MINISTERUL EDUCAȚIEI ȘI CERCETĂRII ȘTIINȚIFICE

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE
CATEDRA CALCULATOARE

FUNDAMENTAL PROGRAMMING TECHNIQUES

ASSIGNMENT 3

Order Management

Student: Torzsa Gabriela

An studiu: II, semestrul II

Grupa: 30226

CUPRINS

I. Obiectivul temei

Obiectivul temei a fost sa proiectam o aplicatie care sa simuleze comenzile de produse ale unor clienti de la un anumit depozit. Detaliile despre fiecare enitate (client, produs, comanda) trebuiau retinute in tabele intr-o baza de date. Fiecarei entitati ii corespunde un tabel. Aplicatia permite introducerea de clienti noi, de produse, iar pe baza acestor informatii se pot face comenzi. In urma fiecare comenzi realizate, se va genera in format *.pdf* un fisier reprezentand *bonul fiscal* al comenzii respective, continand detalii despre persoana care a facut comanda, produsul comandat, cantitatea si pretul total.

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

Aplicatia este structurata dupa modelul prezentat in cerinta, model sub forma de straturi. Modelarea fiind facuta in aceasta maniera, aplicatia contine 5 pachete reprezentand nivelele de executie ale programului:

1. Pachetul *model* – contine clasele *Client, Product, Orders*, clase asociate cu cele 3 tabele din baza de date
2. Pachetul *Connection* – contine clasa *ConnectionFactory* care realizeaza conexiunea cu baza de date
3. Pachetul *dao* – contine clasele *AbstractDAO, ClientDAO, ProductDAO, OrdersDAO*.

   In clasa *AbstractDAO* am folosit tehnica de reflection pentru a crea o clasa generica care va contine implementarea tuturor interogarilor si toate metodele pentru a accesa baza de date: creare, adaugare, editare, stergere si gasire a obiectelor. Datorita genericitatii, interogarile pentru a accesa baza de date si a realiza modificari asupra ei in functie de tabelul pe care se doreste a fi modificat, se vor face in mod dinamic datorita tehnicii de reflection.
4. Pachetul *bll* – contine clasele *ClientBll, ProductBll, OrderBll* si este responsabil pentru definirea actiunilor fiecarui tabel. In fiecare clasa se suprascriu metodele din clasa *AbstractDAO,* acestea referindu-se la obiectul caracteristic clasei.
5. Pachetul *presention* – contine clasele *Controller, ClientView, ProductView, OrderView*, clase care implementeaza interfata grafica si interactiunea cu aceasta.

Clasa *Main* se afla in pachetul *default.*

Scenarii / cazuri de utilizare

La momentul rularii aplicatiei, se vor deschide 3 ferestre, fiecare apartinand unui tabel (Client, Product, Orders).

In fereastra corespunzatoare clientului, utilizatorul trebuie sa introduca un id, un nume si o adresa caracterizand clientul ce se doreste a fi introdus in tabelul din baza de date. Dupa introducerea datelor, utilizatorul poate sa adauge clientul, sa il stearga, sa actualizeze informatiile referitoare clientului si sa vizualizeze tabelul care contine toti clientii adaugati.

In fereastra corespunzatoare produsului, utilizatorul trebuie sa introduca un id, un nume pentru produs, pretul si cantitatea acestuia. Asemenea ferestrei dedicate clientului, utilizatorul poate sa adauge, sa stearga, sa actualizeze informatiile despre produs si sa vizualizeze tabelul care contine toate produsele adaugate.

In fereastra corespunzatoare fiecarei comenzi, utilizatorul trebuie sa aleaga clientul pentru care se face comanda, produsul dorit si trebuie sa introduca cantitatea pe care doreste sa o comande. In momentul in care utilizatorul alege din lista produsul dorit, acesta poate sa vizualizeze toate informatiile referitoare la acel produs, inclusiv cantitatea acestuia, pentru a evita cazul in care acesta doreste sa comande mai mult decat stocul existent. In caz de succes, dupa apasarea butonului de *order*, se va genera un bon fiscal / chitanta, care va contine toate detaliile despre cumparator, produsul comandat, pretul per bucata, respectiv pretul total.

Cazuri de esec:

1. Daca utilatorul doreste sa stearga un client sau un produs si nu introduce id-ul acestuia, stergerea nu se va efectua. Deci stergerea unui client sau produs se poate realiza doar pe baza id-ului.
2. In cazul in care utilizatorul doreste sa comande o cantitate mai mare dintr-un produs decat stocul existent, se va genera o fereastra cu un mesaj de erorare *„Invalid quantity”*.
3. Data fiind regula de cheie primara a bazelor de date, utilizatorul nu va putea introduce un client sau un produs cu un id deja existent in tabela fiecaruia.

III. Proiectare si implementare

Proiectarea claselor a fost facuta dupa modelul structurat pe nivele prezentat in cerinta, astfel fiecare pachet are rolul lui bine definit.

Pachetul *connection*:

1. Clasa *ConnectionFactory* este clasa in care se realizeaza conexiunea cu baza de date. Are ca atribute <Logger> LOGGER, <String> DRIVER, <String> DBURL, <String> USER, <String> PASS. Clasa contine un contructor, metoda *createConnection()* pentru realizarea conectiunii cu baza de date si metoda *close()* cu diferite argumente, pentru inchiderea conexiunii.

Pachetul *model*:

1. Clasa *Client* simuleaza un client care are ca atribute un *ID*, acesta fiind unic in tabel, reprezentand cheia primara a tabelului, un nume, *nameClient*, format din nume si prenume, si o adresa, *addressClient*, reprezentand adresa clientului. Clasa contine un constructor, metodele mutatoare si accesoare pentru atribute si metoda *toString()* pentru afisarea datelor.

2. Clasa *Product* simuleaza un produs vandut de un magazin care are ca atribute un *ID* unic, un nume, *nameProduct,* reprezentand numele produsului, un pret per bucata, *price*, si

numarul de bucati din produsul respectiv, *quantity.* Clasa contine 2 constructori, unul fara parametri, iar altul cu toate atributele date ca parametri, metodele mutatoare si accesoare pentru atribute si metoda *toString()* pentru afisarea datelor.

3. Clasa *Orders* simuleaza o comanda a unui client pentru un singur produs. O comanda

are ca atribute un ID unic, *id*, numele clientului care face comanda, *nameClient,* numele produsului care se doreste a fi comandat, *nameProduct* si cantitatea pe care clientul doreste sa o comande din respectivul produs, *quantity.* Clasa contine 2 constructori, unul fara parametri, iar altul cu toate atributele obiectului date ca parametri, metodele mutatoare si accesoare pentru atribute si metoda *toString()* pentru afisarea datelor.

Pachetul *dao*

1. Clasa *AbstractDAO* este clasa generica in care este folosita tehnica de reflection sicontine implementarea tuturor actiunilor care se pot efectua in baza de date, adica: metodele de tip *String* care creeaza interogarile si metodele aferente finalizarii sau utilizarii acestora: *findAll()* pentru afisarea tuturor liniilor din tabel, *insert()* pentru a insera o noua linie in tabel, *update()* pentru a actualiza datele deja existente dintr-o linie din tabel si *delete()* pentru a sterge o linie din tabel.

   Tot in aceasta clasa este implementata si metoda de creare a obiectelor, *createObjects.*

2. Clasele *ClientDAO, ProductDAO, OrdersDAO* sunt goale (nu implementeaza nimic), ele doar extind clasa *AbstractDAO* cu toate metodele implemenate in aceasta. Datorita tehnicii de reflection in functie de tabelul in care vrem sa lucram (sa adaugam sau sa modificam date) se executa metodele in functie de field-urile / parametri corespunzatori fiecarui tabel.

Pachetul *bll*

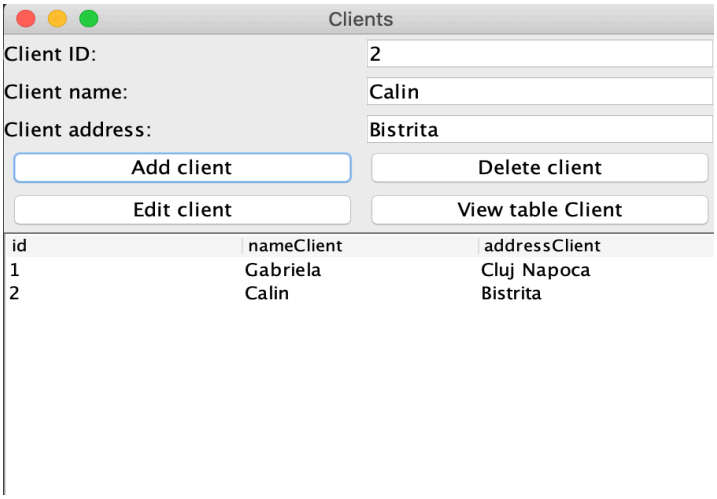
1. Clasa *ClientBll* contine metodele de inserare, editare si stergere a unui client in / din baza de date, in tabelul destinat clientilor.
2. Clasa *ProductBll* contine metodele de inserare, editare si stergere a unui client in / din baza de date, in tabelul destinat produselor.
3. Clasa *OrderBll* implementeaza 2 metode, metoda *createOrder* care creeaza comanda in cazul in care toate datele de intrare sunt valide si o introduce in tabela destinata comenzilor si metoda *generateBill* care genereaza bonul fiscal sub forma de fisier *.pdf*. In metoda *generateBill* se calculeaza pretul total in functie de cantitatea dorita si pretul per bucata.

Pachetul *presentation*

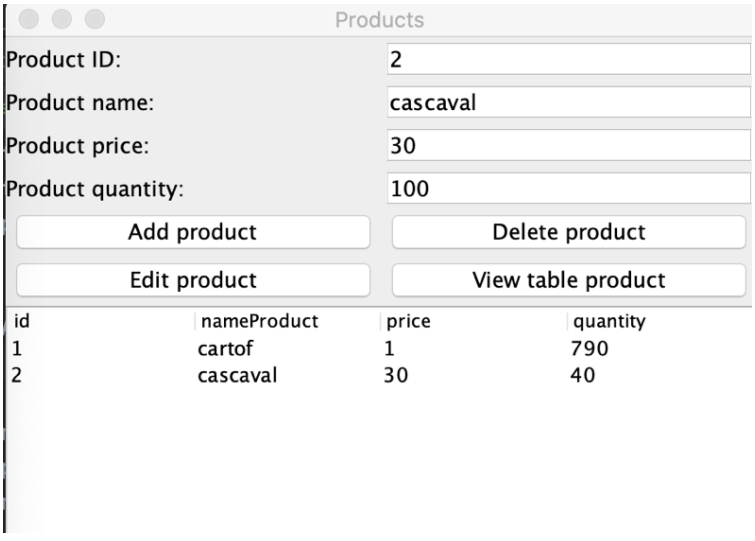
1. Clasa *Controller* contine un constructor cu interfetele grafice corespunzatoare fiecarui tabel.

In clasa *Controller* am implemenetat metodele *getTableHeader* si *getTable* folosind tehnica de reflection pentru a primi o lista cu proprietatile obiectului si apoi pentru a popula tabelul cu elementele din lista.
Clasa *Controller* mai implementeaza metodele pentru creearea tabelului de clienti si tabelului de produse, si metodele care vor realiza actualizarea fiecarui tabel in cazul in care se realizeaza modificari asupra lor.

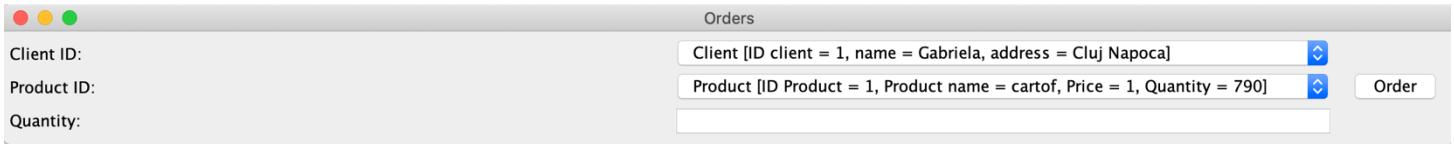
2. Clasa *ClientView* implementeaza interfata grafica cu utilizatorul si ascultatorii butoanelor destinate sa efectueze operatiile pe tabelul din baza de date. Pentru implementarea ascultatorilor am folosit expresii lambda avand scopul de a face codul mai inteligibil (un cod mai frumos). In momentul modificarii tabelului, in zona destinata afisarii tabelului se vor vedea modificarile, iar utilizatorul va putea lucra mai usor cu datele deja existente in tabel.



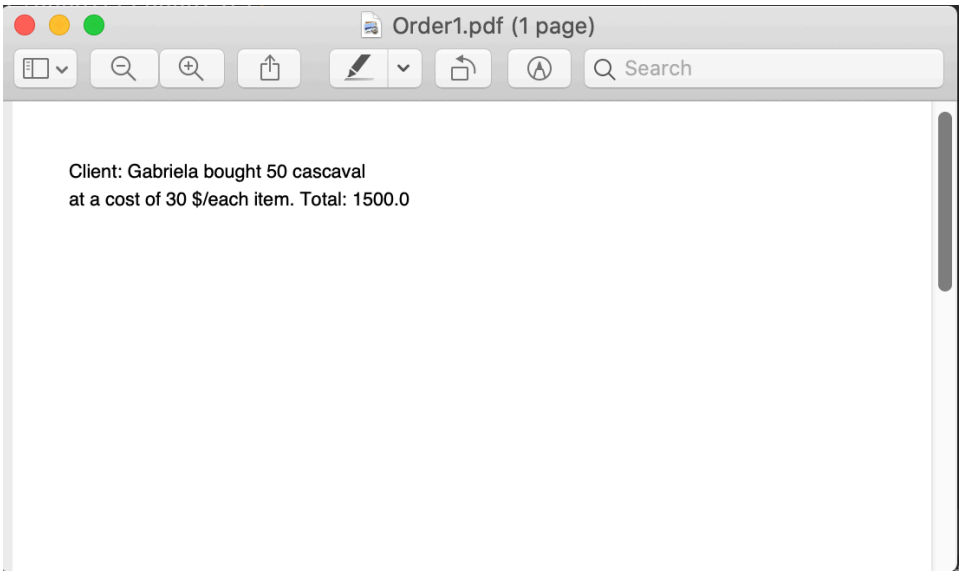
3. Clasa *ProductView* implementeaza interfata grafica cu utilizatorul si ascultatorii butoanelor destinate sa realizeze operatiile pe tabelul din baza de date. La fel ca si in cazul clasei *ClientView* am folosit expresii lambda. Modificarile efectuate asupra tabelului si tabelul in sine vor fi afisate in zona speciala afisarii tabelului.



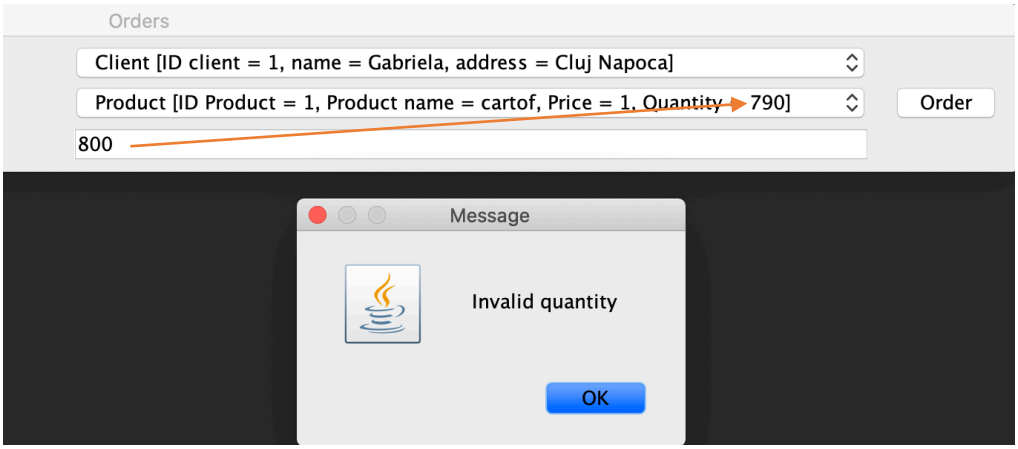
4. Clasa *OrderView* implementeaza interfata grafica pentru tabelul de comenzi, iar utilizatorul poate sa realizeze o comanda avand posibilitatea de a alege clientul care face comanda, produsul pentru care se face comanda si cantitatea. Listele cu clientii si produsele se reprezinta ca valori intr-un *JComboBox* in care prin metodele *setClients* si *setProducts* se actualizeaza datele curente fiecarui tabel.



Dupa apasarea butonului *order* se va genera un bon fiscal sub forma de fisier tip *.pdf* in care se va scrie numele clientului care a efectuat comanda, cantitatea si numele produsului comandat si detaliile despre pretul per bucata al produsului ales si pretul total al comenzii.

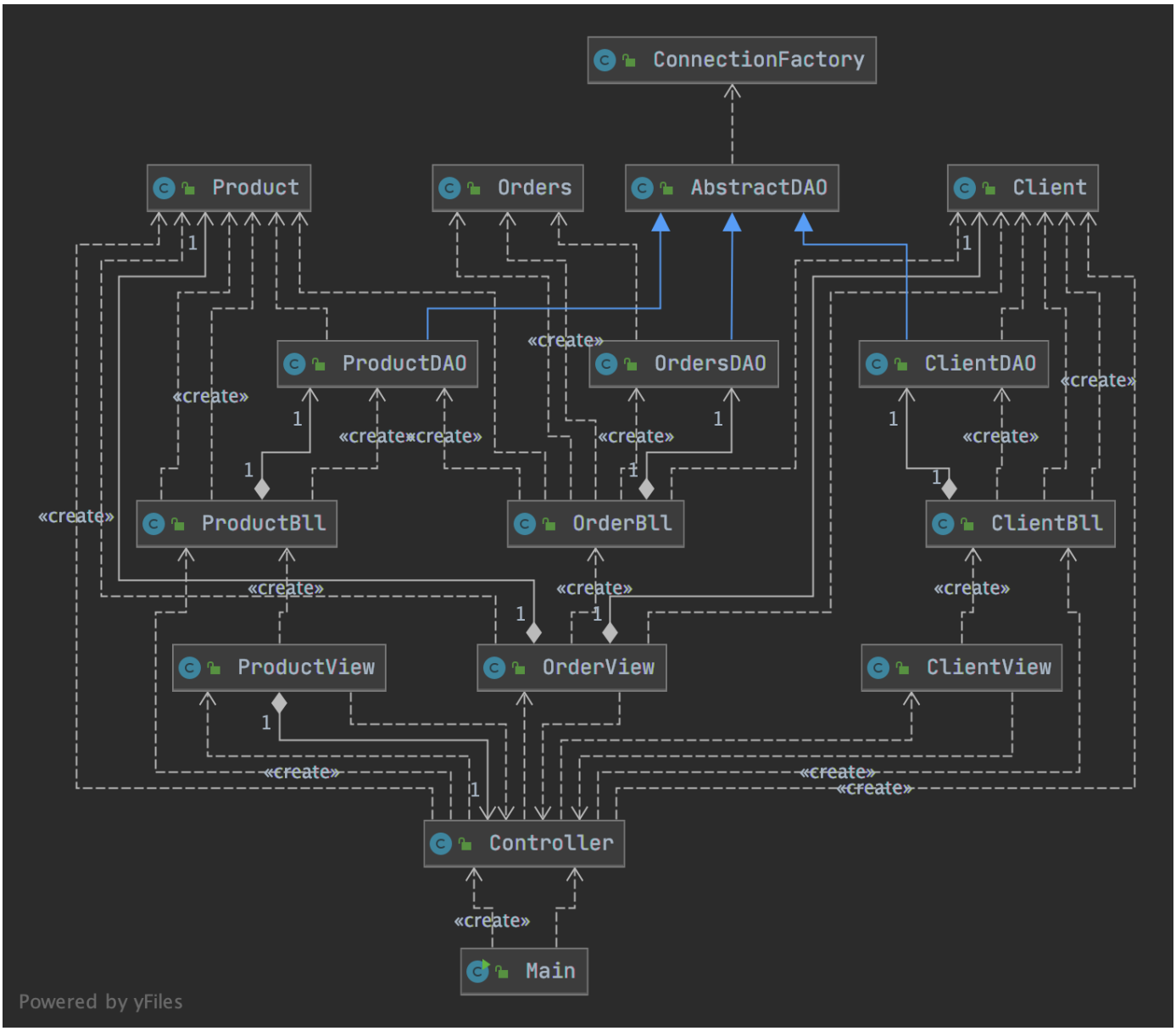


In cazul in care utilizatorul introduce o cantitate mai mare decat cea existenta se va afisa o fereastra cu un mesaj de erorare.



Clasa *Main* se afla in pachetul *default,* clasa in care cream un obiect de tip *Controller* in care se vor deschide cele 3 ferestre pentru interactiunea utilizatorului cu aplicatia.

Diagrama UML



IV. Rezultate

Pentru testarea aplicatiei m-am folosit de functia de *System.out.println()*. In final, am testat toate functionalitatile interactionand cu interfata grafica a aplicatiei, introducand clienti, produse, modificand atributele acestora si generand comenzi pentru a vedea ca aplicatia functioneaza corect.

V. Concluzii

Pentru a putea genera fisierul e tip *.pdf* si a folosi metodele aferente am descarcat de pe internet libraria *itextextpdf.jar.* Acest proiect a avut ca scop lucrul cu bazele de date si utilizarea tehnicii de reflection. Cu ajutorul schemelului de cod de pe *GitLab* am implementat restul aplicatiei, iar conexiunea cu baza de date si ideea de formare a interogarilor si a metodelor pentru efectuarea operatiilor a fost luata din suportul de prezentare al temei.

VI. Bibliografie

Sursele principale au fost stackoverflow, suportul de prezentare al temei, linkurile aferente aceteia si w3schools pentru reimprospatarea lucrului cu bazele de date.