MINISTERUL EDUCAȚIEI NAȚIONALE

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE

GESTIUNEA COMENZILOR

--Tema3--

Student: Mătieși Darius-Andrei

Serie/Grupa: A/30225

Profesor laborator: Moldovan Dorin

An universitar: 2019-2020

Contents

No table of contents entries found.

1.Obiectivul Temei

1.1 Obiectivul principal

Cerinta proiectului

Se cere o aplicație pentru procesarea comenzilor clienților la un depozit. Produsele, clientii și comenzile vor fi ținute în data de baze relaționale. De asemenea, aplicația trebuie să urmeze modelul layered arhitecture și să folosească minim următoarele clase :

•**Model classes** - the data models of the application;

•**Business Logic classes** – implement the application logic;

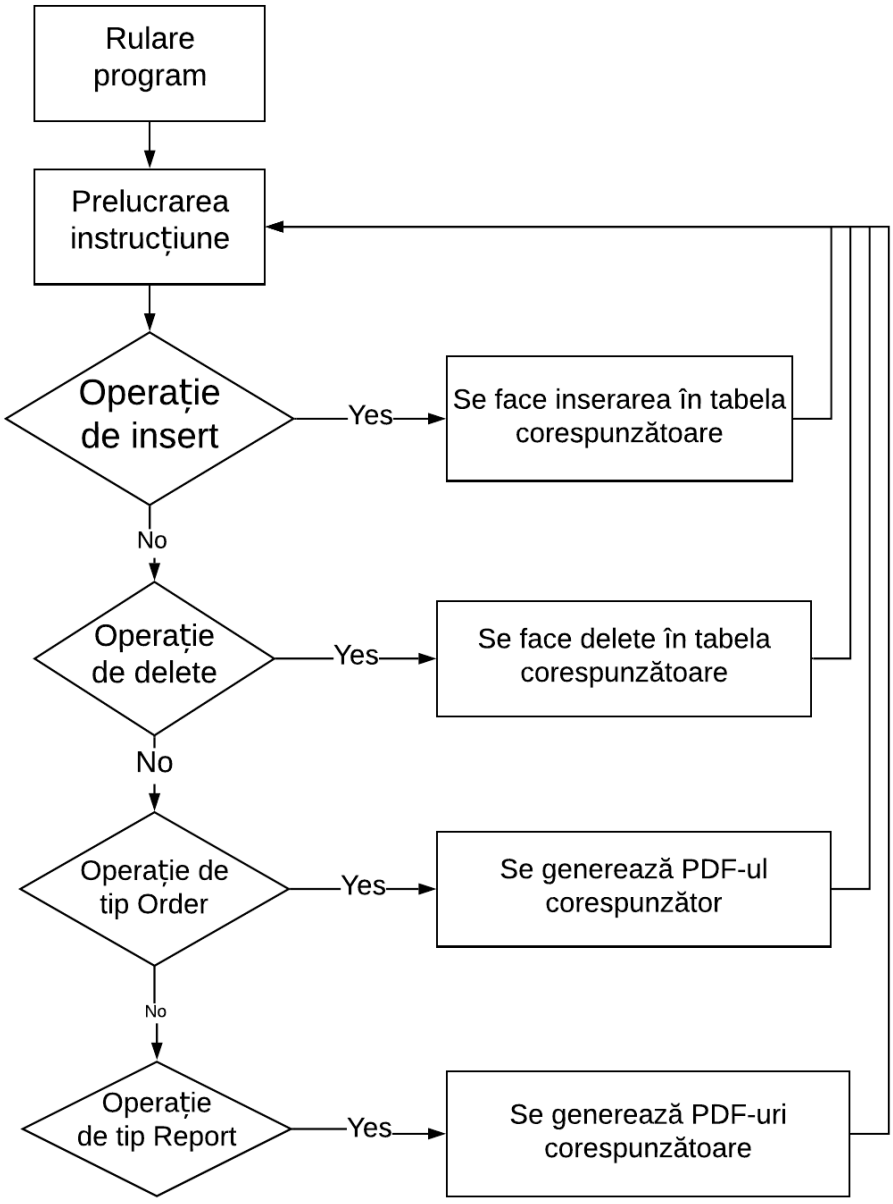
•**Presentation classes** – implement the user input/output;

•**Data access classes** - implement the access to the database;

Obiective secundare:

Obiectiv secundar	Descriere	Capitol
Dezvoltare de use case-uri și scenarii	Printr-un *use-case* se înțelege o listă de acțiuni care definesc interacțiunile dintre un rol (actor) și un sistem cu scopul atingerii unui obiectiv.	2
Proiectare schemă UML	Proiectarea diagramei de clase.	3
Alegerea structurilor de date	Prezentarea structurilor de date pentru atingerea obiectivului principal.	3
Proiectare clase	Utilizarea claselor Client, Client_to_comanda, Comanda , Produs , Produs_to_comanda, pentru a genera un sistem de gestiune a comenzilor pe baza tabelelor din relațional	3
Devoltarea algoritmilor	Descrierea algoritmilor folosiți pentru realizarea gestiunii comezilor	3
Implementarea soluției	Descriere clase și metode și conexiune relațional-program-utilizator	4
Rezultate	Prezentare a câtorva scenarii de testare	5

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



FOLOSIRE PROGRAM!

Pentru a compila și testa programul, trebuie verificat inainte ca parola și utilizatorul conexiunii la baza de date să fie corespunzătoare cu cea a programului de pe calculatorul actual. **Parola setata initial este " ", iar utilizatorul este "root",** din cauza folosirii XAMPP care nu necesită o parolă. Acestea se modifică din cod, în pachetul connection, în clasa ConnectionFactory.

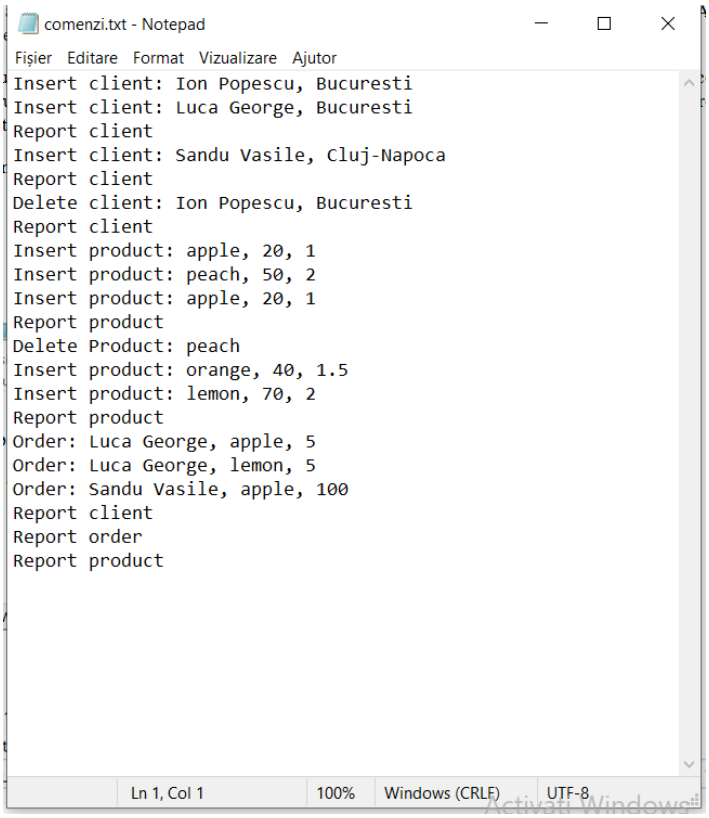
Pentru a se face corect citirea și interpretarea datelor de intrare, acestea trebuie să fie scrise într-un fișier text și să aibă următoarea structură:

- Să înceapă cu un String din multimea "Insert, Delete, ,Report, Order" care indică operația
- Pentru primele 2 String-uri, instrucțiunea să fie urmată de un alt String care sa indice tabela "client sau product".(Pentru Delete in particular e nevoie să fie Delete Product),urmat de caracterul ": ". Comanda se continuă apoi astfel:
  - Insert client*: nume_client, adresă* , unde nume_client este văzut ca un șir de caractere care poate să conțină spații, iar adresă e șir de caracatere care reprezintă orașul de proveniență
  - Insert product*: nume_produs, cantitate, preț* , unde nume_produs este un String reprezentând numele produsului, cantitate este un întreg reprezentând cantitatea de produs și preț este un număr real reprezentând prețul de vânzare al produsului
  - Delete client: nume_client , adresă , unde acestea au aceleași conotații ca mai sus
  - Delete Product: nume_produs, unde acesta are aceași conotație ca mai sus.
- "Report" trebuie să fie urmat de unul din String-urile „client, order sau product".
- "Order" este urmat de ": " și apoi de nume_client, nume_produs și cantitate_cumpărată, unde nume_client și nume_produs sunt String-uri corespunzătoare clientului și a produsului solicitat de acesta, iar cantitate_cumpărată este un întreg simbolizând cantitatea dorită de client din produsul respectiv.

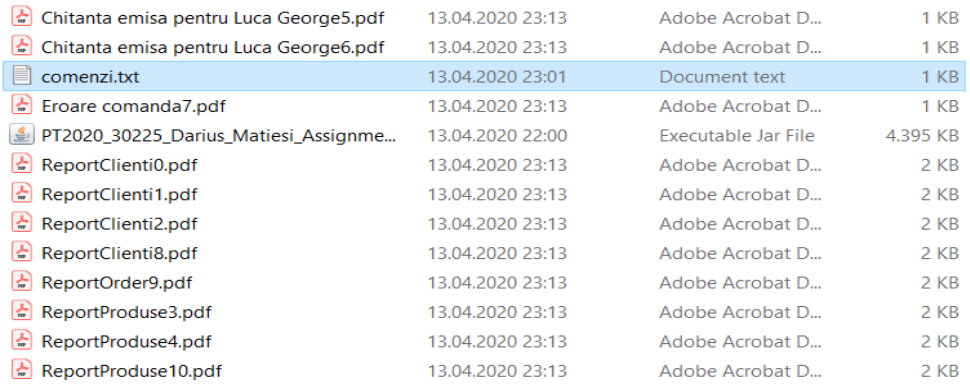
Comanda necesară este: java –jar PT2020_30225_Darius_Matiesi_Assignment_3.jar in.txt

unde in.txt-> fișierul de intrare cu datele menționate

Exemplu fișier intrare:



Conținut fișier după apel



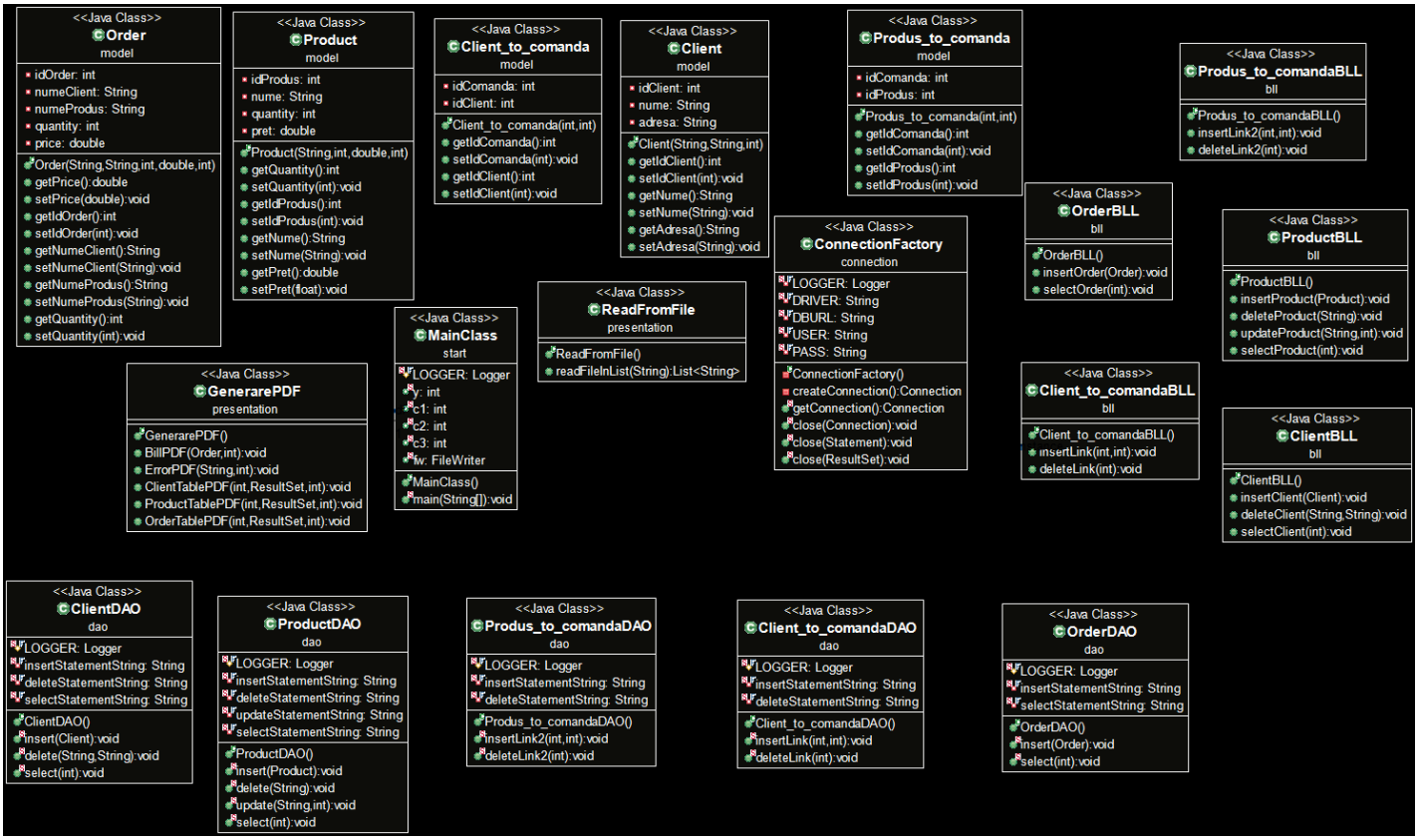
3. Proiectare

3.1 Decizii de proiectare

Datele din fișierul de intrare sunt prelucrate și , dacă acestea sunt corecte , pe baza acestora se execută operația corespunzătoare fiecărei linii.

În funcție de acesta, folosind liste(ArrayList-uri) din obiectele din pachetul “Model”(câte unul pentru fiecare tabel din relațional), se creează un obicect de tip de obiect din pachetul “BLL” care să cheme o metodă din pachetul “DAO” corespunzător și să facă operația pur și simplu atât în tabele cât și in listele corespunzătoare obiectelor din programul principal.

3.2 Diagrama UML



3.3.Structuri de date

Structurile de date pe care am decis să le folosesc sunt din pachetul **java.util.Linked.List** folosite pentru a stoca în paralel informația din tabele mai util, fapt care a ușurat și munca la efectuarea operațiilor pe tabele. Pachetul ne permite să folosim funcții care ne ajută să accesăm și să modelăm foarte ușor termenii unei structuri de genul, precum:

- add(Object): adăugarea unui element în listă(insertLast());
- remove(Obiect): ștergerea unui element din listă;
- isEmpty(): verificare dacă lista e goală;
- getFirst(): returnează primul element din listă;
- get(int): returnează elementul de pe poziția int-ului respectiv și altele.

3.4. Proiectare de clase

Acestă aplicație folosește următoarele clase:

În pachetele:

1) model
   a) Client
      i) Clasa model echivalentă datelor din tabelul client
   b) Order
      i) Clasa model echivalentă datelor din tabelul comanda
   c) Produs_to_comanda
      i) Clasa model echivalentă datelor din tabelul legatura_produs _comanda
   d) Client_to_comanda
      i) Clasa model echivalentă datelor din tabelul legatura_client_comanda
   e) Product
      i) Clasa model echivalentă datelor din tabelul produs
2) dao
   a) ClientDAO
      i) Clasa responsabilă pentru manipularea operațiilor asupra tabelului client al bazei de date asociate
   b) OrderDAO
      i) Clasa responsabilă pentru manipularea operațiilor asupra tabelului comanda al bazei de date asociate
   c) Produs_to_comandaDAO
      i) Clasa responsabilă pentru manipularea operațiilor asupra tabelului legatura_produs _comanda al bazei de date asociate
   d) Client_to_comandaDAO
      i) Clasa responsabilă pentru manipularea operațiilor asupra tabelului legatura_client _comanda al bazei de date asociate
   e) ProductDAO
      i) Clasa responsabilă pentru manipularea operațiilor asupra tabelului produs al bazei de date asociate

3) bll
   a) ClientBLL
      i) menține toate operațiile care încapsulează logica utilizărill tabelului client al bazei de date asociate
   b) OrderBLL
      i) menține toate operațiile care încapsulează logica utilizărill tabelului comanda al bazei de date asociate
   c) Produs_to_comandaBLL
      i) menține toate operațiile care încapsulează logica utilizărill tabelului legatura_produs_comanda al bazei de date asociate
   d) Client_to_comandaBLL
      i) menține toate operațiile care încapsulează logica utilizărill tabelului legatura_client _comanda al bazei de date asociate
   e) ProductBLL
      i) menține toate operațiile care încapsulează logica utilizărill tabelului produs al bazei de date asociate
4) presentation
   a) GenerarePDF
      i) Reprezinta clasa corespunzatoare generării PDF-urilor necesare pentru chitanțe, rapoarte și comenzi nereușite
   b) ReadFromFile
      i) Reprezintă clasa utilizată pentru citirea din fișier
5) start
   a) MainClass
      i) Clasa principală a programului, clasa MainClass din pachetul start este și clasa în care se generează triajul textului și separarea operațiilor, dar și executarea acestora prin apel cu ajutorul obiectelor de tip DAO și BLL
6) connection
   a) ConnectionFactory
      i) Creează conexiunea în program cu baza de date

3.5 Interfețe

Proiectul a fost conceput pe baza utilizării unei anumite interfețe și doar a acesteia, și anume pe baza interfeței List care mă ajută la implementarea ArrayList-urilor necesare pentru realizarea operațiilor pe tabelele bazei de date în corcodanță .

3.6 Relații

Aplicația conține următoarele relații:

- de agregare: Clasa „MainClass”(intreg) conține un obict de tip GenerarePDF(parte)
- de dependență: Clasa Produs_to_comandaBLL implementează metoda “insertLink” corespunzătoare doar clasei respective

3.7 Algoritmi

Pentru realizarea gestiunii comenzilor am optat pentru implementarea următorilor algoritmi:

1. **Separarea cuvintelor**- pentru acest algoritm am dorit să fac separarea folosind metodele puse la dispoziție de Regex si folosind metoda .split după un anumit tipar. Am ales sa separ String-urile după ":" în primul rând si dupa ", " pentru cea de-a doua despartire, în cazul în care comanda este Delete , Insert sau Order pentru a separa corect argumente și a se efectua corect operațiile asupra tabelelor bazei de date.

2. **Verificarea tipului de eroare comandă**- pentru acest algoritm, am separate situațile în care se comanda nu se poate realiza comanda și motivele pentru care aceasta nu se poate realiza, folosind o variabila "valid" care determină dacă suntem pe cazul în care fie produsul nu este disponibi, fie cantitatea de produs este insuficientă, fie persoana nu este înregistrată în baza de date

3. **Stergerea din tabel-** pentru acest algoritm am ales să folosesc Iterator pentru parcurgerea listelor tabelului de legatură cu lista de obiecte de tip Client_to_comanda, respective a tabelului client cu lista de obiecte de tip Client pentru a putea să o traversez și să șterg de pee a în acelali timp. Stergerea se realizează appi și din tabel, prima data din tabelul de legătura și apoi din tabelul client, datorită constrngerilor de cheie strină din baza de date.

4.**Efecturarea Order**- pentru acest algoritm, am ales să parcurg lista de persoane, să verific numele persoanei și pentru aceasta, dacă există, să parcurg lista de produse și pentru produsul comandat, daca există, se face o actualizare atât în tabel cît și in lista de produse a cantității acestuia, dacă cantitatea cerută este mai mica decât cea deja existentă în depozit. În acest caz, se actualizează cantitatea și după se face actualizarea prețului comenzii, se introduce în tabela order, respective in lista orderObj , apoi în cele două tabele de legătura și în final se realizează facturarea într-un document PDF.

4.Implementare

Clasa Client:

- descriere:clasă simulând un consumator real, componentă agestiunii comenzilor, fiind caracterizată de Id(idClient), nume(nume), adresa(adresa) ;
- variabile de instanță: idClient(int), nume(String), adresa(String);
- constructori: unul cu trei paramentri: nume, adresa, idClient;
- getters: getIdClient(), getNume (),getAdresa ();
- setters: setIdClient (int),setNume (String),setAdresa(String)).

Clasa Product:

- descriere: clasă simulând un depozit al produselor fiind caracterizată de id(idProdus), nume(nume), cantitate(quantity), pret(pret);
- variabile instanță: idProdus (int), nume(String), quantity (int), pret(double);
- constructori: unul cu patru paramentri: nume, quantity, pret, idClient;
- getters: getIdProdus(), getPret(), getIdProdus(), getNume();
- setters: setIdProdus(int), setPret(float), setQuantity(int), setNume();

Clasa Order:

- descriere: clasă simulând o gestiune a comenzilor fiind caracterizată de id(idOrder), nume_client(numeClient), numeProdus(numeProdus), cantitate(quantity), pret(pret);
- variabile instanță: id(idOrder), nume_client(numeClient), numeProdus(numeProdus), cantitate(quantity), pret(pret);
- constructori: unul cu cinci paramentri: numeClient , numeProdus, quantity, pret, idOrder;
- getters: getIdProdus(), getPret(), getIdProdus(), getNume();
- setters: setIdProdus(int), setPret(float), setQuantity(int), setNume();

Clasa Client_to_comanda:

- descriere: clasă simulând o legătură între tabelele client și comandă fiind caracterizată de idClient și idComanda;
- variabile instanță: idClient, idComandă;
- contructori: unul cu doi paramentri: idComanda și idClient
- getters : getIdClient () , getdComanda() ;
- setters : setIdClient(), setIdComanda() ;

Clasa Produs_to_comanda:

- descriere: clasă simulând o legătură între tabelele produs și comandă fiind caracterizată de idProdus și idComanda;
- variabile instanță: idProdus, idComandă;
- contructori: unul cu doi paramentri: idComanda și idProdus()
- getters : getIdClient () , getdComanda() ;
- setters : setIdClient(), setIdComanda() ;

Metode caracteristice claselor:

1) Clasele din pachetul DAO
   a. Metoda insert(Client), insertLink(int, int) și insertLink2(int, int) – aceste metode cu nume diferite au functionalități asemănătoare, ele preiau conexiunea cu bază de date și pregătesc statement-ul pentru a putea fi analizat sub formă de comanda MySQL, după care execută inserarea în tabelele corespunzătoare și la final se închid conexiunile.

   b. Metoda delete(Client, Order, Product), insertLink(int, int) și insertLink2(int, int) – aceste metode cu nume diferite au functionalități asemănătoare, ele preiau conexiunea cu bază de date și pregătesc statement-ul pentru a putea fi analizat sub formă de comandă MySQL, după care execută ștergerea în tabelele corespunzătoare și la final se închid conexiunile.

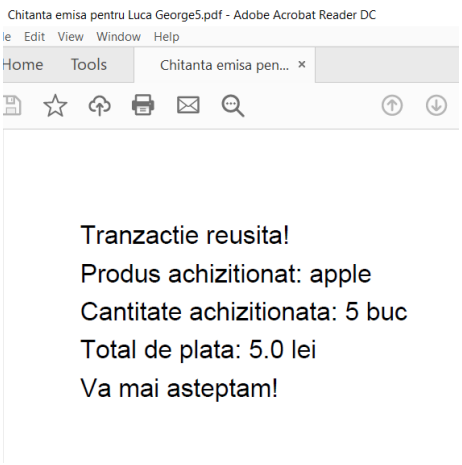
   c. Metoda select(int)- metodă caracteristică fiecărei clase din acest pachet, preia conexiunea cu bază de date și pregăteste statement-ul pentru a putea fi analizat sub formă de comandă MySQL, după care execută selectarea din tabelele corespunzătoare și la final se închid conexiunile.

   d. Metoda update(String, int)- caracteristică clasei Product, este metoda care preia conexiunea cu bază de date și pregăteste statement-ul pentru a putea fi analizat sub formă de comandă MySQL, după care execută update în tabelele corespunzătoare și la final se închid conexiunile.

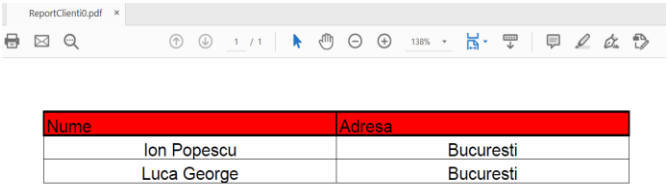
2) GenerarePDF
   a. Metoda BillPDF(Order, int) – metodă care creează un nou document, scrie în el detaliile referitoare la plata efectuată , și afișează un mesaj călduros de la revedere, dupa care închide fișierul.
   b. Metoda ErrorPDF(String, int)- Creează un document în care se scrie un mesaj de Eroare mesaj + String-ul primit ca argument sub formă de paragraph nou în document.
   c. Metoda ClientTablePDF(int, ResultSet, int)- Creează un document în care se creează un nou PDF în care se introd datele din tabelul client sub formă de tabel.
   d. Metoda ProductTablePDF(int, ResultSet, int) - Creează un document în care se creează un nou PDF în care se introd datele din tabelul produs sub formă de tabel.
   e. Metoda ProductTablePDF(int, ResultSet, int)- Creează un document în care se creează un nou PDF în care se introd datele din tabelul comanda sub formă de tabel.

5.Rezultate

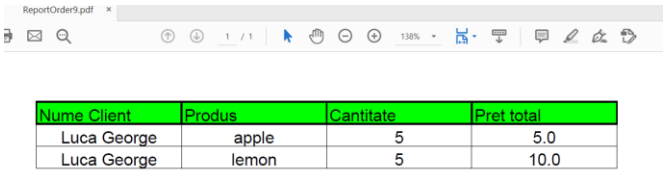
Conținut PDF factură:



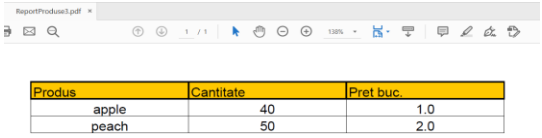
Conținut PDF Rapoarte:



Nume	Adresa
Ion Popescu	Bucuresti
Luca George	Bucuresti

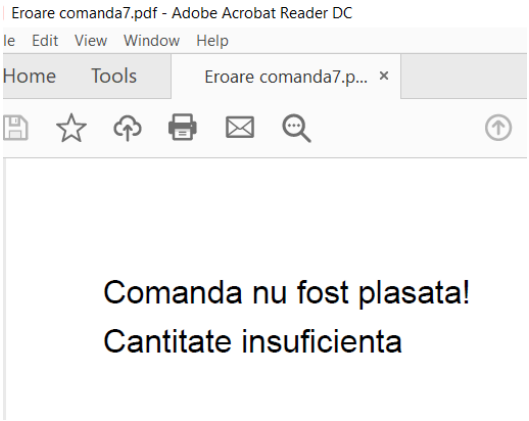


Nume Client	Produs	Cantitate	Pret total
Luca George	apple	5	5.0
Luca George	lemon	5	10.0



Produs	Cantitate	Pret buc.
apple	40	1.0
peach	50	2.0

Conținut PDF eroare comandă :



6.Concluzii

În urma acestui proiect am reușit cu brio să înțeleg și să stăpânesc lucrul programului cu bazele de date, să stabilesc sincronizare și să stabilesc integritatea datelor și manipularea acestora . Prin implementarea claselor am înțeles cum funcționează un sistem de gestiune a comenzilor și ce mecanism are în spate. De asemenea, datorită modului de citire și scriere a informațiilor am deprins abilități precum prelucrarea datelor din și în fișiere, dar și rularea unui fișier .jar folosind argumentele.

Îmbunătățiri ulterioare

- Posibilitatea de a cumpăra mai multe produse deodată și facturarea pe o singură chitanță
- Introducerea unor sugestii legate de cumpărare(reclame)
- Eventualitatea vizualizării de către client a stocului la un moment dat pentru a cunoaște opțiunile

7.Bibliografie

[1] Cursul de Programare Orientată pe Obiecte 2019-2020(PPT)- Conf. Dr. Ing. T. Cioara

[2] https://stackoverflow.com/questions/2540548/how-do-i-get-eclipse-to-use-a-different-compiler-version-for-java

[3] http://coned.utcluj.ro/~salomie/PT_Lic/4_Lab/Assignment_3/Java_Concurrency.pdf

[4] https://www.baeldung.com/reading-file-in-java

[5] https://www.lucidchart.com/documents#/templates?folder_id=home&browser=icon