Tema 4

Food Delivery Management System

UTCN, Orsan Tudor, Grupa 30225

1. Obiectivul temei:

Obiectivul principal al acestei teme este sa creem o interfata gui cu care sa putem gestiona un sustem de management despre food delivery. Avem operatii pentru Administrator, Client si Employee, si in functie de contul pe care il avem, putem actiona in diferite feluri asupra acestui gui.

Obiective principale : Sa folosim la baza structurii claselor noastre diagrama de clase data ca exemplu, sa definim Base Product (ca extindere a clasei MenuItem) cu 7 fielduri specifice, printre care nume, in functie de care recunoastem un anumit produs, si alte 6 care au legatura cu informatii detaliate despre produsul in sine. Sa putem lua dintr-un csv file toate BaseProducturile, si anume sa luam string-ul cu fieldurile si sa il transformam intr-un obiect. Cu distinct (cu cu equals si hashcode) sa luam doar produsele ce nu se repeta, deci daca se repeta sa le luam doar o singura data. In GUI sa avem un log in screen, in functie de contul pe care il avem vor fi vizualizate si restul ferestrelor, iar pentru Administrator si pentru Client sa avem ferestre cu operatiile specifice celor 2 feluri ce conturi. Sa folosim lambda expressions si stream processing, adica programare functionala (nu declarativa), pentru a genera rapoartele administratorului (una din operatiile administratorului), si pentru a cauta produse cu clientul.

Alte obiective ce pot fi considerate principale, insa le voi considera secundare: Sa folosim Composite DP pentru MenuItem, BaseProduct si CompositeProduct (Am extins MenuItem in celelalte 2, si am refolosit constructorul), (iar in cazul CompositeProduct, sa avem mai multe MenuIteme posibile in lista din care este formata CompositeProduct), sa creem un bill in format txt atunci cand creed un Order cu un Client (cu informatiile specifice aceli order), sa implementam pre/post conditii in interfata pentru DeliveryService, ca apoi sa le implementam folosind asserturi; La fel sa facem cu invariantul in clasa DeliveryService, iar cand generam fisierele JavaDoc, sa avem tag-urile custom facute, cu toate informatiile langa. Nu in ultimul rand, sa serializam toata clasa DeliveryService (ea contine toate instantele importante, multe liste cu conturi, produse, orderuri), si doar cand repornim programul sa deserializam datele tocmai serializate.

1. Analiza problemei / cazurile de utilizare:

Pentru ca avem multe windowuri la aceasta tema, apar mai multe use caseuri, care sunt totusi asemanatoare intre ele:

Cazurile de functionare ale programului sunt acestea:

-In primul rand, avem windowul de logare:

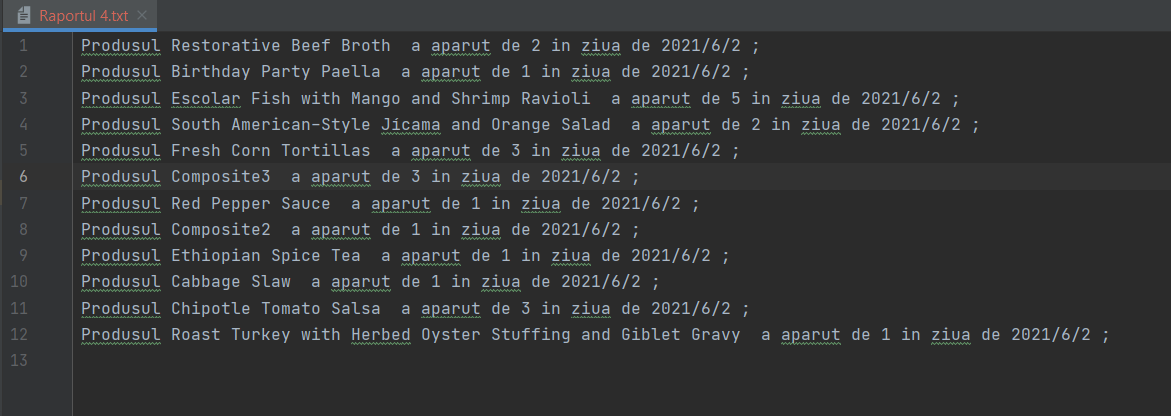


-In acest window, ne putem loga doar daca avem creat un cont. Daca nu avem, mergem pe pagina de creeare cont. Dupa ce am creat un cont valid, ne putem loga drept Client, Administrator sau Employee. Atunci cand se creeaza contul se cunoaste tipul contului.

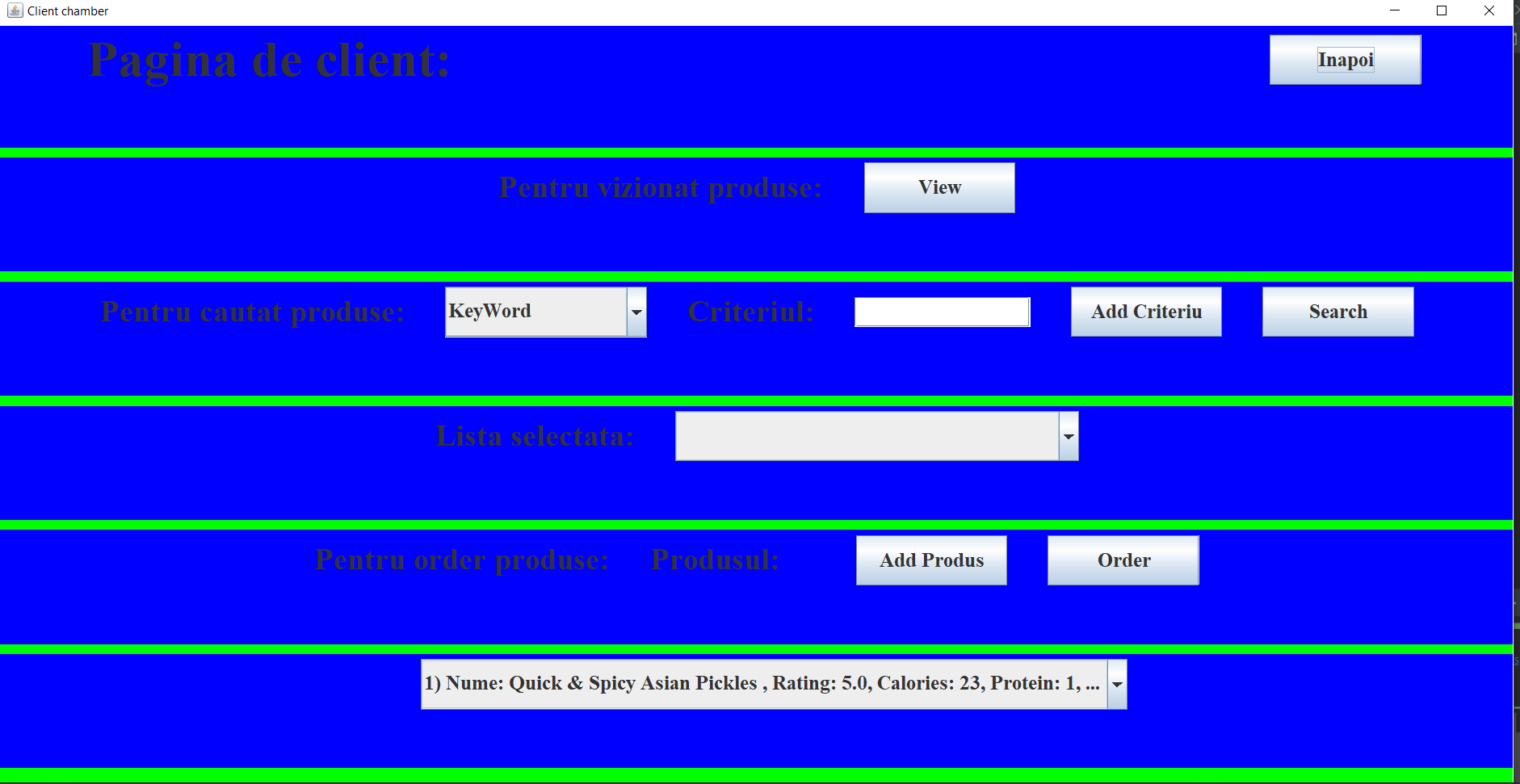
-Dupa logare, primul use case ar fi windowul adiministratorului:



-Acest window permite executarea tuturor operatiilor accesibile de catre administrator. Pentru operatia de import, trebuie doar apasat butonul, dupa care se vor importa datele din fisierul csv. Pentru add, delete, modify, se introduc date de la tastatura legate de ce vrem sa adaugam sau stergem. Daca adaugam, doar introducem date noi, asigurandu-ne ca nu exista deja un produs cu acel nume. Pentru delete, modify, cautam un item existent si ori il stergem din lista, ori ii schimbam proprietatile. Pentru crearea unui produs compus este mai complicat. Prima data, selectam din combo box un Base Product, dupa care il adaugam la lista viitorului Composite Product. Dupa ce adaugam mai multe produse asemenea, vom creea un Composite Product, ce aduna la caracteristicile sale toate caracteristicile Base Producturilor din lista sa. Acum va fi luat acel Composite Product ca un Menu Item ca toate celelalte, si afisat in comboboxuri alaturi de celelalte iteme. Pentru rapoarte nu este asa de complicat. Introduci datele cerute, si dupa procesare, vor fi generate 4 fisiere text, rapoartele, cu datele cerute (ca niste query-uri). Acestea nu se afiseaza pe ecran, ci dupa cum ziceam, in fisiere text.



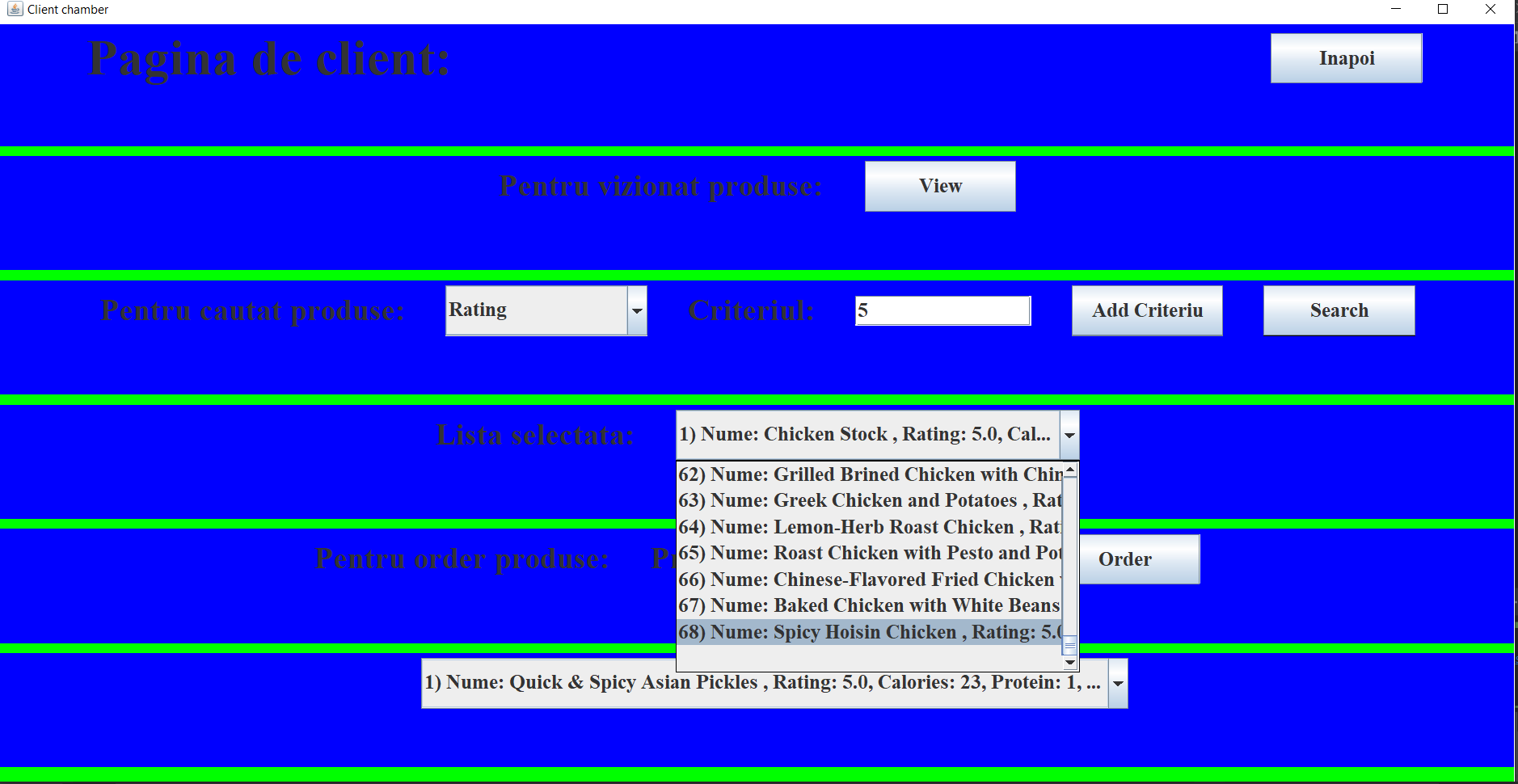
-Pentru client, interfata arata asa:



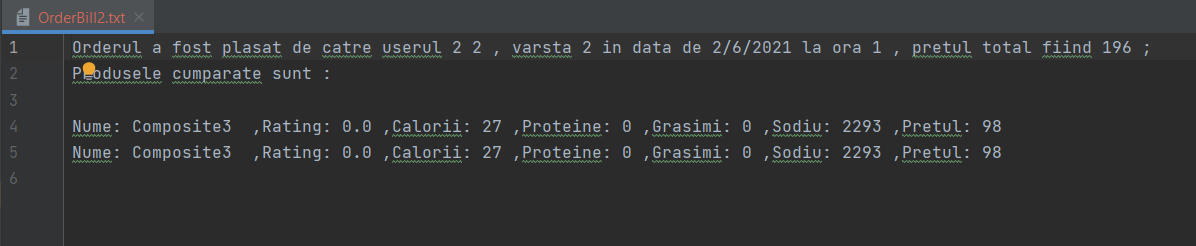
-Nu are la fel de multe operatii ca si administratorul, dar poate executa si el un numar bun de operatii: Cu view se pot vedea toate produsele existente actual:



-Nu se poate interactiona cu ele aici, doar administratorul le poate schimba, dar se pot vedea aici produsele pentru a se putea alege in viitor orderuri potrivite. Pentru cautare, avem un sistem asemanator cu cel legat de composite product. Dupa ce alegi criterii de cautare, poti alege mai multe de la acelasi tip, precum Chicken si Salsa, cu rating 5 si 2, etc... butonul de search va oferi vizual o grupare a tuturor produselor ce contin aceste caracteristici selectate. Nu se poate interactiona cu ele, sunt din nou, doar vizuale, un challenge pentru viitor ar fi sa pot sa aleg pentru order de aici.

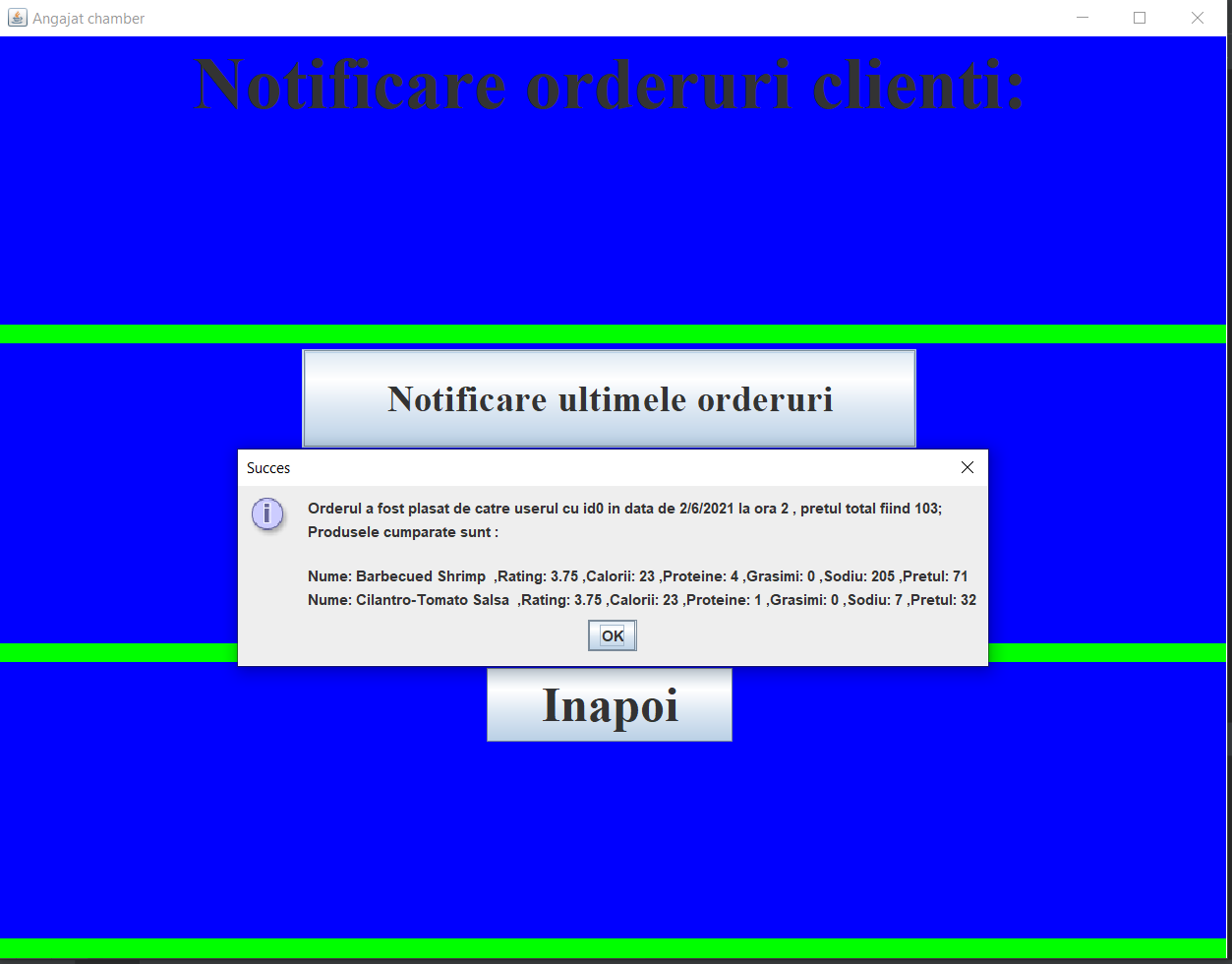


-O ultima operatie importanta a clientului este cea de order. Poti selecta din combo box produse, si sa le adaug la lista de produse din order. Dupa ce apesi pe butonul de order, ai lista de produse selectata la acel order. Poti avea produse duplicate. Orderurile sunt salvate ca billuri, unde sunt trecute toate informatiile necesare, cine a comandat, ce pret a avut comanda, ce a fost comandat etc...:



-Cam acestea au fost operatiile clientului. Este de observat ca orderurile facute de client au repercursiuni pentru ceilalti utilizatori. Administratorul face rapoartele in mare pe baza orderurilor, iar Employee este notificat pe baza acestor orderuri. Deci cand se creeaza un nou order, exista o positibilitate de schimbare a raportului pentru administrator, si o notificare imediata pentru Employee.

-Un ultim use case simplu este cel al Employee:



-Aici tot ce se intampla este ca atunci cand se apasa pe butonul „Notificare ultimele orderuri”, apar toate orderurile, in ordinea in care au fost facute, pe ecranul Employee. De notat este ca nu functioneaza cu serializarea, deoarece nu serializez si datele din EmployeeGui (desi pe viitor pot schimba acest aspect). Si pentru ca nu se serializeaza, la repornirea programului dispar aceste notificari. Deci ca sa poata fi vazute, trebuie apasat butonul inainte de inchiderea apliatiei (Ar trebui schimbata aici functionalitatea). Ce se afiseaza pe ecran este ce se afiseaza si in bill, dar motivul pentru care se intampla asta este ca Employee sa vada ce are de pregatit pentru Client. Dupa ce stie ce produse are de pregatit, le poate prepara si trimite clientului.

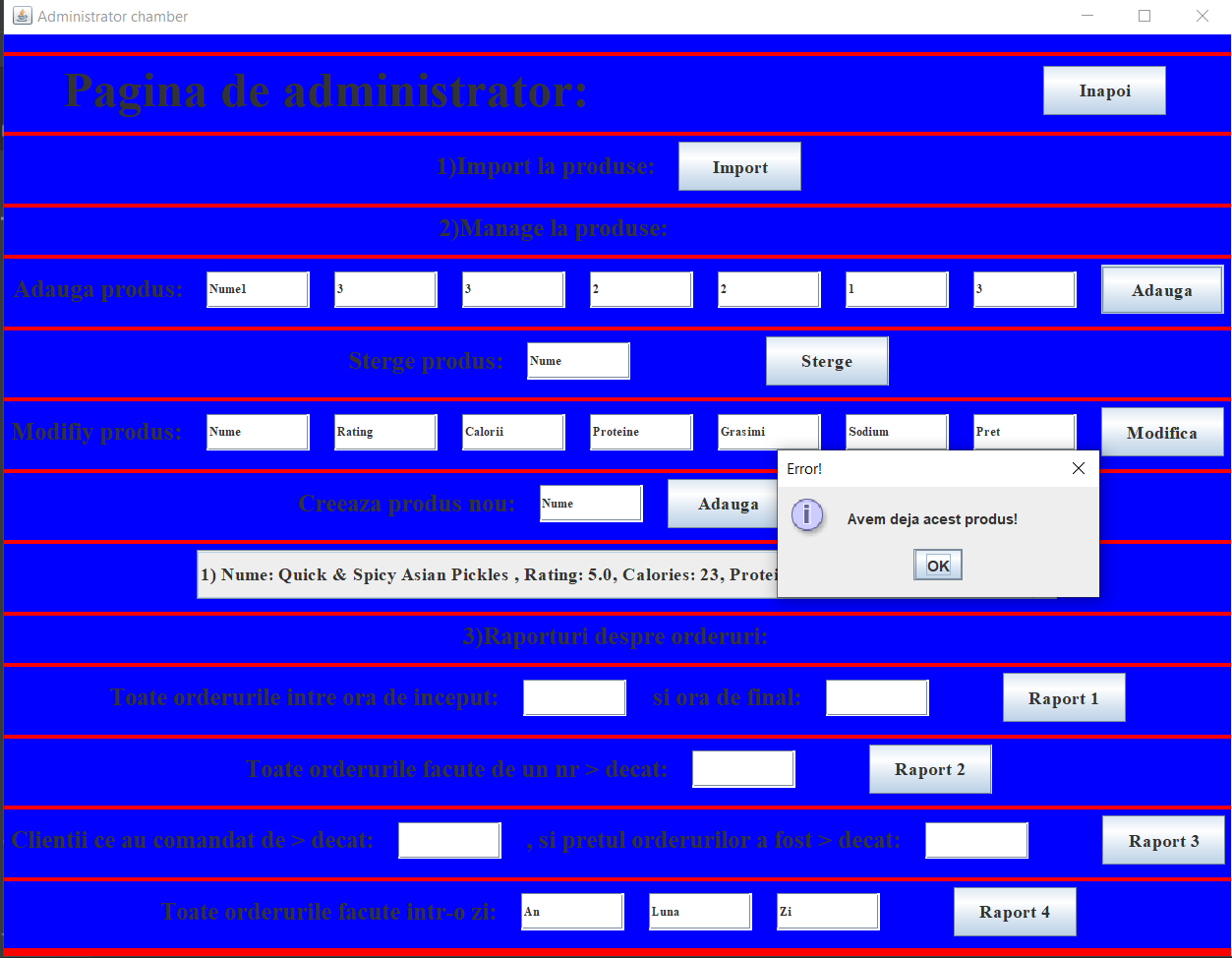
-Acestea au fost toate windowurile functionale ale aplicatiei, sunt multe operatii posibile, desi si guiul si complexitatea operatiilor au fost ridicate la alt nivel. Insa pare ca totul merge bine si ca totul are sens atunci cand te uiti la date si incerci sa le folosesti.

-Resetez de fiecare data cand dau pe un buton de inapoi, si astfel nu raman date salvate de mai devreme nicaieri.

Cerintele non-functionale ale programului sunt:

-Ar trebui sa fie usor de folosit de catre utilizator, sa fie clar unde trebuie sa introduca date, cum functioneaza si unde se afiseaza rezultatul sau rezultatele finale.

-In caz de erori, sa fie evidentiat faptul ca avem acea eroare, si sa cunoasca utilizatorul faptul ca trebuie sa schimbe valorile de intrare, ca sa corespunda la cerintele problemei.



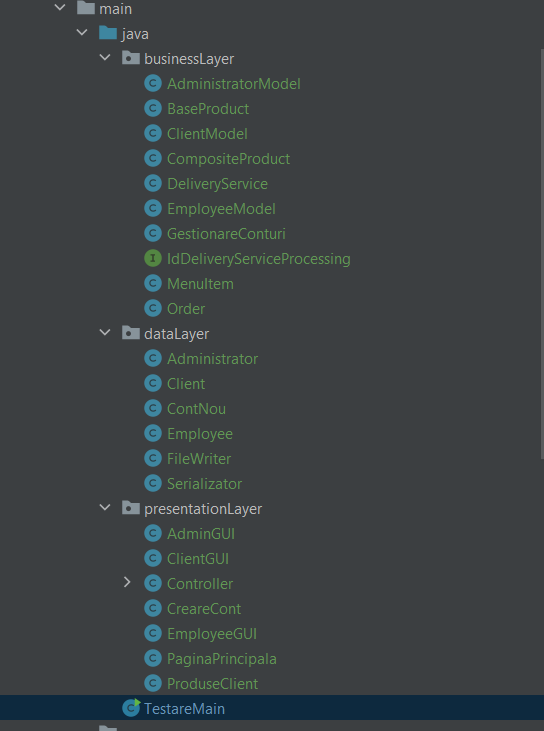
-Sa poata fi reutilizabil, daca se doresc alte date de intrare, sa se poata face accesibila aceasta optiune cat mai usor. (deci cu un buton de inapoi care reseteaza datele se poate obtine usor aceasta cerinta)

-Claritatea datelor este un mare avantaj la un astfel de GUI, pentru ca avem multe date, si vrem ca ele sa fie cat mai usor de inteles si cat mai usor de accesat de catre utilizatori.

1. Proiectarea problemei:

Pentru proiectarea problemei, am sa intru, in detaliu, in mai multe aspecte: (sunt multe aspecte de discutat)

Primul aspect: Pachetele si structura lor:



-Ca si structura, impartirea pe pachete am facut-o astfel: in presentationLayer am pus clasele legate de GUI, ele au cel mai mare acces, in dataLayer sunt cele mai de jos date, ele au acces cel mai restrictiv, iar in businessLayer este stratul de mijloc. Main se afla inafara, si doar initializeaza structura MVC, ca dupa sa se poata rula toate schimbarile din GUI.

Al doilea aspect: Fieldurile:

-Fieldurile sunt private, in majoritatea claselor, pentru a putea exista o incapsulare mai puternica a codului. Pentru toate fieldurile importante si folosite si altundeva decat in clasa proprie, am creat setteri si getteri. Nu ma bazez prea mult pe setteri, dar folosesc des getterii, pentru a extrage date din obiecte folositoare, sau pentru a extrage datele de intrare a utilizatorilor.

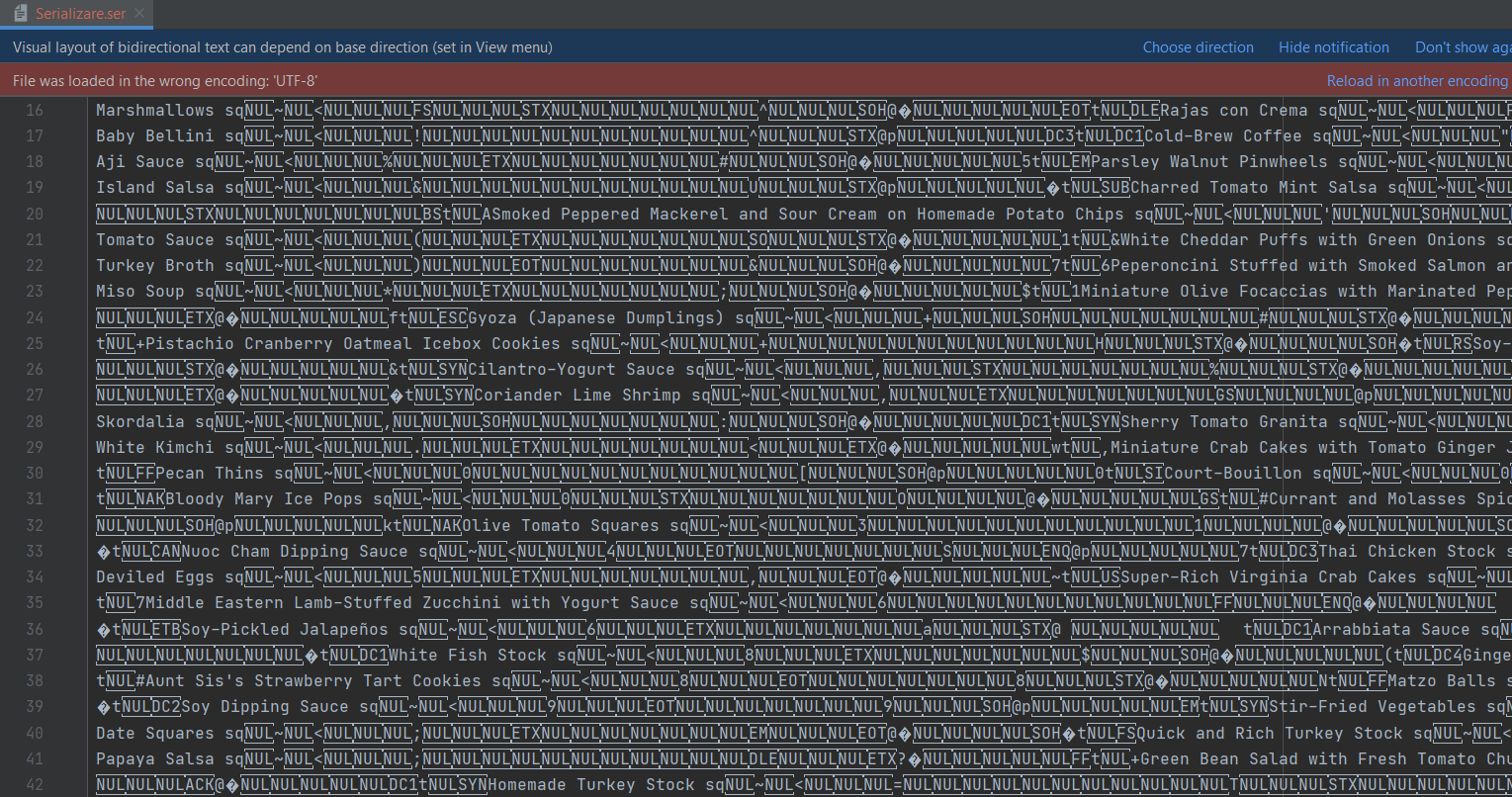
Al treilea aspect: Lambda expressions si Stream processing:

-Un aspect important al temei il reprezinta Lambda Exmpressions si Stream Processing. Ele ajuta codul sa fie mult mai lizibil si mai rapid, dpdv al VM. Codul folosit are doar cateva randuri, si foloseste operatii pe streams, combinate cu operatii lambda, pentru a structura datele dupa cum dorim. Seamana cu niste query-uri, si se bazeaza pe ideea ca ii spunem la compilator ce sa faca, nu si cum. Astfel, putem folosi operatii pe stream simple, in contrast cu bucle si metode imbarligate.

-La toate operatiile clientilor si administratorilor unde am folosit lambda si stream am avut grija sa folosesc cat mai putine bucle sau altfel de conditii declarative. Dar, a fost nevoie pentru scrierea in rapoarte, in fisierele text, de o parcurgere cu for, pentru a scrie in fisiere, iar la search pentru client a fost nevoie de o verificare a listelor, cu un if, pentru a vedea daca sunt sau nu goale inainte de filtrare. Nu am fost 100% nevoit sa folosesc o strategie asemenea, dar mi s-a parut ca ar fi fost prea greu in alte metode, si ca merge bine asa.

Al patrulea aspect: Serializare:

-Un aspect important al temei, desi nu obligatoriu, este serializarea. Pentru a refolosi datele existente deja din alte rulari ale programului, rulari precedente, poti serializa tot timpul datele inainte de inchiderea programului. Asta inseamna ca obiectele instantiate vor fi transformatie, serializare, in sir de bytes, si vor fi salvate in fisiere ce seamana cu fisierele binare la structura, fisiere .ser:



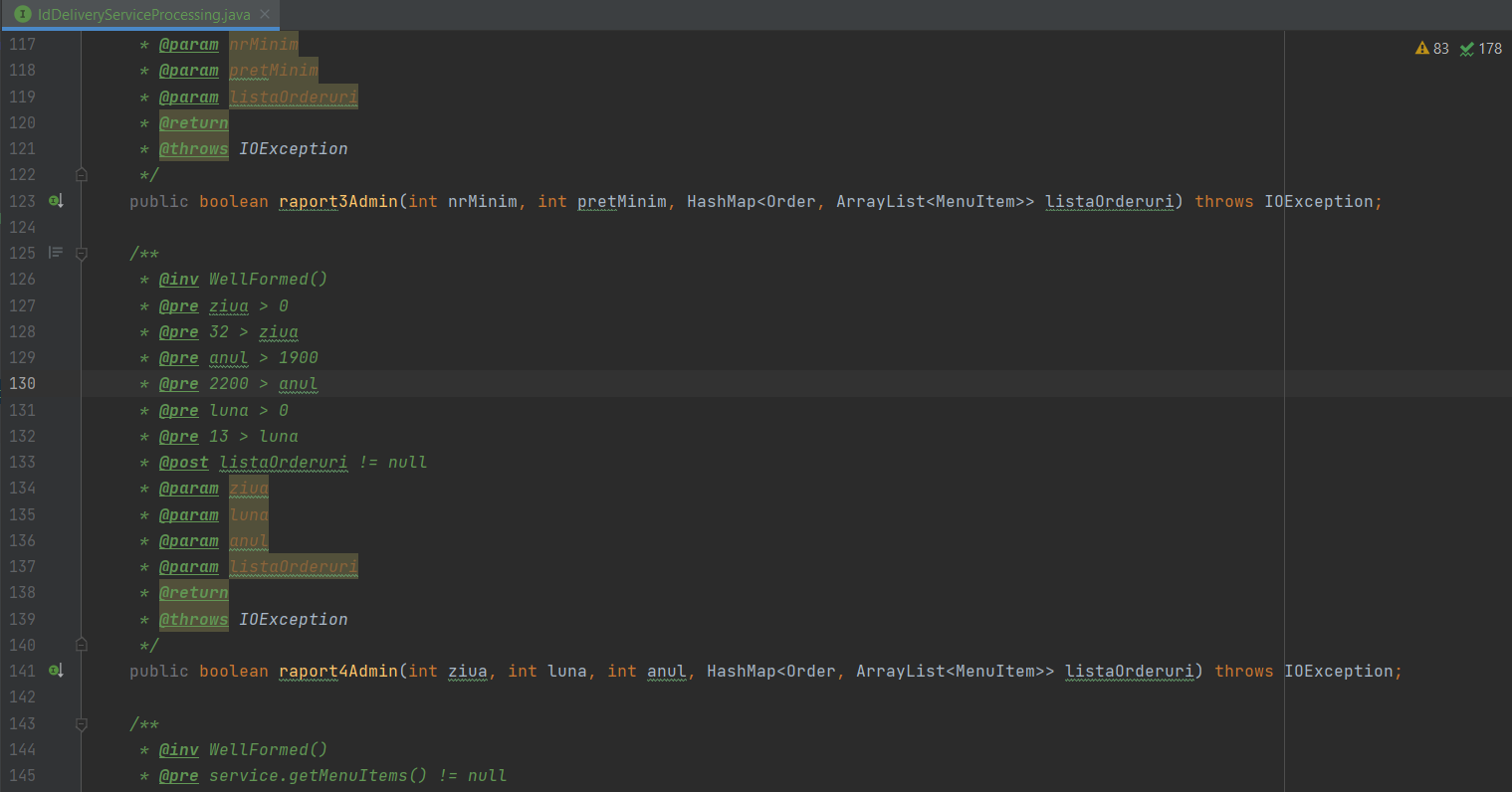
-Dupa serializare, datele raman salvazte in acest fisier. Pentru a refolosi datele, pentru a deserializa aceste date criptate inapoi in obiecte, folosim deserializarea, si extragem inapoi obiectele, folosind casting. Folosind aceasta strategie pe clasa DeliveryService, putem salva toate listele importante, pentru a putea fi refolosite. Astfel nu mai trebuie sa creem conturi noi sau sa dam import la fiecare rulare a programului, ci dupa ce dam import o data si serializam, vor ramane datele acolo atata timp cat fisierul serializat ramane acolo.

Al cincilea aspect: Notificarea Employee, Desing Pattern:

-Pentru a notifica employee atunci cand un client face un order, am folosit un design pattern specific. Am dat la DeliveryService extends la Observable, pentru a putea ca atunci cand dam add la Order sa notificam observerul, adica EmployeeGui, care implementeaza Observer. In clasa DeliveryService, setChanged si notify dupa ce schimbam datele, adica adaugam un order nou, va face ca EmployeeGui sa dea update la obiectul salvat, adica obiectul Observable pe care il observa. La fiecare order nou, un mesaj este trimis in EmployeeGui pentru a fi notificat cu acel nou order.

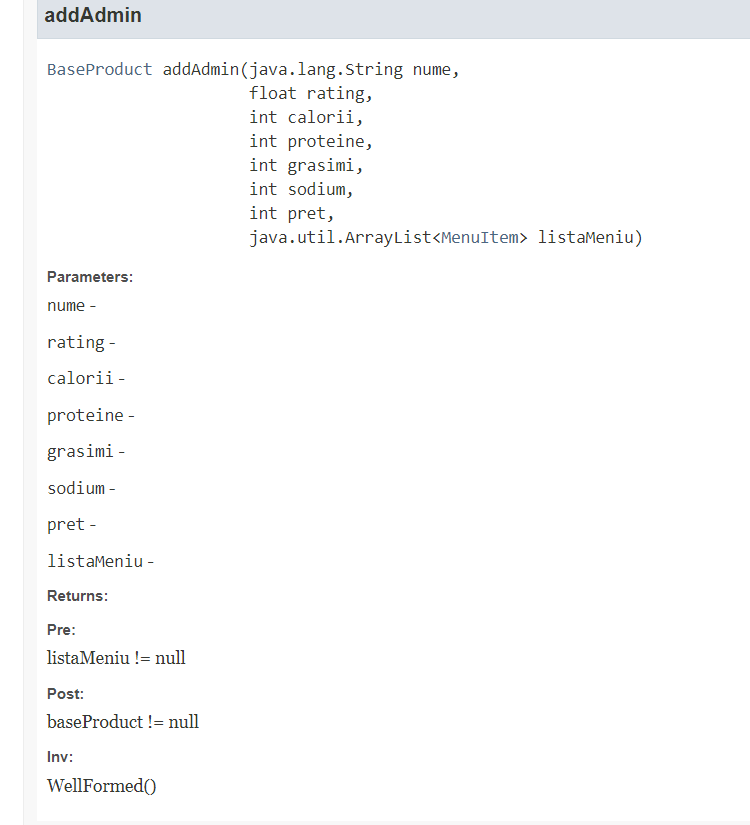
-Este o metoda buna de a lucra, cu aceasta notificare instanta. Ajuta la o comunicare rapida intre obiecte.

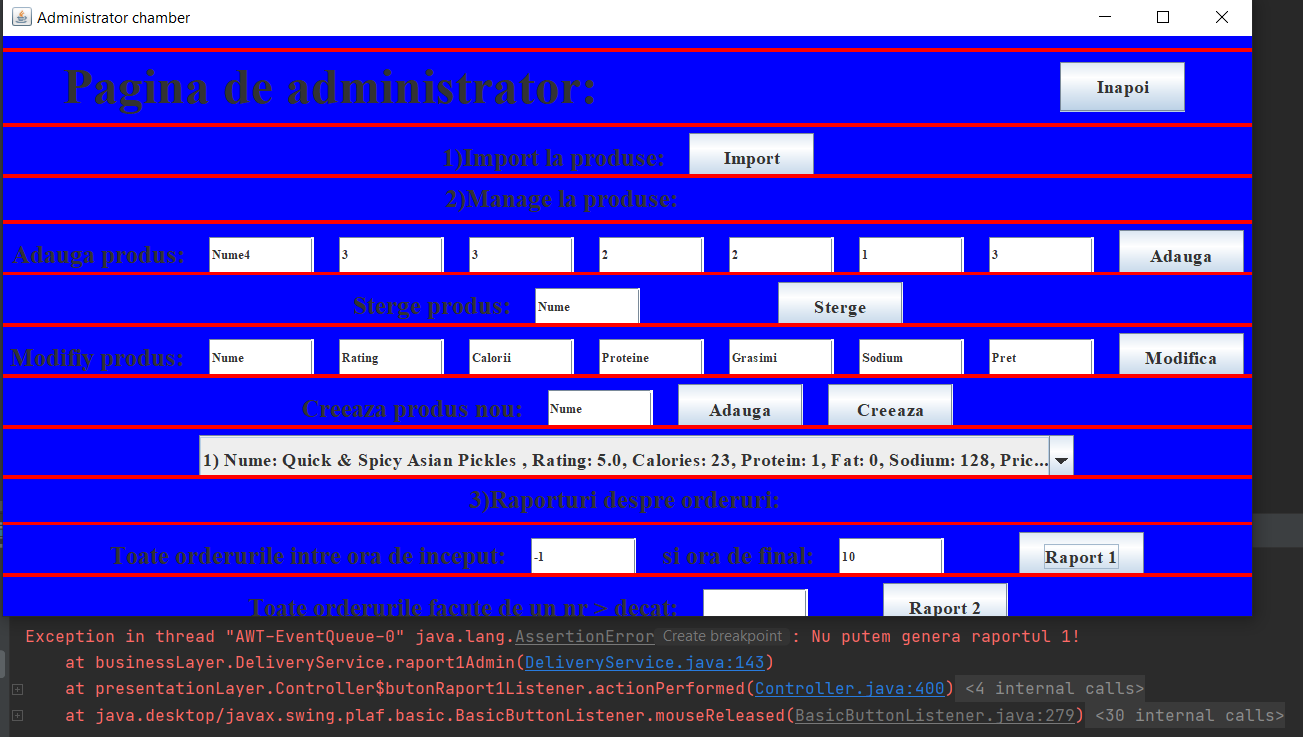
Al saselea aspect: Conditiile JavaDoc:

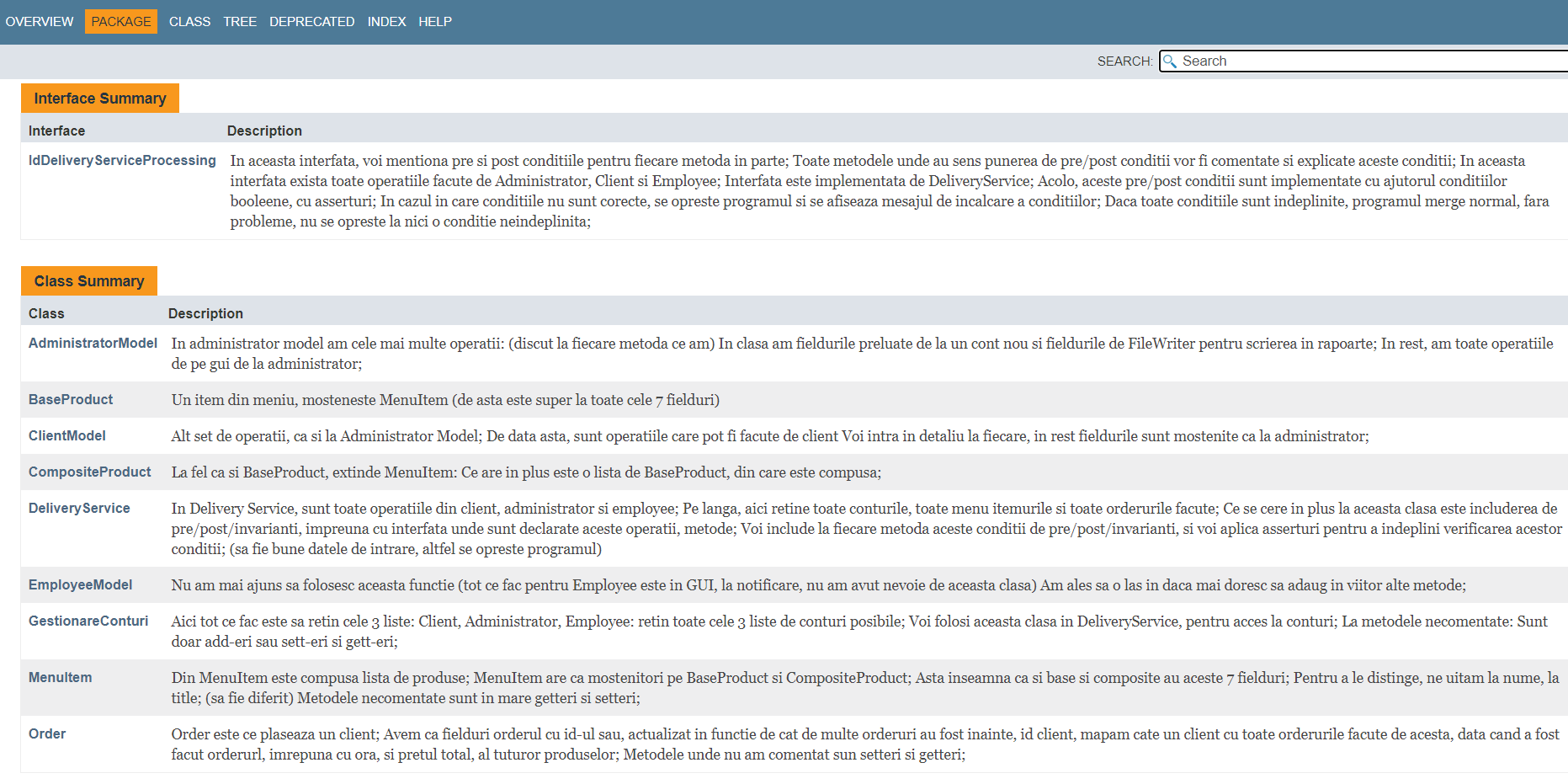


-Cu ajutorul lui JavaDoc, putem sa documentam mai bine proiectul nostru, punand comentarii folositoare la fiecare colt. Sa introducem comentarii inainte de clase, metode, cu taguri legate de parametrii, return, throw, etc... .Pentru aceasta tema, ne-am axat pe pre/post conditii la metode, si anume cum este un obiect inainte si/sau cum este dupa metoda (vrem sa fie intr-o anumita stare inainte si intr-o anumita stare dupa).

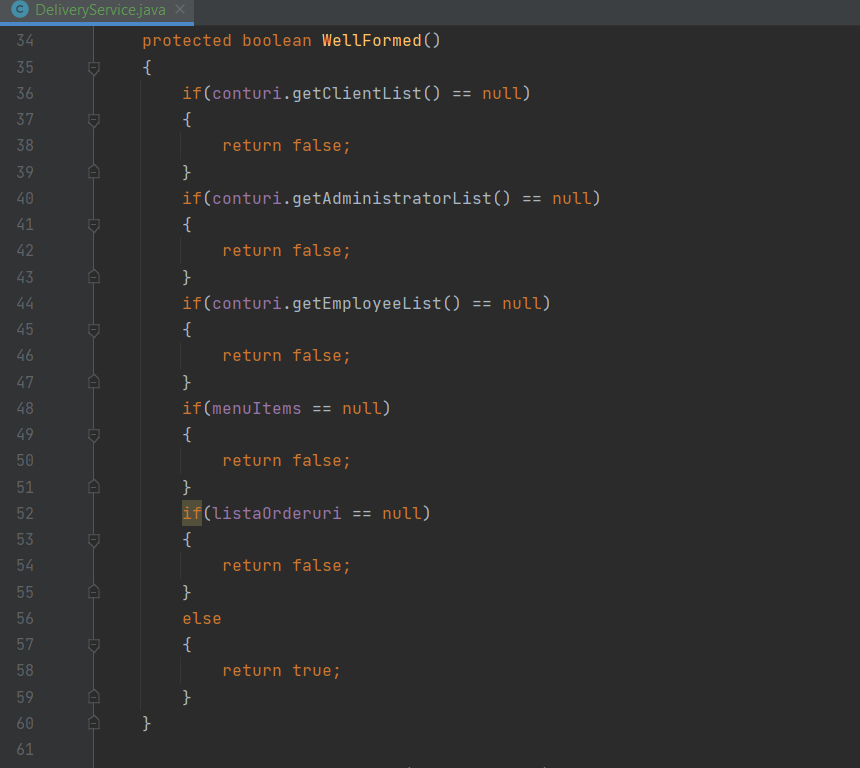
-Folosind asserturi, putem gestiona aceste pre/post conditii. Cu asserturile, putem opri programul atunci cand nu avem datele in regula, si putem regandi unde am fi putut neglija anumite conditii.



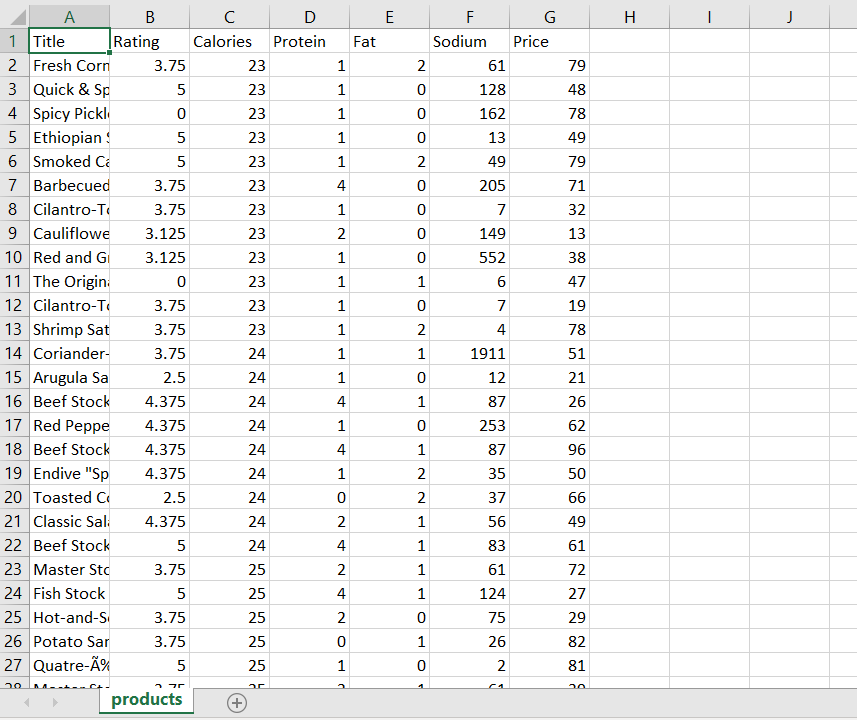




-De asemenea, cu invariantul putem sa gestionam mai multe conditii specifice, pentru mai multe metode, si sa ne asiguram ca tot timpul se termina cu valoarea true. Daca oricand, o va lua pe o valoare false, se va termina ca si mai sus.



Al saptele aspect: Citirea din CSV:



-Nu este mult de remarcat aici, doar ca si aici se poate lucra cu stream processing. Luand fiecare linie pe rand, putem mapa obiectele direct din stringul initial, in obiectele finale, extragand datele cerute din stringurile avute pe fiecare linie. Acele date sunt impartite intre „ , ”. Trebuie dat skip la prima linie, deoarece acea prima linie este doar prezentarea titlurilor categoriilor, si cu distinct pe MenuIteme, te vei asigura ca nu se repeta 2 produse, ca nu sunt 2 produse cu acelasi nume, toate sunt unice. Distinc functioneaza daca suprascrii metodele equals si hashCode.

-Am intalnit acasta suprascriere si la structura HashMap, atunci cand am vrut sa mapam la fiecare order o lista de produse. Am folosit equals si hashCode pentru a defini idOrder ca fiind cheia de care se leaga lista de valori, de produse.

Au fost multe subiecte inedite la aceasta tema, acestea au fost cele de care am dorit sa ma ating cel mai tare.

1. Implementarea problemei:

Voi vorbi pe scurt despre fiecare clasa in parte: In documentatia JavaDoc deja am scris la fiecare clasa si la fiecare metoda niste informatii, si deci nu voi intra in amanunte mult prea detaliate aici, deoarece ar fi prea mult de zis la fiecare:

-ContNou:

Clasa unde scriu la fiecare cont datele personale;

-Administrator/Client/Employee;

Extinde ContNou, doar la Client pun un camp nou, id, pentru mapare cu order, in rest doar salvez datele personale ale fiecaruia, si pe langa retin si clasa care detine acele date personale;

-FileWriter:

Aici scriu in fisier Billul creat la crearea unui Order de catre un client anume;

-Serializator:

Clasa unde serializez si deserializez; O folosesc in DeliveryService, aka model, pentru a serializa cand schimb ceva, si in main deserializez inainte de restul operatiilor, deci la inceputul inceputului;

-MenuItem/BaseProduct/CompositeProduct:

Pentru a tine informatia pentru produse. BaseProduct este un MenuItem simplu, iau CompositeProduct are aceleasi caracteristici ca un BaseProduct, dar acele caracteristici se obin din mai multe BaseProduct adunate la un loc, deci are o lista de BaseProducts in ea.

-AdministratorModel/ClientModel/EmployeeModel:

Aici fac toate operatiile gasite in IdDeliveryServiceProcessing, direct din clasele ce extind cele 3 conturi. Prima data apelez din DeliveryService operatiile in sine, dupa care calculele esentiale se fac in aceste 3 clase;

-GestionareConturi:

Aici tin datele personale a tuturor celor care au cont. Am 3 liste, de Clienti, Administratori si Employees, deci orice cont facut vreodata va ajunge aici;

-DeliveryService/IdDeliveryServiceProcessing:

Una din clasele principale: Folosesc aceasta clasa ca model pentru MVC, aici tin conturile, MenuItemurile si Orderurile. De asemenea, cu aceasta clasa implementez toate operatiile, de ea ma folosesc pentru a apela operatiile in controller, atunci cand este necesar. Este o clasa foarte folositoare, si interfata IdDeliveryServiceProcessing este interfata cu toate operatiile, de la cele 3 tipuri de conturi. Nu face mai multe, decat ca aici sunt puse tagurile de pre/post conditii;

-Order:

In DeliveryServicei exista HashMapul in care se leaga un order, cu id-ul sau de o lista de produse, dar order in sine este doar o clasa cu anumite date speficice in fielduri, si anume id client, order, data actuala si ora actuala, si are ca rol doar maparea unui order cu o lista de produse anume, deci ca o recunoastere a acelei liste prin datele acelui order generat automat. Se genereaza IdOrder in ordine, nu se da de la tastatura, ca si la IdClient.

-AdminGui/ClientGui/EmployeeGui:

Ferestrele de care vorbeam la UseCaseuri: Aici se face get la toate datele date de catre utilizator, si set dupa la datele dorite. Toate 3 au functionalitati complexe, am vorbit deja de ele.

-PaginaPrincipala/CreareCont:

Alte 2 pagini, ferestre de GUI, se ocupa de partea cu creare conturi si logare in conturi. Simple ca si concept, se intalnesc oriunde in ziua de azi, am discutat la JavaDoc si la UseCaseuri de ele;

-ProduseClient:

Pagina GUI pentru view la client, nimic mai mult. Doar un comboBox updatat cand trebuie; (adica setat cand trebuie, la momentele cand conteaza)

-Controller:

Una din cele mai cruciale componente ale proiectului: Cu controllerul legam viewurile de model, deci facem legaturile cele mai importante. La apasarea butoanelor, se intampla ce dorim. Se updateaza datele cand apasam pe buton, dam notificare cand apasam, stergem date cand apasam, etc... . Functioneaza ca si un controller normal, dar pentru ca sunt atat de multe operatii, el are un rol important in fiecare din ele.

Am avut multe clase la aceasta tema, insa toate au avut un rol si si-au facut treaba, fiecare a avut roluri specifice si momente cand sa isi indeplineasca rolurile.

1. Rezultatele obtinute:

Deoarece am lucrat cu GUI, avem multe rezultate ce pot fi obtinute. In mare, avem mesaje de succes si mesaje de eroare.

-Mesajele de succes sunt urmate de schimbari precum updatarea unui comboBox, ca acum sa contina si datele ultima data puse sau modificate.

-Se poate ca dupa un mesaj de succes sa se creeze si un fisier txt cu anumite date, deoarece operatiile au fost corect facute.

-Contrar, daca datele nu sunt bune, programul se sfarseste si nu va ajunge sa isi modifice starea deloc. Se poate sfarsi in multe moduri, asa cum am vazut si mai sus:

-Poate sa primeasca date proaste de intrare, si sa nu poata converti datele cum trebuie, poate sa aiba liste goale cu care nu poate lucra, nu poti cauta daca nu alegi criterii de cautare, nu poti face un order fara produse, etc..., se poate sa nu gasesti un item, pentru ca el exista deja, sau sa nu poti creea un item nou cu un nume existent deja, deci multe posibilitati.

-Din pre/post conditii poti sa primesti warninguri din asserturi, si din nou, sa ai programul oprit. In toate aceste cazuri, vei fi notificat si se vor cere alte date, cele corecte, precum s-a reamintit in cerintele non-functionale.

1. Concluzii:

Aceasta tema a fost foarte interesanta, deoarece am inceput de la aproape nimic, si am ajuns sa avem o aplicatie la un asemenea nivel de complexitate, functionala. Este interesant de vazut evolutia, si toate lucrurile pe care le poti face sa imbunatatesti experienta, sau sa imbunatatesti functionalitatea (precum serializarea). Mi s-a parut foarte mult de lucru, pentru a reusi sa faci toate lucrurile sa functioneze una cu alta, si foarte multe layer-uri de complexitate, insa atunci cand totul merge cum trebuie, este foarte satisfacator.

Pentru viitor, as dori sa imbunatatesc unele functionalitati, sa fie chiar mai flexibile, precum ideea de a putea selecta din itemele cautate, sau sa fac pana si mai multe din date serializabile, dar in ochii mei aceasta aplicatie merge mai bine decat m-as fi asteptat. Am lucrat la multe aspecte OOP, nu toate au fost manevrate la perfectie, dar am inteles si invatat multe.

Au fost 4 teme grele, dar am terminat si am reusit sa le fac pe toate functionale pana la urma.

1. Bibliografie:

* Materialul ajutator oferit pentru aceasta tema.
* <https://docs.oracle.com/javase/tutorial/java/javaOO/lambdaexpressions.html>
* <https://www.oracle.com/technical-resources/articles/java/ma14-java-se-8-streams.html>
* <https://howtodoinjava.com/java8/java-stream-distinct-examples/>
* <https://www.tutorialspoint.com/java/java_serialization.htm>
* <https://docs.oracle.com/javase/8/docs/technotes/guides/language/assert.html>
* <https://docs.oracle.com/javase/7/docs/technotes/tools/windows/javadoc.html#tag>
* <https://www.baeldung.com/java-observer-pattern#:~:text=Observer%20is%20a%20behavioral%20design,channels%20when%20it%20receives%20news>.
* <https://dzone.com/articles/how-to-read-a-big-csv-file-with-java-8-and-stream?fbclid=IwAR0jFHT4cRVCvNbqcKtBZu0oU3mu29Ph_8qiIWfF4UhUaWenLTPJsXvyUPI>