Universitatea Tehnică Cluj-Napoca

Facultatea de Automatică și Calculatoare

Departamentul Calculatoare



**Proiect**

la disciplina

Tehnici de Programare

**Manager de comenzi**

**Pînzariu Răzvan**

**30222**

An academic: 2020 - 2021

Cuprins

[Obiectivul temei 2](#_Toc70069691)

[Analiza problemei, modelare, scenariu, cazuri de utilizare 3](#_Toc70069692)

[Proiectare 5](#_Toc70069693)

[Implementare 6](#_Toc70069694)

[Rezultate 9](#_Toc70069695)

[Concluzii 10](#_Toc70069696)

[Biografie 10](#_Toc70069697)

# Obiectivul temei

Obiectivul temei curente este de a implementa in Java un manager de comenzi (en. Order Management) ce dispune de o interfata grafica interactiva si o conexiune cu o baza de date existenta a unui depozit. Programul permite utilizatorului (managerului bazei) sa adauge clienti sau produse noi, sa le editeze, sa le stearga si sa plaseze comenzi pentru clientii si produsele deja inregistrate, precum si sa elibereze un bon fiscal pe urma acestora.

Pe plan secundar se vor rezolva:

* Analiza problemei si identificarea cazurilor de utilizare
* Proiectarea managerului in clase si pachete specifice tehnicii de programare alese
* Implementarea folosind clase cu functionalitati precise
* Testarea si vizualizarea corectitudinii programului

# Analiza problemei, modelare, scenariu, cazuri de utilizare

Bazele de date sunt prezente pretutindeni: de la depozite, la companii la retele de internet – orice spatiu digital ce are nevoie sa stocheze cantitati mari de informatii si relatiile dintre ele foloseste o baza de date. Interactiunea cu acestea poate fi uneori anevoioasa cand utilizatorul nu are cunostinte suficiente de mari pentru limbajul bazei folosite, fiind adesea nevoie de cel putin un specialist in fiecare context in care bazele de date sunt folosite. Pentru accesul persoanelor mai putin experimentate, propunem folosirea unei interfate grafice prin intermediul unui limbaj de legatura (aici Java) ce reduce complexitatea gestionarii datelor la cateva apasari de buton, disponibile oricui fara foarte mult efort.

Pentru a rezolva o astfel de problema trebuie asigurata simplitatea si intregimea programului folosit. Nu trebuie sa fie doar intuitiv, ci sa cuprinda si aproape toate functionalitatile pe care le-ar fi oferit un limbaj de gestionare al bazelor de date, intr-o forma mai simplificata. Printr-un astfel de program facem operatiile uzuale disponibile oricui, ramanand ca specialistul sa se ocupe doar de situatiile exceptionale.

Managerul de comenzi ia in calcul cateva date introduse de utilizator prin intermediul interfetei:

Pentru clienti: Pentru produse:

1. Nume 1. Nume
2. Adresa 2. Cantitate
3. Email 3. Greutate
4. Varsta Nume 4. Pret

Pentru comenzi:

1. Clientul ce efectueaza comanda
2. Produsul ce va fi achizitionat
3. Cantitatea produsului

Datele acceptate sunt doar valorile numerice pozitive si sirurile de caractere, urmand a fi verificate dupa standardele depozitului ulterior.

Interfata pune la dispozitie multe butoane impartite in trei cadre: la stanga sunt butoanele de adaugare, editare, stergere si examinare clienti, in mijlor sunt analog butoane cu aceleasi functii pentru produse, iar in dreapta butonul de a adauga un produs la comanda si cel ce finalizeaza comanda.

Interactiunea cu sistemul se reduce la urmatorii pasi:

1. A. Utilizatorul selecteaza un client sau un produs sau un client si un produs(pentru a plasa o comanda)

B. Utilizatorul introduce datele unui client nou sau al unui produs

C. Utilizatorul vizualizeaza toti clientii

2. A. Utilizatorul editeaza datele clientului/produsului selectat si apasa butonul de editare

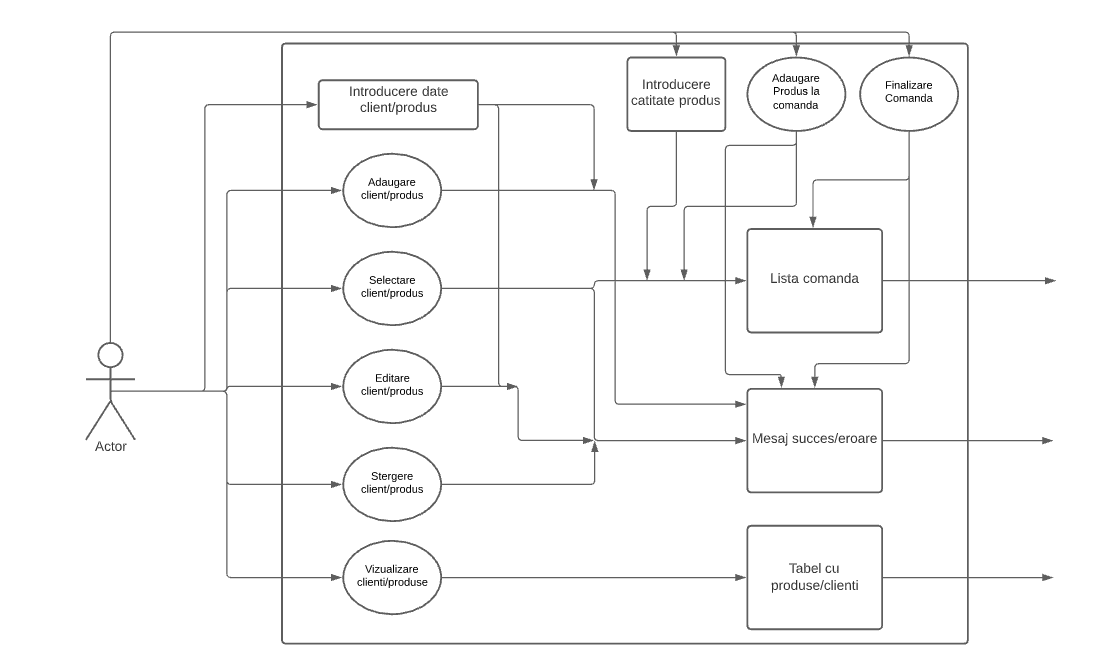
B. Utilizatorul sterge clientul/produsul selectat

C. Utilizatorul adauga produsul la comanda clientului

3. Utilizatorul continua sa adauge produse la comanda si o finalizeaza.

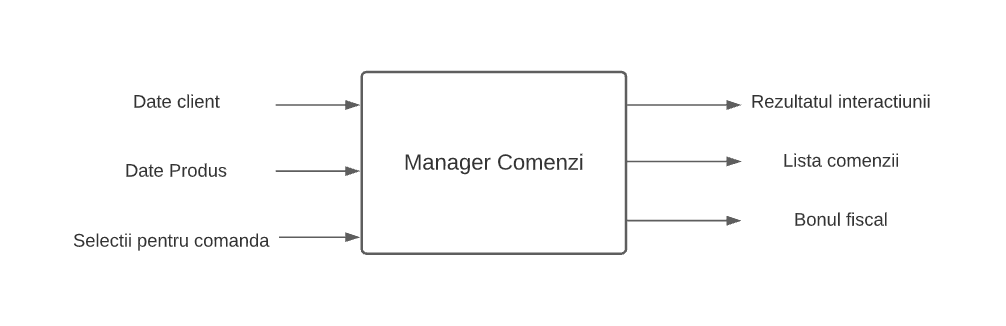
Mesaje de succes sau eroare corespunzatoare se vor afisa in cadrul fiecarei operatii efectuate.

Diagrama de mai jos descrie succint cazurile de utilizare:



# Proiectare

La cel mai inalt nivel avem un manager de comenzi ce primeste datele despre client/produs sau selectiile pentru comanda si o operatie specifica, oferind la iesire un mesaj ce reflecta interactiunea cu depozitul, lista comenzii curente si un bon fiscal pentru comanda, precum arat si diagrama atasata:



Organizarea interna se rezuma la interactiunile dintre sase pachete:

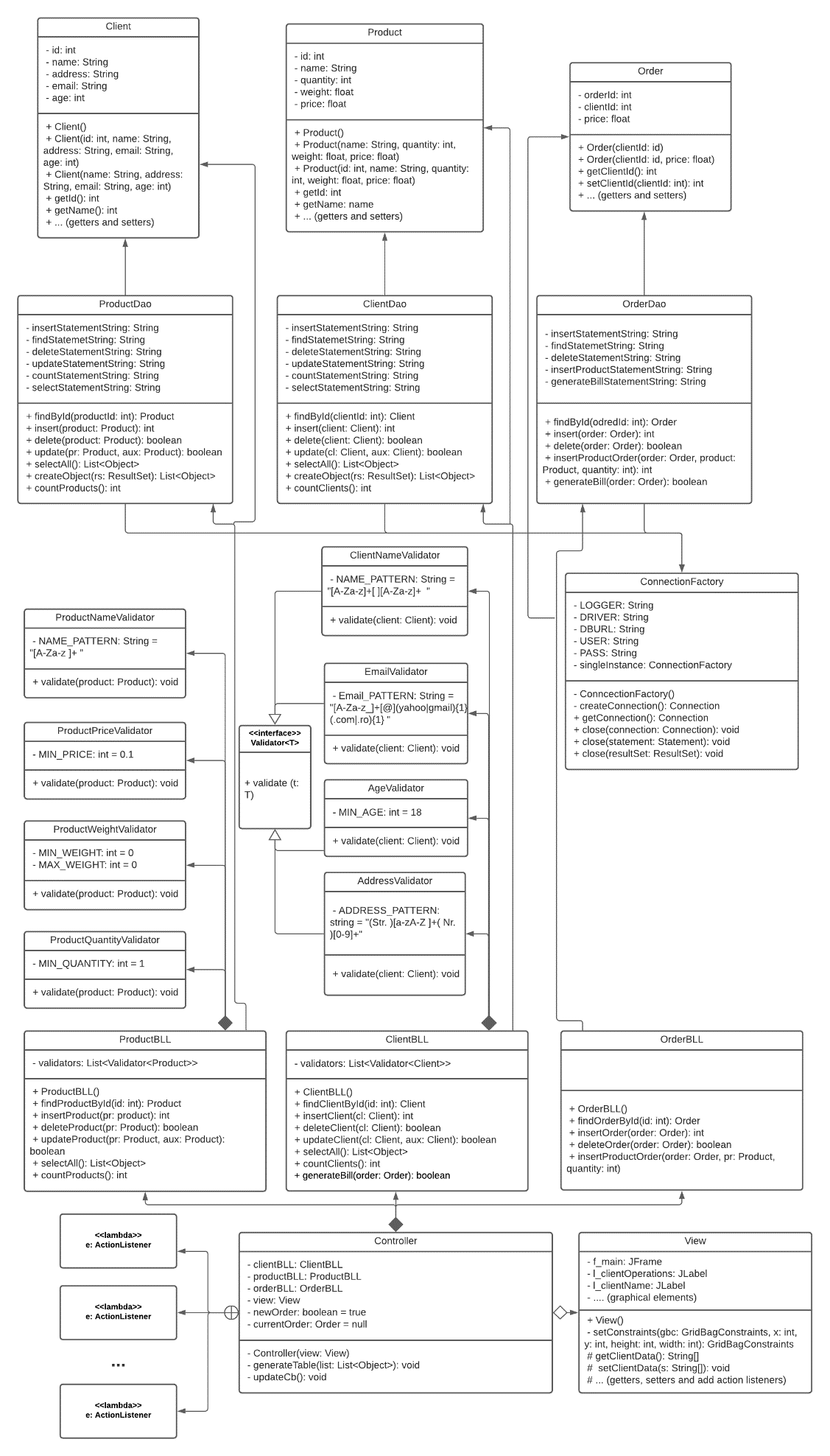
1. **presentation** – contine clasele ce se ocupa de interactiunea cu interfata grafica, structurat dupa un model arhitectural MVC (Model-View-Controller)
2. **model** – contine clasele principale(client, produs, comanda) ale programului si accesul la datele acestora prin gettere si settere
3. **dao –** contine interactiunile cu baza de date pentru fiecare clasa specificata in model. Sunt cuprinse comenzile in sql ce vor fi efectuate si rezultatele acestora in cadrul operatiunilor de cautare, editare, stergere etc.
4. **connection –** contine clasele ce realizeaza conexiunea cu serverul bazei de date si returneaza aceasta conexiune la cererea altor clase
5. **bll –** contine atat clase de validare (pentru datele introduse conform standardelor depozitului) cat si clasele de Business Logic pentru client produs si comanda ce au in principal scopul de a valida datele introduse si a apela metodele din pachetul dao.
6. **start –** contine clasa main ce instantiaza un “controller”, pornind astfel intreaga aplicatie

În cadrul pachetului gui avem o altă subdiviziune datorată arhitecturii MVC. Sumar explicat, Controller-ul gestionează datele primite în cadrul interfeței(clasa View) și le trimite mai departe catre Model pentru procesare. Același Controller primește rezultatele returnate de Model si actualizează corespunzător interfața (View).

In cadrul conexiunii (“ConnectionFactory”) am folosit o arhitectura de tip instanta singulara (“singleton pattern”) cu scopul de a restrictiona instantierea clasei la o unica instanta (aici “singleInstance” prezenta intre variabile) fara a mai putea fi apelata de alt undeva. “Constructorul” este astfel privat si apelat o singura data de propria sa clasa, fiind disponibile “gettere” statice prin care alte parti din program obtin conexiunea la baza de date.

Intregul program foloseste o arhitectura de tip “layered”(“3 tier architecture” – un tip concret de “n-tier architecture”) in care prezentarea, procesarea si managementul datelor sunt separate fizic. Dispunem astfel de “layer-ul” de prezentare (interfata grafica), “layer-ul” de acces de date (“data access”) reprezentat de pachetul dao si “business layer”(logica aplocatiei) reprezentat de pachetul bll. O astfel de arhitectura permite programatorului sa actualizeze un specific “layer” sau sa adauge unul nou fara a altera intreaga functionalitate a programului, fiind in special utila cand se cere o conexiune intre client si server.

Relatiile dintre clase sunt reprezentate de diagrama UML:



# Implementare

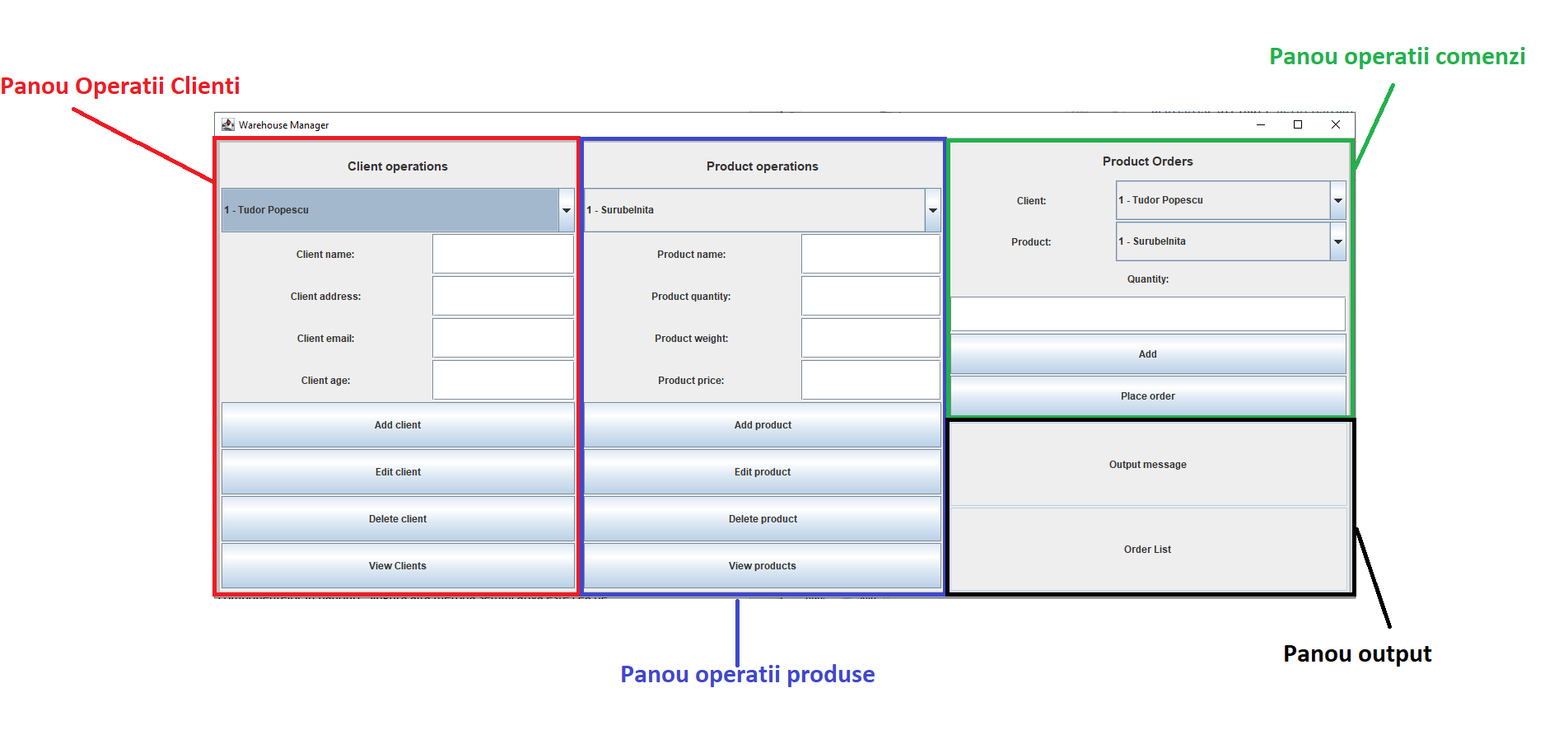
Pentru implementare s-au folosit urmatoarele clase:

1. Start (pachet – start)

Continand metoda “main” a proiectului clasa Start initializeaza un “Controller” si un “View”, afisand astfel interfata grafica si punand la dispozitie toate functionalitatile proiectului.

2. View (pachet - presentation)

“View” este clasa grafica principala a proiectului, continand toate elementele de design(variabile instanta) precum si metode ce permit altor clase din acelasi pachet sa preia datele introduse de utilizator sau sa modifice anumite aspecte cu scopul comunicarii unui “output” de catre sistem.

 In constructor sunt stabilite detalii privind aspectul fizic(aliniere, format text, dimensiuni etc.) precum si organizarea componentelor in panouri. Singura alta metoda semificativa este cea de “setConstraints” care permite utilizatorului sa modifice constrangerile unui “gridBagLayout” pentru fiecare componenta in parte, oferind un mod mult mai flexibil de lucru.

Pentru o mai usoara intelegere a interfetei, s-a facut o separare in patru panouri: operatii clienti – rosu, operatii produse – albastru, operatii comenzi – verde, output – negru. Panourile ce gestioneaza clientii si produsele sunt similare. Fiecare dispune in partea de sus de un “combo box” ce permite selectarea unui client/produs specific. Odata facuta selectia, se vor afisa datele obiectului in campurile de text etichetate de mai jos. Utilizatorul poate apoi sa stearga clientul/produsul selectat sau sa il editeze modificand campuile de text si actionand butonul corespunzator. Utilizatorul poate introduce date complet noi in campurile de text si apoi adauga/crea un nou client/produs apasand butonul “Add client/product”. Pentru a vizualiza toti clientii sau toate produsele de care dispune baza de date, butoanele de “View” vor deschide o noua fereastra cu un tabel ce contine toate datele respective.

Panoul de operatii comenzi dispune de doua “combo boxuri” etichetate in partea de sus prin care se selecteaza clientul ce plaseasa comanda si produsul pe care vrea sa il adauge comenzii. Ca si in cadrul operatiilor anterioare, odata selectat un client sau produs se vor afisa datele sale in campurile de text din panourile operatii clienti sau operatii produse. In campul de text “Quantity” se va afisa cantitatea maxima a produsului selectat, utilizatorul modificand acest camp cu cantitatea dorita. Prin butonul “Add” se adauga produsul la comanda: din acest punct nu se mai poate selecta alt client, doar alte produse pana la finalizarea comenzii(butonul “Place order”).

Panoul de output contine doua campuri de text needitabile: in primul se va afisa statutul comenzii (daca s-a efectuat cu succes sau nu, daca unele date introduse sunt gresite, daca e o eroare la baza de date etc.), iar in celalalt lista de produse adaugate la comanda curenta, precum si cantitatile lor.

3. Controller(pachet - presentation)

Clasa “Controller” gestioneaza datele introduse de utilizator prin interfata grafica si le transmite mai departe spre celelalte clase pentru a se efectua operatiile dorite (identificate dupa butonul apasat). Clasa contine cate o expresie lambda pentru fiecare buton si doua metode semnificative: “generateTable” care primeste ca si argument o lista de obiecte si genereaza un tabel folosind tehnici de reflexie cu toate acele obiecte si metoda “updateCb” ce mentine actualizate “combo boxurile” odata ce are loc o operatie in sistem.

Prin tehnici de reflexie, “generateTable” primeste orice clasa si verifica variabilele instanta ale acesteia, generand un sir de siruri de caractere pentru titlurile coloanelor tabelului. Prin aceleasi tehnici, se acceseaza valoarea fiecarei variabile, chiar daca este privata si se adauga in matricea de siruri de caractere ce va caracteriza datele introduse in randurile tabelului. Este apelata apoi interfata grafica pentru afisarea propriu-zisa.

4. Product (pachet - model)

Clasa product contine informatii despre produsele din baza de date: id, nume, cantitate, greutate si pret. Sunt prezente doar “gettere” si “settere” precum si un “override” al metodei “toString” pentru o afisare mai corecta in “combo boxuri”;

5. Order (pachet - model)

In order sunt stocate informatii despre comenzi: id, id client si pret. Sunt prezente doar “gettere” si “settere”.

De mentionat este ca desi o comanda poate avea mai multe produse, din clasa “Order” nu reiese acest lucru. In baza de date este prezent un tabel de jonctiune intre produse si comenzi ce face aceasta legatura.

6. Client (pachet - model)

Clasa product contine informatii despre produsele din baza de date: id, nume, adresa, eMail si varsta. Sunt prezente doar “gettere” si “settere” precum si un “override” al metodei “toString” pentru o afisare mai corecta in “combo boxuri”;

7. ProductDao (pachet - dao)

Clasa contine datele de acces pentru produse: comenzile in sql specifice si metodele ce le apeleaza si returneaza rezultatul lor. Se pot astfel: insera produse, gasi in functie de id, sterge, edita, numara si selecta cu totul. Fiecare astfel de operatie are metoda specifica, fiind prezenta in plus o metoda de “createObject” care foloseste rezultatele primite de o selectie a intregului tabel si le transforma intr-o lista de obiecte ce va fi folosita pentru crearea tabelelor din interfata grafica.

8. OrderDao (pachet - dao)

Clasa contine datele de acces pentru comenzi(orders): comenzile in sql specifice si metodele ce le apeleaza si returneaza rezultatul lor. Se pot astfel: insera comenzi, gasi in functie de id, sterge,insera jonctiuni intre produse si comenzi si genera bonuri de plata. Fiecare astfel de operatie are metoda specifica, iar metoda “generateBill” deschide un fisier text nou ce contine id-ul comenzii si scrie in el pretul total odata ce utilizatorul a terminat de adaugat produse.

9. ClientDao (pachet - dao)

Clasa contine datele de acces pentru clienti, functionand aproape similar cu ProductDao.

10. ConnectionFactory (pachet - connection)

ConnectionFactory realizeaza conexiunea cu serverul bazei de date. Avand adresele presetate ca si variabile instanta finale se creaza o instanta unica a conexiunii ce este apoi oferita claselor ce o solicita printr-un “getter”. Sunt prezente si metode pentru inchiderea conexiunii si a celorlalte obiecte ce tin de managementul bazei de date.

11. ProductBLL (pachet - bll)

ProductBLL valideaza apoi foloseste clasele din pachetul dao pentru a efectua modificari in baza de date. Mesaje de eroare specifice sunt returnate.

12. OrderBLL (pachet - bll)

OrderBLL foloseste clasele din pachetul dao pentru a efectua modificar in baza de date, returnand mesaje de eraore specifice.

13. ClientBLL (pachet - bll)

Functioneaza similar ca si “ProductBLL”.

14. Validator (pachet – bll.validators)

Interfata generica validator contine o singura metoda: validate pentru obiectul generic primit. Aceasta permite crearea numeroaselor clase de tip “Validator” ce verifica daca datele introduse de utilizator coincid cu standardele depozitului. Sunt prezente astfel: “ProductWeightValidator”, “ProductQuantityValidator”, “ProductPriceValidator”, “ProductNameValidator”, “EmailValidator”, “ClientNameValidator”, “AgeValidator” si “AddressValidator”.

Cum toate aceste clase au functionalitati similare, prezentarea uneia va duce la intelegerea intuitiva a oricareia.“AgeValidator” are tipul generic “Client”, iar metoda sa de validare primeste un client si verifica daca varsta aceasta este mai mare decat varsta minima permisa pentru inregistrarea in depozit (18 ani). Daca varsta e intre 0 si 17 ani (inclusiv) un mesaj de “Trebuie sa ai peste 18 ani sa fii client!” este transmis (si afisat de alte clase). Daca varsta este mai mica decat zero, caz invalid se transmite un mesaj “Varsta nu poate fi un numar negativ!”. Mesajele sunt erori de tip “IllegalArgumentException”

# Rezultate

S-a testat pe rand fieare functionalitate pentru mai multe date, inclusiv cazuri de eroare evidente sau rare, fiind verificata consistenta bazei de date odata ce programul este inchis si redeschis de oricate ori (cele doua sunt independente fizic). Orice actualizare este trasata in baza de date, iar valorile de iesire sunt verificate in functie de schimbarile ce au loc in sistem.

# Concluzii

Putem folosi limbajele si tehnicile de programare pentru a usura lucrul in anumite domenii mai complexe si a le face disponibile utilizatorilor obisnuiti, ce nu au nevoie astfel de ani de experienta pentru a invata sa le gestioneze. De la un manager de comenzi, la gestionarea unui computer sau a unor cai de date aeriene, interfetele grafice interactive si mecanismele din spatele lor reduc semificativ timpul necesar intelegerii unor concepte, lasand specialistii sa se ocupe de probleme mai serioase unde pot aduce beneficii mai mari.

# Bibliografie

1. Cursurile și prezentarea suport de la disciplina Tehnici de Programare

2. <https://docs.oracle.com/en/> - documentatia limbajului Java

3. <http://www.java2s.com/> - tutoriale si exemple de cod

4. <http://www.newthinktank.com/2012/09/java-reflection-video-tutorial/> - exemple tehnici de reflexie