FOOD DELIVERY MANAGEMENT SYSTEM

*Logo, company name

Description automatically generated*

**Profesor: Tudor Cioara**

**Asistent profesor: Alexandru Rancea**

**Student: Stratulat Ildiko Vivien**

**Grupa : 30221**

**1.Obiectivul temei**

Obiectivul temei este de a crea un sistem ce simulează o firmă de livrări de mâncăruri prin intermediul anumitor noțiuni de programare orientate pe obiecte cum ar fi serializarea, folosirea stream-urilor și a expresiilor lambda.

**2.Analiza problemei, modelare, scenarii, cazuri de utilizare**

* **Analiza problemei:**

Această aplicație ar trebui să simuleze livrarea unor produse (mâncăruri) care are trei componente principale: Administratorul care ar trebui să poată să creeze produse noi, să modifice produsele deja existente și să șteargă din acestea, Angajatul care trebuie să fie notificat în cazul unei noi comenzi depuse de către clienți, iar în cele din urmă, Clientul care poate să vadă ce produse sunt disponibile, să caute printre acestea după anumite criterii, să adauge produse la comandă și să depună comanda.

* **Modelarea problemei:**

**3.Scenarii și cazuri de utilizare:**

Un caz de utilizare reprezintă o colecție de scenarii posibile, referitoare la comunicarea între aplicație și utilizatori, caracterizate cu anumite scopuri. Cazurile de utilizare arată ce trebuie să facă sistemul și nu cum

Deoarece cazurile de utilizare sunt puternic conectate cu pașii utilizatorului am încercat să fac interfața grafică cât mai user-friendly.

Graphical user interface

Description automatically generated with medium confidence

-utilizatorul își va introduce datele de identificare în cazul în care acestea există deja, iar în caz contrar are posibilitatea de a se înregistra

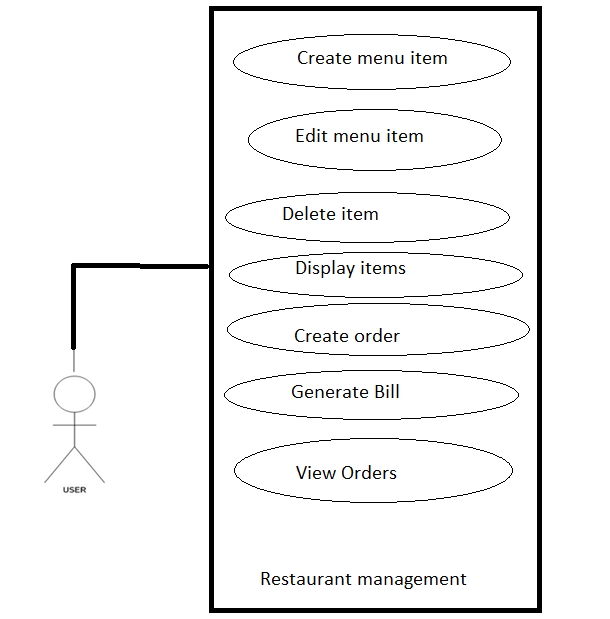
-în cazul în care se vor introduce date necorespunzătoare aplicația nu va face nimic, nu se vor putea efectua operațiile dorite

-scenariu succes: utilizatorul își introduce datele corect => va fi redirecționat la pagina specifică rolului său (admin/client/angajat)

-scenariu de insucces: utilizatorul introduce date invalide iar aplicația nu va avea niciun folos

**3.Proiectare:**

* *Diagrame*
* Diagrame Use Case



* Diagrama claselor

Graphical user interface

Description automatically generated

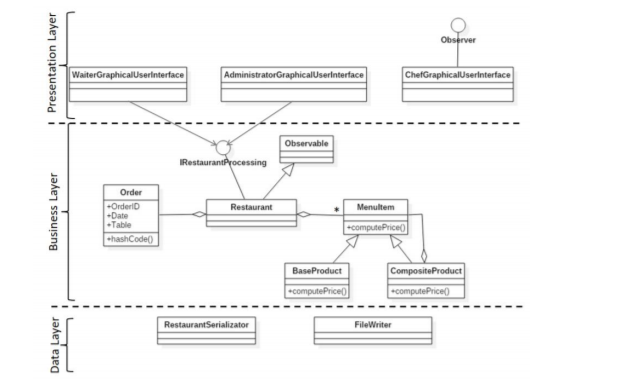
-Diagramele de clase sunt folosite în modelarea orientată pe obiect pentru a a descrie structura statică a sistemului, modului în care este el structurat

-Oferă o notație grafică pentru reprezentarea: claselor, entități ce au caracteristici comune și a relațiilor dintr două sau mai multe clase.

* **Structuri de date**

Structurile de date care au fost folosite în cadrul acestui proiect sunt ori tipuri de date primitive: int, ori un tip mai complex: List,ArrayList, Hashmap și obiectele create de mine, MenuItem, BaseProduct, CompositeProduct, DeliveryService.

Am ales să folosesc HashMap pentru o stocare mai organizată a comenzilor depuse de către clienți, astfel cheia reprezinta Comanda iar valoarea reprezintă produsele din comandă (acestea la rândul lor fiind stocate în niște liste)

**4.Implementare:**

**Descrierea claselor:**

**-USER:**

-clasă care va fi moștenită atât de către client cât și de către angajat

-constructorul: User(int id,String username,String password,String address) -asignează valorile transmise fieldurilor clasei

-conține următoarele fieldu-ri:

-int idUser- id-ului utilizatorului

-String username -numele utilizatorului

-String address -parola utilizatorului

-String role -rolul utilizatorului: client sau angajat

**-metode:**

- public String getUserName() -returnează username -ul user-ului

-public void setUserName(String username)-setează valoarea username-ului

-public String getPassword()-returnează câmpul password al user-ului

-public void setPassword(String password)- setează valoare passwordului userului

-public String getAddress()-returnează adresa userului

-public void setAddress(String address)-setează adresa userului

-public String getRole()-returnează rolul userului(client/angajat)

-public void setRole()-setează valoarea rolului userului (client/angajat)

-public void setIdUser(int id)-setează valoarea id-ului user-ului

-public int getIdUSer()-returnează id-ul userul-ui

**-CLIENT**

-clasă care moștenește atributele și metodele clasei User

-conține informații despre clienți ce doresc să comande

-atribute(pe lângă cele moștenite)- int noOrders- numărul de comenzi date de către client

**-constructor:**

Public Client(int id,String username,String password,String address)

-**metode:**

-public int getNoOrders()- returnează numărul de comenzi date de către client

-public void setNoOrders(int noOrders)- setează număruld de comezni a clientului

**-EMPLOYEE**

clasă care moștenește atributele și metodele clasei User

-conține informații despre angajați

-**constructor:** public Employee(int id,String username,Stirng password,String address)

**ORDER**

-clasă de conține informații despre comenzile date de către clienți

**-atribute**:

-int idClient- id-ul clientului care a pus comanda

-int idOrder- id-ul comenzii

-LocalDateTime date- data la care a fost pusă comanda

-int totalPrice -totalul comenzii

-int noProducts-numărul de produse ce se află în comandăș

**-constuctor**: public Order(int idClient,int idOrder,LocalDateTime date)

**-metode:**

-public int getNoProducts()

-public void setNoProducts(int nr)

-public int setIDClient(int id)

-public int getIdClient()

-public void setDate(LocalDateTime date)

-public LocalDateTime getDate()

-public int getTotalPrice()

-public void setTotalPrice(int price)

-public int calculateTotalPrice(ArrayList <MenuItem > menuItems)

-public boolean equals(Object obj)

-public int hashCode()

**BaseProduct:**

-clasa care moștenește toate atributele și metodele clasei MenuItem

-atribute: float rating, int price, int calories, int proteins, int fats, int sodium

-constructor: public BaseProduct(String title,float rating,int calories, int proteins,int fats, int sodium)

**-metode:**

-public float getRating()

-public void setRating(float rating)

-public int getPrice()

-public void setPrice(int price)

-public void setCalories(int calories)

-public int getCalories()

-public int getProteins()

-public void setProteins(int proteins)

-public int getSodium()

-public void setSodium(int sodium)

**CompositeProduct:**

-clasă ce moștenește toate atributele și metodele clasei MenuItem

-conține o listă ce este alcătuită din obiecte ale clasei BaseProduct

**-atribute**: float rating, int price, int calories, int proteins, int fats, int sodium,ArrayList<BaseProduct> products

**-metode:**

-public float getRating()

-public void setRating(float rating)

-public int getPrice()

-public void setPrice(int price)

-public void setCalories(int calories)

-public int getCalories()

-public int getProteins()

-public void setProteins(int proteins)

-public int getSodium()

-public void setSodium(int sodium)

**LoginGui**

-reprezintă interfața grafică pentru logarea utilizatorilor: Admin/Angajat/Client

-este posibilă și înregistrarea clienților prin intermediul acesteia

-atribute: elemente clasice ale unei clase ce implementează o interfață grafică: butoane,label-uri,textfield-uri, tabel pentru prezentarea produselor, frame

**AdminGUI**

-reprezintă interfața grafica a administratorului, prin care acesta poate efectua anumite operații cu produsele din meniu

-atribute: elemente clasice ale unei clase ce implementează o interfață grafică: butoane,label-uri,textfield-uri, tabel pentru prezentarea produselor, frame

-la apăsarea butonului de importare produsele din fșierul .csv se vor importa în structurile de date folosite in program

-apăsarea butonului show se vor prezenta toate produsele din meniu în interfața grafică

-la apăsarea butonului delete se va șterge rândul selectat din tabelul cu produse

-la apăsarea butonului addTo se va adăuga produsul de bază la un nou produs compus

-la apasarea addComposit se va adăuga în lista de produse noul produs compus creat

-la apăsarea butonului addBase se va adăuga noul produs de baza la liste cu produse

-la apăsarea butonului btnEdit se va permite modificarea unor date a produselor

-la apăsarea butoanelor btnDayReport, btnTimeReport,btnClientReport,btnProduct report se vor genera niște rapoarte cu anumite informații despre comenzile puse de către clienți

**ClientGUI**

-reprezintă interfața grafică a clientului, care îi permite să efectueze un număr de lucruri: să caute produse după anumite criterii, să adauge produse la comandă și să pună comanda

-atribute: elemente clasice ale unei clase ce implementează o interfață grafică: butoane,label-uri,textfield-uri, tabel pentru prezentarea produselor, frame: Jtable table Jtextfield tfname,tfrating,tfcalories,tfprotein,tffats și Jbutton btnCriteria, btnSearchPrice, btnSearchProtein, btnSearcFats,btnSearchCalories,btnAllProducts, btnAddToOrder,btnOrder, btnSearchName, btnSearchRating.

-metoda public void notificare() va fi apelată ori de câte ori va fi pusă o comandă de către un client, aceasta va notifica angajatul că o noua comandă necesită pregătire.

-la apăsarea butonului btnShow – se va afișa în interfața grafică, prin intermediul unui table, produsele ce se află în meniu la acel moment

-la apăsarea butonului btnAddToOrder se vor adăuga produsele selectate de către client în tabel la comanda sa

-la apăsarea butonului btnOrder de către client, se va pune o comandă cu produsele selectate anterior și va fi apelată metodata de notficiare() pentru angajat

-la apăsarea butonului btnSearchName se va căuta prin elementele tabelului produsul ce conține numele oferit de client

-la apăsarea produsului btnSearchPrice se va căuta prin elementele tabelului produsele ce au pretul căutat de către client.

**DeliveryService**

-implementează intefața IDDeliveryServiceProcessing

-aici se află tot de ce avem nevoie pentru simularea cu succes a aplicației: metodele, datele despre clienți și despre produse.

**Stream-uri și expresii lambda**

-expresiile lambda sunt metode anonime (fără nume) utilizate pentru a implementa o metodă definită de o interfață funcțională (o interfață ce conține o singură metodă abstractă)

-Un flux de date (stream) reprezintă un canal de comunicații prin care datele circulă de la o sursă către destinație. În limbajul java, fluxurile și operațiile corespunzătoare sunt implementate prin intermediul claselor din pachetul java.io.

-Fluxurile de procesare (sau de filtrare) sunt responsabile cu preluarea datelor de la un flux primitiv și procesarea acestora pentru a le oferi într-o altă formă, mai utilă dintr-un anumit punct de vedere. De exemplu, BufferedReader poate prelua date de la un flux FileReader și sp ofere informația dintr-un fișier linie cu linie. Fiind primitiv, FileReader nu putea citi decât caracter cu caracter.

**Composit pattern:**

Modelul composit este un model de proiectare de partiționare și descrie un grup de obiecte care este tratat la fel ca o singură instanță a aceuiași tip de obiect. Intenția unui compozit este de a compune obiecte în structuri de copac pentru a reprezenta ierarhii parțiale. Permite structura de copac al cărui fiecare nod poate efectua câte o sarcină.

**Observer:**

Observatorul este un model comportamental de proiectare. Specifică comunicarea între obiecte: observabil și observator.Un observabil este un obiect care notifică observatorii despre schimbările în starea sa.

Modelul observator este folosit atunci când există o relație între mai multe obiecte, cum ar fi dacă un obiect este modificat, obiectele sale dependente trebuie notificate automat. Modelul observator se încadrează în categoria modelului comportamental.

**Invariant:**

În programarea computerizată, în special în programarea orientată pe obiecte, un invariant de clasă (sau invariant de tip) este un invariant utilizat pentru constrângerea obiectelor unei clase. Metodele clasei ar trebui să păstreze invariantul.

Un invariant de obiect, sau invariant de reprezentare, este o costrucție de programare computerizată constând dintr-un set de proprietăți invariante care rămân necompromise indiferent de starea obiectului. Acest lucru asigură că obiectul va îndeplinii întotdeauna condiții predefinite și că, prin urmare, metodele pot face referire întotdeauna la obiect fără riscul de a face prezumții inexacte. Definire invarianților de clasă poate ajuta programatorii și testerii să prindă mai multe erori în timpul testării aplicației. Efectul util al invarianților de clasă a software-ului orientat obiect este îmbunătățit în prezența moștenirii. Invarianții de clasp sunt moșteniți, adică invarianții tuturor părinților unei clase se aplică clasei sine.

Moștenirea poate permite claselor descendente să modifice datele de implementare ale claselor părinte, deci ar fi posibil ca o clasă descendentă să schimbe starea instanțelor într-un mod în care le-a făcut invalide din punct de vedere al clasei părinte. Îngrijorarea pentru acest tip de descendenți care se comportă greșit este unul din motivele pentru care proiectanții de software orientat pe obiect oferă favorizarea compoziției în locul moștenirii.

Cu toate acestea, deoarece invarianțiide clasă sunt moșteniți, invariantul de clasă pentru o anumită clasă constă din orice afirmații invariante codate: imediat pe acea clasă împreună cu toate clauzele invariante moștenite de părinții clasei. Aceasta înseamnă că, chiar dacă clasele descendente pot avea acces la datele de implementare ale părinților lor, invariantul clasei îi poate împiedica să manipuleze acele date în orice mod care produce o instanță nevalidă în timpul rulării.

**Interfața grafică**

-are rolul de a conecta utilizatorul cu aplicația

-acest utilizator poate să introducă date, în cazul de față username, parolă

Graphical user interface

Description automatically generated

**Interfața grafică a adminului**

Graphical user interface, table

Description automatically generated

**Graphical user interface

Description automatically generatedInterfața grafică a clientului**

**5.Rezultate:**

-am atașat o captură de ecran cu bonurile care sunt generate de fiecare dată când o comandă este dată de către clienși

Text

Description automatically generated

**6.Concluzii:**

Acest proiect a reprezentat un exercițiu absolut necesar pentru aprofundarea noțiunilor studiate în primul semestru despre Programarea orientată pe obiecte, pe parcursul realizării acestuia am întâmpinat anumite dificultăți, dar datorită acestora am fost nevoită să mă documentez mai mult, astfel înțelegând mult mai bine aceste noțiuni.

Una dintre noțiunile noi pe care le-am întâlnit este serializarea, o noțiunea care mi-a cam dat bătăi de cap, am avut o eroare de a trebuit sa stau trei zile să reușesc să o rezolv.Totodată, în timp ce m-am documentat pentru această temă am aflat alte chestii noi cum ar fi lambda expressions, care la prima impresie, când am auzit aceasta expresie m-au cam speriat, dar apoi mi-am dat seama că nu-i chiar așa. În plus, am aflat și despre ce probleme pot apărea la folosirea metodei .distinct() de la streamuri în cazul listelor neordonate, probleme pe care am descoperit că se pot remedia prin overriding-ul a două alte metode: hashcode() și și equals().

**7.Bibliografie:**

1. [www.wikipedia.org](http://www.wikipedia.org)
2. [www.geeksforgeeks.com](http://www.geeksforgeeks.com)
3. [www.w3school.com](http://www.w3school.com)
4. [www.stackoverflow.com](http://www.stackoverflow.com)
5. Cursurile de la Programare Orientată pe Obiecte Raluca Brehar
6. [www.youtube.com](http://www.youtube.com)
7. www.tutorialspoint.com