l-am primit ca parametru si apoi serializam din nou lista dupa actualizare.

Metoda public void deleteItem(String str) deserializeaza lista, cauta in lista produsul cu numele corespunazator cu parametrul pe care l-am transmis functiei, il sterge din lista la gasire si apoi serializeaza lista.

Metoda public void modifyItem(String str, String t1, String t2, String t3, String t4, String t5, String t6, String t7) modifica unul sau mai multe campuri din compozitia unui base product, in functie de datele pe care le introduce administratorul.

Metoda public void createItem(List<BaseProduct> lista, String t1, String t2, String t3, String t4, String t5, String t6) lasa administratorul sa creeze un nou produs.

Metoda public TableModel searchItem(String t1, String t2, String t3, String t4, String t5, String t6, String t7) lasa utilizatorul sa caute un produs dupa una sau mai multe categorii.

Metoda public void createOrder(List<MenuItem> listaComanda, int ID, String numeClient) lasa un client sa plaseze o anumita comanda formata din unul sau mai multe produse iar dupa plasare o chitanta este formata in care apare data curente, produsele cumparate, numele clientului si pretul total.

Ultimele 4 metode sunt metode pentru generarea rapoartelor, metode apelate de admin. Public void raport1 afiseasa intr-un fisier txt toate comenzile amplasate intr-un anumit interval orar dar de administrator. Public void raport2 afiseara intr-un fisier txt toate produsele comandate de un numar de ori mai mare fata de cel specificat de admin si de cate ori au fost comandate. Public void raport 3 pune intr-un fisier txt clientii care au comandat de un nr de ori mai mare decat cel metionat de admin si cu o valoarea toatala mai mare decat cea mentionata de admin. Public void raport4 pune in fisier txt produsele cumparate intr-o anumita zi si de cate ori au fost achizitionate.

Mai avem si clasele **AdinistratorForm, LogInForm,ClientForm si ReportsForm** unde facem design-ul si apelam metodele pentru interfata.

# 5.Rezultate

Pentru aceasta tema nu s-a folosit nicio metoda de testare.

# 6.Concluzii *.*

Aceasta aplicatie pentru gestionarea comenziilor companiei de catering poate avea urmatoarele imbunatatiri:

* crearea unei interfetei de employee cu ObserverPattern.

Consider ca din aceasta tema am invatat la ce este utila Serializarea, dar am si imbunatatit lucrurl cu stream-uri si lambda expressions.

# 7.Bibliografie

<https://www.baeldung.com/java-iterate-map>

<https://stackoverflow.com/questions/29888539/java-8-map-lambda-expression>

<https://www.java67.com/2015/01/java-8-map-function-examples.html>

<https://www.baeldung.com/jackson-map>

<https://mkyong.com/java8/java-8-streams-map-examples/>