MINISTERUL EDUCAȚIEI NAȚIONALE

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

5/13/2021

Secția Calculatoare și Tehnologia Informației

TEMA 4
FOOD DELIVERY MANAGEMENT SYSTEM

Paula Sirghi
GRUPA 30223

Table of Contents

1. Obiectivul temei

Obiectivul acestei teme este reprezentat de implementarea unei aplicații ce are ca scop implementarea unui sistem de management al transportului mancarii pentru o companie de catering. Sistemul are 3 tipuri de utilizatori cu diferite funcționalități:

Clientul poate comanda produse din meniu, poate căuta produse ce respectă un anumit criteriu cum ar fi un cuvant cheie în numele produsului, se poate înregistra astfel încât să se poată loga în viitor și poate vedea toate produsele din meniu.

Administratorul poate importa lista inițială de produse ce vor alcătui meniul, poate modifica produsele și poate genera patru tipuri de reporturi (time interval of the orders, the products ordered more than a specific number of times, the client that have ordered more than a specific number of times and the value of the order is higher than a given number, the products ordered within specific day with the number of times they have been ordered).

Modul de citire al datelor introduse se face simplu, urmând ca textul introdus să fie convertit conform unui șablon în concordanță cu cerințele temei, iar rezultatul va fi scris în fișierul ce stochează datele aplicației și în obiectul corespunzător, iar în cazul generării comenzilor, se va face o factură pentru fiecare comandă care va fi scrisă într-un fișier.

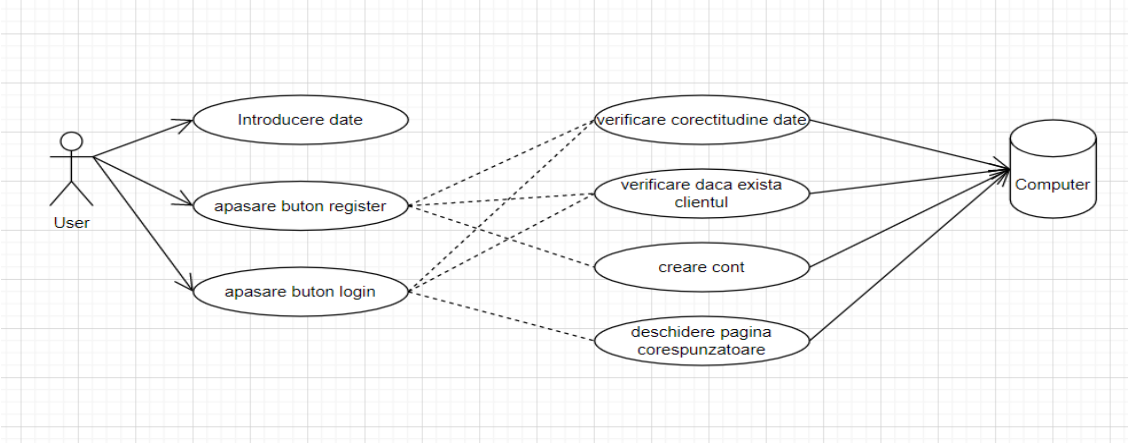
2. Analiza problemei, modelare, scenarii, cazuri de utilizare

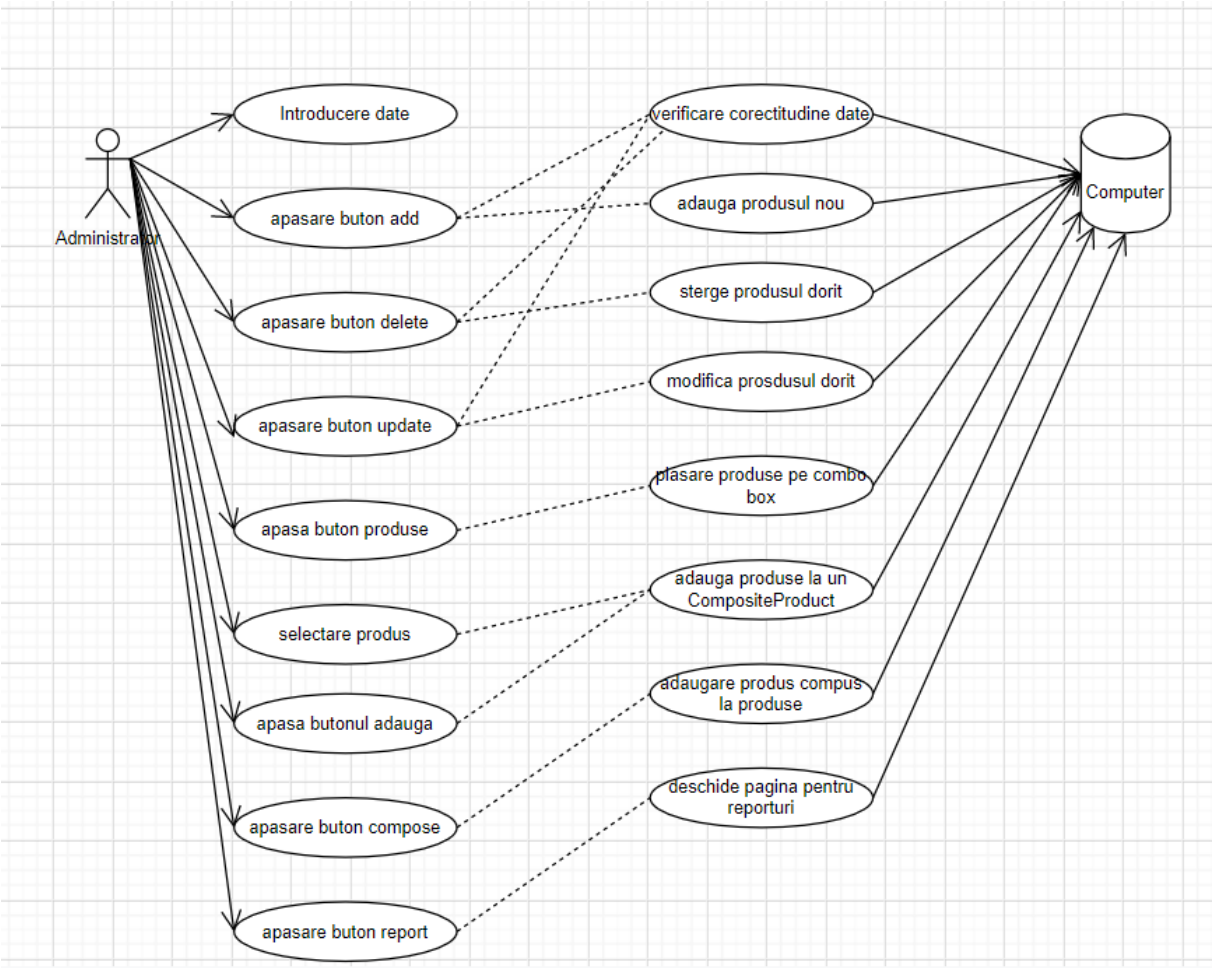
Analiza problemei constă, inițial, în înțelegerea cerinței problemei și implicit implementarea unui plan incipient pentru rezolvarea sa care, după cum am menționat mai sus, necesită implementarea operațiilor cerinței, operații care diferă în funcție de utilizatorul curent al aplicației. Ca prim pas, vom alege clasele necesare (substantive) și metodele reprezentative pentru fiecare (verbe) pentru a putea realiza o aplicație funcțională și ușor de înțeles pentru orice cunoscător de java.

O altă etapă reprezentativă constă în stabilirea intrărilor și ieșirilor aplicației noastre. În cazul de față intrările depind de interfața grafică în care ne aflăm. Dacă suntem în interfața incipientă avem 3 intrări de tip String (nume, parola, tip utilizator), pentru client avem o intrare, un String ce reprezintă criteriul de căutarte, în interfața pentru administrator avem 8 intrări reprezentative pentru operațiile de inserare, modificare și ștergere, iar în interfața pentru angajat avem o intrare care reprezintă notificarea primită la momentul plasării unei comenzi.

Am luat în calcul eventualele erori de introducere a unor date care nu sunt în conformitate cu intrările dorite, iar ca urmare utilizatorul va primi un mesaj de eroare în acest caz.

Use case-urile de mai jos evidențiază pașii ce sunt parcurși atât de utilizator, cât si de computer pentru a se obține rezultatul așteptat în momentul execuției programului.



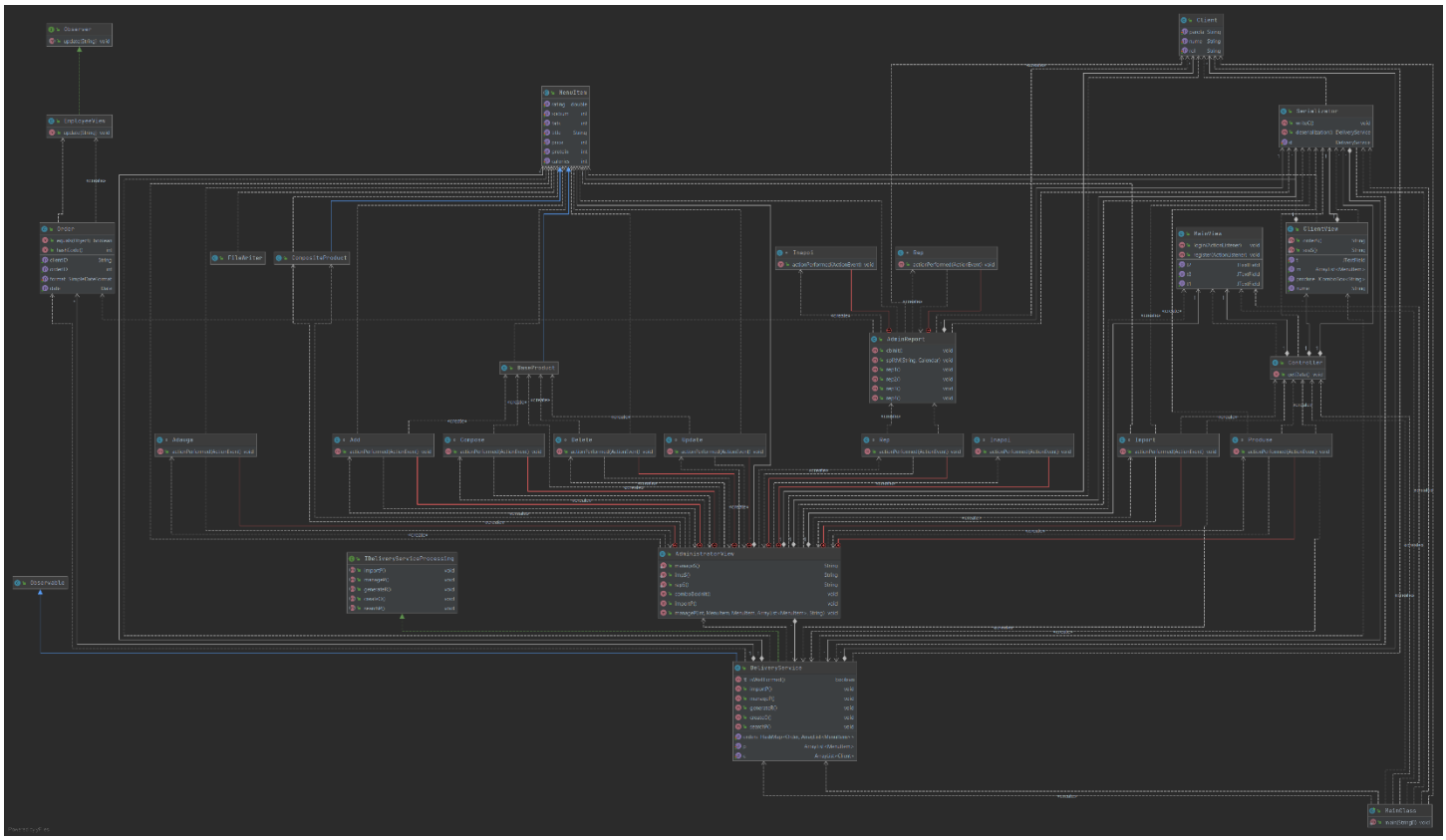


3. Proiectare

Pentru reprezentarea temei am ales o arhitectură structurată. Pachetul businessLayer conține clasele BaseProduct, Client, CompositeProduct, DeliveryService, MenuItem, Observable și

Order și interfața IdeliveryServiceProcess. Pachetul dataLayer conține clasele FileWriter și Serializator. Pachetul presentationLayer conție clasele AdministratorView, AdminReport, ClientView, Controller, EmployeeView, MainView și interfața Observer. Pachetul start montine clasa MainClass.

Diagrama UML este reperezentată mai jos, iar în ea putem observa clasele proiectului cu legăturile specificate, dar și metodele și structurile de date folosite.



4. Implementare

Descrierea claselor se va face pe baza diagramei UML de clase reprezentată mai sus.

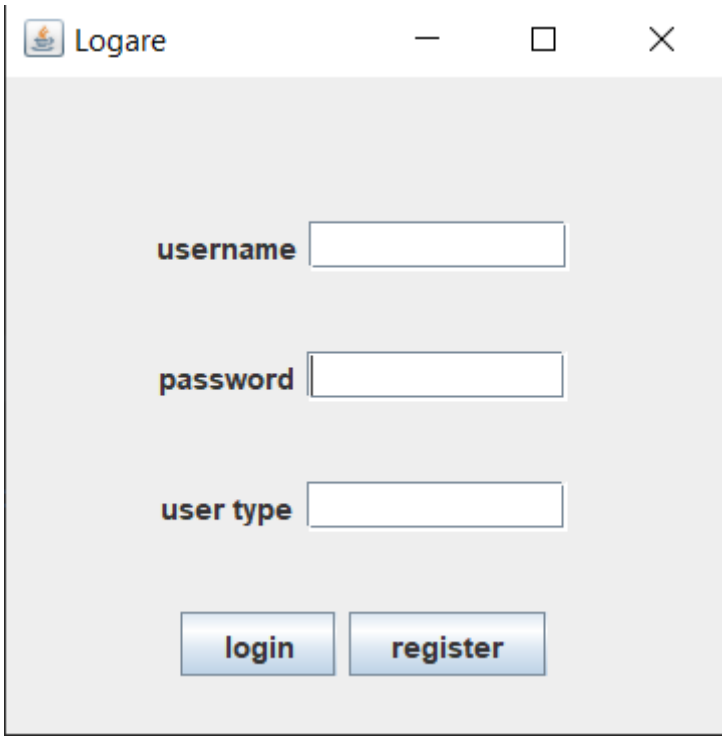
- Clasa MainClass

Această clasă reprezintă clasa de bază a proiectului, clasa de unde se începe simularea, fiind clasa ce conține metoda statica main in care este instanțierea unui obiect MainView și a unui obiect de tip Serializator care este folosit la scrierea în fișier în cazul unor actualizări necesare. Tot aici avem și instanțierea clasei Controller ce va realiza operațiile specifice unui client.

- Clasa MainView

Reprezentativă pentru GUI-ul inițial în care utilizatorul introduce fie un nume și o parolă pentru a se autentifica, fie un nume, o parolă și un rol pentru a-și crea un cont nou.

Aceasta conține 2 Jbuttons, fapt care poate fi observat în poza de mai jos a interfaței grafice.



Constructorul clasei realizează interfața grafică pe care o vede utilizatorul în momentul rulării programului, precum se vede în poza de mai sus.

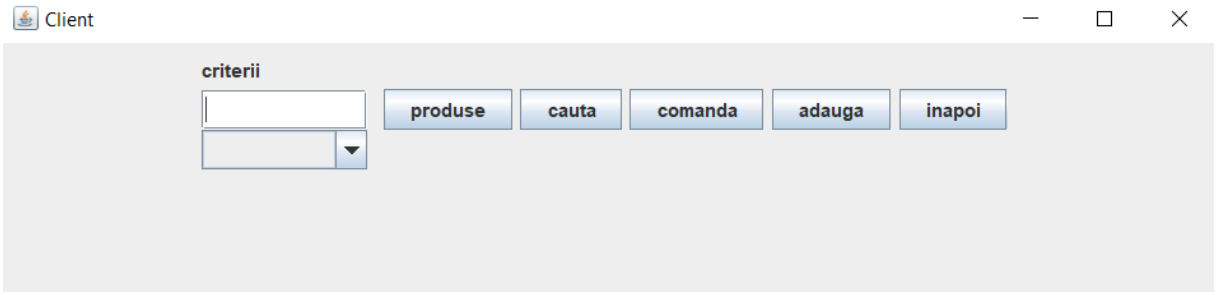
Metodele login și register sunt folosite pentru ca cele două butoane să implementeze ActionListener-uI, fiind folosite în controller și detaliate mai bine.

Pe lângă metodele menționate mai sus mai avem și gettere și settere necesare datorită folosirii tehnicii încapsulării datelor.

- Clasa ClientView

Reprezentativă pentru GUI-ul clientului în care utilizatorul introduce date pentru realizarea operațiilor reprezentative pentru un obiect de tipul client.

Aceasta conține 1 JtextField, 5 JButtons și 1 JComboBox, fapt care poate fi observat în poza de mai jos a interfaței grafice în care utilizatorul introduce noi date pentru o nouă execuție uneia din operațiilor de căutare și plasare comandă.



În constructorul clasei se realizează design-ul interfeței conform modelului de mai sus.

Metodele products, search, comanda, add și Înapoi sunt folosite pentru a adăuga ActionListeners butoanelor interfeței, funcțiile lor fiind stabilite în clasa Controller.

Metoda createO este reprezentativă pentru crarea uei comenzi și implicit a fișierului reprezentativ pentru aceasta. Noua comandă va fi adăugată HashMap-ului reprezentativ pentru aceasta.

Metodele cuv, rating, calories, protein, fat, sodium, price verifică dacă există un anumit câmp în lista de produse, iar dacă există îl adaugă unei liste.

Metoda cazuri se folosește de metodele de mai sus pentru a căuta un anumit produs în lista de produse în funcție de criteriile introduse de client.

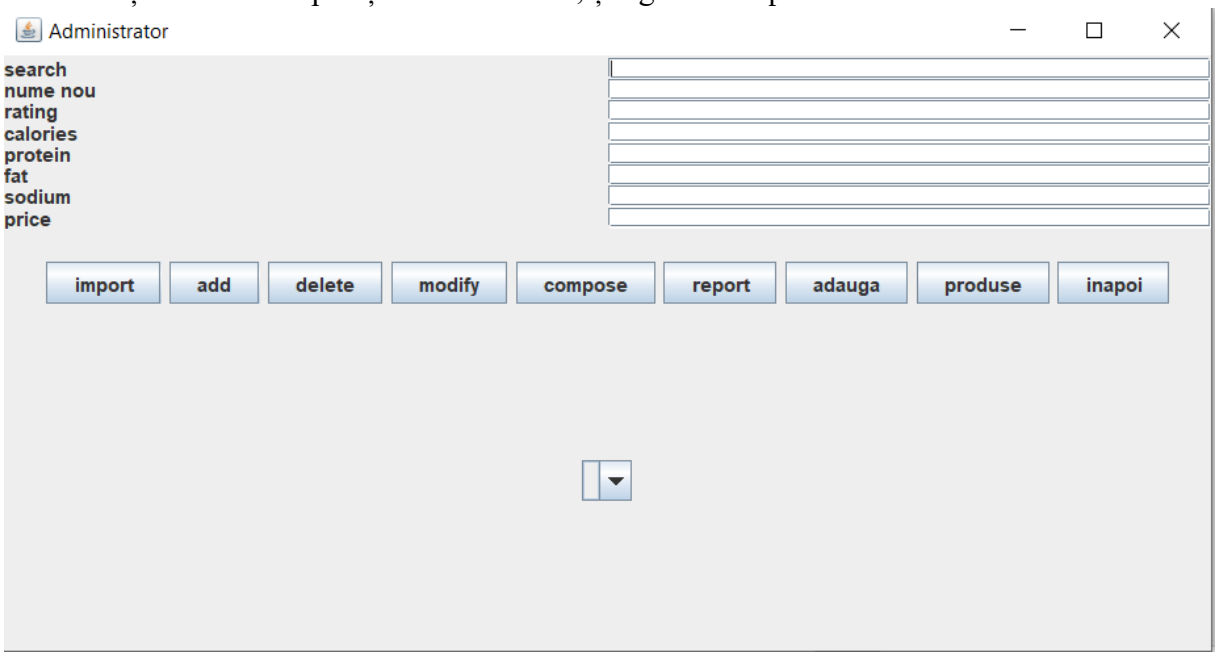
Metoda searchP separă șirul de criterii de căutare cu ajutorul expresiilor lambda și a stram-ului, urmând sa apeleze metoda de mai sus.

Alte metode ale acestei clase sunt getterele getNume, getT, getProduse, getM și settere-le setT, setProduse și setM, metode necesare datorită folosirii încapusulării datelor.

- Clasa AdministratorView

Reprezentativă pentru GUI-ul pentru administrator în care utilizatorul introduce date pentru realizarea operațiilor reprezentative pentru un obiect de tipul produs.

Aceasta conține 8 JtextField-uri, 8 Jlabel-uri, 9 Jbuttons și un JCheckBox, fapt care poate fi observat în poza de mai jos a interfației grafice în care utilizatorul introduce noi date pentru o nouă execuție uneia din operațiilor de inserare, ștergere sau update.



În constructorul clasei se realizează design-ul interfeței grafice pentru administrator conform modelului prezentat mai sus.

Metoda importP este folosită pentru a realiza preluarea produselor din fisierul dat, aceste produse fiind scrise in file.txt si adaugate clasei DeliveryService.

Metoda manageP este folosită pentru a implemena funcționalitățile administratorului. În primul caz se adaugă un produs listei de produse, în al doilea caz se șterge un produs din listă, în al treilea caz se modifică un produs din listă, iar în al patrulea caz se formează un produs compus din mai multe produse de bază.

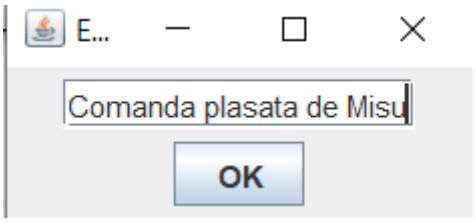
Metoda generateR este folosită pentru generarea de rapoarte.

Tot aici avem și clasele interne Udate, Compose, Inapoi, Delete, Add, Adauga, Import, Produse și Rep folosite pentru butoanele prezente astfel încât la apîsarea lor să se execute task-ul dorit.

- Clasa EmployeeView

Reprezentativă pentru GUI-ul pentru angajat. Aceasta va apărea la fiecare plasare de comandă cu mesajul *Comandă plasată de nume_client*.

Aceasta conține 1 JtextField și 1 Jbutton, fapt care poate fi observat în poza de mai jos a interfaței grafice.



În constructorul clasei se realizează design-ul interfeței acestui tip de utilizator al aplicației conform modelulului de mai sus.

Clasa ButonL este folosită pentru a ieși din acest frame în momentul apăsării butonului.

Clasa implementează interfața Observer, prin urmare avem metoda update folosită pentru a atenționa angajatul de fiecare dată când se plasează o comandă nouă.

- Clasa Controller

Această clasă realizează operațiile specifice clientului.

Metoda getData este folosită pentru a prelua datele introduse de utilizator în pagina rincipală a proiectului.

Clasa LoginListener surascrie metoda actionPerformed și caută în DeliveryService clientul preluat din interfața de start a programului, iar dacă acesta este găsit se va deschide fie interfața pentru administrator, fie cea pentru client în funcție de rolul ocupat de utilizator.

Clasa AddListener este folosită pentru a parcurge lista de produse și a căuta un produs cu un nume specificat pentru a-l adăuga în coșul de cumpărături al clientului.

Clasa ComandaListener este folosită pentru a plasa o comandă cu produsele din coșul menționat mai sus.

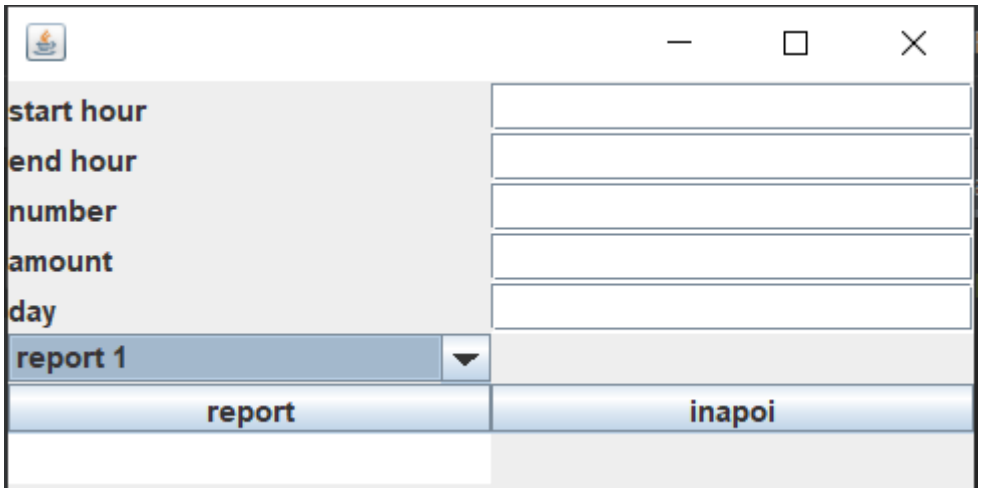
Clasa ViewProducts este folosită pentru a genera un tabel cu produsele curente ale aplicației.

Clasa RegisterListener este folosită pentru a adăuga un nou utilizator în lista de utilizatori ai programului.

- Clasa AdminReport

Această clasă este reprezentativă pentru interfața grafică pentru administrator referitoare la rapoarte.

Aceasta conține 5 JtextField-uri, 5 Jlabels, un JcheckBox, 2 Jbuttons și un JtextField.



Metoda rep apeleaza metodele rep1, 2, 3, 4 în funcție de opțiunea selectată din JcomboBox. Aceste 4 metode generează raporturi pe care le scriu in josul paginii. Raporturile sunt generate folosind Spring si expresii lambda.

- Interfața Observer

Această interfață este reprezentativă pentru Observer Design Pattern. După cum am menționat și mai sus, rolul ei este de a atenționa angajatul de fiecare dată când este plasată o comandă.

- Clasa FileWriter

Această clasă este reprezentativă pentru scrierea în fișierul reprezentativ unei comenzi.

Constructorul său realizează creearea sau suprascrierea unui fișier cu numele clientului ce plasează o comandă. Fișierul va conține toate produsele comandate, prețul total al comenzii și data și ora la care a fost plasată aceasta.

- Clasa Serializator

Această clasă este reprezentativă pentru serializare și deserializare.

Constructorul intanțiază obiectul, atribuindu-i datele corespunzătoare.

Metoda writeC este folosită pentru a suprascrie conținutul fișierului la fiecare modificare a acestuia în cod. Această metodă este reprezentativă pentru serializare.

Metoda deserialization este reprezentativă, după cum îi spune și numele, deserializării. Aceasta realizează transformarea conținutului fișierului într-un obiect de tipul DeliveryService, care va fi folosit mai departe de aplicație.

- ClasaBaseProduct

Această clasă extinde superclasa MenuItem și instanțiază obiectele de tipul BaseProduct.

- Clasa Client

Această clasă este reprezentativă, după cum îi spune și numele, pentru utilizatorul de tipul client.

Constructorul este folosit pentru a instanția un obiect de acest tip.

Setter-ele setPaula, setRol, set Nume și gettere-le getNume, getParola, getRol sunt necesare datorită folosirii tehnicii încapsulării datelor.

- Clasa CompositeProduct

Această clasă extinde superclasa MenuItem și instanțiază obiectele de acest tip.
Un obiect CompositeProduct este constituit din mai multe obiecte BaseProduct.

- Clasa DeliveryService

Această clasa implementează IdeliveryServiceProcessing și Serializable.

Metoda importP, manageP, generateR, createO si searchP sunt folosite pentru javaDoc, verificand preconditiile si postconditiile si apeland metoda isWellFormed. Putem observa aici comentariile specifice pentru javaDoc cu @pre, @post si @isWellFormed. Tot aici am folosit si assert-urile.

Pe lângă aceste metode mai sunt prezente și settere și gettere necesare datorită folosirii tehnicii încapsulării datelor, dar și constructorul care instanțiază un obiect de acest tip.

- Clasa MenuItem

Această clasă este superclasa obiectelor reprezentative pentru produse.

Meotdele sunt settere si gettere necesare datorită folosirii tehnicii încapsulării datelor, dar și un constructor care instanțiază un obiect de tip produs.

- Clasa Order

Această clasă este reprezentativă pentru o comandă și conține constructorul care instanțiază un obiect de acest tip și apelează metoda update a angajatului pentru a-l informa în legătură cu noua comandă plasată.

Tot aici am suprascris metoda equals.

În plus, pot fi observate și aici settere și gettere necesare datorită folosirii tehnicii încapsulării datelor.

- Interfața IdeliveryServiceProcess

Interfață reprezentativă pentru operațiile necesare, aceasta fiind implementată de clasa DeliveryService care suprascrie metodele și le oferă funcționalitățile menționate mai sus, la această clasă.

5. Rezultate

Rezultatele operațiilor au fost fie scrise în câmpuri specifice din interfețele grafice, fie în fișierul principal, fie în fișierele generate pentru fiecare factură, aceasta fiind denumit cu numele clientului ce a plasat comanda și conținând numele produselor comandate, prețul total al comenzii și data și ora la care s-a realizat comanda.

6. Concluzii

Din punctul meu de vedere, această tema a fost destul de interesantă și diferită față de celelalte trei, de data aceasta fiind nevoiți să stocăm informațiile într-un fișier. În ciuda acestui fapt a fost o temă frumoasă deși partea cu expresiile lambda a fost o noutate pentru mulți dintre studenți, nemaifolosind-o până acum.

7. Bibilografie

Resursele bibliografice sunt reprezentate de:

- suportul de laborator;
- suportul de curs de semestrul trecut de la materia Programare Orientată pe Obiect;
- link-urile din suportul de laborator