DOCUMENTATIE

TEMA 4

NUME STUDENT: Dragotoniu Corina Mădălina

GRUPA: 30224

# CUPRINS

[1. Obiectivul temei 3](file:///C:\Users\40770\Downloads\PT2021-2022_Documentation_Template%20(1).doc#_Toc95297885)

[2. Analiza problemei, modelare, scenarii, cazuri de utilizare 4](file:///C:\Users\40770\Downloads\PT2021-2022_Documentation_Template%20(1).doc#_Toc95297886)

[3. Proiectare 5](file:///C:\Users\40770\Downloads\PT2021-2022_Documentation_Template%20(1).doc#_Toc95297887)

[4. Implementare 6](file:///C:\Users\40770\Downloads\PT2021-2022_Documentation_Template%20(1).doc#_Toc95297888)

[5. Rezultate 7](file:///C:\Users\40770\Downloads\PT2021-2022_Documentation_Template%20(1).doc#_Toc95297889)

[6. Concluzii 8](file:///C:\Users\40770\Downloads\PT2021-2022_Documentation_Template%20(1).doc#_Toc95297890)

[7. Bibliografie 8](file:///C:\Users\40770\Downloads\PT2021-2022_Documentation_Template%20(1).doc#_Toc95297891)

# Obiectivul temei

Obiectivul principal al temei consta in proiectarea si implementarea unui sistem de manageriere pentru livrarea de produse alimentare in contextul unei companii de catering. Clientul poate depune comenzi pe baza produselor disponibile in oferta companiei. Sistemul ar trebui sa aiba 3 tipuri de utilizatori, care sa se logheze pe baza unui nume de utilizator si a unei parole, anume administrator, angajat obișnuit si client.

|  |  |  |
| --- | --- | --- |
| Obiectiv Secundar | Descriere | Capitol |
| Dezvoltarea de use case-uri si scenarii | Prezinta o reprezentare grafica a modului in care utilizatorul interactioneaza cu interfata grafica a sistemului de livrare a comenzilor, urmarind anumite spete si scenarii de functionare si utilizare. | 2 |
| Alegerea structurilor de date | Ofera explicatii cu privire la optiunea pentru anumite structuri de date in implementarea solutiei. | 3 |
| Impartirea pe clase | Urmareste din punct de vedere schematic modul de interactiune si asamblare a claselor in vederea functionarii corecte a aplicatei de realizare a livrarilor. | 3 |
| Dezvoltarea algoritmilor | Aduce adnotari imperios necesare asupra modului in care sunt traduse in practica, sub forma algoritmica, anumite metode auxiliare sau fundamentale in implementarea aplicatiei. | 3 |
| Implementarea solutiei | Realizeaza o introspectie asupra functionalitatii si a necesitatii fiecarei clase constituente. | 4 |

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

Un stream reprezintă o secvență de elemente și suportă diferite tipuri de operații care să evalueze elementele respective. Operațiile cu stream-uri sunt intermediare sau terminale. Operațiile intermediare returnează un stream, deci putem înlănțui mai multe operații intermediare fără a folosi punct și virgulă. Operațiile terminale sunt ori de tipul void, ori returnează un rezultat care nu este tot un stream. Tipuri de operații intermediare sunt filter, map și sorted, iar un exemplu de operație terminală este forEach. Cele mai multe operații cu stream-uri acceptă un fel de lambda expresii ca parametrii.

Expresiile Lambda profită de capacitățile de proces paralele ale mediilor multi-core, așa cum se vede cu sprijinul operațiunilor de conducte pe date în API-ul Stream. Sunt metode anonime (metode fără nume) utilizate pentru a implementa o metodă definită de o interfață funcțională. O interfață funcțională este o interfață care conține o singură metodă abstractă.

Expresiile Lambda introduc noul operator săgeată -> în Java. Împarte expresiile lambda în două părți. Partea din stânga specifică parametrii solicitați de expresie, care ar putea fi, de asemenea, goi dacă nu sunt necesari parametri. Partea dreaptă este corpul lambda care specifică acțiunile expresiei lambda.

Analizând enunțul temei propuse, observăm că trebuie să începem prin a citi datele din fișierul de tip csv care constituie meniul cu care vom lucra, folosind stream-uri și expresii lambda. Apoi, vom implementa funcțiile de serializare și deserializare pentru a putea salva meniul într-un fișier separat, pe care să îl putem citi și în care putem adăuga noi intrări. Apoi implementăm funcționalitățile pentru tipurile de utilizatori client și administrator. Administratorul va putea să importe datele din fișierul inițial, să adauge, să șteargă și să modifice produse din meniu, dar și să genereze report-uri în funcție de anumite criterii. Clientul se poate înregistra în aplicație, creându-și un cont, se poate loga, poate vedea meniul, poate căuta anumite produse din meniu în funcție de anumite criterii și să plaseze comenzi, în urma cărora va primi o chitanță.

Trecând la partea de modelare, vom respecta diagrama de clase și pachete prestabilită în cerința proiectului. Așadar, vom avea pachetele: viewLayer si controllerLayer (care se ocupă de interfața grafică), businessLayer (care va conține clasele Order, MenuItem, BaseProduct, CompositeProduct,Client,User, interfața IDeliveryServiceProcessing și clasa DeliveryService) și dataLayer (care va conține clasele Serializator și FileWriter).

În continuare, voi prezenta cazul de utilizare ideal:

1. Utilizatorul își poate crea un cont dacă a ales să fie client sau se poate loga dacă are deja un cont de client sau este admin/angajat.
2. Dacă utilizatorul este client, atunci poate folosi toate funcționalitățile de care dispune clientul: poate vedea meniul, poate căuta produse, poate plasa o comandă și va primi o chitanță.
3. Dacă utilizatorul este administrator, poate folosi toate funcționalitățile de care dispune administratorul: poate importa meniul din fișierul csv, poate sa adauge, să șteargă și să modifice produse din meniu și să genereze un report.
4. Daca utilizatorul este angajat, acesta va putea sa vadă toate comenzile plasate de clienți

# 3. Proiectare

Structura de date primordiala in implementarea aplicatiei o constituie HashMap, utilizat in vederea modului de stocare a perechilor de tipul Comanda – lista de produse aferenta comenzii. Avantajul principal al utilizarii acestei structurii care implementeaza o tabela de dispersie rezida in complexitatea medie de adaugare sau extragere a unor elemente din tabela, anume O(1).

O alta structura de date importanta in vederea implementarii solutiei finale o constituie TreeSet-ul. Optiunea pentru TreeSet, in detrimental unui ArrayList pentru produsele importate din fisierul csv, isi are sorgintea in redundanta stocarii datelor pe care ar fi implicat-o o structura de tip ArrayList, dat fiind faptul ca produsele din cadrul fisierului csv pot avea aparitii multiple. De asemenea, optiunea pentru TreeSet in detrimental unui ArrayList, este justificata prin caracteristicile intrinseci ale primei structuri mentionate. Astfel, desi deficitar in implementarea anumitor metode, unde ArrayList-ul ar putea fi mai benefic, ordonarea implicita a elementelor in momentul adaugarii intr-un TreeSet, l-a facut pe acesta din urma mai dezirabil.

Diagrama UML din randurile ce urmeaza descrie modul in care clasele utilizate in implementarea aplicateie de delivery interactioneaza intre ele.

O imagine care conține text

Descriere generată automat

In cadrul proiectului s-a urmarit implemenatarea a diferite tehnici de programare care exploateaza facilitati oferite de mediul de prgaramre java pe baza unor pattern-uri cunoscute. Pentru implementarea Observer Design Pattern s-a urmarit constituirea unei clase de tip Observer, care are drept scop auxiliar notificarea celorlalte clase de interes care implementeaza interfata Observable despre modificarile care survin la nivelul unor date relevante, in cazul de fata notificarea utilizatorului cu rol de employee in legatura cu depunerea comenzilor. In vederea atingerii conceptului de Composite Design Pattern au fost definite clasele CompositeProduct si BaseProduct care extind functionalitatea clasei MenuItem. In vederea implementarii patternului de Design By Contract au fost definite interfata IDeliveryServiceProccesing si clasa care implementeaza metodele definite in cadrul interfetei, anume DeliveryService. Pentru stocarea datelor de la o rulare la alta a aplicatiei au fost implementate conceptele de serializare si deserializare a datelor, pe baza metodelor din cadrul clasei Serializator. De asemenea, in vederea unei bune gestionari si manipularii a unui set mare de date, clasa DeliveryService uzeaza de stream-uri si de expresii lambda inerente acestora. De asemenea, in clasa LogInController a fost realizata metoda verifyLogInData care determina pe baza username-ului oferit de catre user in fereastra de logare a tpului de utilizator care urmareste lansarea aplicatiei, astfel incat, in masura in care parola concorda cu numele de utilizator, urmatoarea fereastra deschisa sa permita doar functionalitatiile particulare fiecarui tip de utilizator in parte, pastrand astfel integritatea sistemului.

# 4. Implementare

* Clasa **MenuItem** este o clasa abstracta, având o metoda abstracta de computare a prețului, pe care clasele care o moștenesc urmează sa o suprascrie. Aceasta definește la nivel de atribute principalele câmpuri care definesc un produs din cadrul meniului pus la dispoziție de firma de livrare.
* Clasa **BaseProduct** extinde clasa MenuItem suprascriind metoda acesteia abstracta de determinare a pretului in contextual unui produs de acest tip.
* Clasa **CompositeProduct** extinde la rândul ei clasa MenuItem suprascriind metoda acesteia abstracta de determinare a prețului in contextual unui produs de acest tip. De asemenea, pornind de la definiția unui produs compus, ca fiind un produs determinat de mai multe produse obișnuite, clasa CompositeProduct defineste sub forma de atribut o lista de MenuItem-uri si determina in funcție de atributele fiecărui produs din lista, atributele principale ale produsului compus, luând in considerare prețul, numărul de calorii, de grăsimi, rating-ului fiecărui produs constituent etc.
* Clasa **Role** este de fapt un Enum care definește tipurile de utilizatori posibili in cadrul aplicatiei.
* Clasa **User** modeleaza conceptul unui utilizator la nivel de baza, definind atributele sale primordiale, anume parola si numele de utilizatori.
* Clasa **Client** extinde conceptul unui utilizator primar prin adăugarea unui nume si a unui identificator unic de ID setat prin intermediul unei variabile statice de tip contor.
* Clasa **Order** introduce la nivel de cod conceptul unei comenzi prin definirea atributelor sale principale. Elementul unui de identificare, anume orderID-ul este de asemenea setat cu ajutorul unei variabile statice de tip contor. Fiind un element in cadrul tabelei de dispersie, la nivelul acesteia este suprascrisa si funcția de hashCode, calculate in baza ID-ului comenzii, a ID-ului clientului care a depus comanda si a datei in care a fost depusa comanda.
* Clasa **DeliveryService** implementează interfața IDeliveryServiceProccesing la nivelul careia sunt definite principalele operații realizate de administrator si de utilizatori. Astfel metoda importProducts() determina importarea produselor de catre administrator din fișierul csv. Metoda addProductToMenu determina adăugarea unui obiect de tip MenuItem in lista de produse a companiei. Metoda DeleteProductFromMenu determina ștergerea unui produs din lista de produse, iar metoda editProductFromMenu editarea datelor unui produs. Metoda generateTimeReport generează un raport in format txt, filtrând datele prin intermediul stream-urilor si pe baza expresiilor lambda, a comenzilor plasate in intervalul de tip dat de startHour si endHour. Metoda generateFrequencyReport generează un raport in format txt, filtrând datele prin intermediul stream-urilor si pe baza expresiilor lambda a produselor comandate de un număr mai mare de ori decat cel indicat prin numberTimesOrdered. Metoda generateFrequencyAndValueReport generează un raport in format txt, filtrând datele prin intermediul stream-urilor si pe baza expresiilor lambda a clienților care au comandat de un număr mai mare de ori decât cel indicat prin numberTimesOrdered.si a căror comanda a presupus plata unei sume de bani mai mare decât cea indicate prin parametrul amount. Metoda generateDateReport generează un raport in format txt, filtrând datele prin intermediul stream-urilor si pe baza expresiilor labda a comenzilor realizate in data determinate de parametrul localDate. Metodele private filterClientsByFrequency si FilterProductsByFrequency sunt metode auxiliare, care realizează filtrarea după frecventa, utilizate in generarea raporturilor 2 si 3. Metoda createOrder determina crearea unui obiect de tip Order pe baza parametrilor primiți la apel, in timp ce metoda SearchForProduct determina căutarea pe baza anumitor criterii date prin intermediul parametrilor săi a produselor care le îndeplinesc.
* Clasa **FileWriter** este utilizata in vederea scrierii in fișier, utila in generarea rapoartelor de catre administrator, dar si in generarea facturii in urma plasării unei comenzi. Aceasta definește metoda de scriere in fișier – metoda write, respective metoda care închide fișierul – metoda close. Atributul Boolean isClosed oferă informații despre starea fișierului in vederea serializarii si a deserializarii.
* Clasa **Serializator** este utilizata in vederea definirii metodelor prin care sunt serializate si deserializate datele in vederea păstrării integrității acestora din momentul închiderii sesiunii curente si pana in momentul pornirii unei noi sesiuni.
* Clasa **Main** debutează prin deserializarea datelor aferente sesiunii precedente si implicit prin încărcarea lor. Aceasta definește o instanța de tip DeliverySrvice si o fereastra de tip LogInView cu controller-ul aferent care fac posibila funcționarea aplicației. Aprioric încheierii sesiunii datele noi, procesate pana in acest moment sunt serialitate.

# 5. Rezultate

Pentru a verifica corectitudinea acestei aplicații de gestiune a unui restaurant, am încercat sa testez cat mai mult fiecare funcționalitate a butoanelor din interfața. In toate cazurile testate, aplicația a oferit rezolvarea corecta a taskului, simulând cat mai veridic realitatea.

# 6. Concluzii

In final, sistemul de administrare al comenzilor pentru un restaurant își demonstrează utilitatea printr-o paralela cu alte aplicații de food delivery foarte populare in ziua de azi. Chiar daca funcționalitatea sa este limitata, modul in care acesta simulează anumite funcționalități prezente si in aplicații il face o unealta foarte viabila, cu potențial ulterior de progres.

Ca dezvoltări ulterioare ce pot fi aduse aplicației ar fi posibilitatea de delogare in fereastra principala, dezvoltarea conceptului de Observer-Observable prin notificarea clientului care a depus o anumita comanda despre statusul ei si de timpi aproximativi de livrare. Desigur, aceasta din urma presupune dezvoltarea clasei Order prin adăugarea de noi atribute si metode. Nu in ultimul rând, ar putea fi extinse capacitățile administratorului prin conferirea acestuia de noi acțiuni pe care sa le realizeze.

# 7. Bibliografie

<http://javahungry.blogspot.com/2013/08/hashing-how-hash-map-works-in-javaor.html>

<https://howtodoinjava.com/java/collections/hashmap/how-hashmap-works-in-java/>

<http://javahungry.blogspot.com/2014/03/hashmap-vs-hashtable-difference-withexample-java-interview-questions.html>

<http://javahungry.blogspot.com/2013/08/how-sets-are-implemented-internallyin.html>

<https://www.baeldung.com/java-serialization>

<https://www.geeksforgeeks.org/serialization-in-java/>

<https://docs.oracle.com/javase/8/docs/api/java/io/Serializable.html>