

**FOOD DELIVERY MANAGEMENT SYSTEM**

Student: Blajan George-Paul

Grupa:302210

[Obiectivul temei 2](#_Toc578442550)

[Analiza problemei 2](#_Toc341783354)

[Proiectare 3](#_Toc1698308054)

[Implementare 5](#_Toc752169291)

[Rezultate 8](#_Toc474166320)

[Concluzii 8](#_Toc366942283)

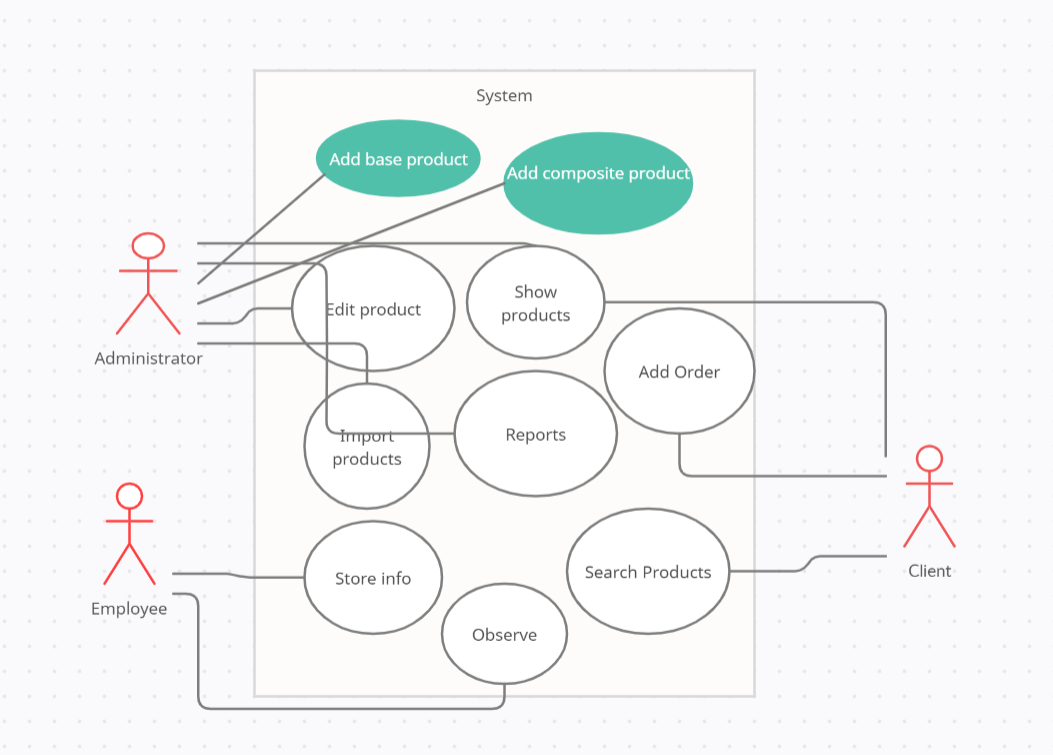
[Bibliografie 8](#_Toc1135390979)

# ***Obiectivul temei***

Obiectivul acestei teme este realizarea unei aplicatii de management a comenzilor pentru o companie de livrare mancare.Pentru a realiza acest lucru a fost nevoie implementarea unei interfete de logare pentru conturi de tip administrator,client si angajat si folosirea unor pattern-uri corespunzatoare cum ar fi composite pattern ,design by contract pattern etc.

# ***Analiza problemei***

Cerinta acestui proiect este implementarea unor operatii de comandare,introducere,afisare produse servite de o companie de cattering,pentru a atinge aceste performante precizate implementarea a fost facut cu structuri de hashing deoarece acestea au o complexitate pentru search de O(1) in general daca este alcatuita o implementarea eficienta(o functie de hash optima).De asemenea au fost implementate 18 interfete grafice cu care utilizatorul poate interactiona pentru buna utilizare a aplicatiei.



Sunt 3 categorii de utilizatori: Administrator,Client si Employee.Administratorul realizeaza operatii cum ar fi introducerea de produse simple sau compuse,editarea de produse,stergerea de produse,realizarea unui import general de produse . Clientul fac operatii precum ,cautarea de produse,comandarea de produse.Employee ul urmareste plasarea comenzilor pentru a putea fi pregatit sa le onoreze.Aceasta urmarire a comenzilor este realizata cu ajutorul interfetei Observer si a clasei Observable.

# ***Proiectare***

Structurile de date necesare pentru acest proiect au fost in mare ArrayList-ul si HashSet-ul.HashSet-ul a fost implementat impreuna cu suprascrierea metodelor hashFunction si equals pentru ca HashSet ul sa poata elimina duplicatele si sa ierahizeze obiectele stocate dupa o functie de hash care le repartiza raportat la un atribut specific al obiectelor.A fost folosit HashSet ca sa stocam comenzile si produsele aflate in meniul restaurantului iar ArrayList ul a fost folosit pentru a stoca conturile inregistrate in aplicatie print interfata de Sign Up.  
Clasa HashSet implementează interfața Set, susținută de un tabel hash care este de fapt o instanță HashMap. Nu se oferă nicio garanție cu privire la ordinea de iterație a setului, ceea ce înseamnă că clasa nu garantează ordinea constantă a elementelor în timp. Această clasă permite elementul nul. Clasa oferă, de asemenea, performanță constantă în timp pentru operațiile de bază, cum ar fi adăugarea, eliminarea, conținerea și dimensiunea, presupunând că funcția hash dispersează elementele în mod corespunzător printre găleți, pe care le vom vedea în continuare în articol.

Câteva caracteristici importante ale HashSet sunt:

• Implementează interfața setată.

• Structura subiacentă a datelor pentru HashSet este Hashtable.

• Pe măsură ce implementează Set Interface, valorile duplicate nu sunt permise.

• Obiectele pe care le introduceți în HashSet nu sunt garantate pentru a fi inserate în aceeași ordine. • Obiectele sunt inserate pe baza codului lor hash.

• Elementele NULL sunt permise în HashSet. HashSet implementează, de asemenea, interfețe serializabile și clonabile.

Acum, pentru menținerea performanței constante a timpului, iterarea peste HashSet necesită timp proporțional cu suma mărimii instanței HashSet (numărul de elemente) plus „capacitatea” instanței HashMap de sprijin (numărul de cupe). Astfel, este foarte important să nu setați capacitatea inițială prea mare (sau factorul de încărcare prea mic) dacă performanța de iterație este importantă.Capacitate inițială: capacitatea inițială înseamnă numărul de cupe atunci când este creat hashtable (HashSet utilizează intern structura de date hashtable). Numărul de găleți va fi mărit automat dacă dimensiunea curentă se umple. Factor de încărcare: factorul de încărcare este o măsură a cât de plin este permis să obțină HashSet înainte ca capacitatea sa să crească automat. Atunci când numărul de intrări în tabelul hash depășește produsul factorului de încărcare și capacitatea curentă, tabelul hash este refăcut (adică structurile de date interne sunt reconstruite), astfel încât tabelul hash are aproximativ de două ori numărul de cupe.

Interfata definita este IDeliveryServiceProcessing care declara antetul metodelor implementate de clasa DeliveryService si este folosita pentru a utiliza Design By Contract Pattern prin aplicarea de tag-uri cum ar fi @pre , @post antetelor metodelor,acest pattern este completat prin verificarea cu ajutorul assert urile a conditiilor in clasa DeliveryService.  
Tehnica de dezvoltare software Design by Contract (DBC) asigură software de înaltă calitate, garantând că fiecare componentă a unui sistem respectă așteptările sale. În calitate de dezvoltator care utilizează DBC, specificați contractele de componente ca parte a interfeței componentei. Contractul specifică la ce se așteaptă această componentă de la clienți și la ce se pot aștepta clienții de la aceasta.

O alta tehnica folosita este Composite Design aplicata pe clasele MenuItem,BaseProduct si CompositeProduct.Aceasta metoda presupune o ierarhire sub forma de arbore ,MenuItem fiind nodul composite,nodul parinte iar BaseProduct si CompositeProduct sunt copii.In aceasta strategie sarcina principala revine claselor copil.

Modelul compozit este un model de proiectare de partiționare și descrie un grup de obiecte care este tratat la fel ca o singură instanță a aceluiași tip de obiect. Intenția unui compozit este de a „compune” obiecte în structuri de copac pentru a reprezenta ierarhii parțiale. Vă permite să aveți o structură de copac și să solicitați fiecărui nod din structura de copac să efectueze o sarcină.

Modelul compozit are patru participanți:

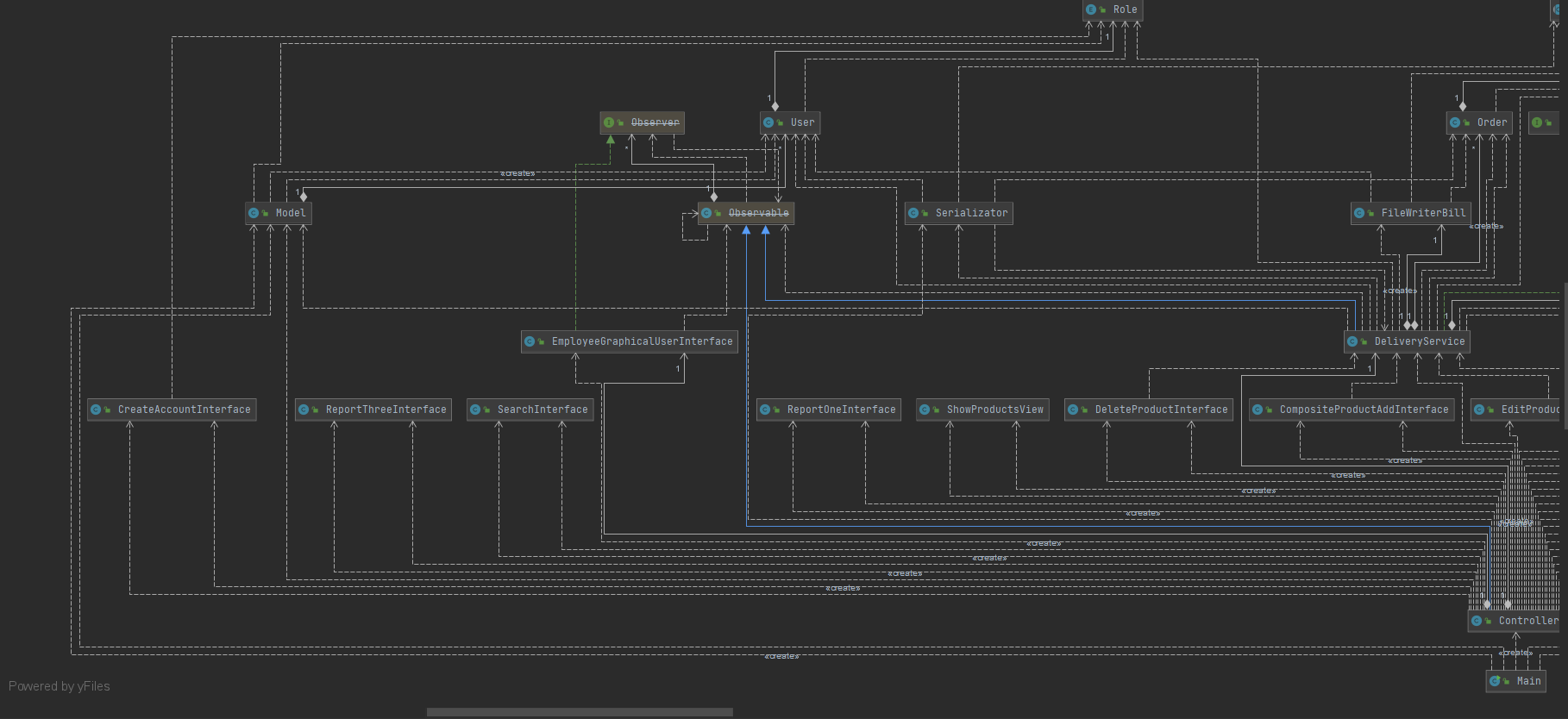
Component - Component declară interfața pentru obiectele din compoziție și pentru accesarea și gestionarea componentelor sale copil. De asemenea, implementează comportamentul implicit pentru interfața comună tuturor claselor, după caz.

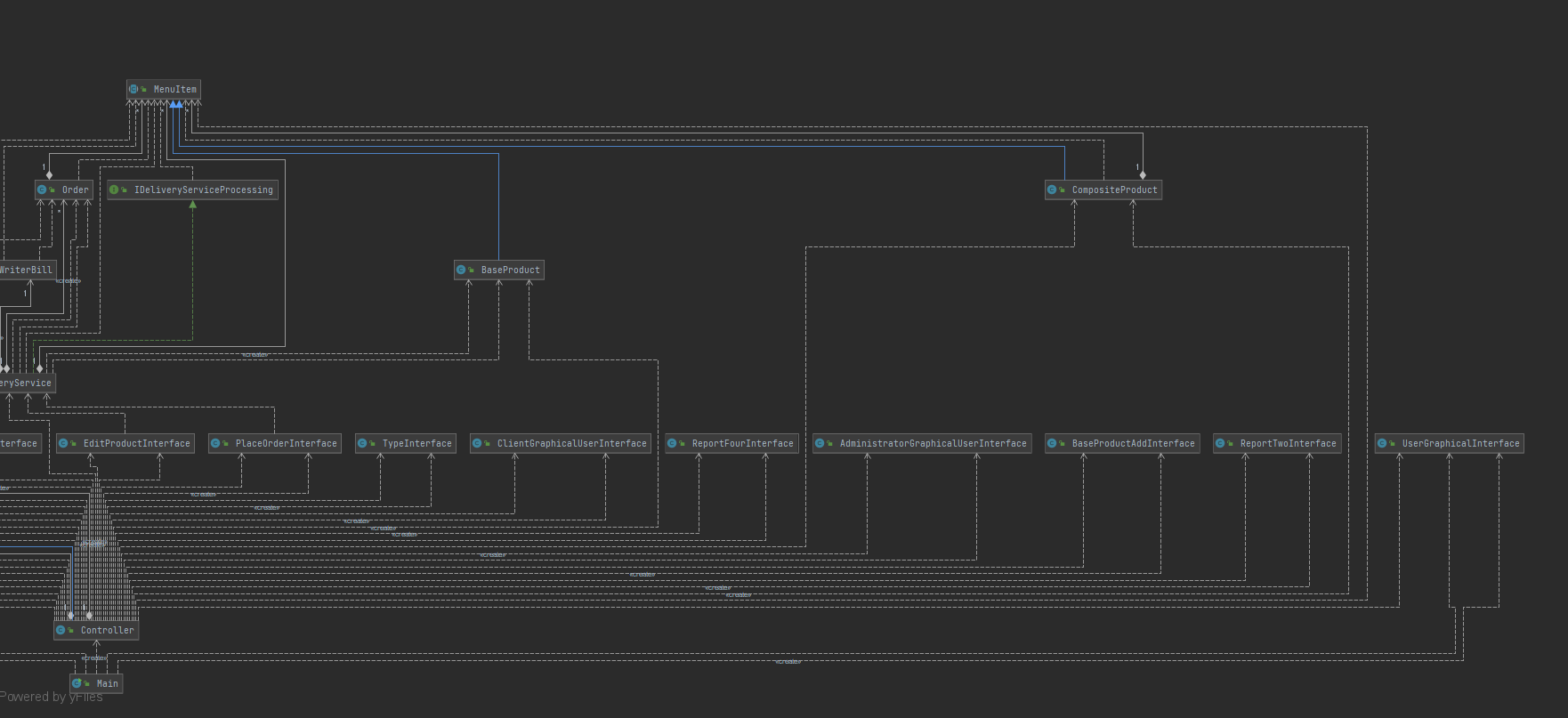
Leaf - Leaf definește comportamentul obiectelor primitive din compoziție. Reprezintă obiecte frunze în compoziție.

Compozit - Compozit stochează componentele copil și implementează operațiuni legate de copil în interfața componentelor.

Client - Clientul manipulează obiectele din compoziție prin interfața componentei.

Clientul folosește interfața clasei de componente pentru a interacționa cu obiecte din structura compoziției. Dacă destinatarul este o frunză, atunci cererea este tratată direct. Dacă destinatarul este un compozit, atunci acesta transmite de obicei cererea către componentele sale copil, efectuând eventual operații suplimentare înainte și după redirecționare.





O alta interfata implementata estea interfata Observer folosita pentru ca clasa Employee sa primeasca update de fiecare data cand clasa Client genereaza o comanda noua.

# ***Implementare***

Una din clasele implementate este clasa User,aceasta clasa are ca variabile instanta username , password ,role si ClientID, aceasta clasa este responsabila cu mimarea conceptului de utilizator al aplicatiei ,este folosita pentru a tine minte clientii care isi fac cont in aplicatie.Aceasta clasa implementeaza in principala gettere si settere.

O alta clasa implementata este clasa Model,aceasta clasa tine un array de useri si este responsabila pentru validarea datelor introduse la logare si la inregistrare de catre utlizatori,aceasta clasa returneaza ID ul unui user gasit cu ajutorul username ului si a parolei si incarca Useru luatii din fisier prin deserializarea in memoria programului.

O alta clasa implementata este clasa Order,aceasta clasa este una dintre principalele clase ale aplicatiei,are ca variabila instanta un array de iteme pe care clientul le a selectat in comanda respectiva,aceasta clasa implementeaza metode de get si de set si de asemenea suprascrie metodele hashFunction si equals pentru ca HashSet sa poata stoca eficient si cu succes obiectele de tip Order.

O alta clasa implementata este MenuItem,MenuItem este o clasa abstracta parinte a claselor BaseProduct si CompositeProduct,aceasta clasa declarata antetul metodelor implementate de cele doua clase copil ca si metode abstracte.Metodele declarate de aceasta clasa sunt: public abstract int computePrice(); public abstract String toString(); public abstract String getTitle(); public abstract float getRating(); public abstract int getCalories(); public abstract int getProtein(); public abstract int getFat(); public abstract int getSodium(); public abstract void setTitle(String a); public abstract void setRating(float a); public abstract void setCalories(int a); public abstract void setProtein(int a); public abstract void setFat(int a); public abstract void setSodium(int a); public abstract void setPrice(int a); public abstract boolean equals(Object o); public abstract int hashCode(); public abstract int getOrderedTimes();  
public abstract void increaseOrderedTimes(); public abstract int getReport4();public abstract void increaseReport4(). Aceasta clasa implementeaza interfata Serializable,interfata care este mostenita de cei 2 copii ai ei ,este nevoie de asta pentru a putea serializa si deserializa structurile ce contin acest tip de obiecte.

O alta clasa implementata este clasa BaseProduct,un copil al clasei MenuItem,fiind copil al aceste clase aceasta clasa extinde clasa Menuitem. Clasa BaseProduct are ca variabile instanta atributele unui produs prezent in restaurant,atributele sunt title,rating,calories,protein,fat,sodium si price.

Clasa implementeaza gettere si settere stabilite de clasa MenuItem si pe langa suprascrie metodele hashfunction si equals pentru ca structura HashSet sa poata realiza corect si eficient operatii cu acest tip de clasa.De asemenea mai implementeaza si metode ajutatoare pentru reportul4 generat de administrator.

O alta clasa implementata este clasa CompositeProduct,aceasta clasa mosteneste si ea clasa MenuItem si implementeaza metodele declarate de aceasta,gettere si settere, de asemenea suprascrie si metodele de hashFunction si equals pentru ca HashSet sa poata implementa corect si eficient acest tip de obeicte.

Clasa CompositeProduct tine de asemenea un array de MenuItem deoarece un composite product este alcatuit din multiple base product ,exemplu : meniul zilei alcatuit din supa de rosii, pui la gratar ,paste ,suc etc.

O alta clasa implementata este DeliveryService,aceasta este cea mai importanta clasa a proiectului,aceasta clasa extinde interfata IDeliveryService care declara antetul metodelor implementate de DelievryService si serveste ca metode de implementare a Design by Contract Pattern prin introducerea tag-uri post,pre conditionale,de asemenea in Clasa Delivery Service sunt implementate si testarile corectitudinii cu ajutorul assert urilor.

Clasa Delivery Service este principala clasa functionala a acestui proiect,aceasta implementeaza metoda de addProduct pentru adaugarea unui produs de catre administrator,aceasta clasa implementeaza metoda importProducts pentru a importa produsele prezente in fisierul .csv ,pentru a realiza aceasta importare de informatii din csv file sunt folosite streamuri si expresii lamba.

Aceasta clasa tine minte un HashSet de comenzi,este nevoie de HashSet pentru eficienta search ului si pentru ca elimina duplicatele,un HashSet de produse din meniu,un obiect de tip fileWriterBill cu ajutorul caruia genereaza nota de plata si rapoartele generate de administrator.

Metodele implementate de aceasta clasa sunt addOrder , importProduct,generateRaport1 prin care genereaza un raport intr un fisier .txt care contine produsele comandate intre anumite ore de catre clienti,generateReport2 prin care genereaza un raport care contine produsele comandate de un numar de ori mai mare decat o valoarea data de administrator,generateRaport3,generateRaport4,search,loadInfo.

O alta clasa implementata este Serializator,aceasta clasa implementeaza doua metode si indeplineste o functionalitate foarte importanta pentru aplicatie.Datorita acestei clase si a metodelor implementate de ea aplicatia poate sa conserver datele generate fara a fi legata la o baza de date.

Serializarea este un mecanism de conversie a stării unui obiect într-un flux de octeți. Deserializarea este procesul invers în care fluxul de octeți este utilizat pentru a recrea obiectul Java real în memorie. Acest mecanism este utilizat pentru a persista obiectul.  
Fluxul de octeți creat este independent de platformă. Deci, obiectul serializat pe o platformă poate fi deserializat pe o altă platformă. Pentru a face un obiect Java serializabil, implementăm interfața java.io.Serializable. Clasa ObjectOutputStream conține metoda writeObject () pentru serializarea unui obiect.

Serializarea este realizata la pornirea aplicatiei pentru a incarca datele stocate precedente in fisiere .txt,astfel putem pastra starea anterioara a aplicatiei fara a pierde date,deci nu avem inconsistenta.

Deserializarea este realizata de catre interfata angajatului prin apasarea unui buton.Aceasta operatie asigura incarcarea informatiilor prezente la acel moment in aplicatie in fisiere.

Se comporta asemanator unei baze de date dar utilizeaza fisiere.

O alta clasa implementata este FileWriterBill,aceasta clasa implementeaza metode de scriere a celor 4 rapoarte produse de administrator in fisiere .txt cu ajutorul streamurilor si a fisierelor.  
Clasa Java FileInputStream obține octeți de intrare dintr-un fișier. Este folosit pentru citirea datelor orientate pe octeți (fluxuri de octeți bruti), cum ar fi date de imagine, audio, video etc.

Puteți citi, de asemenea, date de flux de caractere. Dar, pentru citirea fluxurilor de caractere, se recomandă utilizarea clasei FileReader.Dacă trebuie să scrieți valori primitive într-un fișier, utilizați clasa FileOutputStream. Puteți scrie date orientate atât pe octeți, cât și pe caractere, prin clasa FileOutputStream. Dar, pentru datele orientate spre caractere, este de preferat să utilizați FileWriter decât FileOutputStream.

Avem un numar de 18 clase reprezentate pentru interfete,astfel incat sa satisfaca cerinta de complexitate a aplicatiei,fiind atribuit un numar destul de mare de interfete fiecarui tip de utilizatori.

O alta clasa implementata este clasa Controller,aceasta clasa este si cea mai lunga clasa ca numar de cuvinte din proiect,ea manageriaza comportamentul interfetelor si interactioneaza cu acestea,de ex la apasarea de butoane le ascunde pe unele si le face vizibile pe altele si notifica clasa DeliveryService sa indeplineasca o anumita actiune.

# ***Rezultate***

Aplicatia a fost testa cu ajutor asserturilor prezente in clasa DeliveryService,in rest testarea s-a efectuat manual.

# ***Concluzii***

Aceasta aplicatie poate fi extinsa la un nivel profesional pentru a servii unui lant de supermarketuri sau restaurent,cum sunt aplicatiile lidl plus ,love kaufland , food panda service etc.

# ***Bibliografie***

[Tutorials List - Javatpoint](https://www.javatpoint.com/)

[InfoWorld - Technology insight for the enterprise](https://www.infoworld.com/)

[Overview (Java Platform SE 7 ) (oracle.com)](https://docs.oracle.com/javase/7/docs/api/overview-summary.html)