UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

ORDER MANAGEMENT

Facultatea de Automatică și Calculatoare
Departamentul Calculatoare
Curs: Tehnici de Programare Fundamentale
An universitar 2020-2021

Student:
Bîrluțiu Claudiu-Andrei
CTI-ro, An 2, Grupa 30226

Cuprins

1.Problema și soluția dată

În zilele noastre consumerismul a ajuns la cote înalte. Numărul lanțurilor de magazine a crescut, produsele sunt din ce în ce mai diverse și pe de altă parte și concurența a crescut. Din aceste motive se pune mult accentul pe publicitate și încredere, de aceea este nevoie ca site-urile sau depozitele de produse să gestioneze cât mai bine stocul de produse și comenzile plasate de către clienți.

În aceste condiții este nevoie de baze de date, în care producătorii/vânzătorii să țină evidența produselor, a clienților și comenzilor plasate de aceștia. Având în vedere faptul că actualizarea bazelor de date se poate face în principiu de cei care știu sintaxa comenzilor (SQL, MicrosoftSql), este nevoie de o aplicație interactivă care să permită oricărui utilizator să gestioneze baza de date prin intermediul unei interfețe grafice intuitive și ușor de folosit, fără a scrie comenzi.

2.Obiectivul temei

Obiectivul principal se referă la proiectarea și implementarea unei aplicații de management a comenzilor pentru un depozit. Acesta va permite gestionarea unei baze de date prin intermediul unei interfețe grafice în care utilizatorul poate să introducă de la tastatură valorile pe care vrea să le scrie în baza de date și prin apăsarea unor butoane să realizeze operații specifice. Utilizatorul poate gestiona prin intermediul acestei aplicații în principiu 3 tabele de date. Utilizatorul poate să vadă toți clienții din baza de date, poate să îi șteargă, să introducă noi clienți sau să le modifice celor introduși deja, anumite informații. De asemenea, poate manipula produsele stocate în baze de date, să șteargă din ele sau să le modifice. Cel mai important, utilizatorul poate să creeze comenzi prin selecția clientului și a produselor împreună cu cantitatea. În momentul finalizării comenzii se va crea o factură cu produsele cumpărate și prețul pe care să-l achite cumpărătorul.

Obiectivele secundare, ce reprezintă pașii de urmat în îndeplinirea obiectivului principal, se regăsesc în următorul tabel. Acestea vor fi abordate în amănunt în următoarele capitole ale acestui document.

Obiectiv	Scurtă descriere	Capitol
Analiza problemei și identificarea cerințelor	-analiza se referă la descompunerea problemei în componente mai mici prin procesul de abstractizare. -se poate vorbi și de o inginerie a cerințelor care stabilește serviciile cerute sistemului precum și constrângerile.	3. Analiza problemei, modelare, scenarii, cazuri de utilizare
Proiectarea sistemului de management al comenzilor	-obținerea unui sistem bun prin utilizarea unor resurse existente și noi resurse. -reprezintă o activitate de tipul divide-and-conquer. -presupune folosirea paradigmei OOP, realizarea de diagrame UML, proiectarea claselor etc.	4. Proiectare
Implementarea sistemului de management al comenzilor	-presupune dezvoltarea și scrierea codului în limbajul de programare Java pentru clasele și metodele acestora.	5. Implementare
Testarea sistemului de management al comenzilor	-realizarea unor operații prin intermediul interfeței și observarea modificării tabelelor din baza de date	6. Rezultate

Tabel 1-Obiective secundare

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

1.1 Analiza problemei

Este important să înțelegem problema și domeniul ei. Procesul de analiză are drept scop descompunerea problemei mari în componente mici și de înțeles.

Domeniului problemei prezente se concentrează în jurul gestiunii bazelor de date, prin operații de manipulare și interogare a bazelor de date.

Prin interogarea unui tabel se înțelege selecția datelor dintr-un tabel care corespund unor criterii date de utilizator.

Prin manipulare se înțelege actualizarea tabelelor din baza de date prin introducerea de noi date, modificarea unor informații pentru datele existente sau excluderea lor definitivă din tabel.

De asemenea, o proprietate destul de importantă a bazelor de date trebuie avută în vedere, și anume legăturile dintre tabele și realizarea unor interogări pe baza relaționării dintre acestea.

Pentru creare bazei de date s-a folosit MySql Workbench, iar prin instructiuni de Data Definition Language s-au definit tabele necesare aplicației de management. S-a identificat necesitatea existenței a 4 tabele: tabelul *client* care contine clienții depozitului, tabelul *product* cu produsele din stoc, tabelul *order* cu comenzile generate și un tabel *soldproducts* pentru stocarea informațiilor despre produsele vândute la nivelul fiecărei comenzi.

În ceea ce privește aspectul interfeței simulatorului, interacțiunea cu utilizatorul trebuie sa fie cât mai prietenoasă și intuitivă astfel încât operațiile de inserare și interogare a tabelelor să se realizeze cât mai ușor. De asemenea, utilizatorii trebuie să fie atenționați în momentul în care introduc date invalide și să fie informați de greșelile pe care le-au făcut în timpul operației efectuate.

1.2 Modelare

După determinarea domeniului problemei, datele și procesele sunt translatate prin abstractizare în domeniul soluției ducând la modele de reprezentare, algoritmi și programe.

Folosind modelarea OOP se poate facilita alinierea obiectelor din lumea reală cu obiectele software, deoarece obiectele cu relațiile și interacțiunile din lumea reală relaționează și interacționează similar și într-o aplicație software. Clasele din implementare pot fi inspirate din domeniul problemei (nume similare, relații și responsabilități similare) ceea ce va asigura o mai bună înțelegere a designului și a implementării.

În cadrul sistemului de management al comenzilor pentru un depozit s-au creat clase cu denumiri specifice: *Client, Product, Order*. Relațiile dintre aceste obiecte s-au corelat cu situația din lumea reală. În cadrul aplicației, un client când cumpără un produs, cantitatea produsului respectiv va scădea din stocul depozitului, iar clientul trebuie să plătească produsul respectiv în funcție de cantitatea cumpărată. De asemenea, în momentul achitării acestuia, clientul primește o factură sau un bon fiscal în care apar informațiile despre produs și suma de plată. Acestea demonstrează consistența procesului de cumpărare a unui produs.

Denumirea sugestivă a claselor facilitează înțelegerea lor și definirea atributelor corespunzătoare.

1.3 Scenarii și cazuri de utilizare

Identificarea cerințelor și setarea lor clară esențiale și trebuie realizate riguros înainte de proiectarea și implementare programului software. Se poate vorbi chiar de o inginerie a cerințelor care se ocupă cu stabilirea serviciilor cerute sistemului și a constrângerilor(acest proces este unul ciclic). În continuare mă voi axa pe două etape ale acestui proces: elicitarea cerințelor și specificarea cerințelor în cadrul temei de laborator.

Elicitarea cerințelor: descoperirea cerințelor sistemului: activități precum interviuri și analiză bazată pe scenarii.

Specificarea cerințelor: implică modelarea riguroasă a cerințelor: use-case, diagrame UML

Din enunțul problemei reies următoarele cerințe pe care trebuie să le îndeplinească sistemul:

a) Existența unei interfețe grafice prin care utilizatorii pot să modifice tabela clienților dintr-o baza de date prin inserări de noi clienți, modificarea informațiilor celor existenți, ștergerea de clienți și afișarea tuturor clienților în interfața grafică.

b) Existența unei interfețe grafice prin care utilizatorii pot să modifice tabela produselor dintr-o baza de date prin inserări de noi de produse, modificarea informațiilor celor existente, ștergerea de produse și afișarea tuturor produselor în interfața grafică.
c) Existența unei interfețe grafice prin care utilizatorii pot să selecteze un client, iar apoi să selecteze produsele pe care le cumpără și să introducă această informație în baza de date , în tabela destinată comenzilor.
d) La finalul unei comenzi să se genereze o factură cu produsele cumpărate

În următorul tabel sunt definite cerințele sistemului:

	Cerințe
Cerințe funcționale (descriu ce trebuie să facă sistemul)	✓ Sistemul de management al unui depozit le permite utilizatorilor să insereze în interfața grafică datele unui nou client și să îl introducă în baza de date ✓ Sistemul de management al unui depozit le permite utilizatorilor să ceară afișarea tuturor clienților din baza de date. ✓ Sistemul de management al unui depozit le permite utilizatorilor să selecteze și să seteze noile valori pentru câmpurile unui client existent în baza de date și să îl actualizeze. ✓ Aplicația permite utilizatorilor să selecteze un client și să îl șteargă din baza de date ✓ Sistemul de management al unui depozit le permite utilizatorilor să insereze în interfața grafică datele unui nou produs și să îl introducă în baza de date ✓ Sistemul de management al unui depozit le permite utilizatorilor să ceară afișarea tuturor produselor din baza de date. ✓ Sistemul de management al unui depozit le permite utilizatorilor să selecteze și să seteze noile valori pentru câmpurile unui produs existent în baza de date și să îl actualizeze. ✓ Aplicația permite utilizatorilor să selecteze un produs și să îl șteargă din baza de date ✓ Aplicația permite utilizatorilor să realizeze o comandă prin selecția clientului și a produselor pe care acesta le cumpără. ✓ Aplicația va genera a finalul comenzii o factură în format pdf cu datele corespunzătoare
Cerințe non-funcționale (descriu cu ce calitate ar trebui să funcționeze sistemul)	○ Sistemul de management al unui depozit să fie intuitiv și ușor de folosit ○ Semnalizarea unei nereguli să se facă într-un fel cât mai prietenos și ușor de identificat de către utilizator. Se va avea în vedere și restricțiile impuse cu privire la email, gen și număr de telefon. De exemplu un număr valid de telefon în cepe cu 07 și conține 10 cifre
Constrângeri (descriu limitele de dezvoltare a sistemului)	▪ Operațiile pe tabele se fac pe rând. Nu exista simultaneitate

Tabel 2 Definirea cerințelor sistemului

Use-case UML capturează grafic actorii sistemului, cazurile de utilizare și relațiile dintre acestea.
Actorii- orice tip de utilizator

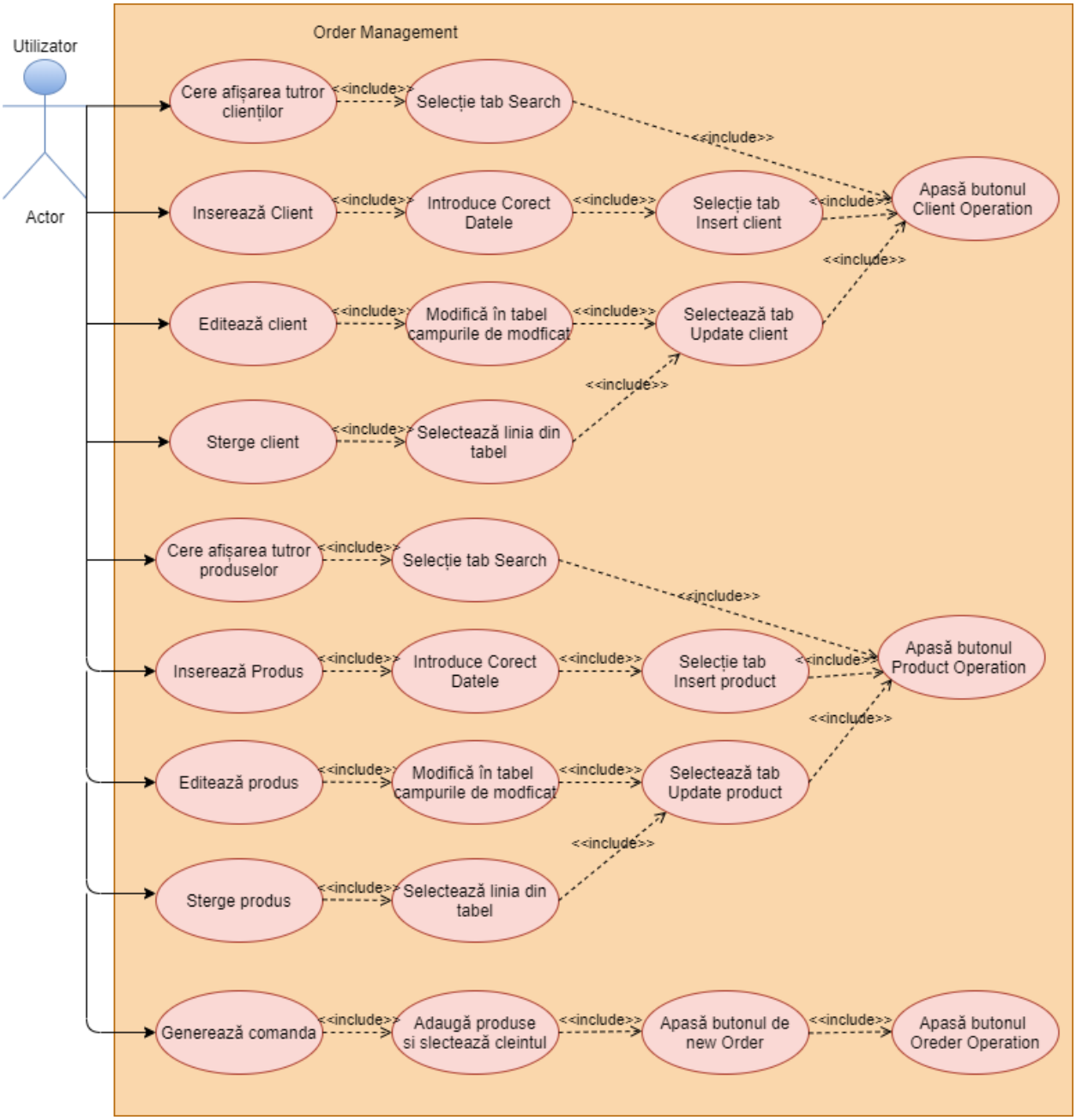

Figure 1 Use-Case UML

Cazurile de utilizare ale aplicației:

Use case: afișare clienți
Actor principal: utilizator
Princpalul scenariu de succes:

1. Utilizatorul apasă din fereastra principală butonul Client Operation
2. Utilizatorul selectează tab-ul *Search*
3. Utilizatorul apasă butonul *find all* din fereastra ClientWindow

Secvențe secundare

Use case: inserare client
Actor principal: utilizator
Princpalul scenariu de succes:

1. Utilizatorul apasă din fereastra principală butonul Client Operation
2. Utilizatorul selectează tab-ul *Insert Client*
3. Utilizatorul introduce id-ul, numele, genul(M,m,F sau f), email-ul, telefonul și adresa clientului nouâ
4. Utilizatorul apasă pe butonul *Insert Client*

Secvențe secundare

- Utilizatorul introduce id deja folosit. Va primi un mesaj de informare
- Utilizatorul introduce un email deja existent in baza de date sau un email invalid. Va fi informat si poate modifica câmpul respectiv
- Utilizatorul introduce un număr de telefon deja existent în baza de date sau un număr de telefon invalid (valid 07 urmat de încă 8 cifre). Va fi informat și poate modifica câmpul respectiv

Use case: editare client
Actor principal: utilizator
Princpalul scenariu de succes:

1. Utilizatorul apasă din fereastra principală butonul Client Operation
2. Utilizatorul selectează tab-ul *Update Client*
3. Utilizatorul modifică unul din câmpurile din tabel corespunzătoare clientului căruia îi vrea să modifice informațiile.
4. Utilizatorul apasă pe butonul *Edit Client*

Secvențe secundare

- Utilizatorul introduce informații ce nu corespund (cele indicate use case-ului anterior). Va primi un mesaj de informare, iar baza de date nu se va modifica până când nu introduce date valide

Use case: ștergere clienți
Actor principal: utilizator
Princpalul scenariu de succes:

1. Utilizatorul apasă din fereastra principală butonul Client Operation
2. Utilizatorul selectează tab-ul *Update Client*
3. Utilizatorul setează clientul pe care doreste sa îl șteargă și apasă butonul de *Delete Client*

Secvențe secundare

- Clientul nu poate fi șters fiind implicat într-o relație cheie primara-cheie străină. Utilizatorul va fi informat

Use case: afișare produse
Actor principal: utilizator
Princpalul scenariu de succes:

1. Utilizatorul apasă din fereastra principală butonul Product Operation
2. Utilizatorul selectează tab-ul *Search*
3. Utilizatorul apasă butonul *find all* din fereastra ProductWindow

Secvențe secundare

Use case: inserare produs
Actor principal: utilizator
Princpalul scenariu de succes:

1. Utilizatorul apasă din fereastra principală butonul Product Operation
2. Utilizatorul selectează tab-ul *Insert Product*
3. Utilizatorul introduce id-ul, numele, cantitatea și prețul
4. Utilizatorul apasă pe butonul *Insert Product*

Secvențe secundare

- Utilizatorul introduce id deja folosit. Va primi un mesaj de informare

Use case: editare produs
Actor principal: utilizator
Princpalul scenariu de succes:

1. Utilizatorul apasă din fereastra principală butonul Product Operation
2. Utilizatorul selectează tab-ul *Update Product*
3. Utilizatorul modifică unul din câmpurile din tabel corespunzătoare produsului căruia îi vrea să modifice informațiile.
4. Utilizatorul apasă pe butonul *Edit Product*

Secvențe secundare

- Utilizatorul introduce informații ce nu corespund (cele indicate use case-ului anterior). Va primi un mesaj de informare, iar baza de date nu se va modifica până când nu introduce date valide

Use case: ștergere produs
Actor principal: utilizator
Princpalul scenariu de succes:

1. Utilizatorul apasă din fereastra principală butonul Product Operation
2. Utilizatorul selectează tab-ul *Update Product*
3. Utilizatorul setează clientul pe care dorește sa îl șteargă și apasă butonul de *Delete Product*

Secvențe secundare

- Produsul nu poate fi șters fiind implicat într-o relație cheie primara-cheie străină. Utilizatorul va fi informat

Use case: creare comanda
Actor principal: utilizator
Princpalul scenariu de succes:

1. Utilizatorul apasă din fereastra principală butonul Order Operation
2. Utilizatorul apasă butonul *New Order*
3. Utilizatorul selectează clientul
4. Utilizatorul adaugă produse și cantitatea lor.
5. Utilizatorul apasă butonul de *Generate Order*

Secvențe secundare

- Orice în neregulă legată de selecția greșita a produselor sau cazul în care nu mai sunt suficiente pe stoc va fi semnalată utilizatorului.

4.Proiectare

Obiectivul etapei de design sau proiectare este obținerea unui sistem bun prin utilizarea eficientă a unor resurse existente, precum și a unor resurse noi. De asemenea, scopul ei este de a sparge complexitatea folosind fie abordări de tipul top-down sau bottom-up. S-a folosit paradigma OOP.

Se pot defini mai multe nivele în cadrul etapei de design:

Design level 1

În cadrul primului nivel regăsim imaginea globală a sistemului. Grafic se reprezintă sub forma unei „cutii negre” asupra căreia se trimit date de intrare și procesele ascunse din interiorul cutiei furnizează un rezultat. Principalul scop al sistemului de management este gestiunea bazei de date a depozitului. Astfel un utilizator poate să realizeze mai multe operații prin intermediul acestei aplicații, având ca efect modificarea datelor din baza de date.

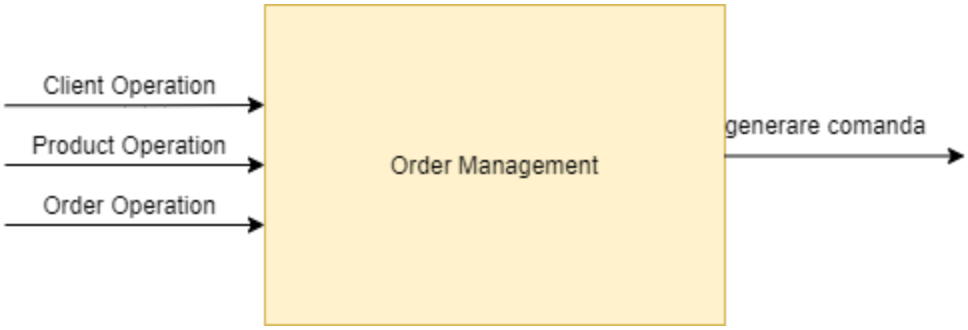

Figure 2 Cutia neagră a sistemului

Design level 2

Sistemul de management este divizat în 5 subsisteme (+1 pentru clasa ce conține metoda *main* de start a aplicației). Această arhitectură pe layere permite reutilizare codului și minimizează dependențele/interacțiunile între subsisteme și se evită relațiile circulare între acestea.

presentation- conține clasele care definesc interfața grafică cu controllerul asociat, pentru a gestiona semnalele date de client prin butoane.

bll– conține clasele care încapsulează logica aplicației. Prin intermediul lui se cer claselor ce manipulează direct baza de date să facă anumite modificări în raport cu modelul. Această clasă conține și un subsitem numit *validators* ce conține validatorii pentru obiectele de tipul Client.

model- conține clasele modelelor de date (client, comanda, produse)

dao-conține clasele ce conțin interogările cu cererea conexiunii la baza de date și prin intermediul cărora se fac actualizările bazei de date. Principalele interogări sunt cele de tipul CRUD.

connection-conține clasa prin intermediul căreia se poate face conexiunea la baza de date cu serverul local și cu parola

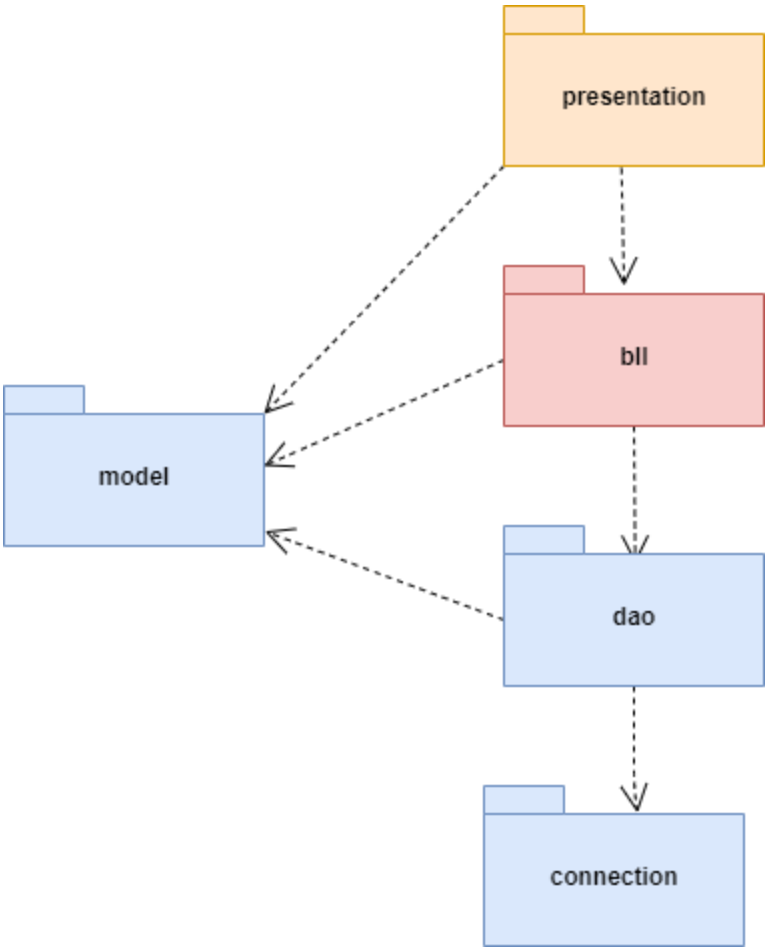

Figure 3 Diagrame pachete

Design level 3

În cadrul acestui nivel se definesc clasele fiecărui subsistem și interacțiunile dintre ele. Denumirea claselor, e sugestivă și inspirată din domeniul problemei *(Client, Order, Product, SoldProducts*)

.

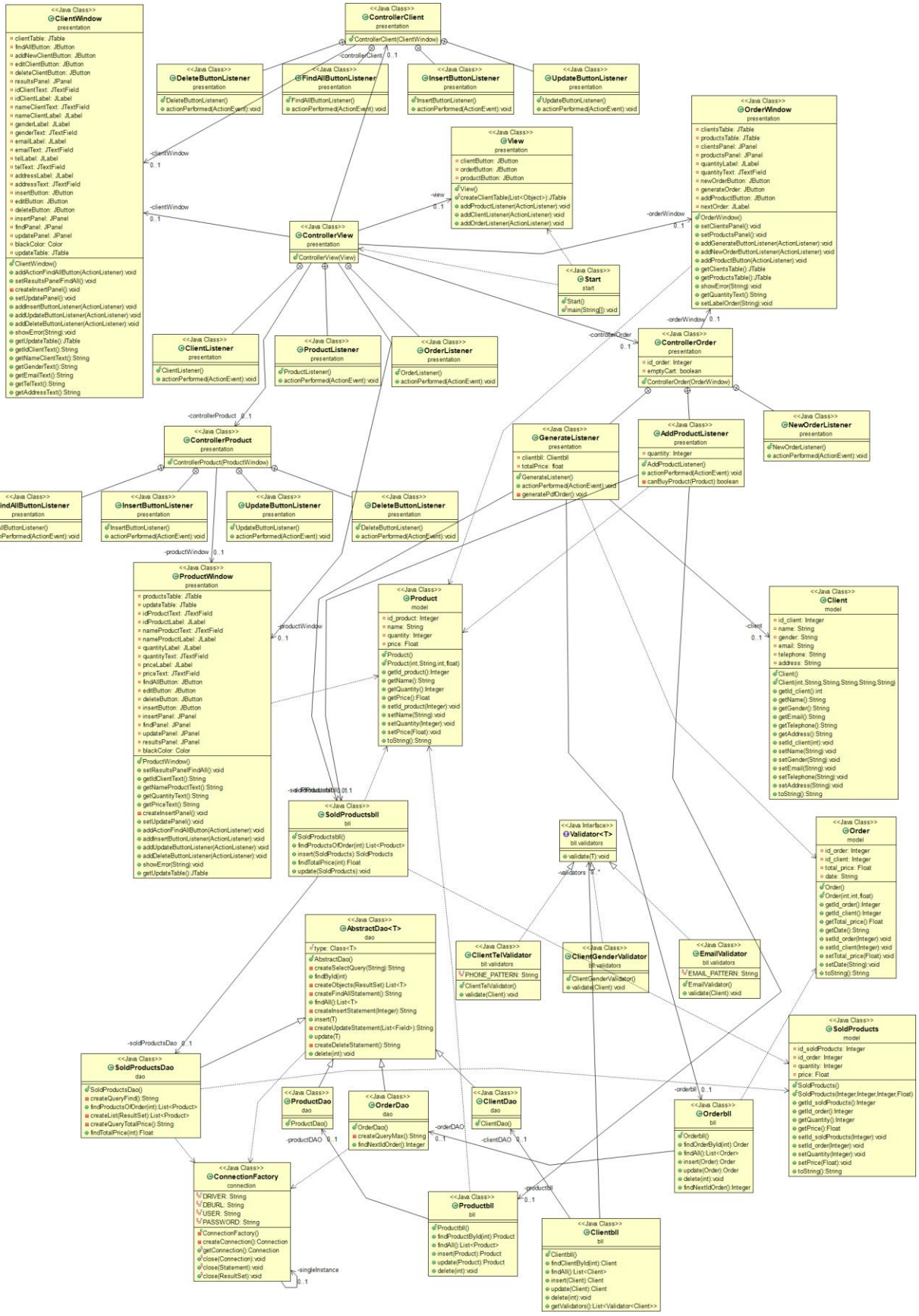
ClientWindow
presentation
ControllerClient
presentation
DeleteButtonListener
presentation
FindAllButtonListener
presentation
InsertButtonListener
presentation
UpdateButtonListener
presentation
View
presentation
OrderWindow
presentation
ControllerView
presentation
Start
start
ClientListener
presentation
ProductListener
presentation
OrderListener
presentation
ControllerOrder
presentation
ControllerProduct
presentation
GenerateListener
presentation
AddProductListener
presentation
NewOrderListener
presentation
ProductWindow
presentation
Product
model
Client
model
Order
model
SoldProductsbll
bll
Validator<T>
bll.validators
AbstractDao<T>
dao
ClientTelValidator
bll.validators
ClientGenderValidator
bll.validators
EmailValidator
bll.validators
SoldProducts
model
SoldProductsDao
dao
ProductDao
dao
OrderDao
dao
ClientDao
dao
Orderbll
bll
ConnectionFactory
connection
Productbll
bll
Clientbll
bll

Figure 4 Diagrame UML

Design-ul de nivel 4, care se referă la împărțirea claselor în rutine este ilustrat în cadrul diagramei UML. Există o metodă private în clasa generică *AbstractDao (createObjects(ResultSet resultSet))*, în intermediul căreia se utilizează tehnica de reflection pentru extragerea rezultatelor din cursorul interogării executate și returnează o listă de obiecte din *model* care au câmpurile identice cu coloanele din tabele din baza de date. Alte metode private au fost folosite în clase de Controller pentru operații specifice, cum ar fi metoda de crearea a documentului pdf cu valorile specifice comenzii generate de utilizator.

Având partea de proiectare finalizată se poate trece implementarea claselor cu funcționalitățile acestora.

5.Implementare

1.4 Clase și metode

În ceea ce privește denumirea claselor și a metodelor s-au ales nume sugestive care să reflecte domeniul problemei (managementul unui depozit de marfa), un exemplu fiind cel prezentat în subcapitol *Modelare*. De asemenea am pus accentul pe proprietatea de reutilizarea a codului, de aceea clasele din pachetul model, *Client* și *Order, Product, SoldProducts* pot fi folosite independent de celelalte clase ale sistemului în alte programe, câștig obținut datorită folosirii arhitecturii pe layere. În continuare sunt prezentate clasele din cele 5(7) pachete ale sistemului împreună cu câmpurile și capabilitățile lor importante.

a. **Clase din pachetul „model”**

*Clasa **Client*** -este clasa care descrie modelul real de client. Acesta clasa va avea câmpurile identice (și ca de numire) cu coloanele tabelului asociat din baza de date. Aceasta va descrie caracteristicile necesare pentru definirea unui client printr-un id propriu, nume, gen, email, număr de telefon și adresa. Aceste câmpuri prezintă datele necesare pentru procesarea unei comenzi.

Clasa Client are doi **constructori**:

Constructori	Descriere
public Client()	-acest constructor e necsar pentru tehnica de reflection folosită pentru extragerea din rezultatele interogării a câmpurilor pentru un client (din clasa AbstractDao)
public Client(int id_client,String name,String gender,String email, String telephone,String address)	-acest constructor are rolul de a seta atributele obiectului nou cu parametrii transmiși

Acestă Clasă conține toate metodele accesoare și mutatoare pentru câmpurile obiectului.

*Clasa **Product*** -este clasa care descrie modelul real de produs. Acesta clasa va avea câmpurile identice (și ca de numire) cu coloanele tabelului asociat din baza de date. Aceasta va descrie caracteristicile necesare pentru definirea unui produs printr-un id propriu, nume, cantitate, pret. Aceste câmpuri prezintă datele necesare pentru procesarea unei comenzi.

Clasa Product are doi **constructori**:

Constructori	Descriere
public Product()	-acest constructor e necesar pentru tehnica de reflection folosită pentru extragerea din rezultatele interogării a câmpurilor pentru un produs (din clasa AbstractDao)

public Product(int id_prod, String name, int quantity,float price)	-acest constructor are rolul de a seta atributele obiectului nou cu parametrii transmiși

Acestă Clasă conține toate metodele accesoare și mutatoare pentru câmpurile obiectului.

*Clasa **SoldProducts***-este clasa care descrie modelul real de produs vandut. Acesta clasa va avea câmpurile identice (și ca de numire) cu coloanele tabelului asociat din baza de date. Aceasta va descrie caracteristicile necesare pentru definirea unui produs vândut printr-un id propriu, id comenzii, cantitate, pret. Aceste câmpuri prezintă datele necesare pentru procesarea unei comenzi.

Clasa SoldProducts are doi **constructori**:

Constructori	Descriere
public **SoldProducts** ()	-acest constructor e necesar pentru tehnica de reflection folosită pentru extragerea din rezultatele interogării a câmpurilor pentru un produs vandut (din clasa AbstractDao)
public **SoldProducts** (Integer id_soldProducts, Integer id_order, Integer quantity, Float price)	-acest constructor are rolul de a seta atributele obiectului nou cu parametrii transmiși

Acestă Clasă conține toate metodele accesoare și mutatoare pentru câmpurile obiectului.

*Clasa **Order***-este clasa care descrie modelul real comandă. Acesta clasa va avea câmpurile identice (și ca de numire) cu coloanele tabelului asociat din baza de date. Aceasta va descrie caracteristicile necesare pentru definirea unei comenzi printr-un id propriu, id client, prețul total de achitat și data procesării comenzii.

Clasa Order are doi **constructori**:

Constructori	Descriere
public **Order** ()	-acest constructor e necesar pentru tehnica de reflection folosită pentru extragerea din rezultatele interogării a câmpurilor pentru o commanda procesată (din clasa AbstractDao)
public Order(int id_order, int id_client,float total_price)	-acest constructor are rolul de a seta atributele obiectului nou cu parametrii transmiși

Acestă Clasă conține toate metodele accesoare și mutatoare pentru câmpurile obiectului.

b. Clase din pachetul *dao*

*Clasa **AbstractDao***-este clasa generică care conține metodele ce realizează interogările de tip CRUD în baza de date. Aceasta clasă folosește tehnici de reflection pentru extragerea rezultatelor interogărilor având în vedere faptul că nu se cunoaște concret tipul clasei și implicit cum să se extragă câmpurile pentru crearea de obiecte Client, Product etc.

Clasa Order are un **constructor**:

Constructori	Descriere
public AbstractDao()	-se ia tipul concret al clasei ce moștenește clasa AbstractDao

Metode:

- *public T findById(int id)* – această metodă se conectează la baza de date și execută interograea SQL de căutare a unei linii de tabel ce are ca identificatoru id-ul transmis ca parametru
- *private List<T> createObjects(ResultSet resultSet)*- utilizează tehnica de reflection pentru a extrage conform tipului concret al clasei câmpurile din cursorul rezultat în urma unei interogăeri a tebelului corespunzător din baza de date
- public List<T> findAll() -va returna o listă cu toate obiectele de tipul concret ce au fost extrase din tabela specifică din baza de date
- public T insert(T t) throws SQLException, IllegalAccessException -prin intermediul acestei metode se va insera un nou obiect în tabela asociată din baza de date, setând câmpurile din tabel cu valorile campurilor obiectului transmis ca parametru.
- public T update(T t) throws SQLException, IllegalAccessException- se vor actualiza informațiile din tabela asociată pentru obietul transmis ca parametru. Se va lua id-ul ca principal identificator a liniei din tabel ce va fi modificată.
- public void delete(int id) throws SQLException, IllegalAccessException -va șterge obiectul corespunzător id-ului din tabela corespunzătoare

Claselele ClientDao și ProductDao moștenesc metoda generică AbstractDao și îi asociază parametrului tipul specific Client, respectiv Product. Acestea nu mai conțin alte interogări specifice față de cele de tip CRUD standard.

Clasa OrderDao la fel ca cele de mai sus, doar că mai implementează o interogare specifică care returnează identificatorul nou eligibil ca fiind maximul dintre cele existente +1.

Clasa SolProducts la fel moștenește clasa generică AbstractDao asociindu-i tipul corespunzător. Această clasă implementează două metode specifice de interogare *public Float findTotalPrice(int id_order), public List<Product> findProductsOfOrder(int id_order)* prin care se calculează prețul total pentru o comandă și respectiv, returnarea listei produselor unei comenzi.

Toate aceste Clase conțin metode *private* ce vor genera textul instrucțiunilor SQL specifice, folosind un obiect de tipul *StringBuilder*.

c. Clase din pachetul bll

Clasele Clientbll, Orderbll, Productbll, SoldProductsbll vor avea ca și câmp un obiect de tipul Dao corespunzător. Astfel metodele lor vor cere prin intermediul acelui câmp sa facă operațiile de interogare sau manipulare a bazei de date. Clasa *Clientbll* conține și o list de validatori pentru caoutile de email, numar de telefon și gen pentru clienți. Înainte de a se apela metodele instanței de manipulare a bazei de date se verifică dacă e eligibil clientul iar în caz contrar se va arunca o excepție.

Metodele acestor clase:

- public Client findClientById(int id) → Clientbll
- public List<Client> findAll()
- public Client insert(Client client) throws SQLException, IllegalAccessException, IllegalArgumentException
- public Client update(Client client) throws SQLException, IllegalAccessException,IllegalArgumentException
- public void delete(int id) throws SQLException, IllegalAccessException
- public List<Product> findProductsOfOrder(int id_order) → SoldProductsbll
- public Float findTotalPrice(int id_order) throws SQLException, IllegalAccessException → SoldProductsbll
- public Integer findNextIdOrder() →Orderbll
- public Product findProductById(int id) → Productbll

Subpachetul **validators**

- interfața **Validator** cu paramtru generic
  - metoda de implementat este public void validate(T t)
- Clasa *ClientGenderValidator* va implementa metoda interfeței Validator
  - Metoda validate verifică dacă în câmpul **gender** al obiectului de tip Client se află M,m, F sau f, în caz contrar se aruncă o excepție

- *Clasa ClientTelValidator* se va verifica numărul de telefon cu următorul pattern(regex)
  - [0][7][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]-numărul începe cu 07 și conține 10 cifre
- Clasa *EmailValidator* folosește patternul de validare din exemplul furnizat în prezentarea temei
-

d. Clase din pachetul connection

În cadrul acestui pachet se află o singură clasă ConnectionFactory. Prin intermediul acestei clase se cere conexiunea la baza de date pin intermediul unui driver și serverul local. Atributele acestei clase sunt static final (*DRIVER, DBURL, USER, PASSWORD*).

Constructorul fără paramteri: încarcă driverul

Metode importante:

- public static Connection getConnection() - va returna conexiunea cu baza de date prin apelul metodei private *createConnection* ce va crea conexiunea cu atributele de clasă specificate anterior
- public static void close(Connection connection)-va inchide conexiunea cu baza de date
- public static void close(Statement statement) -va închide interogarea/instrucțiunea SQL
- public static void close(ResultSet resultSet) -închiderea cursorului pentru rezultate

e. Clase din pachetul presentation

Clasa ClientWindow -moștenește clasa JFrame și va descrie fereastra pentru operațiilor executate pe tabele clienților. Consructorul va aranja panel-urile și butoanele corespunzătoare. Se vor crea un panel de tipul JTabbedPane pentru operația de seleție totală, inserare și editare clienti(+delete).

Metode importante:

- *setResultsPanelFindAll(),setUpdatePanel()* -prin aceste metode publice controller-ul va actualiza fiecare panel în parte, și respectiv tabelele cu valorile din baza de date
- *createInsertPanel()*-va crea panel-ul pentru operația de insert

Clasa ControllerClient- în intermediul constructorului va adăuga listeneri pentru butoanele din fereastra clientWindow

Metodele listenerilor din clasele interne:

- clasa internă FindAllButtonListener - va conține implementarea logicii pentru butonul de *find All*
- clasa internă *InsertButtonListener*- - va conține implementarea logicii pentru butonul de insert
- clasa internă *UpdateButtonListener* -- va conține implementarea logicii pentru butonul de edit
- clasa internă DeleteButtonListener -- va conține implementarea logicii pentru butonul de delete

Clasa ProductWindow și ControllerProduct sunt similare cu cele din cadrul logicii de prezentare pentru client.

Clasa OrderWindow descrie interfața grafică pentru fereastra dedicată creării comenzii. Va conține în principiu două panel-uri cu tabelele clienților și respectiv a produselor. Două metode, *public void setClientsPanel(), public void setProductsPanel()* vor fi apelate când apare o modificare în cadrul celor două tabele, pentru păstrarea consistenței.

Clasa ControllerOrder implementează logica de gestionare a operațiilor ce se fac în cadrul ferestrei ordreWindow.

Clase interne pentru listeneri butoane:

- *NewOrderListener* – va determina id-ul comenzii de generat. Neapăsarea butonului de new order nu va permite generarea comenzii
- *AddProductListener*- va gestiona vânzarea produselor și actualizarea tabelului pentru produse după ce o anumită cantitate dintr-un produs s-a vândut
- *GenerateListener* – va descrie logica de creare a comenzii, introcerea ei în baza de date și generarea unui pdf cu informațiile comenzii care include și prețul total de achitat.

Clasa View-va conține 3 butoane prin care se selectează una din cele trei ferestre pentru operații pe tabela de clienți, produse sau pentru generarea unei comenzi.

Această clasă conține și metoda *public static JTable createClientTable(List<Object> objects)*, metodă ce folosțe tehnica de reflection pentru a extrage valorile dintr-o listă de obiecte și va crea un JTabel cu valori extrase.

Clasa ConrolllerView în cadrul constructorului său se vor crea cele 3 ferestre cu controllerele lor. Va implementa logica de selecție a ferestrei ce va fi vizibilă la un moment dat.

Metoda *main* din pachetul start va crea un obiect de tipul View și controllerul acestei ferestre.

1.5 Interfața grafică cu utilizatorul (GUI)

Interfața grafică cu utilizatorul este destul de intuitivă și folosește elemente din pachetul *javax.swing*. Am încercat să ofer un aspect cât mai prietenos pentru utilizator pentru ca acesta să își dea seama cât mai ușor de modul de funcționare. În cazul în care va introduce valori necorespunzătoare acesta va fi informat. În timp ce face operații în interfața grafică pentru actualizare bazei de date, și în cadrul interfeței grafice tabelele se vor actualiza cu valorile specifice. Din fereastra principală, utilizatorul poate selecta pe ce tabel din baza de date să facă operațiile dorite sau poate opta pentru plasarea de comenzi.

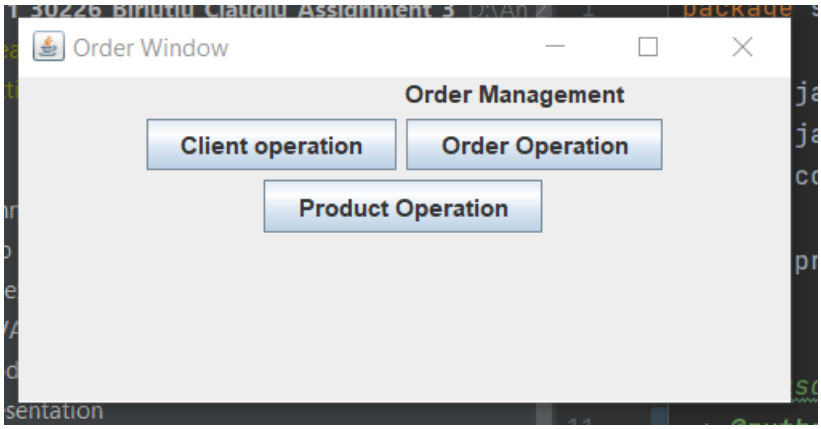

Figure 5 Fereastra principală

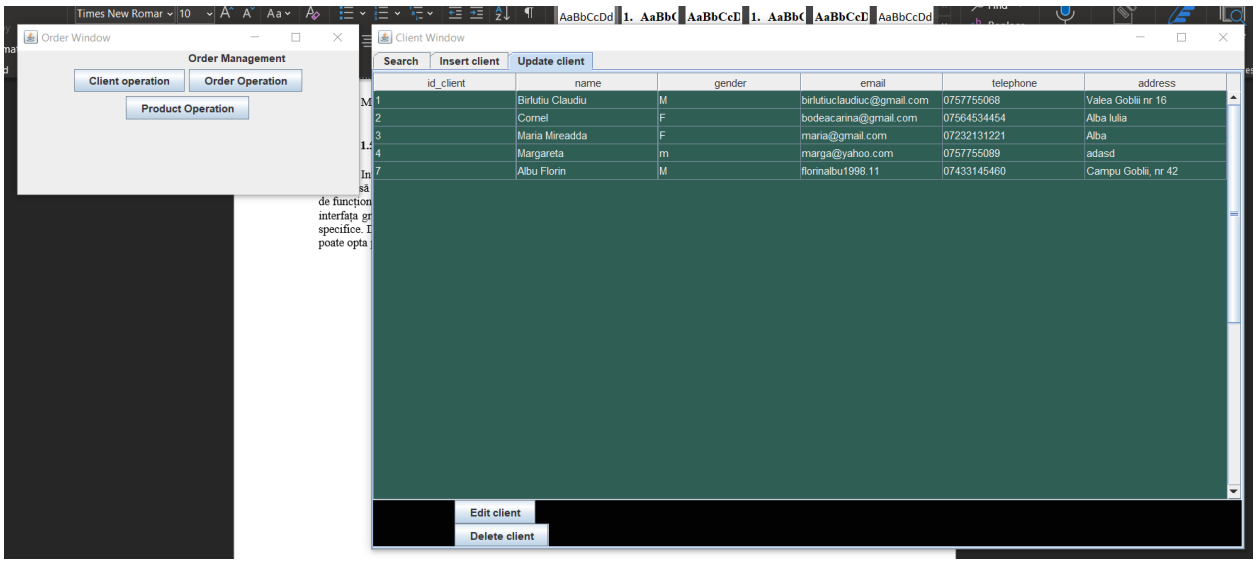

Figure 6 Fereastra Client Window - update

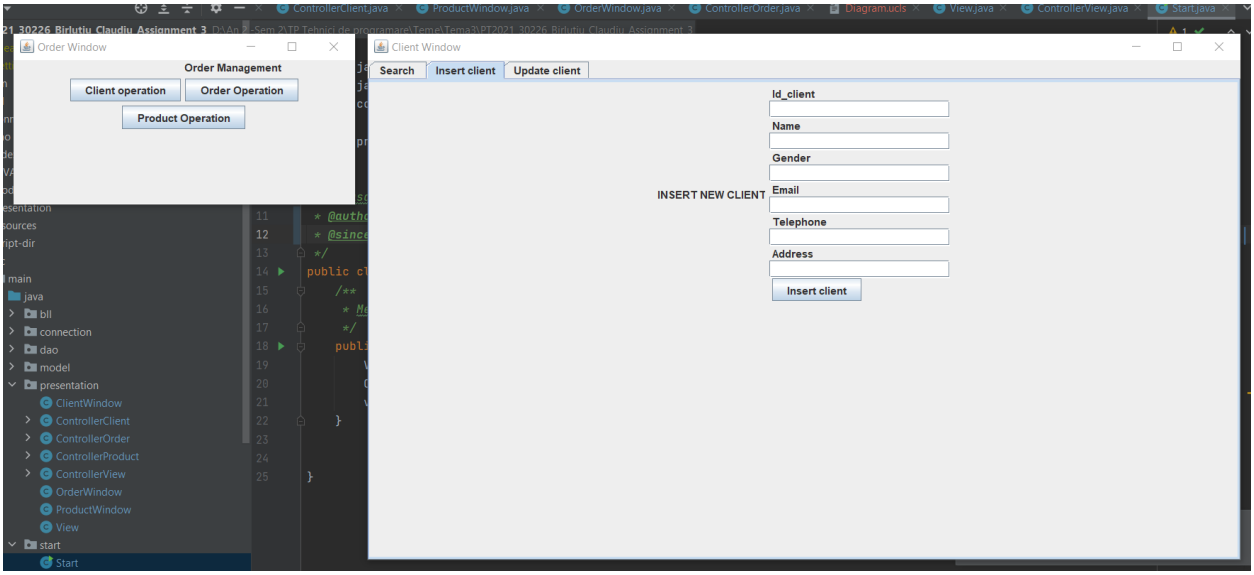

Figure 7 Fereastra Client Window - insert

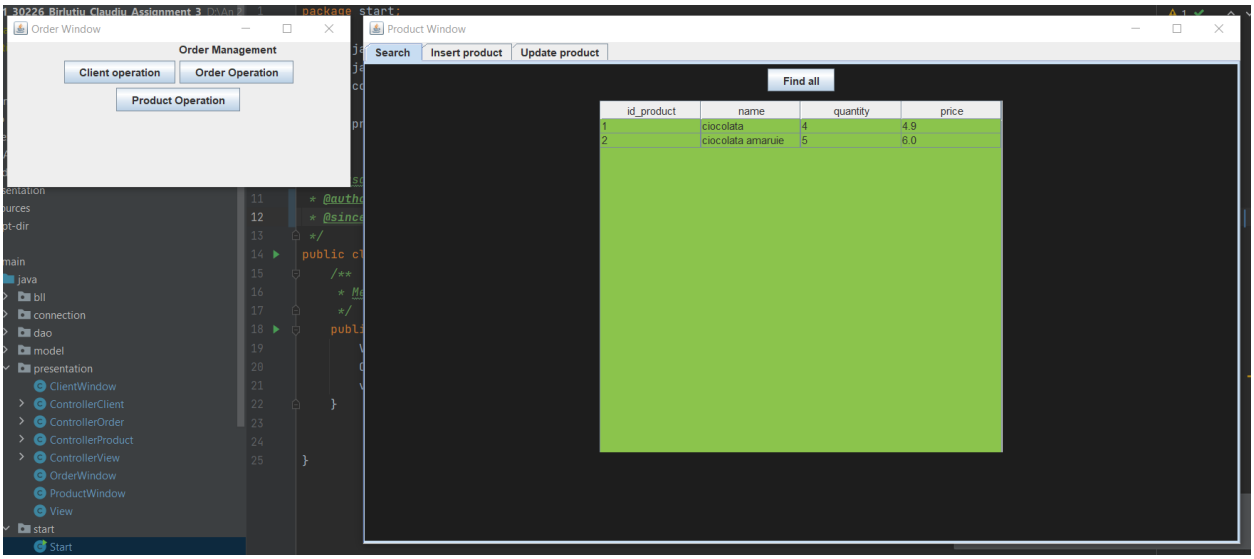

Figure 8 Fereastra Product Window – find all

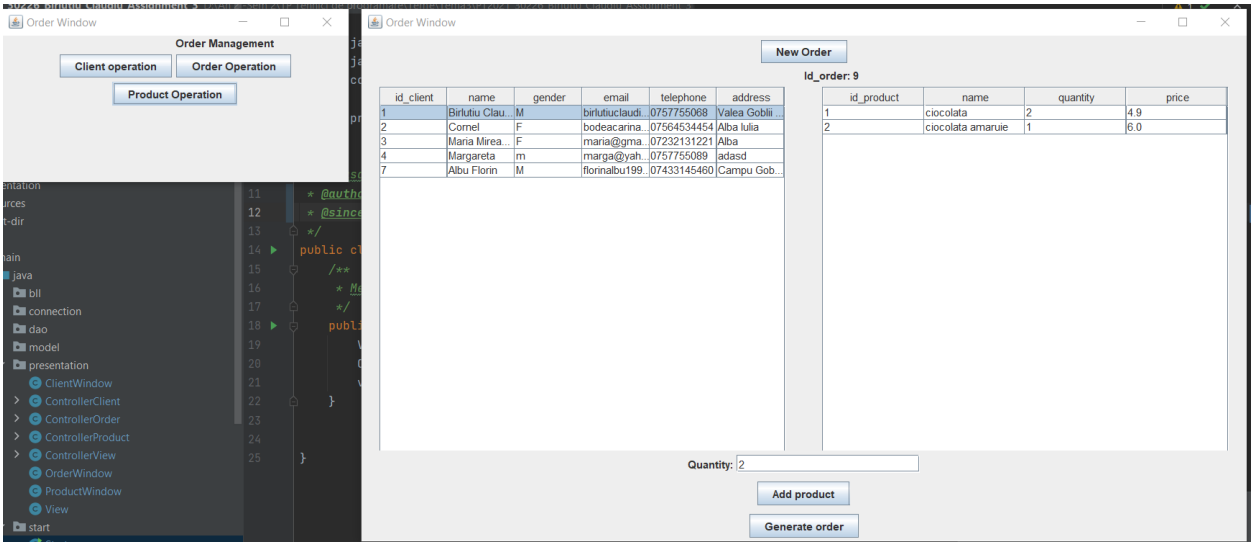

Figure 9 Fereastra Order Window

```
ORDER nb: 9
Client: Birlutiu Claudiu
   Email: birlutiuclaudiuc@gmail.com
   Telephone: 0757755068
   Address: Valea Goblii nr 16
 PRODUCTS
ciocolata amaruie   4 buc.  24.0 lei
ciocolata   2 buc.  9.8 lei


TOTAL PRICE: 33.8    Date:21/04/23 01:15:32
```

Figure 10 Format din PDF

6.Rezultate

Modul de testare a fost să verific dacă modificarea tabelelor din baza de date are loc în conformitatea cu cererile din interfața grafică. De asemenea, factura generată trebuie să conțină informațiile corecte cu privire la client, produsele cumpărate (cantitate și preț), precum și prețul total să fie eligibil.

7.Concluzii

În concluzie, prin realizarea acestei teme de laborator, s-a realizat un sistem de management al comenzilor pentru un depozit de produse, prin intermediul căruia un utilizator poate să gestioneze baza de date prin intermediul unei interfețe grafice.

Pentru proiectarea aplicației s-a folosit paradigma orientată pe obiect, folosindu-se layered Architectures, care îi oferă proprietate de reutilizare a codului/ claselor în alte aplicații. Întreținerea codului este mai ușor de realizat, iar modelul poate fi reutilizat de alte aplicații. De asemenea și dezvoltarea aplicației de față se poate efectua independent la nivel de clase, fie la nivelul interfeței grafice, fie în cadrul modelului.

Pentru cel care a realizat aplicația, dezvoltarea ei l-a familiarizat mai mult cu conceptul de Object Oriented și cu modul de modelare, proiectare și implementare a unei aplicații folosind această paradigmă de programare. De asemenea, l-a obișnuit cu urmarea pașilor logici în procesul de dezvoltare a unei aplicații software , de la analiza problemei, fixarea cerințelor, proiectarea, implementarea și testarea ei. A învățat noțiunile de bază cu privire la conexiunea la o bază de date printr-un server local și de asemenea la execuția de instrucțiuni SQL care au efect asupra tabelelor din baza de date asociată. L-a familiarizat cu tehnica de reflection, tehnică prin care s-a reușit generalizarea unor metode pentru evitarea redundanței.

Ca posibilă dezvoltare ar fi adăugarea unui tabel ce reprezintă coșul de cumpărături și astfel clientul se poate răzgândi, adică pune produsul la loc. O altă îmbunătățire ar fi ca pe factură să nu-mi apară de două ori numele aceluiași produs, dacă s-a cumpărat din același produs la momente diferite.

8.Bibliografie

- „ASSIGNMENT 3-SUPPORT PRESENTATION” , material auxiliar (furnizat pe platforma Microsoft Teams)
- Cursuri de la disciplina Tehnici de Programare Fundamentală, studiată în semestrul al 2-lea al anului universitar 2020-2021
- https://gitlab.com/utcn_dsrl/ptreflectionexample
- https://docs.oracle.com/javase/7/docs/api/java/awt/Graphics.html
- https://docs.oracle.com/javase/7/docs/api/
- http://www.mkyong.com/jdbc/how-to-connect-to-mysql-with-jdbc-driver-java/
- https://www.baeldung.com/javadoc
- https://www.baeldung.com/java-pdf-creation
- https://www.vogella.com/tutorials/JavaPDF/article.html