

Universitatea Tehnică Cluj-Napoca

ORDER MANAGEMENT

~ Tema 3 ~

Stroe Mădălina Ionela

Grupa 302210

Semigrupa2

CUPRINS

- Obiectivul temei
- Analiza problemei, modelare, scenarii și cazuri de utilizare
- Proiectare
- Implementare
- Rezultate
- Concluzii
- Bibliografie

~ Obiectivul temei ~

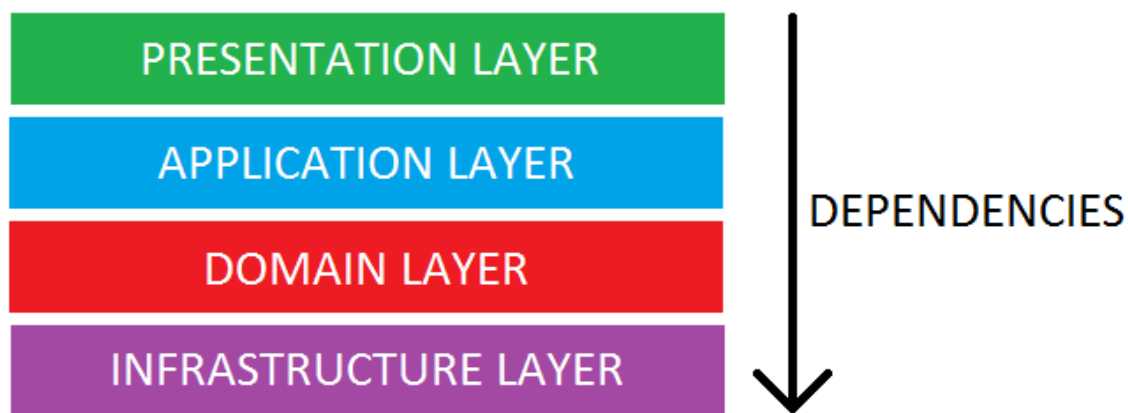
Obiectivul principal al temei este de a crea o aplicație **OrderManagement** pentru procesarea comenzilor clienților pentru un depozit(baza de date).

Command Name	Command Syntax	Description
Add client to the database	Insert client: Ion Popescu, Bucuresti	Insert in the database a new client with name Ion Popescu and address Bucuresti
Delete Client from the database	Delete client: Ion Popescu	Delete from database the client with name Ion Popescu
Add product to the database	Insert product: apple, 20, 1	Add product apple with quantity 20 and price 1
Delete product from the database	Delete product: apple	Delete product apple from database
Create order for client	Order: Ion Popescu, apple, 5	Create order for Ion Popescu, with apple quantity 5. Also update the apple stock to 15. Generate a bill in pdf format with the order and total price of 5
Generate reports	Report client Report order Report product	Generate pdf reports with all clients/orders/products displayed in a tabular form. The reports should contain the information corresponding to the entity for which reports are asked (client, order or product) returned from the database by a SELECT * query, displayed in a table in a PDF file.

Printre obiectivele secundare ale temei se numără structurarea aplicației în pachete utilizând arhitectură pe straturi, prezentată în materialul suport. De asemenea, aplicație trebuie să permită procesarea comenzilor dintr-un fișier primit ca argument, salvarea rezultatelor în baza de date și generarea rapoartelor în format .pdf.

Se dorește, desigur, și perfecționarea și aprofundarea cunoștințelor de programare orientată pe obiecte dobândite până în acest punct.

O arhitectură pe straturi (**layered architecture**) permite organizarea structurii proiectului în 4 mari categorii: *presentation layer* – conține clasele responsabile pentru input-ul și output-ul datelor, *model layer* – conține modelul pentru aplicație (clase ale căror obiecte vor fi folosite), *business logic layer* – conține implementarea logică a aplicației, algoritmi cu care se lucrează, *data acces* – conține instrucțiuni ce ne permit să accesăm baza de date.

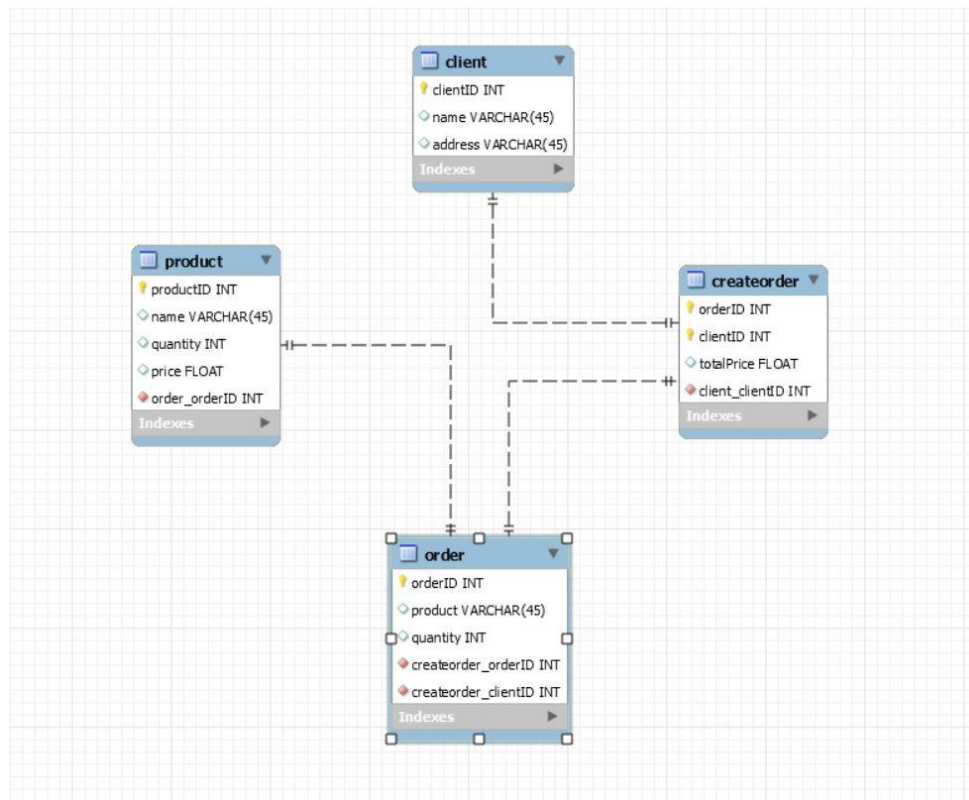


O bază de date relațională (**relational database**) este de fapt o colecție de informații aflate în relații, care sunt organizate în tabele în funcție de tipul lor pentru un acces mai ușor. Tabele folosite în baza de date stochează informații despre obiectele aflate în relații.

SQL este limbaj de interogare structurat, specific pentru manipularea datelor în sisteme de manipulare a bazelor de date relaționale, iar la origine este un limbaj bazat pe algebra relațională. Acesta are ca scop inserarea datelor, interogații, actualizare și ștergere, etc.

Un primary key este reprezentat de un set de attribute(coloane) care identifică unic o înregistrare în baza de date.

Un tabel care conține unul sau mai multe foreign keys se numește tabel-copil iar tabelul care conține primary key este denumit tabel de referință sau părinte. Așadar, un foreign key permite crearea de relații între tabele.



~ Analiza problemei ~

Pentru ca informațiile să ajungă în baza de date, este necesar ca acestea să fie citite din fișierul .txt și să fie procesate ulterior.

În fișier există comenzile de Insert, Delete, eventual Update, acestea fiind operațiunile care implică baza de date. De asemenea, putem face report, unde va trebui să selectăm toate datele din tabel și să scriem într-un fișier pdf rezultatul.

În poza de mai jos putem vedea care ar fi pașii execuției programului:

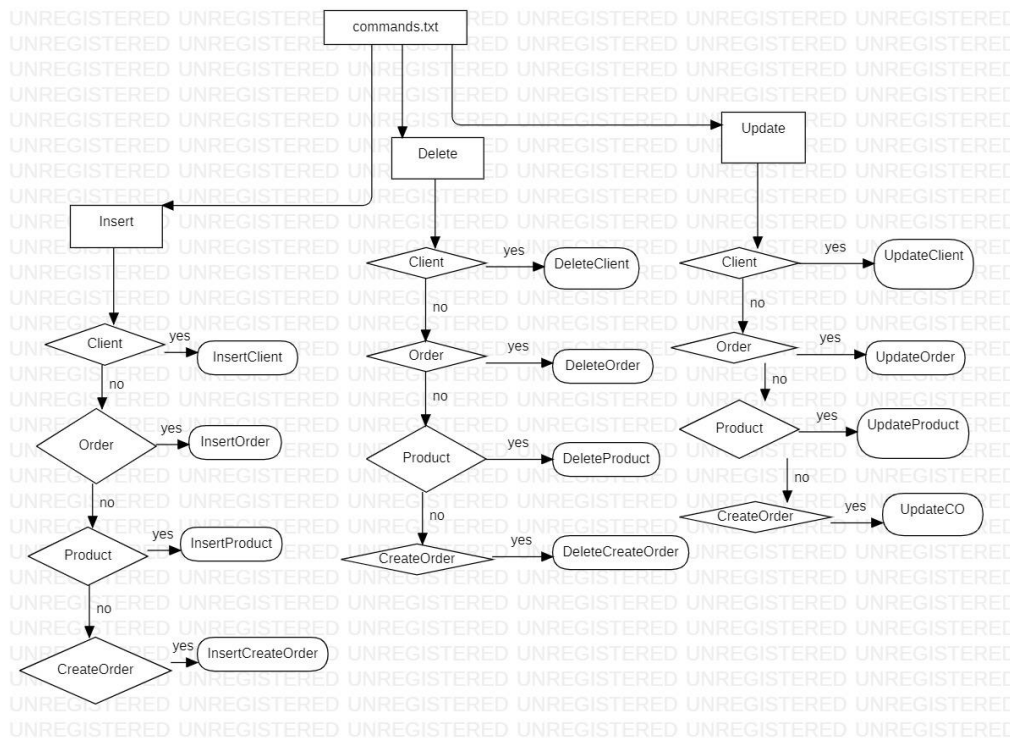
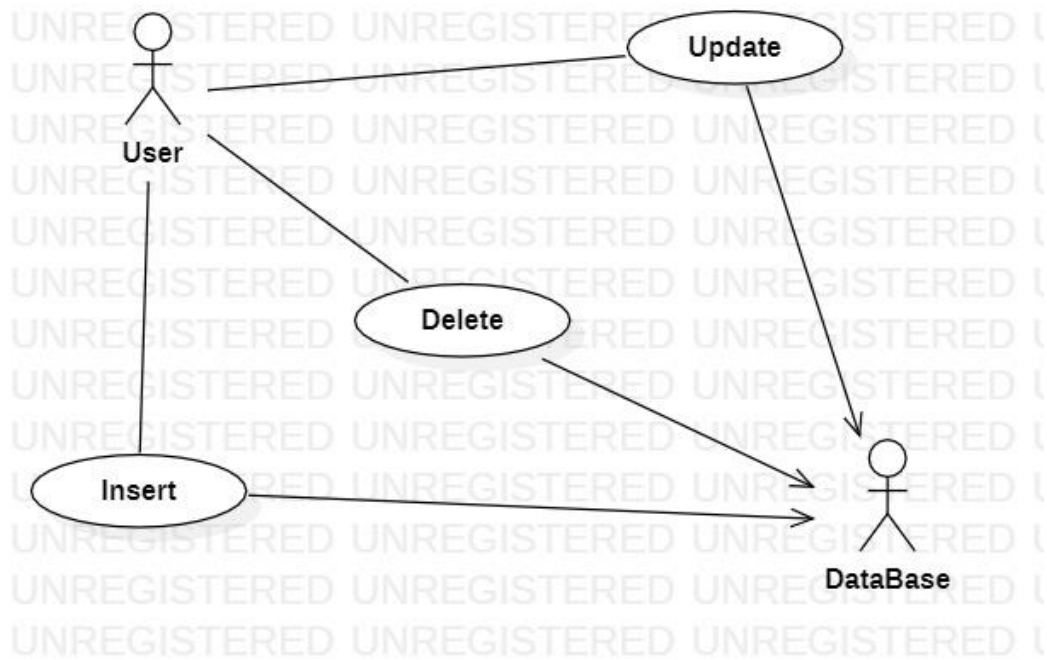


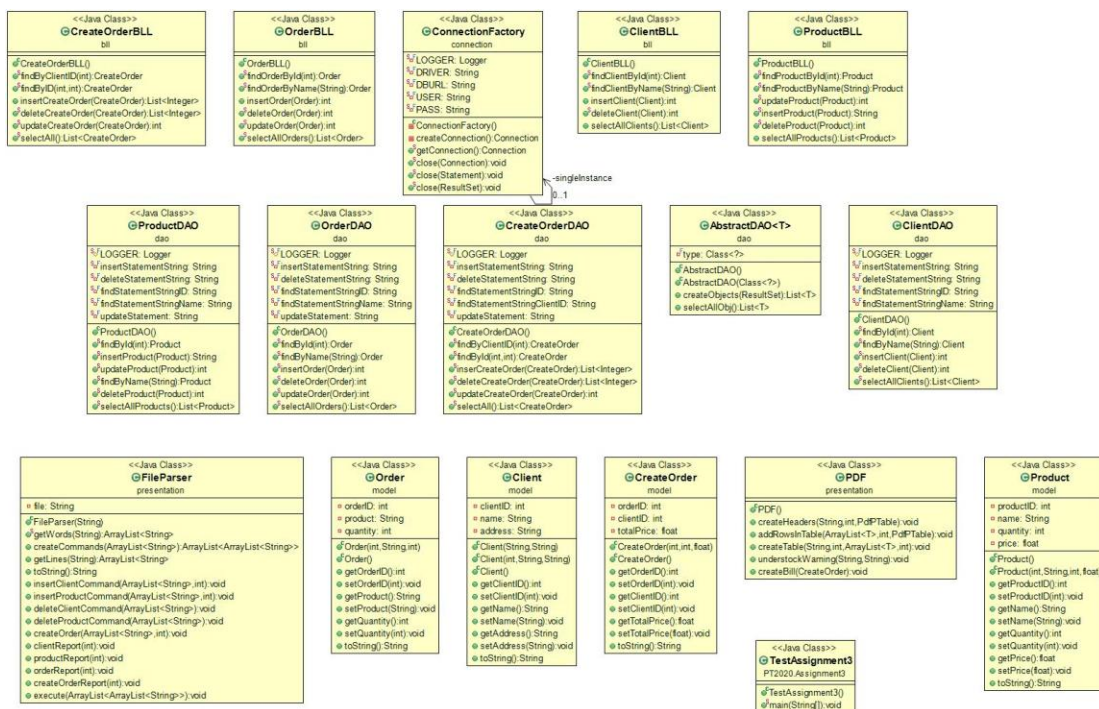
Diagrama use-case pentru această aplicație:



~ Proiectare ~

Unified Modeling Language (prescurtat UML) este un limbaj standard pentru descrierea de modele si specificatii software. Diagrama de clase UML Este folosită pentru reprezentarea vizuală a claselor și a interdependențelor, taxionomiei și a relațiilor de multiplicitate dintre ele. Diagramele de clasă sunt folosite și pentru reprezentarea concretă a unor instanțe de clasă, așadar obiecte și a legăturilor concrete dintre acestea.

În imaginea de mai jos puteți vedea diagrama de clase:

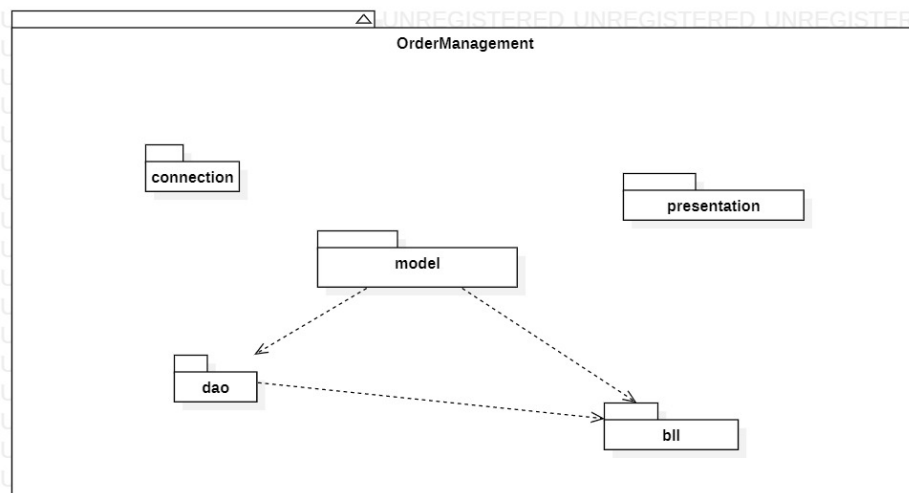


Am structurat programul în 6 pachete(straturi):

- BLL
 - Client BLL
 - CreateOrderBLL
 - OrderBLL
 - ProductBLL
- Connection
 - ConnectionFactory
- Dao

- ClientDAO
- CreateOrderDAO
- OrderDAO
- ProductDAO
- Model
 - Client
 - CreateOrder
 - Order
 - Product
- Presentation
 - FileParser
 - PDF
- PT2020.Assignment3
 - TestAssignment3

Diagrama de pachete:



Structurile de date folosite au fost List, ArrayList, etc, in funcție de necesitate.

Ca algoritmi putem considera algoritmi de adăugare în listă.

Calculul prețului final comandă= $\text{prețProdus} \times \text{cantitate}$.

~Implementare~

Această temă își propune să aducă în discuție manipularea bazei de date, realizând operații de inserare, ștergere și actualizare(depinzând de caz).

În primul rând e nevoie de o clasă care să implementeze metode care să ajute la conectarea la baza de date. Așadar, clasa ConnectionFactory are metode de creare conexiune cu baza de date, închidere conexiune cu baza de date, închidere statement, închidere resultSet.

Un Statement este un obiect folosit pentru a executa o interogare SQL și care returnează rezultatul produs.

Un obiect de tip ResultSet menține un cursor poziționat spre un rând al tabelului. Inițial, cursorul pointează înainte de primul rând.

În pachetul Model descriem obiectele pe care le vom folosi pe tot parcursul aplicației.

În clasa Client, avem attributele: id-ul clientului, nume, adresă.

În clasa CreateOrder avem attributele: id comandă, id client și preț total.

În clasa Order avem attributele: id comandă, nume produs, cantitate.

În clasa Product avem attributele: id produs, nume produs, cantitate, preț.

În pachetul DAO.

Clasa ClientDAO implementează următoarele metode:

- findById: cu ajutorul interogărilor SQL, vom căuta în baza de date clientul cu id-ul primit ca parametru.
- FindByName: cu ajutorul interogărilor SQL, vom căuta în baza de date clientul cu numele primit ca parametru.
- insertClient: cu ajutorul interogărilor SQL, vom insera în tabela Client un obiect de tipul client cu id primit.
- deleteClient: cu ajutorul interogărilor SQL, vom șterge din tabela Client un client cu id primit.
- selectAllClient: metodă de afișare a tuturor înregistrărilor din tabel.

Clasa CreateOrderDAO are următoarele metode:

- findByClientID: căutare în tabelă după id-ul clientului primit ca parametru

- findByID: căutare în tabelă a obiectului după o combinație de orderID și clientID.
- insertCreateOrder: inserare după id
- deleteCreateOrder: ștergere după id
- updateCreateOrder: actualizare după id
- selectAll: metodă de afișare a tuturor înregistrărilor din tabel.

Clasa OrderDAO:

- findById: căutare după id
- findByName: căutare după numele produsului
- insertOrder: inserare după id
- deleteOrder: ștergere după id
- updateOrder: actualizare după id
- selectAll: metodă de afișare a tuturor înregistrărilor din tabel.

Clasa ProductDAO:

- findById: căutare după id
- insertProduct: inserare după nume
- updateProduct: actualizare după id
- findByName: căutare după nume
- deleteProduct: ștergere după id
- selectAllProducts: metodă de afișare a tuturor înregistrărilor din tabel.

În pachetul BLL implementăm metodele din clasele DAO:

Clasa ClientBLL:

- findClientByID: returnează un client cu id-ul primit ca parametru
- findClientByName: aceeași poveste, dar pentru nume
- insertClient: returnează id clientInserat
- deleteClient: returneaza id
- selectAllClients: returnează o listă cu toate înregistrările

Clasa CreateOrderBLL:

- findByClientID: returnează un CreateOrder după id client
- findByID: returnează CreateOrder după combinație de id client și id order
- insertCreateOrder: returnează listă de id
- deleteCreateOrder: returnează listă de id pt obiect de șters
- updateCreateOrder: returnează id
- selectAll: listă cu toate obiectele

! Analog pentru OrderBLL și ProductBLL.

Pachetul presentation vine cu câteva lucruri noi de implementat:

Clasa FileParser:

- getWords: pentru linia primită ca parametru, vom folosi regular expressions cu scopul de a o separa în cuvinte cheie, pe care le vom adăuga într-o listă.
- getLines: primește ca parametru tot fișierul citit, dar transformat în string. Fișierul este citit linie cu linie. Rezultatul este un array de String-uri.
- createCommands: primește ca parametru lista de linii citite din fișier, pe care o transformă în comenzi.
- insertClientCommand: primește ca parametrii linia și id cu care va fi inserat clientul.
- insertProductCommand: primește ca parametrii linia și id cu care va fi inserat produsul. Dacă produsul cu id primit ca parametru nu există, va fi inserat. Altfel, el va fi actualizat.
- deleteClientCommand: primește linia cu comanda de ștergere
- deleteProductCommand: primește linia cu comanda de ștergere
- createOrder: primește linia cu comanda și id comandă.

Căutăm clientul după nume. Dacă acesta nu există, un pdf cu mesaj corespunzător va fi creat.

Altfel, verificăm dacă produsul există. Dacă nu se află în tabel, un mesaj corespunzător va fi afișat.

Dacă pentru acel produs, cantitatea este mai mică decât cantitatea cerută, un mesaj understock va fi creat.

Altfel, înseamnă că totul este în regulă și putem continua cu calculul prețului total pentru CreateOrder, apoi vom insera un CreateOrder cu id primit ca parametru, id client și preț calculat.

Pentru fiecare comandă, vom crea un bill.

Un Order simplu va fi căutat după order id.

Dacă nu există, se va insera, dacă există se va actualiza.

După ce comanda a fost creată, cantitatea produsului se va actualiza la cantitateInițială – cantitateComandată.

- Următoarele metode sunt pentru generarea raporturilor, toate lucrând după același tipar pe care îl voi explica mai jos.
- Execute: pentru fiecare linie verifică cuvintele cheie cu scopul de a realiza corect operația și a transmite valori corecte în baza de date.

Clasa PDF:

- createHeaders: este metoda pentru creare a coloanelor cu nume specifice din tabel. Primește ca parametru numele tabelului, iar numele fiecărei coloane din tabel va fi adăugată într-o listă. Dimensiunea este numărul de coloane din tabel.
- addRowsInTable: metodă de adăugare a înregistrărilor în tabel. Lista depinde de tipul de obiecte: client, product, order sau CreateOrder. Pentru fiecare obiect din listă, parcurgem fiecare coloană și o adăugăm în celula tabelului pdf.
- createTable: metoda de creare tabel pdf în funcție de nume și dimensiune.
- understockWarning: metoda de afișare a unui mesaj în anumite situații.
- createBill: metodă de creare chitanță. Vom adăuga numărul, numele Clientului și suma totală a order-ului. Am ales ca pentru fiecare order, indiferent dacă există de mai multe ori același client, să se genereze un bon.

~ Rezultate ~

Rezultatele pot fi văzute pe de o parte, direct în baza de date, iar pe de altă parte în rapoartele create.

Rezultate în baza de date:

CLIENT

	clientID	name	address
	2	Luca George	Bucuresti
	3	Sandu Vasile	Cluj-Napoca
▶*	NULL	NULL	NULL

PRODUS

	productID	name	quantity	price
	1	apple	35	1
	4	orange	40	1.5
	5	lemon	65	2
▶*	NULL	NULL	NULL	NULL




ORDER

	orderID	product	quantity
	1	apple	5
	2	lemon	5
▶*	NULL	NULL	NULL




CREATE ORDER

Result Grid			
	orderID	clientID	totalPrice
	1	2	5
	2	2	10
▶*	NULL	NULL	NULL

Rezultate în rapoarte:




















client_table2.pdf	1 / 1			
-------------------	-------	---	---	---

clientID	name	address
1	Ion Popescu	Bucuresti
2	Luca George	Bucuresti
3	Sandu Vasile	Cluj-Napoca




















client_table4.pdf	1 / 1			
-------------------	-------	---	---	---

clientID	name	address
2	Luca George	Bucuresti
3	Sandu Vasile	Cluj-Napoca

productID	name	quantity	price
1	apple	40	1.0
2	peach	50	2.0

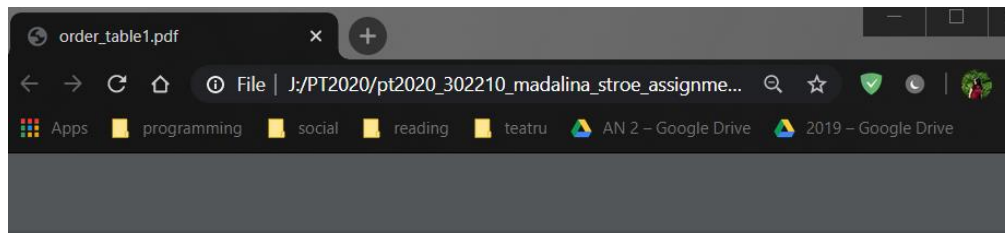
product_table2.pdf					
→    File J:/PT2020/pt2020_302210_madalina_stroe_assignm...     					
Apps  programming  social  reading  teatru  AN 2 – Google Drive  2019 – Google Drive					

productID	name	quantity	price
1	apple	40	1.0
4	orange	40	1.5
5	lemon	70	2.0

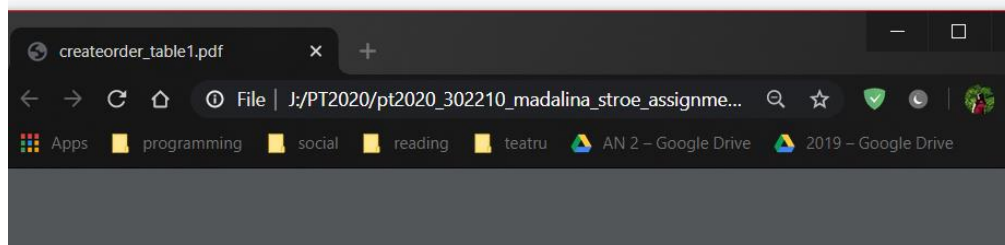
product_table3.pdf					
→    File J:/PT2020/pt2020_302210_madalina_stroe_assignme...     					
Apps  programming  social  reading  teatru  AN 2 – Google Drive  2019 – Google Drive					

productID	name	quantity	price
1	apple	35	1.0
4	orange	40	1.5
5	lemon	65	2.0

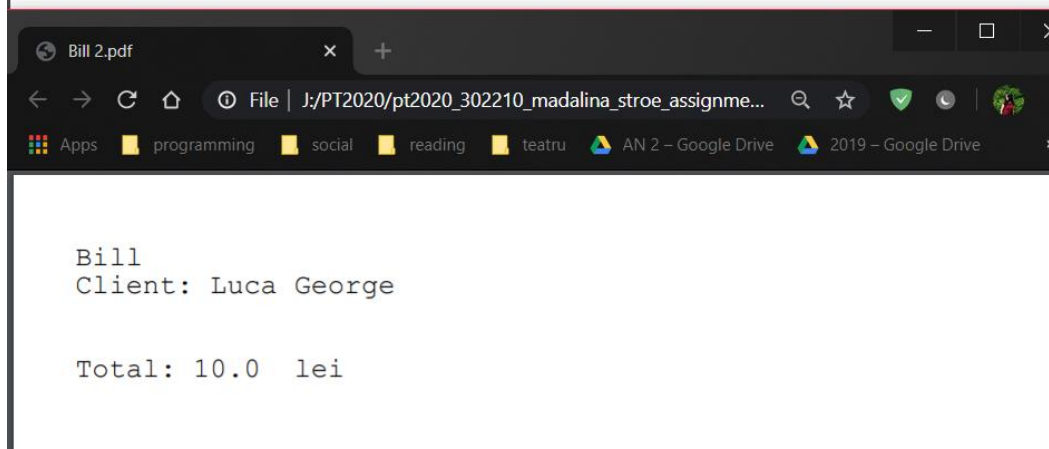
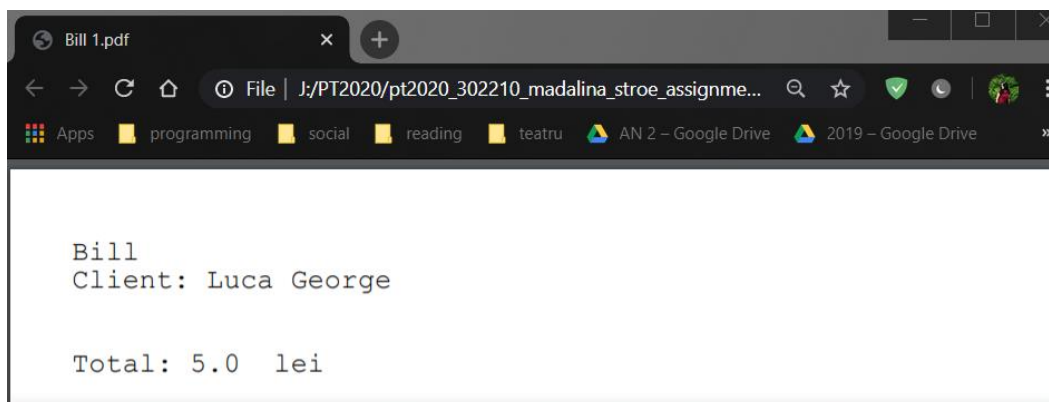


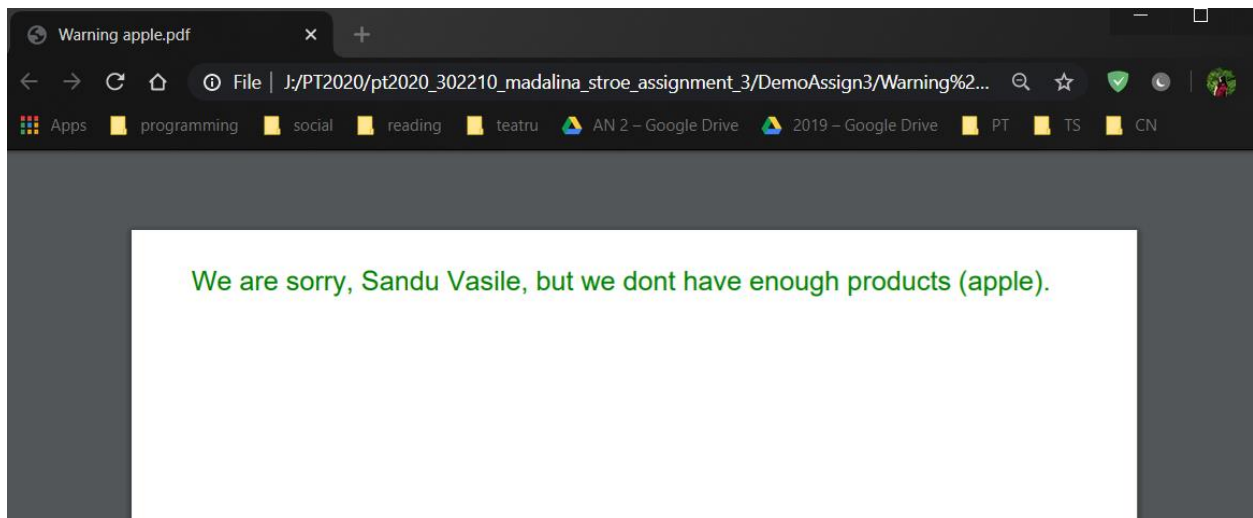


orderID	product	quantity
1	apple	5
2	lemon	5



orderID	clientID	totalPrice
1	2	5.0
2	2	10.0





~ Concluzii ~

Concluziile pentru această temă ar fi următoarele:

Deși a lucra cu baze de date nu este ceva în totalitate nou, modul în care am lucrat de această dată a adus cu sine concepte noi și interesante.

O arhitectură pe straturi (layer) a programului este un lucru bun, mai ales atunci când numărul de clase nu este unul redus, cum a fost aici.

Generarea rapoartelor pdf sub formă de tabel a fost un lucru nou și ușor de înțeles, până la urmă, iar după asta, a urmat organizarea estetică a tabelului.

Am lucrat cu mai multă ușurință cu obiecte de tip `StringBuilder`, care par a fi din ce în ce mai utile și mai potrivite în rezolvarea temelor și a altor programe.

La partea de SQL, am aflat că pentru a crea și executa o interogare este nevoie de obiecte de tip `Statement`. După ce a fost creată conexiunea cu baza de date, interogarea este inițializată.

Simbolurile `?` din interogare vor fi înlocuite cu valori din aplicație.

Rezultatul execuției interogării este stocat într-un `resultSet`. Fiecare element din `resultSet` corespunde unui rând din tabel. `ResultSet` poate fi iterat(am vorbit mai sus de cursor).

Proprietățile sau valorile coloanei din tabel pot fi extrase dacă numele coloanei este știut.

La închiderea conexiunii, trebuie să se închidă: `resultSet`, `statement`, `connection`.

~ Bibliografie ~

<https://docs.oracle.com/javase/7/docs/api/java/sql/ResultSet.html>

<https://docs.oracle.com/javase/7/docs/api/java/sql/Statement.html>

<https://www.codeproject.com/Questions/1239655/How-can-I-align-center-two-or-more-chunk-using-ite>

<https://stackoverflow.com/questions/30929733/itext-text-pdf-center-alignment>

<https://stackoverflow.com/questions/14373269/align-paragraph-at-the-center-of-the-page>

<https://stackoverflow.com/questions/16966629/what-is-the-difference-between-getfields-and-getdeclaredfields-in-java-reflectio>

<https://www.youtube.com/watch?v=pqwG36bHFEI>

<https://www.youtube.com/watch?v=fPr6HUDC37I>

https://utcn_dsrl@bitbucket.org/utcn_dsrl/pt-layered-architecture.git

https://utcn_dsrl@bitbucket.org/utcn_dsrl/pt-reflection-example.git