MINISTERUL EDUCAŢIEI NAŢIONALE
UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

ASSIGNMENT 3
Order management

Ilovan Bianca-Maria
Grupa 302210

Cuprins:

1.Obiectivul temei

Obiectivul principal al temei era realizarea unei aplicatii de order management, pentru procesarea unor clienti, a unor produse si a unor comenzi(ale clientilor). Pentru stocarea acestor "elemente" aveam nevoie in prima faza de o baza de date relationala ce contine clientii, produsele din stoc, cat si comenzile plasate de clienti.

In ceea ce priveste alte **obiective secundare** ale temei, aici ne putem referi la citirea comenzilor dintr-un fisier dat ca argument si parsarea informatiilor continute de acesta, realizarea cu succes a comenzilor intalnite in fisier, salvarea datelor in database, iar mai pe urma generarea unor PDF-uri ce sa corespunda cu datele din tabel la momentul actual. De asemenea, organizarea aplicatiei in cele 4 pachete: model classes, business logic classes, presentation classes si data access classes, poate fi considerat un obiectiv destul de important.

2.Analiza problemei, modelare, scenarii, use case-uri

2.1.Analiza problemei

Dupa cum rezulta din analiza cerintei assignment-ului, aplicatia s-ar concretiza prin implementarea unei baze de date care simuleaza si proceseaza comenzile clientilor pentru un depozit. Pentru a stoca produsele, clientii si comenzile se vor folosi baze de date relationale. Aceste comenzi sunt regasite in fisierul de intrare care va fi dat ca argument in linia de comanda. Astfel, principalele operatii pe care aplicatia ar trebui sa le faca posibile sunt:

- inserarea in baza de date a unui client si a unui produs
- generarea unui PDF ce contine fie clientii, produsele sau comenzile actuale ale bazei de date
- stergerea clientilor si a produselor dupa nume
- generarea unei facturi in urma plasarii comenzii

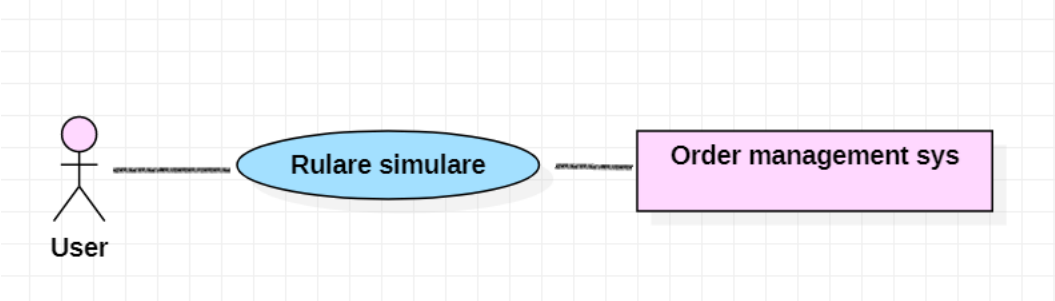
2.2.Solutie aleasa pentru modelarea problemei

Avand in vedere cerintele, am considerat potrivit ca baza mea de date **OrderManagement** sa contina 4 tabele: **client**, **product**, **order** si **orderitem**. **Fiecare client** are un id unic, care este primary key si se va incrementa automat, un nume si o adresa. **In cazul produsului**, asemanator, cu exceptia faptului ca nu va mai fi vorba de o adresa, ci de un pret al produsului, cat si de cantitate. Pentru a efetua **legaturile logice dintre clienti si produse**, am avut nevoie de celelate 2 tabele si anume: **order** care contine un id unic, id-ul clientului, numele clientului si totalul si **orderitem**, ce contine un id unic, id-ul produsului si cantitatea ce doreste a fi cumparata.

2.3.Use case-uri

Functionalitatea aplicatiei e descrisa de diferite cazuri de utilizare ale acesteia, in cazul “depozitului de produse” fiind vorba despre plasarea comenzilor clientilor, iar mai pe urma, de actualizare a stocului produselor, in functie de comenzi. Pentru a simula aceasta aplicatie, toate datele de intrare sunt intalnite in fisierul de intrare. Ce sta la latitudinea utilizatorului este in a-si alege fisierul de intrare cu care va rula aplicatia. Astfel, in functie de aceste date(intalnite in fisierul de intrare), aplicatia va simula ce se va intampla in momentul in care se executa comenzile intalnite in acel fisier. Pentru a rula aplicatia este necesara introducerea urmatoarei comenzi:

java -jar <the_absolute_path.jar> textFile.txt.txt



User-ul trebuie sa ruleze practic aplicatia, dand ca argument fisierul de intrare(in Eclipse). Pentru a rula din linia de comanda, se foloseste comanda precizata mai sus. In functie de alegerea fisierului de intrare, pe care utilizatorul are dreptul sa-l aleaga, aplicatia incearca sa dea raspunsul dorit si totul sa mearga conform “scenariului”.

2.4.Scenarii

Use case: se rezuleaza fisierul .jar
Primary actor: user
Scenarii de succes:

Pentru a utiliza aplicatie sunt necesare urmatoarele:

1. Crearea unui fisier text, ce va fi dat ca input, in care se introduc practic anumite comenzi: inserari de clienti si produse, stergeri de clienti si produse, generari de PDF-uri. Un model de fisier de intrare este:

```
Insert client: Ion Popescu, Bucuresti
Insert client: Luca George, Bucuresti
Report client
Insert client: Sandu Vasile, Cluj-Napoca
Report client
Delete client: Ion Popescu, Bucuresti
Report client
Insert product: apple, 1, 20
Insert product: peach, 2, 50
Insert product: apple, 1, 20
Report product
Delete Product: peach
Insert product: orange, 1.5, 40
Insert product: lemon, 2, 70
Report product
Order: Luca George, apple, 5
Order: Luca George, lemon, 5
Order: Sandu Vasile, apple, 100
Report client
Report order
Report product
```

2. Se ruleaza aplicatia, dandu-se ca argument fisierul de intrare
3. In functie de ce comenzi sunt intalnite in fisierul de intrare aplicatia va face modificarile necesare in baza de date.

a) insert client: se va insera clientul precizat in baza de date
b) delete client: va sterge clientul precizat din baza de date
c) report client: va genera un PDF cu clientii din baza de date la acel moment
d) insert product: se va insera produsul precizat in baza de date
e) delete product: se va stereg produsul din baza de date
f) report product: va genera un PDF cu produsele din baza de date la acel moment
g) order: se va plasa o comanda pentru clientul cu numele precizat, cu produsul si cantitatea dorita de acesta
h) report order: va genera un PDF cu comenzile

4. Continutul tabelelor va fi practic ilustrat de PDF-urile generate in urma rularii fisierului de intrare.

Scenarii de esec:

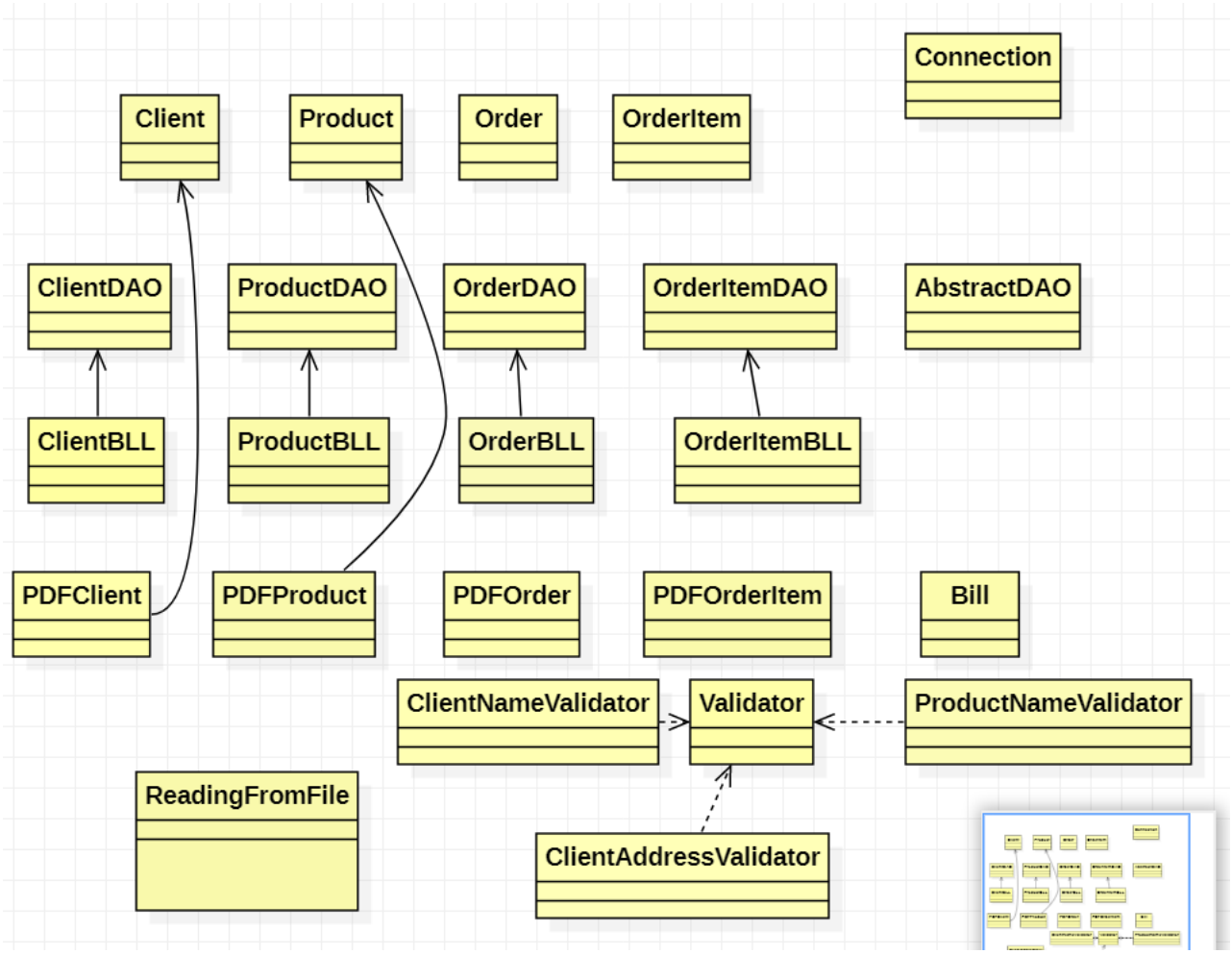
• Numele fisierului dat ca input nu corespunde cu numele unui fisier existent, in acest caz va fi aruncata o exceptie care va informa utilizatorul.

3.Proiectare

3.1. Decizii de proiectare

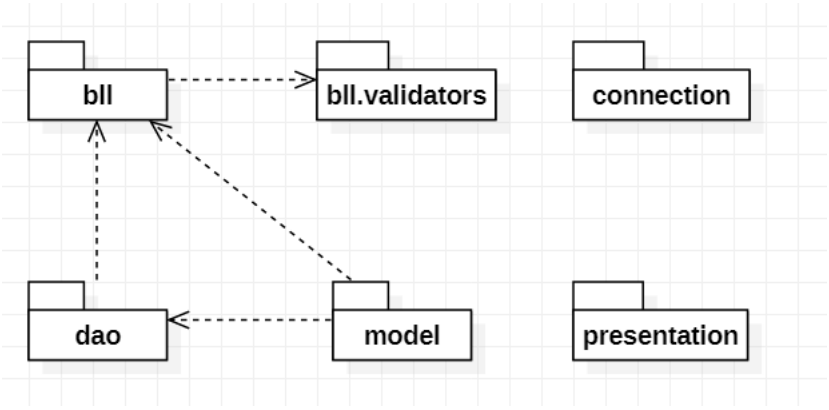
Am considerat ca fiind potrivit sa ma structurata aplicatia in mai multe pachete, tocmai pentru a ilustra ideea de "**Layerd Architecture**". Astfel. pachetul **bll(business logic classes)** contine clasele ClientBLL, OrderBLL, OrderItemBLL, ProductBLL; **bll.validators** contine clasele Validator, ClientNameValidator, ClientAddressValidator, ProductNameValidator, OrderTotalValidator; connection contine clasa ConnectionFactory; **dao(data access classes)** contine clasele AbstractDAO, ClientDAO, OrderDAO, OrderItemDAO, ProductDAO; **model** contine clasele Client, Order, OrderItem, Product; **presentation** contine clasele Bill, PDFClient, PDFOrder, PDFOrderItem, PDFProduct, ReadingFromFile; iar ultimul pachet, **start** contine clasa Start care este practic **main class** in care am facut diverse teste pentru a testa pe rand functionalitatea functiilor implementate, inainte de a face citirea din fisier.

3.2 Diagrama de clase



3.4. Diagrama de pachete

Am decis sa urmez structura din modelul primit ca exemplu, astfel, pachetele acestui proiect sunt: **bll**(business logic classes), pachet ce contine practic clasele ce contribuie la realizarea logica a operatiilor pe tabele bazei de date, **bll.validators**, ce contine anumite clase cu rol de a valida unele campuri inainte de a le insera in baza de date, **connection** care contine clasa ce realizeaza conexiunea cu baza de date, **dao**(data access classes), pachet a carui clase sunt raspunzatoare de interactiunea cu data base-ul, **model** a carui clase corespund cu tabelele bazei de date, fiecare cu atributele sale specifice, **presentation**, ce contine clase ce au legatura cu afisarea rezultatelor sub forma de PDF-uri si **start**, ce contine o clasa in care am facut diverse verificari pentru corectitudinea operatiilor de inserare sau stergere in sau din tabelele existente.



3.5. Structuri de date si algoritmi

Ca **structuri de date** am folosit ArrayList sau List de Client, Product, Order sau OrderItem in functie de necesitate.

La **algoritmi** folositi am putea sa consideram modul in care am introdus clientii si produsele in baza de date, stergerea clientilor si produselor cat si modificarile necesare ce trebuie facute.

De asemenea, am decis ca pretul comenzii sa fie considerat ca fiind produsul dintre pretul produsului si cantitatea cumparata.

4. Implementare

4.1. Clase

Pachetul connection

• clasa **Connection factory:** destul de importanta, intrucat primul pas pentru a putea realiza aplicatia propriu zisa era necesitatea unei conexiuni la baza de date creata in MySql, astfel, metodele din aceasta clasa sunt cele care duc la bun sfarsit aceasta prima necessitate. Ca metode in aceasta clasa apar **createConnection()**, **getConnection()**, **close()**.

Pachetul model

Dupa ce conexiunea s-a realizat cu success, aveam nevoie sa legam cumva tabelele bazei de date de aplicatia propriu zisa. Astfel, acest pachet contine clase ce coincid cu cele 4 tabele ale bazei mele de date ordermanagement.
• clasa **Client**: are 3 atribute ce definesc un posibil client care este raspunzator de cum decurg mai departe lucrurile in baza mea de dat: un **id** unic, **numele** acestuia si **adresa** (**id, name, address**)
Ca metode in aceasta clasa avem metodele de **get** si **set**, cat si o metoda **toString().**
• clasa **Product**: are 3 atribute menite sa ilustreze caracteristicile unui produs: **idproduct**(reprezentand un id unic al produsului), **nameP**(reprezentand numele produsului), **price**, de tip double(pretul unui produs) si **quantity**(cantitatea produsului respectiv). Ca metode in aceasta clasa avem metodele de **get**, **set** si **toString()**.
• clasa **Order**: am ales ca 4 aceasta clasa sa am 4 atribute, menite sa imi defineasca o comanda a unui client, astfel: **idorder**(id-ul comenzii), **idclient**(id-ul clientului care a plasat comanda), **nameClient**(numele clientului) si total(pretul total al comenzii).
• clasa **OrderItem**: are ca atribute **idorder** ca fiind un id unic, **idproduct**(id-ul produsului), **quantity**(cantitatea).

Pachetul dao (data acces classes)

• clasa **ClientDAO**: este cea care se ocupa cu interactiunea tabelei Client din baza de date. Pentru asta aveam nevoie de diferite statement-uri sql pentru inserare, stergere, selectarea tuturor coloanelor pentru afisare, sau selectarea doar anumitor campuri.
Ca metode in aceasta clasa avem:

• **findById**: metoda care ne ajuta sa cautam un client dupa id-ul sau, lucru pentru care vom folosi un statement de genul: findStatementString = “SELECT * FROM Client where idclient=?”. Ca prim pas avem nevoie de o conexiune la baza de date. Folosind un obiect de tip ResultSet, ce mentine cursorul pozitionat spre un rand al tabelei ne vom folosi de acel statement pentru a obtine ceea ce ne dorim. Foarte important este ca la final trebuie inchise toate conexiunile deschise initial: resultSet, findStatement, cat si conexiunea cu baza de date.

• **findByName**: in acest caz, metoda ne ajuta sa cautam un client dupa numele sau, fiind nevoie sa inlocuim campul “idclient” cu “name”, in rest procedansu-se la fel.

• **insert**: este cea care ne ajuta sa inseram un client in baza de date. De aceasta data vom avea nevoie de un statement de genul “INSERT INTO client (name, address) values (?, ?)”, adaugandu-se parametrii lipsa in locul necunoscutelor. Din nou vom avea nevoie de o conexiune la baza de date.

• **delete**: aceasta metoda ne ajuta sa stergem un client existent din baza de date. De aceasta data vom avea nevoie de un statement cum e “DELETE FROM client WHERE name=?”. Nu vom mai avea nevoie de executeQuery ca in cazul cautarii unui client dupa id, ci de executeUpdate(). Trebuie sa ne asiguram ca, clientul pe care vrem sa il stergem este deja in baza de date.

• **findAllClients()**: e o metoda de tipul ArrayList<Client>, fiind cea care ne ajuta sa cautam toti clientii din baza de date. Pentru acest lucru, de ajutor este statemntul “SELECT * from client”. Metoda va returna ArrayList-ul de clienti gasiti in baza de date. Aceasta functie ne va fi de folos in momentul in care vom vrea sa generam un PDF cu clientii din baza de date.

• clasa **ProductDAO**: este cea care se ocupa cu interactiunea tabelei Product din baza de date. Pentru asta aveam nevoie de diferite statement-uri sql pentru inserare, stergere, selectarea tuturor coloanelor pentru afisare, sau selectarea doar anumitor campuri.
Ca metode, in aceasta clasa avem:

• **findById**: gaseste un produs dupa id-ul sau. La fel ca si in cazul clientului, doar ca de aceasta data este vorba despre un produs.

• **findByName**: cauta un produs dupa numele sau.

• **insert**: insereaza un produs in baza de date

• **delete**: sterge unb produs existent din baza de date

• **findAllProducts**: returneaza un ArrayList de Products, reusind sa gaseasca toate produsele din baza de date

• **update1:** metoda are 2 parametrii: cantitatea cu care se va mari cantitatea actuala a produsului din baza de date si numele produsului. Datorita statement-ului ales, in final metoda ne va ajuta ca in cazul in care se insereaza acelasi produs in baza de date, sa nu se insereze de 2 ori, ci sa se mareasca cantitatea acestuia.

• **update2**: metoda are 2 parametrii: cantitatea cu care se va micsora cantitatea actuala a produsului din baza de date si numele produsului a carei cantitate necesita modificarea. Datorita statement-ului ales, in final metoda ne va ajuta ca in cazul in care exista o comanda in care intalnim produsul respective, cantitatea acestuia sa scada si sa se actualizeze la cat ar trebui sa fie in mod normal dupa procesarea comenzii.

• clasa **OrderDAO**: este cea care se ocupa cu interactiunea tabelei Order din baza de date. Pentru asta aveam nevoie de diferite statement-uri sql pentru inserare, stergere, selectarea tuturor coloanelor pentru afisare, sau selectarea doar anumitor campuri.
Ca metode, in aceasta clasa avem:

• **findByIdClient**: aceasta metoda este folosita pentru a gasi o comanda dupa id-ul unui client, id care este dat ca parametru in antetul functiei

• **insert**: de tip Order, ajuta la inserarea unei comenzi in baza de date

• **findAllOrders**: de tip ArrayList<Order> gaseste toate comenzile din baza de date

• clasa **OrderItemDAO**: este cea care se ocupa cu interactiunea tabelei OrderItem din baza de date. Pentru asta aveam nevoie de diferite statement-uri sql pentru inserare, stergere, selectarea tuturor coloanelor pentru afisare, sau selectarea doar anumitor campuri.
Ca metode, in aceasta clasa avem:

• **findByProduct**: ne ajuta sa gasim un order item dupa id-ul produsului

• **findByOrder**: ne ajuta sa gasim un order item dupa id-ul comenzii

• **findByOrderAndProduct**: este cea care ne ajuta sa gasim un order item dupa id-ul comenzii si id-ul produsului, ambele primite ca parametrii in antetul metodei

• clasa **AbstractDAO**: aceasta clasa am facut-o ulterior, dupa ce erau gata in mare parte toate operatiile necesare pe baza mea de date, incercaand sa aplic in acest mod si reflection, insa nu am avut timpul necesar sa fac toate modificarile necesare, astfel, e doar o incercare. Toate celelalte clase din pachetul dao ar trebui sa extinda aceasta clasa

Pachetul bll(business logic classes)

• clasa **ClientBLL**: implementeaza operatiile logice care tin de tabela Client a bazei mele de date.
Important de precizat este ca, in constructorul acestei clase am decis sa validez un client, lucru despre care voi discuta in clasa destinata acestui lucru.
Ca metode, in aceasta clasa am:

• **findClientById**: aceasta metoda gaseste un client in functie de id-ul sau, apeland metoda findById() din clasa ClientDAO, metoda despre care am discutat mai sus.

• **findClientByName**: aceasta metoda gaseste un client in functie de numele sau, apeland metoda findByName() din clasa ClientDAO.

• **insertClient()**: metoda insereaza un client in baza de date, apeland metoda insert din ClientDAO. Inainte de a insera un client, m-am asigurat ca este un client considerat valid.

• **deleteClient()**: metoda apeleaza findByName() si sterge un client din baza de date.

• **update()**: este cea care actualizeaza informatiile in cazul unor modificari.

• **findAll()**: gaseste toti clientii din baza de date.

• clasa **ProductBLL**: implementeaza operatiile logice ce tin de tabela Product a bazei mele de date.
Ca metode, in aceasta clasa intalnim:

• **findProductById**: metoda gaseste un produs dupa id

• **findProductByName**: metoda gaseste un produs dupa nume

• **insertProduct**: metoda insereaza un produs in baza de date

• **updateProduct**: metoda actualizeaza informatiile in baza de date in cazul in care exista o comanda si cantitatea produsului trebuie modificata

• **updatePlusProduct**: metoda actualizeaza informatiile in baza de date in cazul in care se doreste inserarea unui produs deja existent in baza de date, trebuind modificata cantitatea acestuia

• **deleteProduct**: metoda sterge un client din baza de date

• clasa **OrderBLL**: implementeaza operatiile logice ce tin de tabela Order a bazei mele de date.
Ca metode, in aceasta clasa apar:

• **insert**: insereaza o noua comanda in baza de date

• **update**: acutualizeaza informatiile legate de comenzi in baza de date

• clasa **OrderItemBLL**: implementeaza operatiile logice ce tin de tabela OrderItem a bazei mele de date.
Ca metode, in aceasta clasa avem:

• **findByOrder**: metoda este cea care cauta in baza de date anumite order item-uri care au id-uri de comanda la fel

• **insert**: metoda insereaza o comanda in baza de date. In aceasta metoda se verifica daca clientul care a plasat comanda a mai comandat produsul cu id-ul respectiv. Apelez metoda findByOrderAndProduct din OrderItemDAO caz in care rezultatul returnat nu este null, voi schimba pretul obiectului returnat cu pretul order item-ului. Cantitatea noua va fi modificata, considerate ca fiind suma dintre vechea cantitate si noua cantitate a order item-ului, dupa care vom seta aceasta noua cantitate. Pnetru ca apar anumite modificari, se va apela metoda de update din aceeasi clasa, OrderItemDAO. In caz contrar, atunci cand e vorba de un rezultat null, doar se va insera order item-ul in baza mea de date, apelandu-se metoda insert din clasa OrderItemDAO.

Pachetul bll.validators

Clasele acetui pachet verifica anumite detalii legate de numele clientului sau produsului, clase care, in final au rol de a se asigura ca nu se vor insera obiecte in baza de date care nu respecta conditiile impuse. Vom considera valide obiectele care indeplinesc conditiile, fie ca vorbim despre client, produse sau comenzi.

• clasa **Validator:** este o interfata si contine metoda **validate**, metoda care va fi implementata in restul claselor din acest pachet
• clasa **ClientAddressValidator**: implementeaza interfata Validator<Client>. Aici regasim metoda **validate**, in care se verifica faptul ca adresa unui client, ca string ar trebui sa fie intre 3 si 45 de caractere ca dimensiune.
• clasa **ClientNameValidator**: implementeaza interfata Validator<Client>. Aici regasim metoda **validate**, in care se verifica faptul ca numele unui client, ca string ar trebui sa fie intre 3 si 45 de caractere ca dimensiune.
• clasa **ProductNameValidator**: implementeaza interfata Validator<Product>. Aici regasim metoda **validate**, in care se verifica faptul ca numele unui produs, ca string ar trebui sa fie intre 3 si 10 de caractere ca dimensiune.
• clasa **OrderTotalValidator**: implementeaza interfata Validator<Order>. Aici regasim metoda **validate**, in care se verifica faptul ca totalul comenzii nu ar trebui sa fie negative, ci mai mare decat 0. Daca nu se respecta aceasta conditie, va fi aruncata o exceptie ce va preciza acest lucru.

Pachetul presentation

Acet pachet contine clase care au legatura cu afisarea rezultatelor: rezultatele se vor afisa sub forma de PDF-uri.

• clasa **PDFClient**: este cea care ne ajuta sa generam un PDF pentru un client, PDF in care vor aparea clientii din baza de date citiri din fisierul dat.
• clasa **PDFProduct**: este cea cea care ne ajuta sa generam un PDF pentru un produs, PDF in care vor aparea produsele din baza de date in urma citirii din fisierul dat.
• clasa **PDFOrderItem**: este cea care ne ajuta sa generam un PDF pentru un order.
• clasa **Bill**: este cea care ne ajuta sa generam o facture pentru un client, factura in care va aprea id-ul clientului si cat trebuie sa plateasca acesta pentru comanda.
• clasa **ReadingFromFile**: este una din clasele de baza ale proiectului, intrucat aici este locul in care se “leaga” majoritatea lucrurilor, intrucat, pana la urma, rezultatele obtinute stau la baza fisierului de intrare primit ca argument.
Ca metode, in aceasta clasa intalnim:

• **insertClient**, **insertOrder, insertProduct**: metode care vor fi apelate mai departe, in metoda readFromFile, in momentul in care intalnim cuvantul “insert” printre comenzile din fisier. Fiecare metoda trebuie sa se asigure ca argumentele sunt separate prin virgula, intrucat asa apar in fisier.

• **deleteClient**, **deleteOrder**, **deleteProduct**: metode care vor fi apelate mai departe, in metoda readFromFile, in momentul in care intalnim cuvantul “delete” printre comenzile din fisier.

• **reportClients**, **reportProducts**, **reportOrders**, **reportItems**: metode care vor fi si ele apelate mai departe in aceeasi metoda readFromFile, in momentul intalnirii cuvantului “report” printer comenzile fisierului.

• **getBill**: ajuta la generarea unei factyri, fiind apelata in metoda makeOrder.

• **makeOrder**: aceasat metoda se ocupa cu “plasarea” unei comenzi in baza de date. Trebuie sa cautam clientul in baza de date, apeland metoda findByName, iar in cazul unui produs, din nou, vom cauta produsul dupa numele sau apeland metoda findProductByName. Trebuie sa vedem daca exista un anumit stoc pentru produsul respectiv(cantitate). In cazul in care nu sunt suficiente produse in stoc, va fi afisat un mesaj in consola in care se va preciza ca nu mai este produsul cu numele respective. Altfel, va trebuie actualizat stocul produsului, cu ajutorul metodei updateProduct, dupa are stabilim noul total al comenzii prin creearea unui nou obiect de tip Order. Se va stabili pretul comenzii, dupa care se va creea un nou obiect de tip order item, iar la final se va genera o factura cu detaliile obtinute.

• metoda **readFromFile**: se asigura ca toate metodele din aceasta clsa vor fi apelate doar in momentul in care se gaseste in fisier comanda resprectiva, verificand daca in fisier apar anumite cuvinte “cheie”.

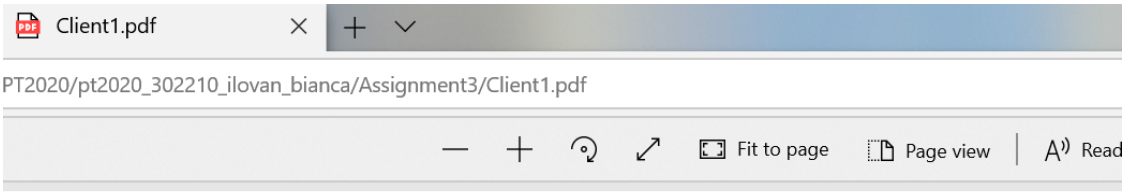
In aceasta clasa apare si metoda **main**, de unde se va rula programul.

Pachetul start:

• clasa **Start**: contine diverse verificari facute pe parcursul proiectului.

5. Rezultate

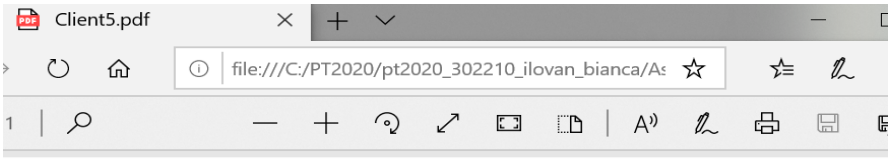
In urma rularii, se vor obtine niste pdf-uri de forma:



IdClient	Name	Address
380	Ion Popescu	Bucuresti
381	Luca George	Bucuresti

La fel pentru produse si comenzi.

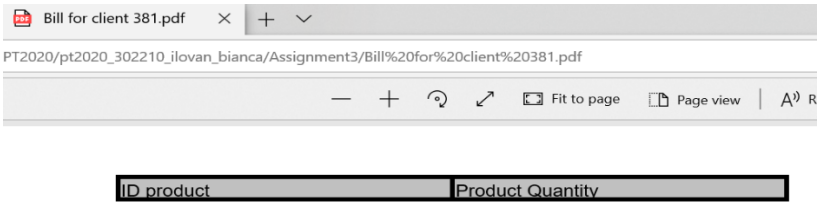
Comenzile pot fi observate intr-un PDF numit “Client5.pdf”, fisier care se genereaza in momentul intalnirii sintagmei “Report orders”, fiind afisati clientii care au comenzi.



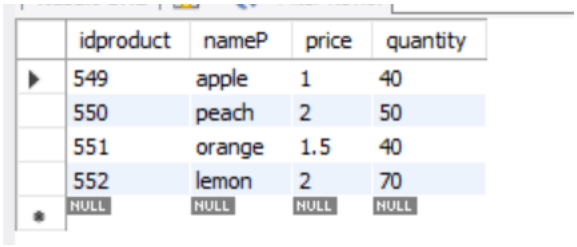
IdClient	Name	Address
435	Luca George	Bucuresti
436	Sandu Vasile	Cluj-Napoca

Produsele comandate vor putea fi vazute in PDF-ul “OrderItem1.pdf”, unde vor aparea id-ul si cantitatea dorita.

In cazul unei facturi, de exemplu pentru clientul cu id-ul 381, in fisier aparand cu numele de “Luca George”, va avea de platit un total de 10 euro.



In urma executarii comanzilor din fisier, rezultatele pot fi observate si in tabelele din MySQL. De exemplu, pentru tabela Product, in urma executarii comenzilor din fisier vom avea:



idproduct	nameP	price	quantity
549	apple	1	40
550	peach	2	50
551	orange	1.5	40
552	lemon	2	70
NULL	NULL	NULL	NULL

Daca produsele nu sunt sterse din baza de date, in momentul unei noi rulari cantitatea produselor existente in baza de date se va mari(spre exemplu daca se ruleaza din nou avand ca argument acelasi fisier).

6. Concluzii

Consider ca din aceasta tema, la fel ca din toate de altfel am invatat multe lucruri si din nou, m-am intalnit cu multe chestii cu care nu mai lucrasem pana acum absolut deloc.
Cel mai important, imi place ca am invatat sa imi structurez aplicatia in pachete si sa stiu sigur sic lar de la inceput rolul fiecarui pachet si clase.

Pentru mine tema asta cred ca a fost cea mai dificila si stresanta dintre toate, in principal pentru faptul ca am avut de lucrat si cu baze de date.

7. Bibliografie

- Bitbucket
- Stackoverflow
- https://mkyong.com/jdbc/how-to-connect-to-mysql-with-jdbc-driver-java/
- https://dzone.com/articles/layers-standard-enterprise
- baeldung.com/java-pdf-creation
- http://coned.utcluj.ro/~salomie/PT_Lic/4_Lab/Assignment_3/Assignment_3_Indications.pdf