**FOOD DELIVERY**

**MANAGEMENT SYSTEM**

**IRIMIA LAURA-MARIA**

**GRUPA 302210**

**PROFESOR LABORATOR: MITREA DAN**

Cuprins

[Obiectivul temei 3](#_Toc72757467)

[Obiectivul principal 3](#_Toc72757468)

[Obiective secundare 3](#_Toc72757469)

[Analiza problemei, modelare, scenarii, cazuri de utilizare 4](#_Toc72757470)

[Proiectare (decizii de proiectare, diagrame UML, structuri de date, proiectare clase, interfețe, relații, packages, algoritmi, interfața utilizator) 5](#_Toc72757471)

[Package -> Class 5](#_Toc72757472)

[Class -> Descriere 5](#_Toc72757473)

[Diagrama UML: 8](#_Toc72757474)

[Interfața: 8](#_Toc72757475)

[Interfața administrator: 8](#_Toc72757476)

[Interfața client: 9](#_Toc72757477)

[Interfața angajat: 9](#_Toc72757478)

[Implementare 10](#_Toc72757479)

[Interfața „IDeliveryServiceProcessing”: 10](#_Toc72757480)

[Clasa „DeliveryService”: 11](#_Toc72757481)

[Clasa „Controller”: 15](#_Toc72757482)

[Rezultate (testare) 17](#_Toc72757483)

[Concluzii 17](#_Toc72757484)

[Bibliografie 17](#_Toc72757485)

# Obiectivul temei

## Obiectivul principal

Obiectivul principal al acestei teme este proiectarea și implementarea unei aplicații pentru gestionarea livrării de mâncare pentru o companie de catering. Clienții pot comanda produse din meniul companiei. Sistemul implementat trebuie sa aiba trei tipuri de utilizatori: administrator, angajat obișnuit și client.

Administratorul poate încărca toate produsele în meniul companiei dintr-un fișier de tip .csv. Totodată, poate să modifice produsele din meniu: să adauge, să șteargă, să modifice detaliile produselor și să creeze noi produse compuse formate din produse de bază. De asemenea, poate să genereze raporturi despre comenzile plasate în companie după anumite criterii:

* Intervalul de tip al comenzilor: comenzile plasate între o oră minimă de start și o ora maximă de sfârșit;
* Produsele comandate de mai mult de un număr dat de ori;
* Clienții care au comandat mai mult de un număr dat de ori și a căror comandă a fost mai mare decât un preț dat;
* Produsele care au fost comandate într-o zi specificată împreună cu frecvența lor.

Clientul poate să se înregistreze și să se logheze în aplicație folosind un nume de utilizator și o parolă. De asemenea, poate să vizioneze toate produsele din meniul companiei. Totodată, poate să caute anumite produse din meniu după diferite criterii: titlu, rating, calorii, proteine, grăsimi, sodiu și preț. De asemenea, poate plasa o comandă formate din diferite produse. Pentru fiecare comandă va fi salvată data și ora și se va genera o factură care o să conțină toate produsele comandate și prețul total al comenzii.

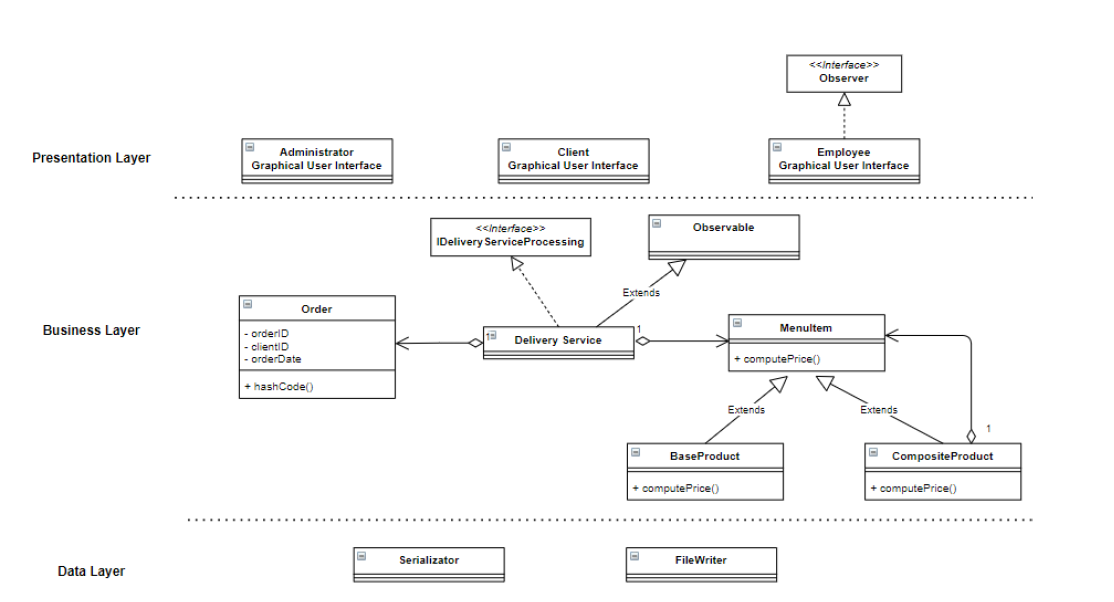
Angajatul va fi notificat de fiecare dată când o nouă comandă este plasată pentru a pregăti toate produsele comandate.

## Obiective secundare

|  |  |
| --- | --- |
| **Obiectiv secundar** | **Descriere** |
| Dezvoltarea de use case-uri și scenarii | Căutarea de scenarii în care aplicația dezvoltată o să fie folosită și de către cine va fi folosită. |
| Alegerea structurilor de date | Proiectarea unor structuri eficiente și potrivite cerințelor date. |
| Împărțirea pe clase | Alegerea unor clase care vor ușura proiectarea problemei. |
| Dezvoltarea algoritmilor | Implementarea algoritmilor necesari rezolvării problemei și rezolvării cazurilor speciale. |
| Implementarea soluției | Combinarea tuturor algoritmilor implementați in vederea rezolvării problemei. |
| Testare | Găsirea cazurilor care pot crea probleme aplicației dezvoltate și testarea acestora. |

# Analiza problemei, modelare, scenarii, cazuri de utilizare

Pentru a implementa sistemul de gestionare a companiei trebuie să declarăm o interfață care va conține toate operațiile pe care administratorul și clienții le pot face. Administratorul poate încărca produsele, le poate modifica și poate genera rapoarte. Clientul poate crea o nouă comandă care presupune calcularea prețului și generarea unei facturi cu formatul .txt și poate căuta produse după diferite criterii.



Trebuie să definim și să implementăm clasele din diagrama de mai sus. Trebuie să folosim Composite Design Pattern pentru definirea claselor pentru produsele de baza, produsele compuse și pentru meniu. Trebuie să folosim Observer Design Pattern pentru a notifica angajatul de fiecare dată când o nouă comandă este plasată.

Trebuie să implementăm clasa care suprascrie toate metodele pentru operațiile administratorului și clienților utilizând o colecție predefinită de tip hashtable. Cheia hashtable va fi generată automat, fiind bazată pe clasa corespunzătoare comenzilor, având asociată mai multe produse din meniu. Trebuie să definim o structură de tipul Map<Order, Collection<MenuItem>> în clasa DeliveryService care să stocheze informațiile legate de comenzi. Cheia va fi formată din obiecte de tipul Order, pentru care metoda care returnează valoarea cheii va fi suprascrisă pentru a calcula valoare respectivă din atributele clasei Order. Trebuie să definim o structură de tip Collection<MenuItem> în care vom salva toate produsele din meniul de la compania de catering. Trebuie să alegem o colecție potrivită pentru felul în care ne-am implementat noi aplicația. Trebuie să definim o metodă de tip “well formed” pentru clasa DeliveryService. Trebuie să implementăm clasa DeliveryService utilizând metoda de Design by Contract care presupune pre și post condiții, invarianți și assert-uri.

Produsele de bază utilizate inițial pentru popularea meniului vor fi încărcate din fișierul .csv aferent utilizând expresii lambda și procesarea stream. (Administratorul poate, de asemenea, să adauge noi produse de bază.)

Produsele din meniu, comenzile și utilizatorii vor fi salvați utilizând serializarea pentru a putea fi disponibili pentru viitoare utilizări ale aplicației utilizând deserializarea.

# Proiectare (decizii de proiectare, diagrame UML, structuri de date, proiectare clase, interfețe, relații, packages, algoritmi, interfața utilizator)

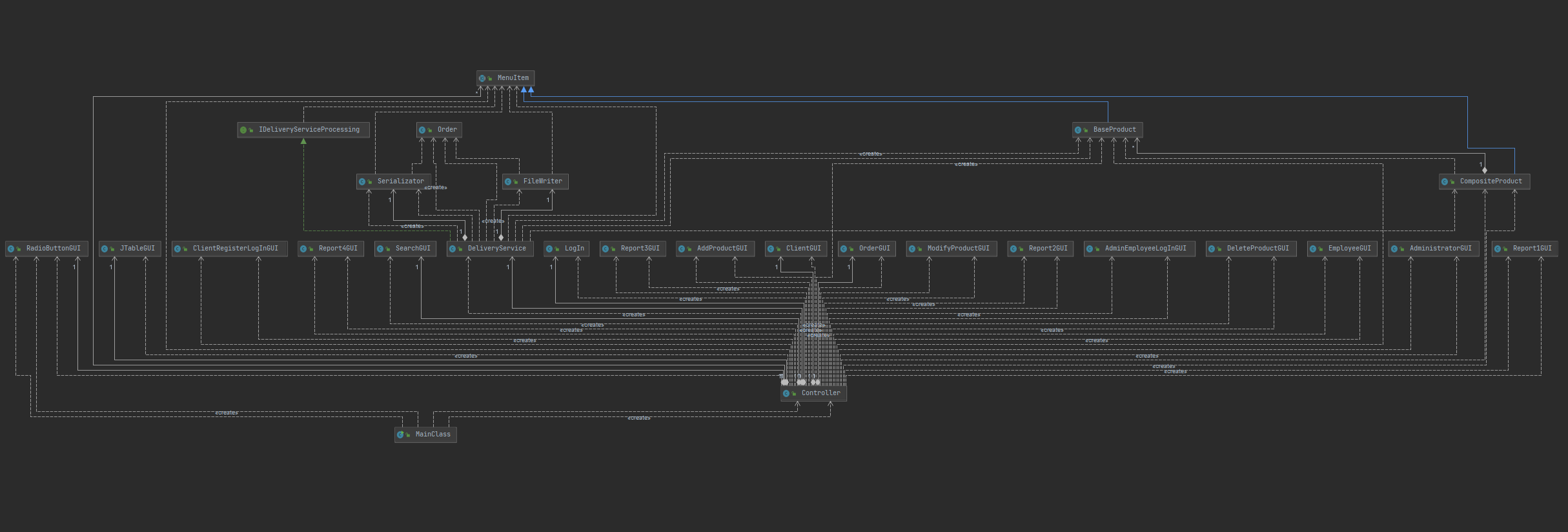
## Package -> Class

|  |  |
| --- | --- |
| **Package** | **Class** |
| main -> java -> **BusinessLayer** | BaseProduct  CompositeProduct  DeliveryService  IDeliveryServiceProcessing (interface)  LogIn  MenuItem  Order |
| main -> java -> **DataLayer** | FileWriter  Serializator |
| main -> java -> **PresentationLayer** | AddProductGUI  AdminEmployeeLogInGUI  AdministratorGUI  ClientGUI  ClientRegisterLogInGUI  Controller  DeleteProductGUI  EmployeeGUI  JTableGUI  ModifyProductGUI  OrderGUI  RadioButtonGUI  Report1GUI  Report2GUI  Report3GUI  Report4GUI  SearchGUI |
| main -> **java** | MainClass |

## Class -> Descriere

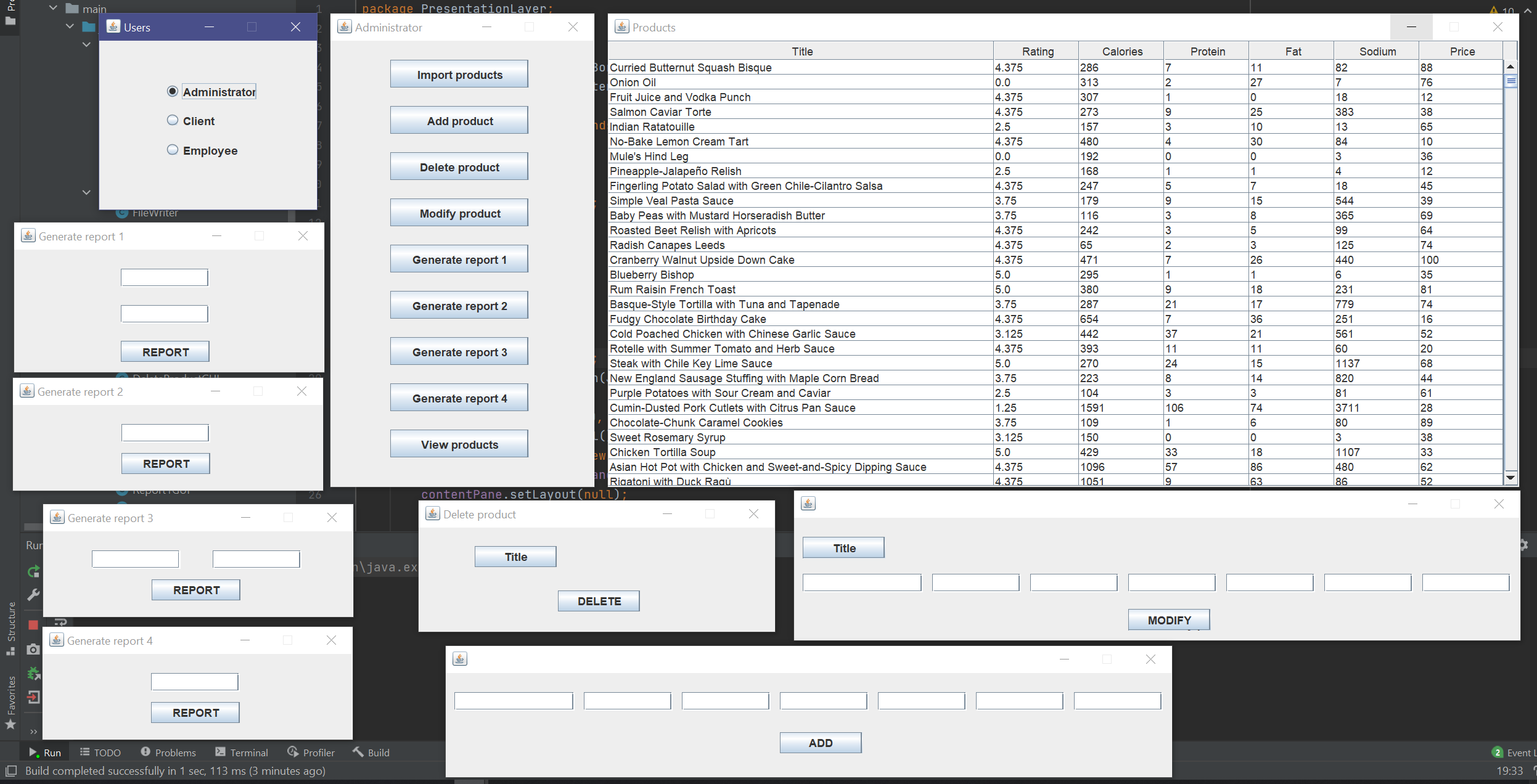
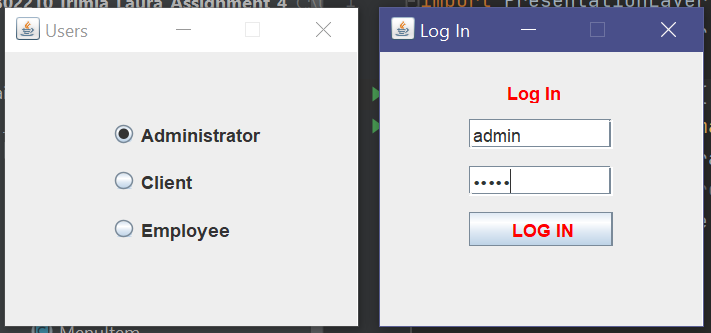
|  |  |
| --- | --- |
| **Class** | **Descriere** |
| BaseProduct | Clasa care extinde clasa MenuItem și care conține un constructor cu toți parametrii corespunzători unui produs de bază și o metodă care calculează și returnează prețul produsului. |
| CompositeProduct | Clasa care extinde clasa MenuItem și care are un singur parametru, o listă de produse de bază. Clasa conține și un constructor cu doi parametrii, titlul produsului compus și lista de produse de bază pe care trebuie să o conțină. În cadrul constructorului setez titlul și calculez restul parametrilor aferenți. Adun raiting-ul, caloriile, proteinele, grăsimile și sodiul tuturor produselor din lista primită ca parametru în variabilele aferente. Prețul îl calculez cu ajutorul metodei abstractă suprascrisă din clasa MenuItem care adună prețul fiecărui produs din lista de produse. Rating-ul îl setez ca fiind suma rating-urilor calculate mai sus împărțită la câte produse sunt în total în lista de produse. Restul parametrilor îi setez cu suma calculată în variabilele aferente de mai sus. |
| DeliveryService | Clasa care conține metodele suprascrise din interfața IDeliveryServiceProcessing pe care o impementează. Are doi parametrii, un obiect de tipul Serializator utilizat pentru serializare și deserializare și un obiect de tipul FileWriter utilizat pentru crearea fișierelor asociate facturilor comenzilor. Cele zece metode sunt suprascrise aferent operațiilor pe care trebuie să le execute. Clasa mai conține și metoda isWellFormed care returnează false numai dacă liste de produse ale meniului rămâne goală. Mai am și două metode care mă ajută să populez tabelul afișat în interfața grafică, o metodă care returnează true dacă titlul unui produs primit ca parametru este produs compus și o metodă care ne ajută să alegem doar produsele distincte din fișierul .csv în funcție de titlu. |
| IDeliveryServiceProcessing | Interfața care conține declararea metodelor pentru acțiunile administratorului și ale clienților. |
| LogIn | Clasa asociată înregistrării și logării clienților. Conține două metode, una pentru register și una pentru login. Metoda pentru register verifică în fișierul care conține toate numele de utilizatori și parolele asociate din cadrul companiei dacă numele de utilizator pe care clientul vrea să îl folosească este deja utilizat sau nu. Dacă este utilizat returnează false și true în cazul contrar. Metoda de login caută în fișierul de care am spus mai sus dacă numele de utilizator introdus există și dacă parola introdusă este cea aferentă. Dacă totul coresunde, returneză true, altfel false. |
| MenuItem | Clasa abstractă care implementează interfața Serializable. Clasa are toți parametrii asociați unui produs din meniu: titlu, rating, calorii, proteine, grăsimi, sodiu și preț. Avem metoda abstractă care calculează prețul și metoda suprascrisă toString. |
| Order | Clasa aferentă comenzilor. Avem trei parametrii importanți, ID-ul comenzii, ID-ul clientului și data la care a fost plasată comanda. ID-ul comenzilor este incrementat automat. Avem și metoda suprascrisă de hashCode care calculează cheia unde trebuie să fie asociată comanda și setul de produse comandate. |
| FileWriter | Clasa care conține metoda care scrie în fișierul corespunzător factura comenzii plasate. |
| Serializator | Clasa care conține metodele de serializare și deserializare pentru produsele din meniu și comenzile plasate. |
| AddProductGUI | Clasa care proiectează fereastra grafică aferentă adăugării unui produs în meniul companiei. Pentru a adăuga un produs de bază vom completa toate câmpurile aferente și vom apăsa butonul de adăugare. Pentru a adăuga un produs compus vom selecta din tabelul de produse produsele pe care le vrem în produsul compus, vom pune titlul dorit în câmpul aferent și vom apăsa butonul de adăugare. |
| AdminEmployeeLogInGUI | Clasa care proiectează fereastra grafică de login pentru administrator sau angajat. Conține două metode care verifică numele de utilizator și parola pentru fiecare în parte. Pentru administrator avem combinația „administrator”- „administrator” și pentru angajat avem combinația „employee”-„employee”. Metodele returnează un text corespunzător cazului în care este greșit numele de utilizator, cazului în care este greșită parola și cazului în care totul se potrivește. |
| AdministratorGUI | Clasa care proiectează fereastra grafică cu butoanele aferente fiecărei operații pe care o poate face administratorul și ascultătorii aferenți. |
| ClientGUI | Clasa care proiectează fereastra grafică cu butoanele aferente fiecărei operații pe care o poate face clientul și ascultătorii aferenți. |
| ClientRegisterLogInGUI | Clasa care proiectează fereastra grafică pentru înregistrarea și logarea clienților în aplicație împreună cu metodele și ascultătorii aferenți. |
| Controller | Clasa care implementează fiecare metodă ascultătoare din toate ferestrele corespunzătoare. |
| DeleteProductGUI | Clasa care proiectează fereastra grafică asociată ștergerii unui produs din tabel. Selectăm un produs din tabel, apăsăm butonul pentru titlul care pune titlul într-o etichetă și apoi apăsăm butonul de delete care va șterge produsul cu titlul selectat. |
| EmployeeGUI | Clasa care proiectează fereastra grafică pentru angajat. Atunci când se plasează o nouă comandă, aceasta va apărea după comenzile deja plasate înaintea ei sau chiar ea va fi prima. |
| JTableGUI | Clasa care proiectează fereastra grafică care conține tabelul folosit de fiecare operație care are nevoie de afișare sub formă de tabel. |
| ModifyProductGUI | Clasa care proiectează fereastra grafică aferentă modificării unui produs din tabel. Se selectează produsul corespunzător, se apasă butonul de titlu pentru scrierea în eticheta corespunzătoare a titlului produsului selectat și apoi se completează câmpurile care se vor schimbate și se apasă butonul de modificare. |
| OrderGUI | Clasa care proiectează fereastra grafică corespunzătoare plasării unei comenzi. Se selectează produsele care se vor comandate, se introduce ID-ul clientului și se plasează comanda. |
| RadioButtonGUI | Clasa care proiectează fereastra grafică principală a aplicației unde se alege ce tip de utilizator vrea să intre în aplicație. |
| Report1GUI | Clasa care proiectează fereastra grafică pentru generarea primului raport. |
| Report2GUI | Clasa care proiectează fereastra grafică pentru generarea celui de-al doilea raport. |
| Report3GUI | Clasa care proiectează fereastra grafică pentru generarea celui de-al treilea raport. |
| Report4GUI | Clasa care proiectează fereastra grafică pentru generarea celui de-al patrulea raport. |
| SearchGUI | Clasa care proiectează fereastra grafică aferentă căutării unui produs de către clientul corespunzător. |
| MainClass | Clasa care conține metoda main și care dă startul aplicației implementate. |

## Diagrama UML:

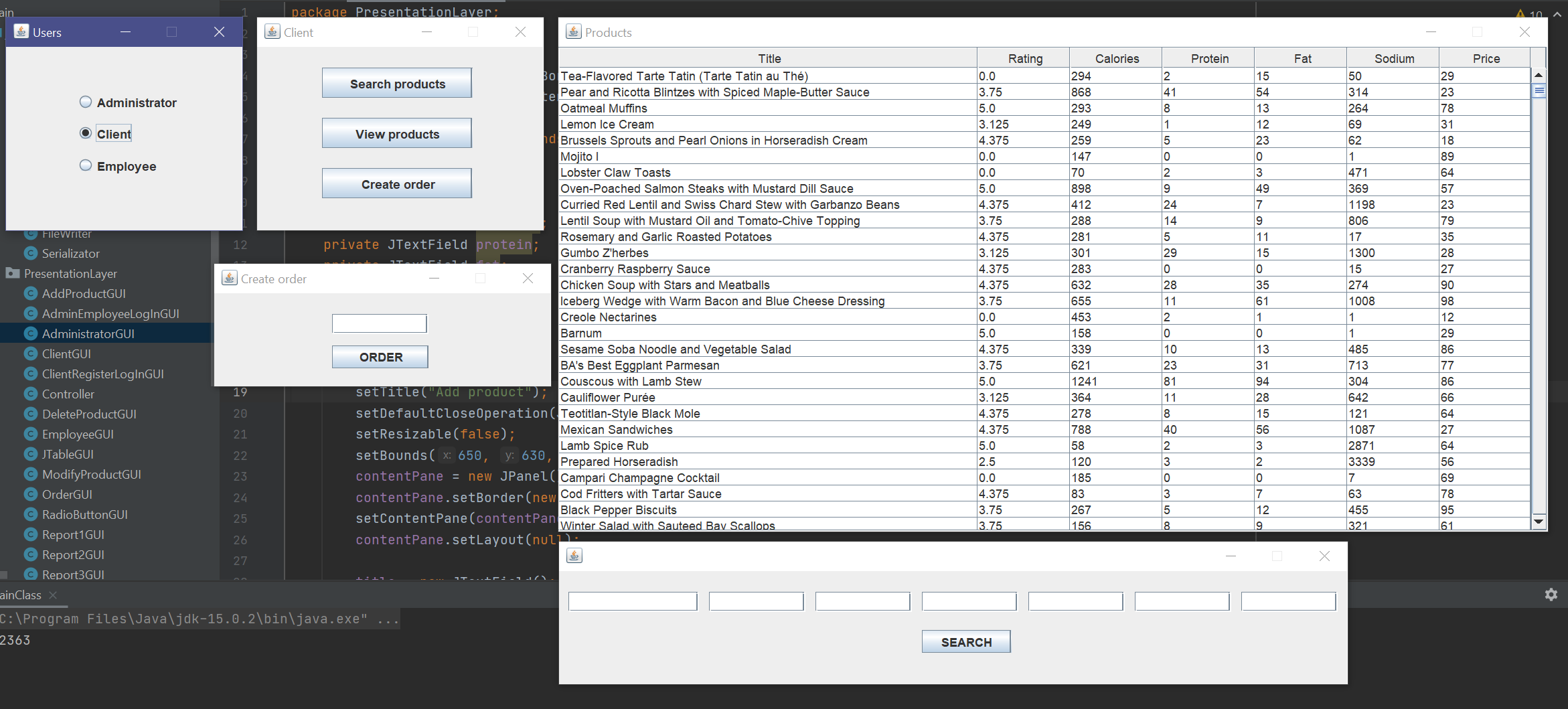
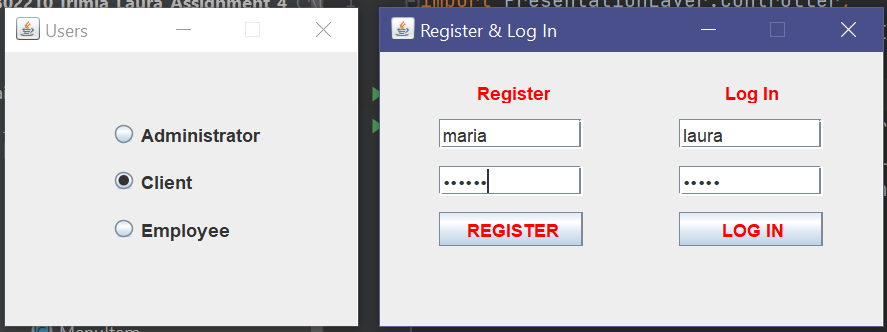


## Interfața:

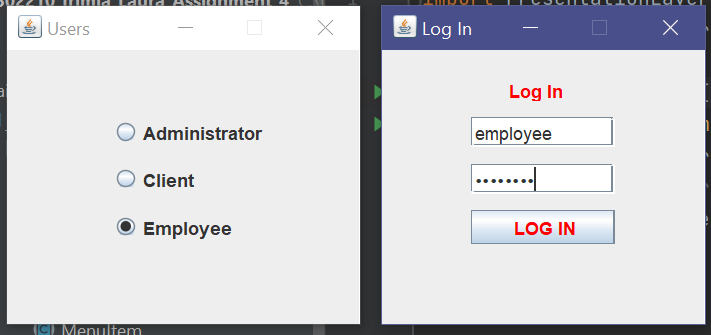
### Interfața administrator:

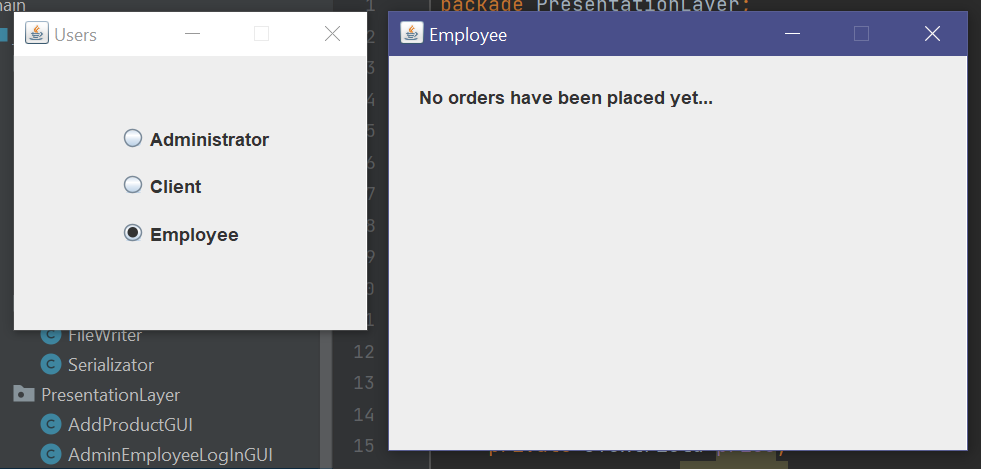


### Interfața client:



### Interfața angajat:



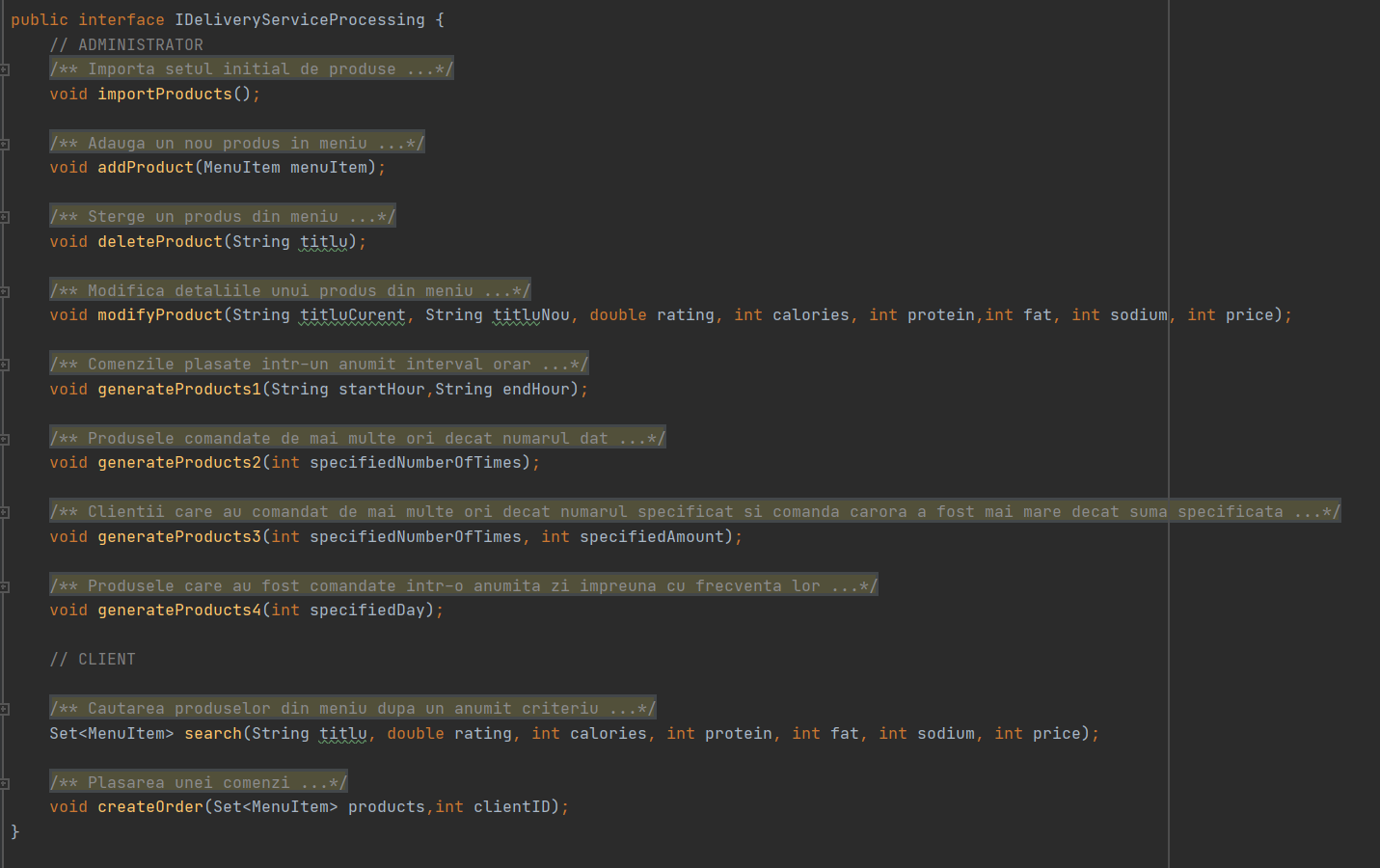


# Implementare

Voi adăuga doar anumite clase și metode care mi se par mie cele mai importante pentru a nu încărca inutil documentația.

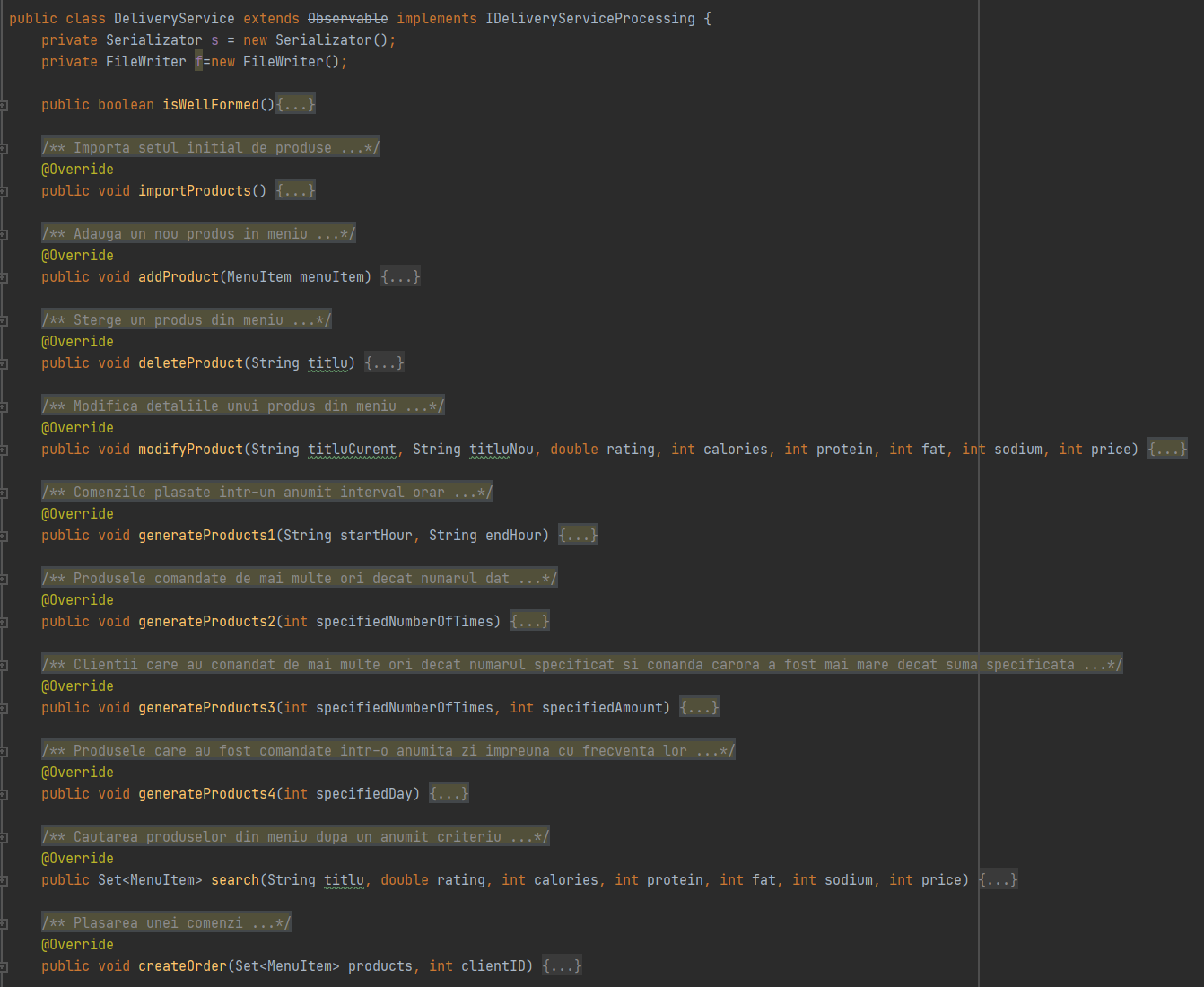
## Interfața „IDeliveryServiceProcessing”:

Interfața conține declararea tuturor metodelor aferente operațiilor pe care le poate executa administratorul, respectiv clientul împreună cu comentariile JavaDoc aferente.

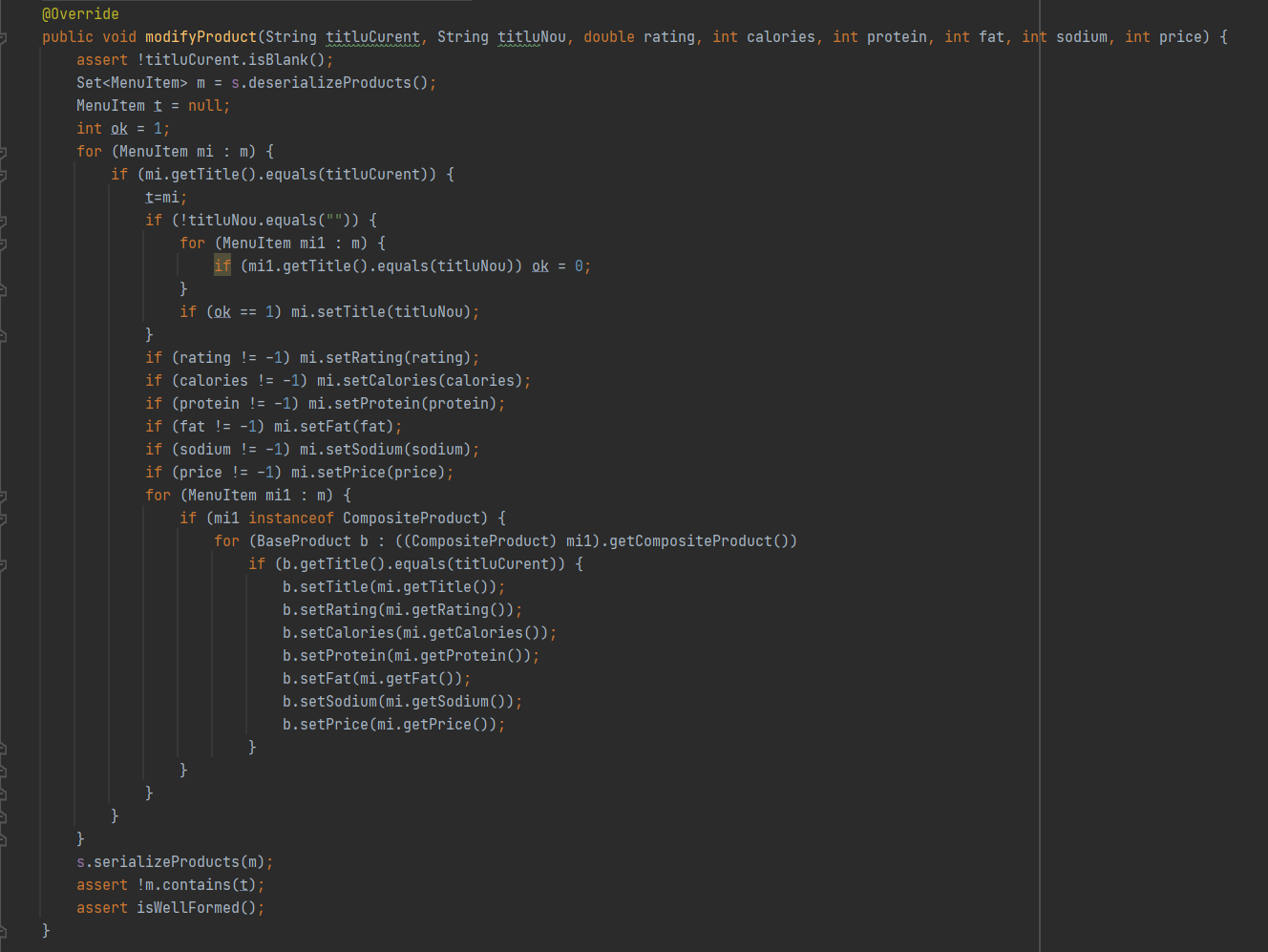


## Clasa „DeliveryService”:

Clasa „DeliveryService” împreună cu cei doi parametrii și toate metodele suprascrise din interfață și metoda isWellFormed().

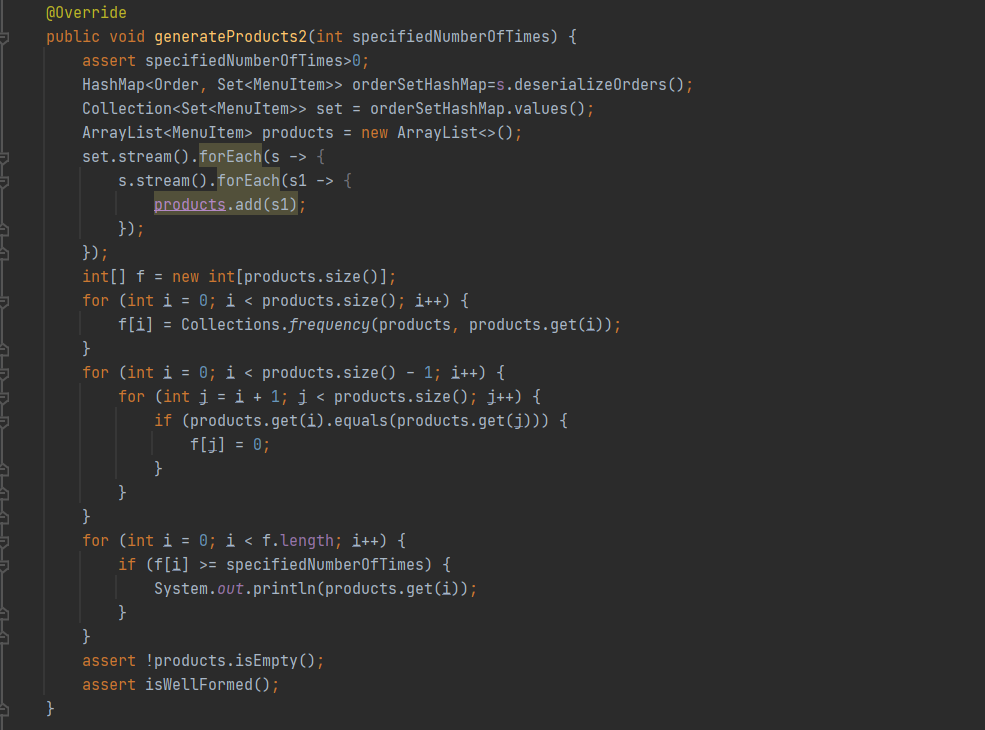


Metoda de modificare a unui produs din meniu am implementat-o în funcție de fiecare câmp al produselor din meniu și al titlului curent. Dacă valoarea câmpului primit ca parametru este diferită de -1, atunci el modifică câmpul aferent cu valoarea primită ca parametru. Dacă titlul nou nu este gol, atunci se modifică și titlul. Apoi, după ce produsul a fost modificat, caut produsul cu titlul curent sau cu titlul vechi, dacă titlul curent a fost modificat, în toate produsele compuse din meniu și verific dacă produsul cu titlul curent se află în lista de produse de bază din produsul compus. Dacă se află în lista respectivă, atunci îl modific și în lista din produsul compus aferent.

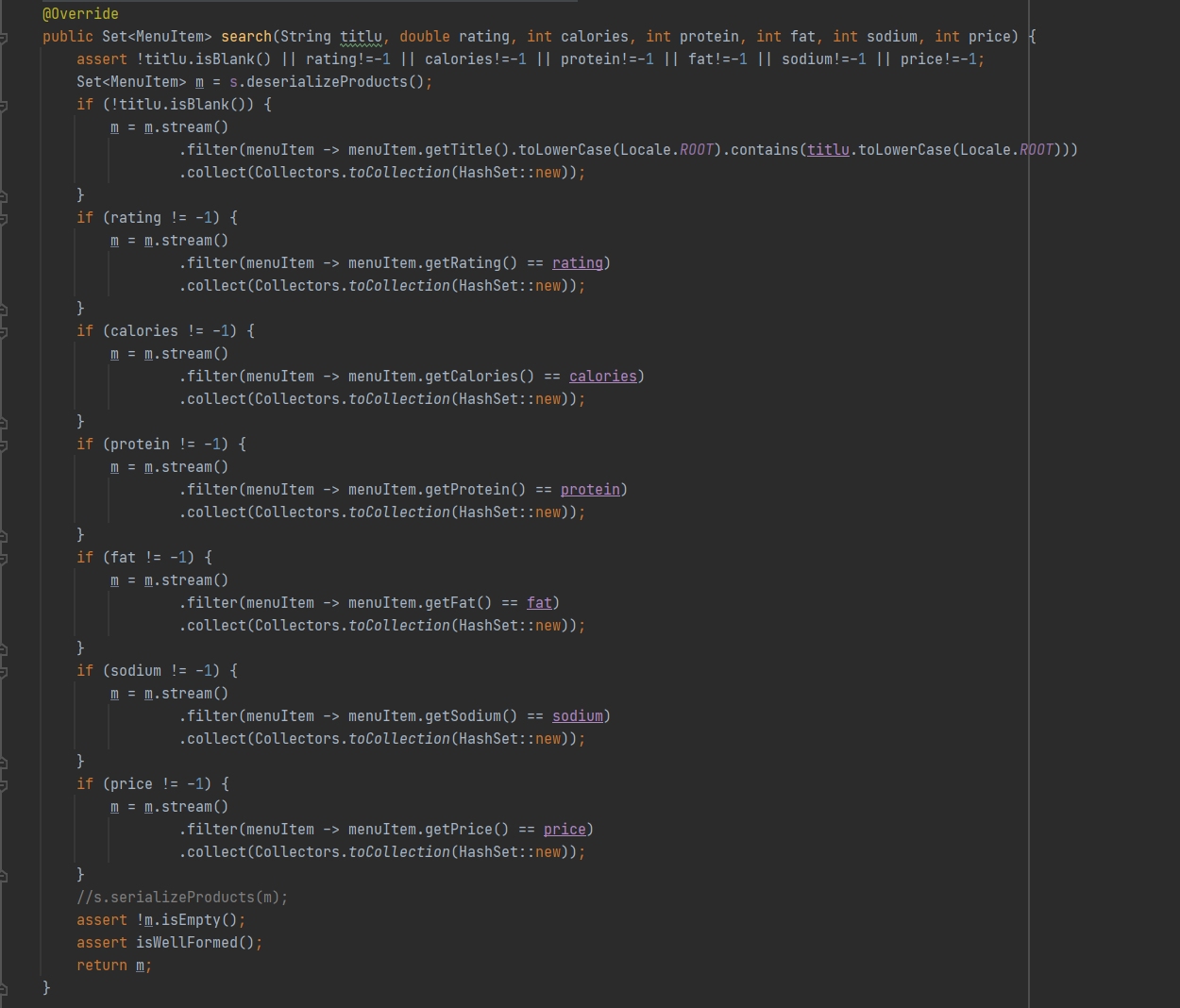


Metoda care afișează produsele care au fost comandate de mai mult de un număr specificat de ori. Pentru fiecare comandă din hashMap-ul de comenzi, adaug toate produsele comandate într-o listă mare de produse. Pentru fiecare produs din lista, îi calculez frecvența cu ajutorul metodei frequency() din clasa Collections. Apoi parcurg vectorul de frecvențe și setez valoarea 0 pe pozițiile unde se găsește un duplicat pentru a rămâne numai frecvențe unice pentru produse unice cu valori diferite de 0 în vectorul de frecvențe. Apoi parcurg vectorul de frecvențe și dacă frecvența de la poziția curentă este mai mare sau egală cu numărul dat de ori, atunci afișez produsul corepunzător.

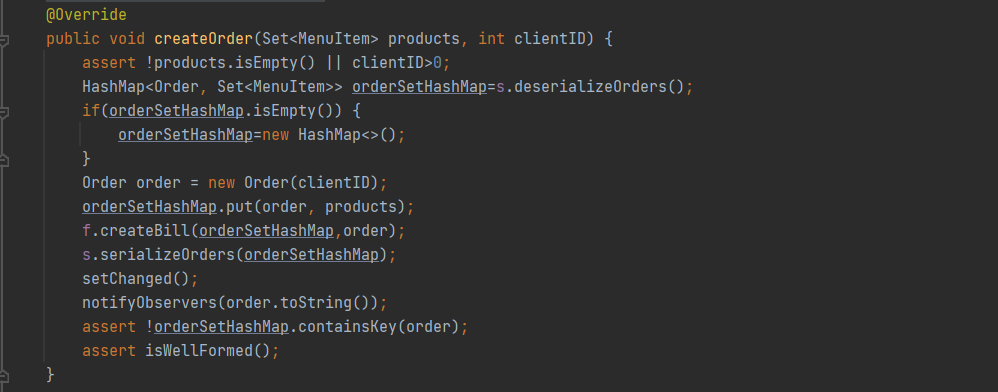
După acest mod de implementare am implementat și restul raporturilor.



Metoda care caută un anumit produs am implementat-o astfel încât să caute după orice parametru al unui produs din meniu. Dacă titlul nu este gol, atunci el caută produsele care conțin în titlu valoarea titlului primit ca parametru. Dacă restul parametrilor sunt diferiți de -1, atunci el caută după valoarea lor corespunzătoare primită.

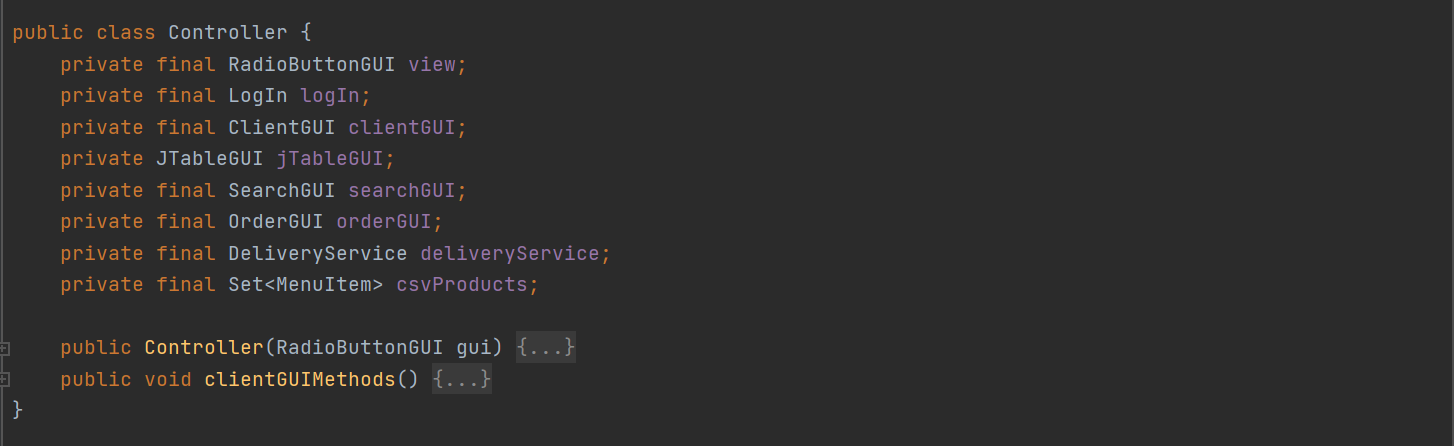


Metoda de plasare a unei comenzi primește ca parametru lista de produse care vor să fie comandate și ID-ul clientului care vrea să plaseze comanda. Se creează o comandă nouă și apoi se adaugă comanda în hashMap-ul de comenzi și apoi se generează factura .txt aferentă comenzii plasate.



## Clasa „Controller”:

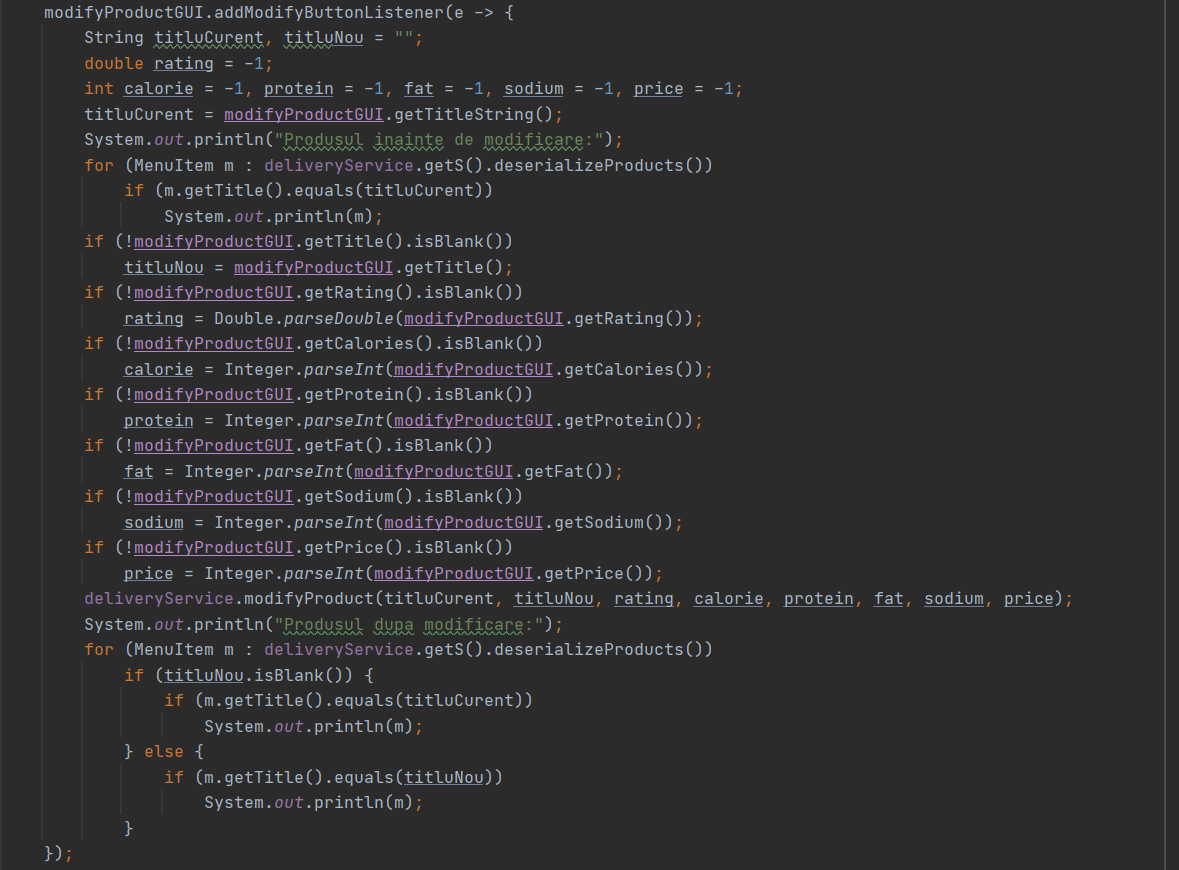
Clasa „Controller” împreună cu parametrii și metodele implementate. În constructor implementez metodele ascultătoare pentru interfețele și operațiile administratorului și angajatului și în metoda clientGUIMethods() implementez cele trei operații disponibile pentru clienți, atât după înregistrare cât și după logare.



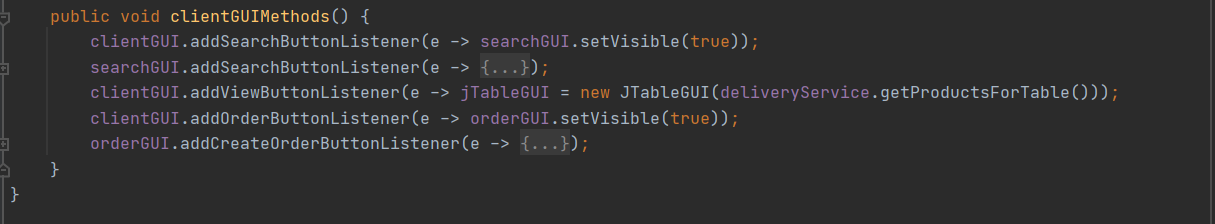
La începutul constructorului îmi creez obiecte noi pentru fiecare parametru declarat mai sus și îmi creez și obiecte pentru care nu a fost necesară declararea în lista de parametrii.



Ascultătorul pentru metoda de modificare l-am implementat astfel încât dacă câmpul din fereastra grafică este gol, atunci să rămână -1 variabila aferentă, altfel să se schimbe cu valoarea introdusă. Produsul curent și produsul modificat vor fi afișate în consolă pentru a se observa modificările.



În metoda pentru metodele clientului am implementat metodele ascultătoare pentru fiecare operație pe care poate să o facă clientul. Am pus cele trei metode într-o metodă separată și nu în consolă pentru că aceste metode trebuie să funcționeze și după ce clientul se loghează și după ce un nou client s-a înregistrat.



# Rezultate (testare)

Toate rezultatele asociate cu această aplicație sunt date de ferestrele grafice, tabelele care apar în interfață și fișierele text generate.

# Concluzii

În urma rezolvării acestei teme, pot spune că mi-am însușit mai bine cunoștințele din semestrul întâi despre Java și am acumulat și informații noi, cum ar fi utilizarea expresiilor lambda și a procesarea cu stream-uri, diverse design pattern-uri și JavaDoc. Dezvoltarea treptată a aplicației m-a ajutat să vin cu idei cât mai creative și eficiente pentru fiecare problemă care apărea. Ca și dezvoltări ulterioare, aș mai putea finisa anumite metode implementate și poate aș face și o interfață mai plăcută vizual, nu doar minimalistă.

# Bibliografie

Nu m-am ajutat de nicio sursă de inspirație pentru redactarea documentație. Tot ce am scris aici sunt ideile care mi-au venit pe moment. Nu pot garanta că în aceste idei nu se găsesc rezumate a altor idei citite pe diverse site-uri de domeniu sau în diverse cărți.