Documentație de temă

Food Delivery Service

Proiect realizat de: Anton Ioana-Carina

Grupa 30225

Cuprins

[Obiectivul temei 4](#_Toc73522676)

[Obiectivul principal 4](#_Toc73522677)

[Obiectivul secundar 4](#_Toc73522678)

[Analiza problemei 4](#_Toc73522679)

[Diagrama use-case 4](#_Toc73522680)

[Administrator Operations 4](#_Toc73522681)

[Client Operations 6](#_Toc73522682)

[Cerinte functionale 7](#_Toc73522683)

[Cerinte non-functionale 7](#_Toc73522684)

[Decizii de proiectare 7](#_Toc73522685)

[Java Streams & Lambda Expressions 7](#_Toc73522686)

[Java serialization 8](#_Toc73522687)

[HashMap 8](#_Toc73522688)

[Diagrama UML 8](#_Toc73522689)

[Pachete 9](#_Toc73522690)

[Implementare 9](#_Toc73522691)

[Bussineslayer 9](#_Toc73522692)

[BaseProduct 9](#_Toc73522693)

[CompositeProduct 9](#_Toc73522694)

[DeliveryService 9](#_Toc73522695)

[IDeliveryService 9](#_Toc73522696)

[MenuItem 9](#_Toc73522697)

[Order 9](#_Toc73522698)

[User 10](#_Toc73522699)

[Datalayer 10](#_Toc73522700)

[Serializator 10](#_Toc73522701)

[Presentationlayer 10](#_Toc73522702)

[AdminGUI 10](#_Toc73522703)

[ClientGUI 10](#_Toc73522704)

[EmployeeGUI 10](#_Toc73522705)

[RegisterGUI 10](#_Toc73522706)

[HomePage 10](#_Toc73522707)

[Controller 10](#_Toc73522708)

[Start 10](#_Toc73522709)

[Start 10](#_Toc73522710)

[Rezultate 11](#_Toc73522711)

[Interfete 11](#_Toc73522712)

[Operatii 11](#_Toc73522713)

[Concluzii 12](#_Toc73522714)

[Dezvoltari ulterioare 12](#_Toc73522715)

[Ce am invatat 12](#_Toc73522716)

[Bibliografie 12](#_Toc73522717)

# Obiectivul temei

## Obiectivul principal

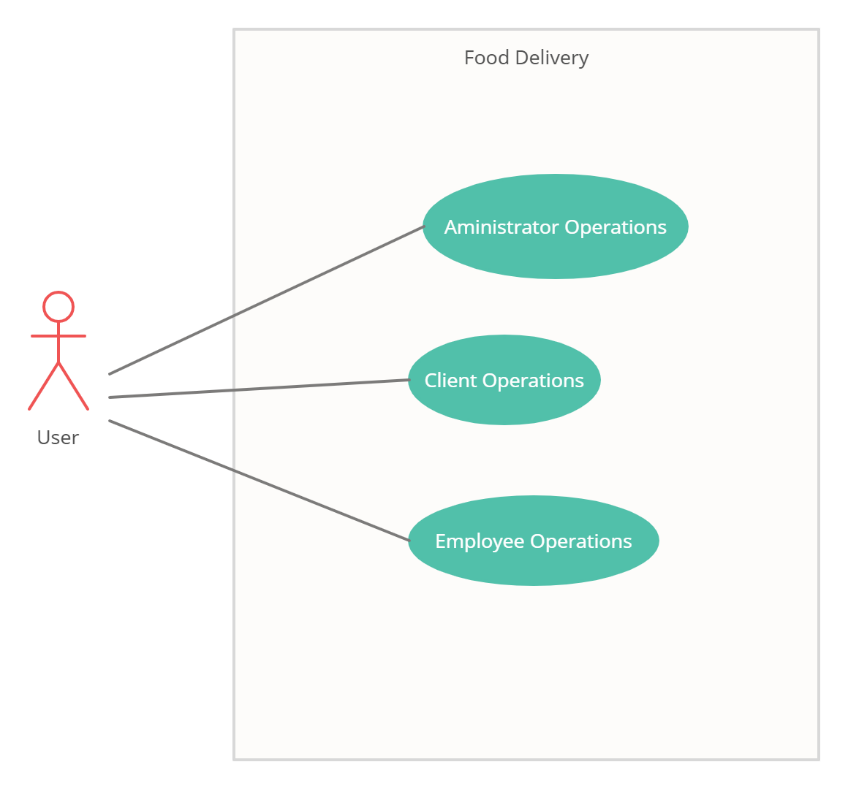
Implementarea unei aplicatii de FoodDelivery pentru o companie de catering. Clientul poate sa comande produse din meniul companiei. Sistemul ar trebui sa aiba 3 tipuri de utilizatori care se vor loga utilizand un nume de utilizator si o parola: administrator, angajat normal si client.

## Obiectivul secundar

|  |  |  |
| --- | --- | --- |
| **Analiza problemei si identificarea cerintelor** | Se stabileste modul de functionare al aplicatiei, se construiesc diagramele use-case si se stabilesc cerintele functionale, respectiv nefunctionale ale problemei | Se detalieaza in capitolul 2 |
| **Proiectare** | Se stabileste structura modelului dupa care se va construi programul, se stabilesc pachetele, clasele, relatiile dintre acestea (Diagrama UML), etc. | Se detaliaza in capitolul 3 |
| **Implementare** | Se transcriu in cod efectiv diagramele de la etapa de proiectare | Se detaliaza in capitolul 4 |
| **Testare** | Se utilizeaza un framework de testare pentru a verifica functionalitatea aplicatiei si cazurile acoperita de acestea | Se detaliaza in capitolul 5 |

# Analiza problemei

## Diagrama use-case



### Administrator Operations

**Use Case**: Import the initial set of products

**Primary Actor**: utilizatorul

**Main Success Scenario**:

1. Utilizatorul apasa butonul de generare produse

2. Programul va executa operatia si va prelua produsele fin fisierul .csv

**Use Case**: Add new product

**Primary Actor**: utilizatorul

**Main Success Scenario**:

1. Utilizatorul introduce datele pentru inserarea unui nou produs

2. Utilizatorul apasa butonul de confirmare

3. Programul va executa interogarea si va insera un nou produs in lista

**Alternative Sequence**: Incorrect input data

- Utilizatorul introduce datele in mod gresit

- Scenariul se intoarce la pasul 1

**Use Case**: Add new composite product

**Primary Actor**: utilizatorul

**Main Success Scenario**:

1. Utilizatorul alege componentele meniului zilei

2. Utilizatorul apasa butonul de confirmare

3. Programul va executa interogarea si va insera un nou produs compus in lista

**Alternative Sequence**: Incorrect input data

- Utilizatorul introduce datele in mod gresit

- Scenariul se intoarce la pasul 1

**Use Case**: Modify product

**Primary Actor**: utilizatorul

**Main Success Scenario**:

1. Utilizatorul introduce datele pentru modificarea unui produs din lista

2. Utilizatorul apasa butonul de confirmare

3. Programul va executa interogarea si va actualiza produsul din lista

**Alternative Sequence**: Incorrect input data

- Utilizatorul introduce datele in mod gresit

- Scenariul se intoarce la pasul 1

**Use Case**: Delete

**Primary Actor**: utilizatorul

**Main Success Scenario**:

1. Utilizatorul introduce index ul din lista al produsul pe care doreste sa il stearga

2. Utilizatorul apasa butonul de confirmare

3. Programul va executa interogarea si va elimina din lista produsul respectiv

### Client Operations

**Use Case**: Register

**Primary Actor**: utilizatorul

**Main Success Scenario**:

1. Utilizatorul introduce datele cerute

2. Utilizatorul apasa butonul de confirmare

3. Programul va executa interogarea si va salva datele de logare

**Alternative Sequence**: Incorrect input data

- Utilizatorul introduce datele in mod gresit

- Scenariul se intoarce la pasul 1

**Use Case**: View menu

**Primary Actor**: utilizatorul

**Main Success Scenario**:

1. Utilizatorul apasa butonul de vizualizare a meniului

2. Programul va executa interogarea si va insera afisa pe ecran meniul

**Use Case**: Search product

**Primary Actor**: utilizatorul

**Main Success Scenario**:

1. Utilizatorul introduce numele produsului cautat

2. Utilizatorul apasa butonul de confirmare

3. Programul va executa interogarea si va afisa pe ecran produsele care corespund numelui

**Alternative Sequence**: Incorrect input data

- Utilizatorul introduce datele in mod gresit

- Scenariul se intoarce la pasul 1

**Use Case**: Place order

**Primary Actor**: utilizatorul

**Main Success Scenario**:

1. Utilizatorul introduce numele a 3 produse

2. Utilizatorul apasa butonul de confirmare

3. Programul va executa interogarea si va crea o noua comanda

**Alternative Sequence**: Incorrect input data

- Utilizatorul introduce datele in mod gresit

- Scenariul se intoarce la pasul 1

## Cerinte functionale

* Programul ar trebui sa permita utilizatorului sa se inregistreze
* Programul ar trebui sa permita utilizatorului sa intre in contul personal
* Programul ar trebui sa permita clientilor sa vizualizeze meniul
* Programul ar trebui sa permita clientilor sa cauta produse
* Programul ar trebui sa permita clientilor sa isi puna o comanda
* Programul ar trebui sa permita administratorilor sa adauge produse
* Programul ar trebui sa permita administratorilor sa stearga produse
* Programul ar trebui sa permita administratorilor sa modifice produse
* Programul ar trebui sa anunte angajatii cand a fost plasata o noua comanda

## Cerinte non-functionale

* Aplicatia ar trebui sa aiba o interfata grafica atractiva, intuitiva si usor de utilizat atat pentru fiecare categorie de operatii cat si pentru pagina de pornire.

# Decizii de proiectare

## Java Streams & Lambda Expressions

Streamurile reprezinta o secventa de elementa asupra carora se pot efectua diferite tipuri de operatii pentru a face computatii asupra acelor elemente.

Operatiile pe streamuri sunt fie intermediare sau terminale. Operatiiel intermediare returneaza un stream deci putem insira mai multe operatii intermediare fara sa folosim ; .

Operatiile terminale sunt fie void, fie returneaza un rezultat care nu este stream.

Majoritatea operatiilro pe strea accepta ca parametru o expresie lambda, o interfata functionata care specifica exact comportamentul operatiei. Majoritatea acestor operatii trebuie sa fie atat non-interfering cat si stateless.

O functie este non-interfering cand nu modifica sursele de baza ale streamului.

O functie este stateless cand executia operatiei este determinista, de exemplu cand nicio expresie lamba nu depinde de orice variabile mutabile sau stari din exterior care s-ar putea schimba in timpul executiei.

## Java serialization

Java ne ofera un mecanism numit serializarea obiectelor unde un obiect poate fi reprezentat ca o secventa de biti care include atat datele obietului cat si informatii despre tipul obiectului si tipurile de date stocate in obiect.

Dupa ce un obiect serializat a fost scris intr-un fisier, acesta poate fi citi din fisier si deserializat, adica informatiile legate de tipul obiectului si bitii care reprezinta obiectul port fi utilizati pentru a recrea obiectul in memorie.

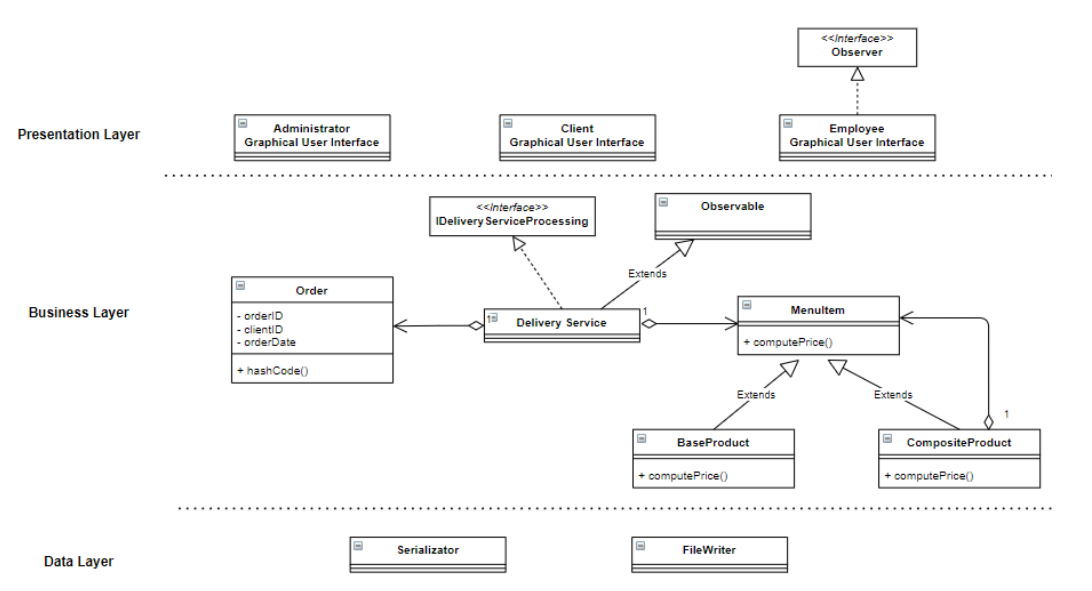
Pentru a serializa un obiect clasa obiectului trebuie sa implementeze metoda **writeObject()** pentru a scrie obiectul, respectiv metoda **readObject()** pentru a deserializa obiectul.

## HashMap

Hashmap-ul in Java functioneaza pe principiile de hashing. Este o structura de date care ne permite sa stocam obiecte si sa le preluam in timp constant O(1) cu conditia ca noi sa stim cheia respectiva valorii. In hashing, functiile hash sunt folosite sa lege cheile de valori in HashMap. Obiectele sunt stocate utilizand functia **put(key,value)**, metoda a HashMap si se preia obiectul apeland **get(key)**. Cand apelam metoda  **put**, metoda **hashcode()** al obiectului cheie este apelata ca functia hash a map-ului sa poata gasi un loc unde sa stocheze obiectul.

Hashmap-ul stocheaza intern maparea sub forma unui obiect de tipul **Map.Entry**, obiect care contine atat cheia cat si valoarea obiectului.

## Diagrama UML



## Pachete

* **Bussineslayer**- clasele legate de modelul de date si logica programului, aici sunt incluse majoritatea claselor, inclusiv cea mai importanta clasa, DeliveryService
* **Datalayer-** instrumentele pentru serializare, acestea au fost utilizate doar pentru protejarea datelor de logare ale utilizatorilor
* **Presentationlayer-** interfetele grafice pentru homepage, pentru fiecare tip de utilizator si pentru anumite operatii care au nevoie de niste date auxiliare pentru a putea fi executate
* **Start-** main-ul aplicatiei unde unde se initializeaza interfetele si se porneste aplicatia

# Implementare

## Bussineslayer

### BaseProduct

**Variabile instanta:**

* **Title**
* **Rating**
* **Calories**
* **Proteins**
* **Fats**
* **Sodium**
* **Price**

**Metode:**

* Clasa BaseProduct reprezinta modelul de date pentru produsele de baza din meniul restaurantului;
* Aceasta extinde clasa MenuItem si ii implementeaza metodele abstracte
* Contine toate caracteristicile produselor de baza din fisierul dat

### CompositeProduct

* Reprezinta modelul de date pentru un produs compus al restaurantului
* Si aceasta clasa extinde clasa MenuItem si ii implementeaza metodele abstracte

### DeliveryService

* Aceasta este clasa principala a aplicatiei
* Aici se stocheaza produsele disponibile in firma, conturile utilizatorilor si tine evidenta comenzilor
* Aceasta clasa implementeaza interfata **IDeliveryService**
* Contine metodele pentru generarea produselor – **generateProducts()**, manipularea lro de catre administrator (add/ modify/ delete) , crearea de comenzi noi – **createNewOrder,** generarea rapoartelor- **createReport**
* In variabila **products** se pastreaza toate produsele de baza din fisierul dat, extrase cu ajutorul streamurilor

### IDeliveryService

* Aceasta clasa contine headerele functiilor implementate in DeliveryService pentru a pastra private datele legate de implementare

### MenuItem

* Clasa care este extinsa de BaseProduct si CompositeProduct, ea contine sabloanele functiilor care trebuie implementate de celalte doua clase care o extind
* Avem de asemenea si niste copii la valorile din BaseProduct pentru a le putea accesa in cadrul structurii de stocare a comenzilor

### Order

* Modelul de date pentru o comanda
* In aceasta functie de calculeaza si cheile hashCode pentru stocarea datelor comenzilor cat mai eficient
* Functia hashcode() prelucreaza un sub forma de int utilizand id ul comenzii, id ul clientului si data si luna in care a fost plasata comanda

### User

* Modelul de date pentru conturile de utilizator
* Ca variabile instanta avem tipul de utilizator sub forma de int: 0 este pentru admin, 1 pentru employee si 2 pentru client si idUser folosim al comenzi, acesta din urma este generat folosind clasa Random
* username ul si parola sunt incluse in clasa Serializator care este extinsa de clasa User

## Datalayer

### Serializator

* Clasa extinsa de clasa User pentru a serializa datele de logare ale utilizatorilor
* Contine o metoda pentr scris si una pentru citit date de utilizator

### Presentationlayer

* In acest pachet sunt incluse toate interfetele grafice necesare pentru rularea aplicatiei
* Pentru interfatele auxiliare necesare la operatiile de la admin si client, au fost categorizare in doua subpachete

### AdminGUI

* Interfata de pornire pentru operatiile specifice administratorilor
* Contine cate un buton pentru fiecare tip de operatie care poate fi facut de catre un administrator, aproape fiecare operatie are cate o interfata separata in cazul in care trebuie introduse si alte date de catre utilizator

### ClientGUI

* Interfata de pornire pentru operatiile specifice clientilor
* La fel ca al interfata de la administrator, avem cate un buton pentru fiecare tip operatie care poate fi facut de catre un client ; un buton poate dupa la o alta fereastra in cazul in care trebuie introduse date in plus pentru operatia respectiva

### EmployeeGUI

* Interfata de pornire pentru angajati
* Interfata aceasta a fost facuta pentru a putea fi anuntati angajatii in cazul in care este inregistrata o comanda noua

### RegisterGUI

* Interfata pentru crearea unui cont de utilizator
* Aici utilizatorii isi aleg un username si o parola si ce fel de cont doresc sa isi creeze
* Aceste date vor fi salvate intr-un fisier serializabil pentru a le pastra intr-un mod sigur
* Ele vor fi deserializate in momentul logarii

### HomePage

* Pagina de pornire/ de logare
* In aceasta pagina utilizatorii isi vor introduce username ul si parola pentru asi putea accesa contul si a efectua operatii specifice acestuia
* In momentul in care va fi apasat butonul logat, programul va verifica daca exista contul si daca ce tip de cont ii si va deschide fereastra potrivita tipului de cont

### Controller

* Clasa Controller este cea unde se face legatura dintre modelul de date si interfetele grafice
* Aici ne asiguram ca fiecare utilizator va avea acces doar al operatiile specifice tipului de cont pe care il are.

## Start

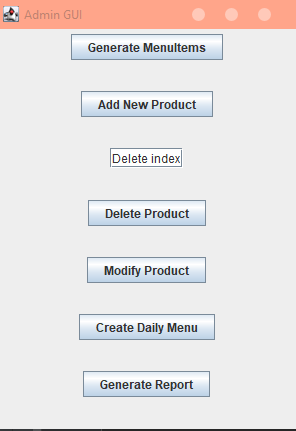
### Start

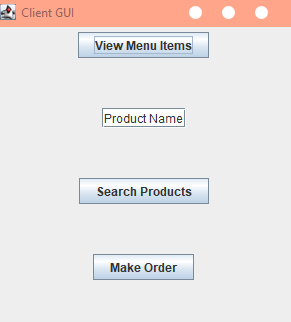
* Aceasta clasa contine codul main al aplicatiei, unde se initializeaza toate interfetele si controller ul.
* Aici s-ar fi salvat si citit datele din DeliveryService dintr-un fisier serializabil

# Rezultate

## Interfete

-Interfata de la Admin



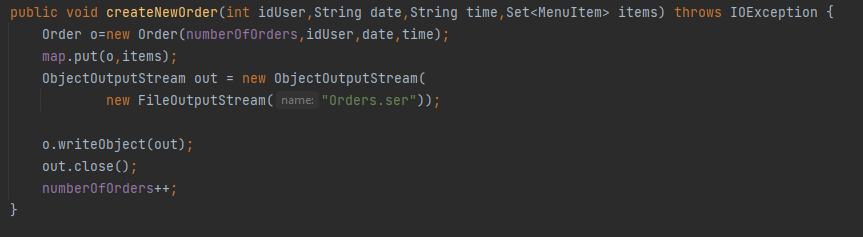
* Interfata de la Client
* 

## Operatii

* **Populare DeliveryService din fisier:**
  + In urma utilizarii streamurilor s-au introdus 12359 din 14002;



* **Make Orders:**
* Crearea unui obiect de tipul Order folosind parametrii primiti de la interfata si scrierea acesteia intr-un fisier serializat ;
* Obiectul Order este utilizat la crearea codului hashcode() sa stim unde sa punem datele din items



# Concluzii

## Dezvoltari ulterioare

* Inlcudere butoane de navigare printre interfete
* Afisare mesaj de eroare la introducerea datelor de logare
* Afisare mesaj de eroare la introducerea datelor necorespunzatoare in anumite JTextField

## Ce am invatat

* Utilizare java streams si a expresiilro lambda in contextul acestora
* Serializarea obiectelor in java si deserializarea
* Colectii care folosesc hashCode, pentru o stocare mai eficienta a datelor
* Citirea dintr-un fisier dat, utilizand streamuri

# Bibliografie

1. <https://www.tutorialspoint.com/java/java_serialization.htm>
2. <https://winterbe.com/posts/2014/07/31/java8-stream-tutorial-examples/>
3. <https://howtodoinjava.com/java8/java-stream-distinct-examples/>
4. https://javarevisited.blogspot.com/2011/02/how-hashmap-works-in-java.html#axzz6wbTRKGav