Documentație tema 4.
Simularea unui serviciu de catering.

1. Obiectivul temei

Obiectivul temei este simularea unui serviciu informatic de catering, prin care pe baza unui cont te poti conecta, atat ca administrator cat si ca client si angajat, fiecare cu propriile actiuni deservite de aplicație. De la gestiunea aplicația, produsele existente, creare de noi produse, de produse compuse din mai multe produse simple, modificarea produselor existente și inclusiv generarea de rapoarte bazate pe activitatea clientilor de catre administrator, la fereastra angajatilor care sunt notificati de fiecare dată când apare o noua comandă, până la clientul care de asemenea se poate conecta, caută produse dupe diferite criterii si plasa comenzi. Aplicația având sistem implementat de gestiunea comezilor, comenzi persistente pe sistem intre executii.

2. Analiza problemei

În primul rând trebuie un mecanism de interacțiuni între utilizator și aplicație, se pune la dispoziție o interfață grafică prin care utilizatorul poate facil interacționa cu aplicația. Interfața grafică trebuie să fie diferită în funcție de de tipul utilizatorul, așadar este împărțita în 3 subinterfețe fiecare asociată unui tip de utilizator.

Așadar și use-case-urile se vor împarți în 3 tipuri diferite în funcție de tipul utilizatorului. Doar un use-case este comun, alegerea ferestrei și autentificarea.

Use case 1: Autentificarea

Actorul principal: Administratorul/Clientul

Principalul scenariu de succes:

1. Utilizatorul introduce datele contului
2. Date introduse sunt corecte și se regăsesc în sistem
3. Utilizatorul reușeste să se conecteze și apare fereastra aleasă

Scenariu alternativ:

1. Utilizatorul introduce un cont care nu este în sistem
2. Se afișează o căsuță în care este specificat acest lucru și i se oferă șansa să introducă din nou datele contului

Use case pentru administrator:

Use case 2: Adaugarea unui produs

Actorul principal: Administratorul

Principalul scenariu de succes:

1. Administratorul introduce în interfața grafică datele noului produs.
2. Administratorul apasă pe buton de adaugare produs.
3. Se citesc datele introduse și se construiește un nou obiect.
4. Se scrie noul obiect în sistem.

Scenariu alternativ:

1. Datele introduse nu sunt corecte.
2. Se afisează o căsuța text cu incovientul.

Use case 3: Modificarea unui produs

Actorul principal: Administratorul

Principalul scenariu de succes:

1. Se aleg un numar aleatoriu de produse din meniu.
2. Se introduc datele care se vor a fi modificate.
3. Se citesc date și se realizează modificările aferente tuturor produselor selecteate.

Scenariu alternativ:

1. Datele introduse nu sunt corecte.
2. Se afisează o căsuța text cu incovientul.

Use case 4: Stergerea unui produs

Actorul principal: Administratorul

Principalul scenariu de succes:

1. Se aleg un numar aleatoriu de produse din meniu.
2. Se apasă pe butonul de ștergere
3. Se șterg produsele din sistem

Scenariu alternativ:

1. Datele introduse nu sunt corecte.
2. Se afisează o căsuța text cu incovientul.

Use case 5: Crearea unui produs compus

Actorul principal: Administratorul

Principalul scenariu de succes:

1. Se aleg un numar aleatoriu de produse din meniu.
2. Se apasă pe butonul creare produs compus
3. Se adauga produsul compus din produsele selectate din sistem

Scenariu alternativ:

1. Datele introduse nu sunt corecte.
2. Se afisează o căsuța text cu incovientul.

Use case 6: Importarea meniului dintr-un fisier csv

Actorul principal: Administratorul

Principalul scenariu de succes:

1. Se apasă pe butonul de importare
2. Se adauga toate produsele din fisier care nu sunt duplicate in noul meniu

Scenariu alternativ:

1. Fisierul nu există
2. Se aruncă o excepție

Use case 7: Generarea de rapoarte

Actorul principal: Administratorul

Principalul scenariu de succes:

1. Se apasă pe butonul de generare rapoaterte
2. Apare o fereastră în care sunt căsute în care se introduc criteriile pentru care se doresc generarea de rapoate
3. Se apasă pe buton corespunzător raportului care se dorește a fi generat
4. Se generează raportul și se scrie într-un fișier pe sistem.

Scenariu alternativ:

1. Datele introduse nu sunt corecte.
2. Se afisează o căsuța text cu incovientul.

Use case pentru angajat:

Use case 8: Angajatul primește o noua comandă

Actorul principal: Angajatul

Principalul scenariu de succes:

1. Angajatul are fereastra respectivă angajaților deschisă
2. În momentul în care un client oarecare plasează o comandă angajatul este notificat în interfața grafică cu date despre comandă și produsele din aceasta.

Use case-uri pentru client:

Use case 9: Cautarea de produse

Actorul principal: Clientul

Principalul scenariu de succes:

1. Clientul introduce criteriile pe baza cărora dorește să caute produsele
2. Apasă pe butonul de căutare
3. Se afișează doar produsele care îndeplinesc criteriile selectate

Scenariu alternativ:

1. Datele introduse nu sunt corecte.
2. Se afisează o căsuța text cu incovientul.

Use case 10: Crearea unei comenzi

Actorul principal: Clientul

Principalul scenariu de succes:

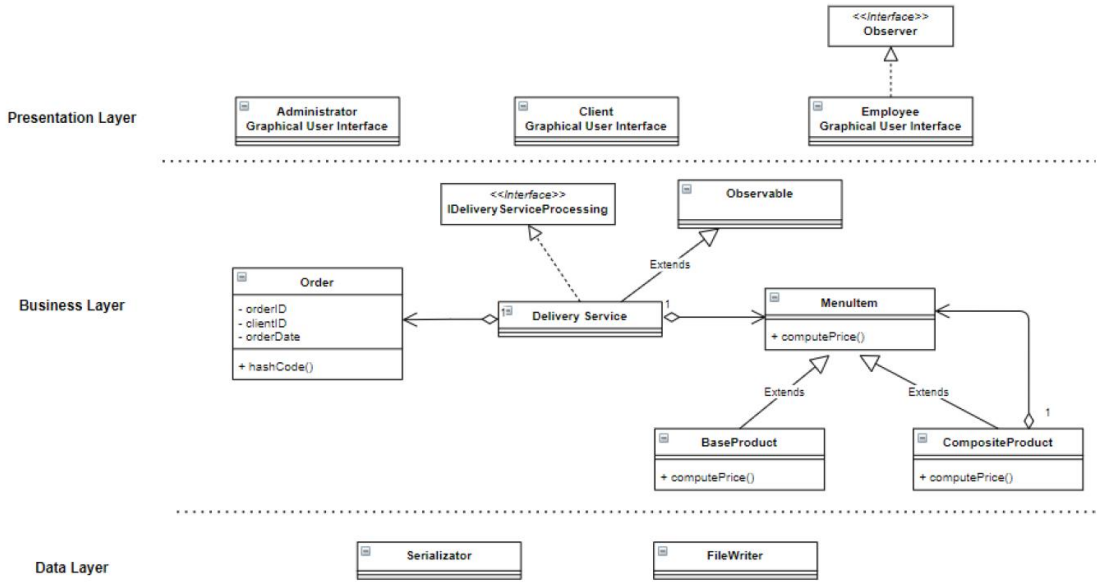
1. Clientul caută produsele dorite
2. Selectează produsele pe care dorește să le comande și specifică de asemenea cantiatea din fiecare produs
3. Apasă pe butonul de plasare a comenzii
4. Se plasează comandă si este notificat că comanda a avut loc.

Scenariu alternativ:

1. Cantiatea selectată nu este validă
2. Se afisează o căsuța text cu incovientul.

3. Proiectare

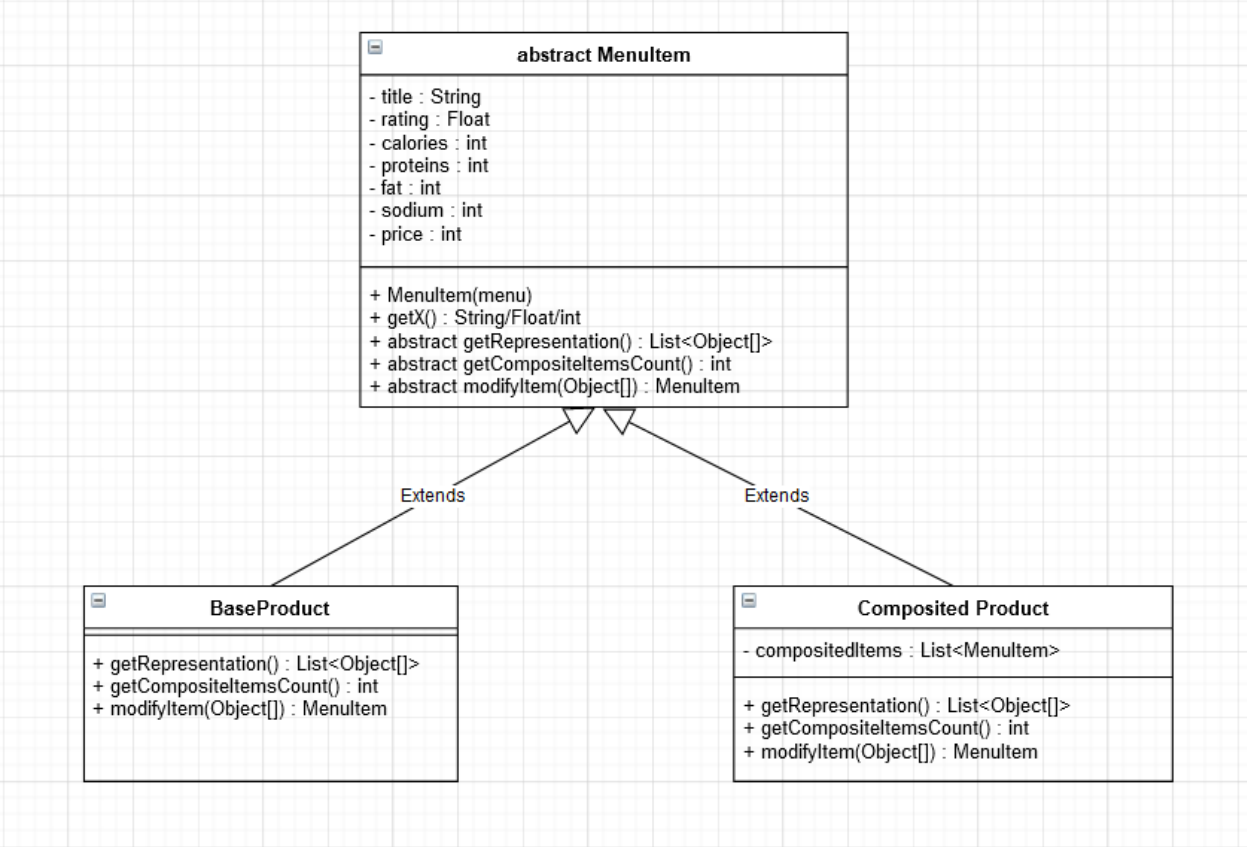
Aplicația se împarte în 3 pachete, pachetul **businessLayer, dataLayer** și **presentationLayer**. Pachetul **businessLayer** conține clasele care modelează logica aplicației, clasa **DeliveryService**, clasele prin care se reprezintă meniunirile, clasa **Order** cât și cateva clase ajutătoare. Pachetul **dataLayer** conține clasa **Serializator** și clasa **Writer,** clase care manipulează obiectele la nivelul sistemului de date, prin scrierea/citirea lor de pe disk.



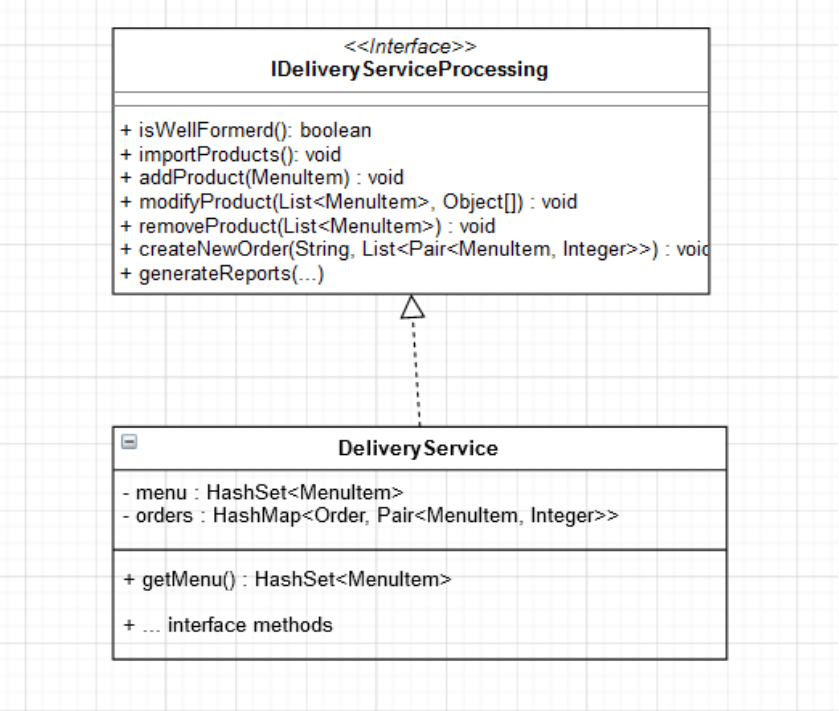
În continuarea se v-a prezenta clase principale din pachetul **businessLayer:**

Clasele **MenuItem, BaseProduct și CompositeProduct** implementează designul **Composited**, **MenuItem** reprezentând clasa abstractă care definește campurile fiecărui produs ca unitate, respectiv getteri și metode abstracte care sunt implementate de claseele **BaseProduct și CompositeProduct** pentru a manipula produsele în funcție de tipul acestora.

Clasele sunt immutabile pentru a facilita lucrul pe stream atât secvențiale cât și paralel. În consecință modificarea unui obiect implică înlocuirea lui cu o altă instanță cu field-urile modificate.



Clasa **DeliveryService** implementează interfața **IDeliveryServiceProcessing** care este de tipul contract.



De asemenea clasa **DeliveryService** trebuie să fie observabilă, implentând patternul Observer cu fereastra angajatorilor, care sunt notificați în momentul în care se plasează o nouă comandă. Patternul s-a implementat folosint **PropertyChangeListener** atât pentru meniul, în momentul în care se fac modificări să se reflecte și in fereastra administratorului cât și asupra comenzilor.

4. **Implementare**

- Se prezintă implementarea realizată in cadrul clasei **DeliveryService:**

- Fiecare metodă care modifică meniul după modificare serializează noul obiect DeliveryService pentru execuții ulterioare

Variabile:

- **menu -** S-a ales implementearea meniului cu un HashSet pentru a avea acces O(1) la elemente pe baza hash-urilor sale și ne având nevoie de a menține meniul într-o ordine.

- **orders –** S-a ales implementarea comenzilor cu un HashMap, care mapează fiecare comandă la o listă de perechi, fiecare pereche reprezintă MenuItem-ul și cantitatea comandată din acesta în comanda respectivă

Metoda: isWellFormed()

- Metoda verifica dacă se pastrează invariantul clasei

Metoda: importProducts()

- Metoda caută fișierul products.csv, il deschide și creează un **BufferedReader** asupra căruia generază un stream care mapează fiecare linie la o listă de String-uri, String-uri care reprezintă field-uri din produse, din fiecare lista se creează un nou obiect de tipul BaseProduct care este introdus în meniu. La sfârșit tot meniul este serializat într-un fișier

Metoda: addProduct(MenuItem)

- Metoda primește un obiect de tipul MenuItem pe care îl introduce în meniul și notifică fereastra administratorul ca s-a produs modificarea pentru a actualiza de asemenea datele.

Metoda: modifyProduct(List<MenuItem>, Object[])

- Metoda primește o listă de obiecte MenuItem care se doresc a fi modificate, și un array de tipul Object în care sunt field-uri care se doresc a fi modificare, penru fiecare obiect din listă se creează un nou obiect cu field-uri modificare care înlocuiește obiectul vechi in meniu și se notifică observatorii

Metoda: removeProduct(List<MenuItem>)

- Metoda primește o listă de obiecte MenuItem care se doresc a fi șterse, fiecare obiect este șters pe urmă și se notifică observatorii.

Metoda: createNewOrder(String, List<Pair<MenuItem, Integer>>)

- Metoda primește un string care reprezintă numele clientului care a plasat comanda și o listă de **Pair,** fiecare pereche reprezintă MenuItem-ul selectat și cantitatea. Metoda creează un nou obiect Order și mapează noul obiect in HashMap-ul orders cu lista de obiect comandate.

Metoda: generateReportOfOrdersInInterval(int startHour, int endHour)

- Se generează toate comenzile plasate între ora de inceput și ora de sfâșit indiferent de zi
- Pe stream-ul de key se realizează operația de filter pe ora comenzii și toate comenzile care îndeplinesc condiția sunt colectate într-o listă care pe urmă este strică în fișier.

Metoda: generateReportsOfProductsOrderedMoreThan(int threshold)

- Se generează toate produsele care s-au comandat mai mult de limita primită ca parametru
- Se realizează un HashMap în care se mapează pentru fiecare MenuItem comandat numărul de comenzi, realizând acest lucru printr-o colectare specifică

Metoda: generateReportsOfClient(int orderedThreshold, int valueThreshold)

- Se generează toți clienții care au comandat de mai multe ori o comandă decât pragul dat fiecare comandă cu o valoarea mai mare decât pragul dat
- Pentru fiecare comandă din stream-ul de comenzi se filtrează doar comenzile care au o valoarea mai mare ca pragul dat, și pe urmă rezultatul filtării se colectează într-un HashMap care mapează fiecare client la numarul de comenzi puse de această, structura care este filtrată si colectată într-o listă formată din numele clienților care indeplinesc cerinta.

Metoda: generateReportOfProductsOrderedInDay**(**LocalDate date**)**

- Metoda generează toate produsele comandate într-o zi împreuna cu numărul de câte ori au fost comandate în acea zi
- De asemenea se mapează într-un HashMap fiecare MenuItem comandat în aceea zi cu numărul de comezi efectuate

Clasa **ProductsPanel:**

- Această clasă facilitează afișarea produselor din meniul și realizează cautarea lor pe baza criteriilor introduse
- De asemenea implementează interfața PropertyChangeListener pentru a fi notificat în momentul în care se produce o modificare în obiectul DeliveryService pentru a reflecta schimbarea și in reprezentarea sa internă

Variabile:

- menuMap – reprezintă un HashMap care mapează fiecare MenuItem din meniu la un JFrame care reprezintă reprezentarea acestuia grafic. S-a implementat prin HashMap pentru a facilita de accesare în O(1) în momentul în care se filtrează meniul

Metode:

- filterItems(Object[] fields) – metoda realizează operația de filtrare în funcție de field-urile specificate, fiind competentă a filtra în funcție de orice combinație de criterii. Acest lucru realizându-se prin crearea unui predicat de filtrare pentru fiecare field nenull și aplicarea unui predicat compus din acestea asupra obiectelor din meniu.

5. Rezultate

S-au testat toate use-case-urile prezentate anterior si modul în care aplicația raspunde în cazul acestora. De asemenea pentru fiecare metoda din DeliveryService s-au testat condițiile pre si post stabilite în contract prin intermediul interfeței IdeliveryServiceProcessing și de asemenea testarea metodei isWellFormed() care testează la apelul fiecărei metode că obiectul îndeplinește invariantul.

Generarea rapoartelor de asemenea s-a testat cu mai multi canditați în rândul clienților și cu un număr relativ mare de comenzi, 40, în limita gestionării manuale pentru a testa daca rezultate produse de aplicație sunt în concordantă cu cerința.

6. Concluzii

Din această temă s-a învațat cum se pot gestiona structuri de date complexe folosind stream și modul în care lucrul cu stream facilitează de multe ori atât implementarea cât și timpul de execuție și avantajul oferit de aceastea prin posibilitatea paralelizării opeațiilor, paralelizare realiazată de compilator, care poate mai ales pentru un set mare de date să realieză un câștig semnificativ de performantă pentru aplicația dezvoltată.

De asemenea s-a invățat modalitatea prin care se pot salva date pe sistem prin intermediul serializării se refolosi aceste date la o execuție anterioară, de asemenea datele sunt salvate într-un fisier binar necitibil de un utilizator simplu așadar oferă și o mică securitate față de un utilizator care ar dori să modificarea fișierului din afara aplicației.

Pentru o dezvoltare ulterioară s-ar putea îmbunătății interfețele grafice atât vizual cât și la nivel de funcționalitate. De asemenea o interfața mai utilă pentru utilizatorii de tip angajat, o interfață în care angajatul poate semnala că a finalizat comanda. Conectarea sistemului de plasare a comenzilor la un driver de imprimantă care poate tipări fiecare comandă pentru angajatori pentru ai informa ce trebuie să pregatească.

7. Bibliografie

https://www.baeldung.com/java-observer-pattern

https://docs.oracle.com/javase/7/docs/api/java

https://docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html#filter-java.util.function.Predicate-

https://www.baeldung.com/java-predicate-chain

https://docs.oracle.com/javase/tutorial/jndi/objects/serial.html

https://stackoverflow.com