MINISTERUL EDUCAȚIEI ȘI CERCETĂRII ȘTIINȚIFICE
UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

ORDER MANAGEMENT

ASSIGNMENT 3

Documentatie

MOLDOVAN ADELINA-STEFANIA

GRUPA 30226

CUPRINS

1. Obiectivul temei 3

1.1 Obiectivul principal 3

1.2 Obiectivul secundar 3

2. Analiza problemei, modelare, scenarii, cazuri de utilizare 4

3. Proiectare 5

3.1 Diagrame UML 5

3.2 Structuri de date 6

3.3 Packages 6

4. Implementare. 7

4.1 Pachetul model 7

Clase: Clients, Orders, Product, FailedOrder

4.2 Pachetul dataAccessLayer. 8

Clase: ConnectionDB, DBOperations, ClientDAO,OrderDAO, ProductDAO ,FailedOrderDAO

4.3 Pachetul businessLogicLayer 10

Clase: ClientBLL, OrderBLL, FailedOrderBLL, ProductBLL

4.4 Pachetul presentation. 11

Clase: GenerateReports, Parser

4.5 Pachetul main 11

Clasa main

5. Rezultate. 12

6. Concluzii. 13

7. Bibliografie. 13

1. Obiectivul temei

Obiective principale

Se considera o aplicatie pentru procesarea comenzilor unor clienti. Bazele de date relationale sunt folosite pentru a stoca produse, clienti si comenzi. Clase minime folosite de catre aplicatie:

- Business Logic classes – logica aplicatiei;
- Presentation classes – clasele care contin interfata cu utilizatorul;
- Data access classes – clasele in care se face accesul la baza de date, de aici se executa interogarile asupra datelor;
- Mode classes – contine modelele de date cu care se lucreaza: date de tip client, produs si comanda.

Obiective secundare

1 . - dezvoltarea de use case-uri : folosim use case-uri (caz de utilizare) pentru a reprezenta cerinţe ale utilizatorilor. Sunt descrise acţiuni pe care un program le execută atunci când interacţionează cu actori şi care conduc la obţinerea unui rezultat .

2 . - dezvoltarea de diagrame UML : folosim diagrame UML pentru a înregistra relaţiile dintre clase .

3 . - implementarea unor clase in java : sunt descrise clasele folosite si rolul acestora .

4. – implemetarea bazelor de date relationale pentru a stoca informatiile necesare pentru aplicatie precum si modul in care se face accesul la datele stocate in baza de date, cum se extrag si se prelucraeza informatiile.

5. – utilizarea javadoc-urilor pentru documentarea claselor

6. - Layered architecture – este un model de arhitectura cu n nivele, unde componentele sunt organizate in straturi orizontale. Aceasta este o metoda traditionala pentru proiectarea mojoritati aplicatiilor software si este menita sa fie independenta de sine. Aceasta inseamna ca toate componentele sunt interconectate, dar nu depind unele de altele.

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

Faza de rezolvare a problemei

1. *Analiza* înseamnă înţelegerea, definirea problemei şi a soluţiei ce trebuie dată;

2. *Algoritmul* presupune dezvoltarea unei secvenţe logice de paşi care trebuie urmaţi pentru rezolvarea problemei;

3. *Verificarea* este parcurgerea paşilor algoritmului pe mai multe exemple pentru a fi siguri că rezolvă problema pentru toate cazurile.

Faza de implementare

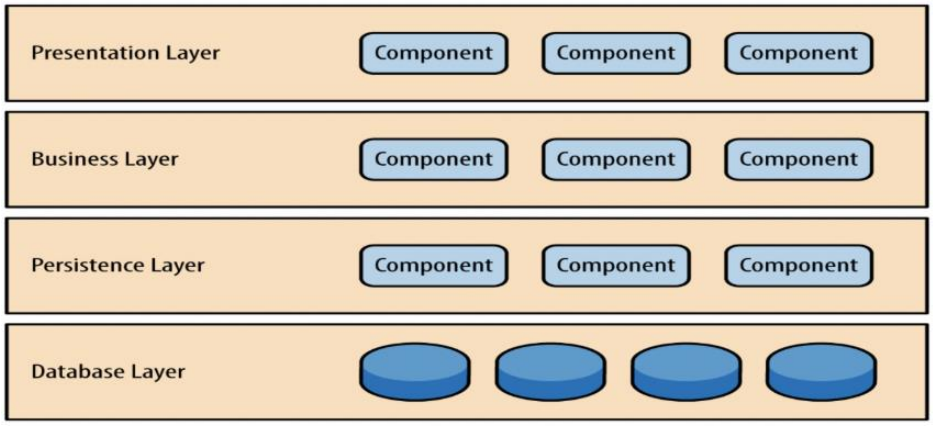
1. *Programul* reprezintă translatarea algoritmului într-un limbaj de programare

2. *Testarea* este etapa în care ne asigurăm că instrucţiunile din program sunt urmate corect de calculator. În situaţia în care constatăm că sunt erori, trebuie să revedem algoritmul şi programul pentru a determina sursa erorilor şi pentru a le corecta. Aceste două faze de dezvoltare a programului sunt urmate de faza de utilizare a programului care înseamnă folosirea acestuia pentru rezolvarea problemelor reale, cele pentru care a fost conceput. Ulterior pot interveni modificări ale programului fie pentru a răspunde noilor cerinţe ale utilizatorilor, fie pentru corectarea erorilor care apar în timpul utilizării şi care nu au putut fi găsite în faza de testare.

Algoritmul reprezinta o succesiune de pasi care duc la rezolvarea unei probleme. In cazul problemei noastre avem urmatoarea succesiune de pasi:

1. Crearea bazei de date in care vom stoca informatia necesara programului;
2. Conectarea bazei de date cu programul nostru;
3. Implementarea metodelor care vor executa interogarile asupra bazei de date;
4. Citirea comenzilor din fisier (insert, delete, report, order);
5. Executarea comenzilor si generarea de pdf-uri in functie de comanda primita.

Layered architecture – tipul de arhitectura folosit pentru implementarea aplicatiei



PresentationLayer – contine clasele care genereaza pdf-urile.

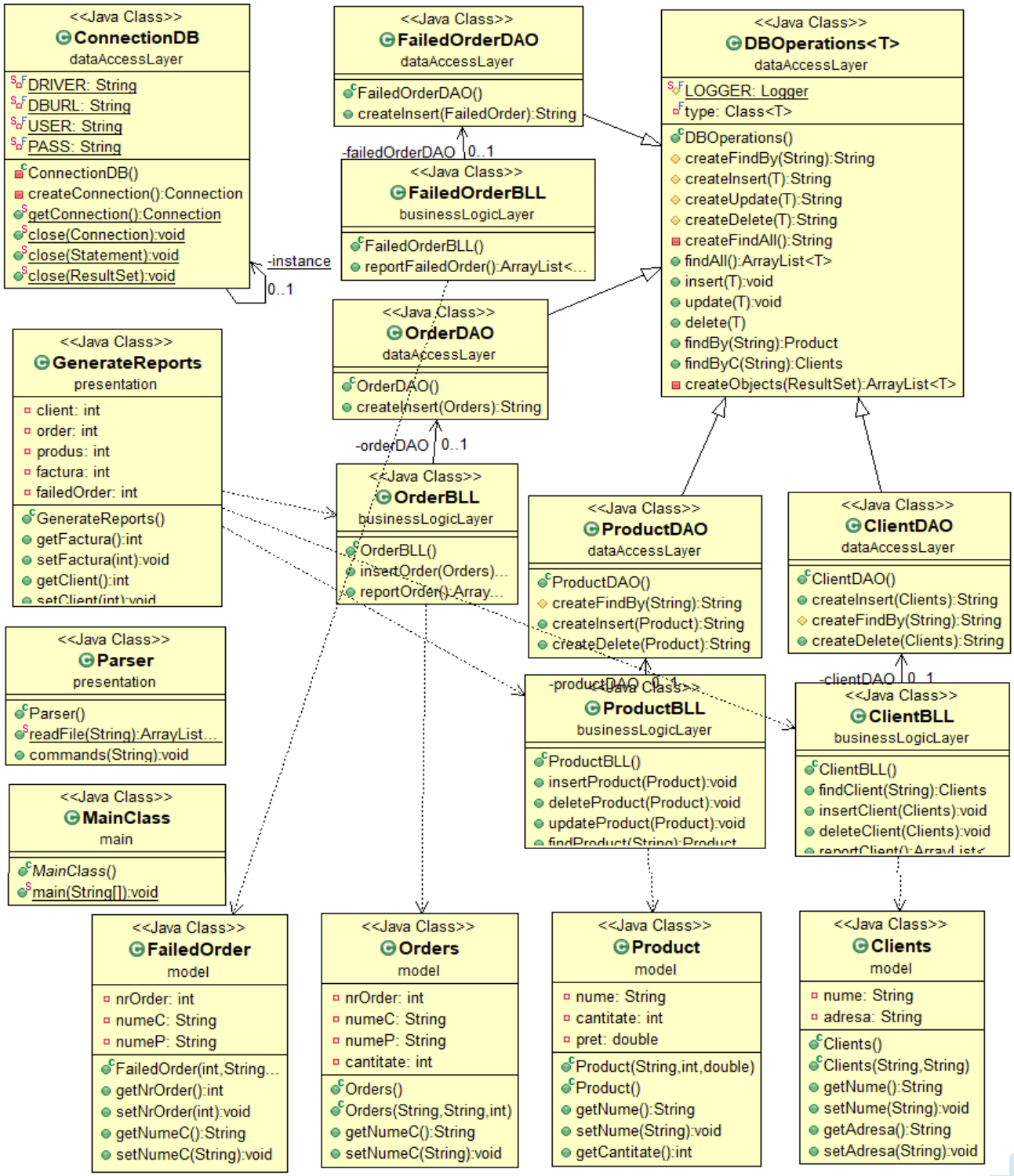
BussinesLayer – contine clasele care descriu logica aplicatiei.

DataBaseLayer – contine clasele care definesc interogarile si conexiunea pentru baza de date.

DatabaseLayer – aici sunt stocate datele.

3.Proiectare

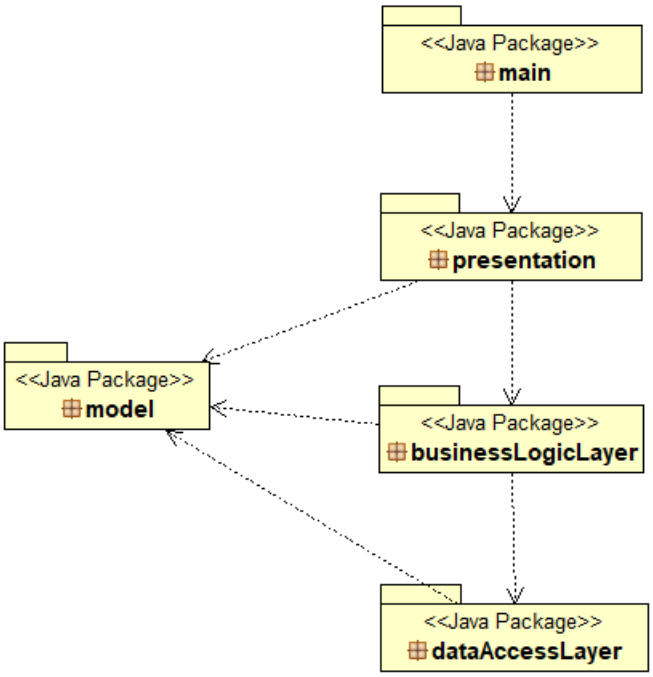
Diagrama UML



Structuri de date

Structurile de date folosite in acest proiect sunt ArrayList -urile, care contin elemente de tipul Clients sau Product, fiind folosite pentru a memora rezultatul interogarilor care returneaza mai multe obiecte de acelasi tip si pentru a putea efectua mai usor si mai eficient operatiile specifice. Deci, ArrayList-urile vor fi folosite pentru cateva operatii din cadrul programului, acestea memorand elementele care trebuie afisate in urma comenzilor de tipul report order, report product sau report client .

Packages



Pachetul (package) este o grupare de elemente ale unui model şi reprezintă baza necesară controlului configuraţiei, depozitării şi accesului . Este un container logic pentru elemente între care se stabilesc legături . Toate elementele UML pot fi grupate în pachete (cel mai des pachetele sunt folosite pentru a grupa clase). Un element poate fi conţinut într-un singur pachet.

Putem observa prin intermediul diagramei de pachete ca programul respecta arhitectura de tipul “Layered architecture” , si anume: imparte aplicatia in diferite straturi, iar fiecare strat are un scop special si apeleaza functii ale straturilor de sub acesta.

Pachetele sunt:

- Model – contine datele mapate in tabelele din baza de date
- Presentation Layer – contine clasele care afiseaza rezultatele programului
- Business Layer – contine clasele care incapsuleaza logica aplicatiei
- Data Access Layer – contine clasele care vor executa interogarile si conexiunea catre baza de date

4.Implementare

Programul este organizat conform șablonului “Layered architecture”.

Pachetul Model contine clasele : Clients, Product, Orders si FailedOrder . Acestea sunt clasele care contin datele mapate in tabelele din baza de date.

Clasa Clients

- contine doua atribute : nume si adresa de tipul String;

- atributele nume si adresa vor memora numele clientului, respective adresa acestiua; deasemenea nume este si cheie primara in tabela clients

- contine gettere si settere , metode care permit citirea si scrierea numelui si a adresei;

```
package model;
public class Clients {
	private String nume;
	private String adresa;
	public Clients(String nume, String adresa) {
		this.nume = nume;
		this.adresa = adresa;
	}
	public String getNume() {
		return nume;
	}
	. . . .
```

Clasa Product

- contine trei atribute : nume de tipul String, cantitate de tipul int si pret de tipul double;

- atributul nume memoreaza numele clientului, atributul cantitate retine cate produse se afla in stoc iar pret reprezinta pretul unui singur produs; atributul nume este cheie primara pentru tabela product;

- contine gettere si settere , metode care permit citirea si scrierea numelui, cantitatii si a pretului;

Clasa Orders

- contine patru atribute : nrOrder de tipul int, numeC de tipul String, numeP de tipul String si cantitate de tipul int;

- nrOrder reprezinta cheia primara a tabelei Orders si se incrementeaza automat din baza de date; atributul numeC memoreaza numele clientului, numeP memoreaza numele produsului comandat, iar atributul cantitate retine cate produse a comandat clientul respectiv;

- contine gettere si settere , metode care permit citirea si scrierea numelor, cantitatii si a numarului comenzii;

Clasa FailedOrder

- FailedOrder este o tabela menita sa stocheze comenzile care nu au putut fi efectuate deoarece cantitatea de produse comandata este mai mare decat cantitatea din stoc;

- contine trei atribute : nrOrder de tipul int, numeC de tipul String, numeP de tipul ;

- nrOrder reprezinta cheia primara a tabelei FailedOrder; atributul numeC memoreaza numele clientului, numeP memoreaza numele produsului comandat,;

- contine gettere si settere , metode care permit citirea si scrierea numelor, cantitatii si a numarului comenzii;

Pachetul dataAccessLayer contine clasele : ConnectionDB, DBOperations, ClientDAO, ProductDAO, OrderDAO si ProductDAO.

Clasa ConnectionDB

- este clasa in care se realizeaza conexiunea intre aplicatie si baza de date
- contine numele driverului, locatia bazei de date (DBURL), utilizatorul(USER) si parola(PASS)
- contine si metode pentru crearea unei conexiuni, obtinerea unui conexiuni active si inchiderea unei conexiuni, a unui statement sau a unui ResultSet

```
public class ConnectionDB {
	private static final String DRIVER = "com.mysql.cj.jdbc.Driver";
	private static final String DBURL = "jdbc:mysql://localhost:3306/bazadedate?"+"&useSSL=false";
	private static final String USER = "root";
	private static final String PASS = "Adelina22*";
	private static ConnectionDB instance = new ConnectionDB();
	private ConnectionDB() {
		try {
			Class.forName(DRIVER);
		} catch (ClassNotFoundException e) {
			e.printStackTrace();
		}
	}
	private Connection createConnection() {
		Connection connection = null;
		try {
			connection = DriverManager.getConnection(DBURL, USER, PASS);
			if (connection == null) {
				System.out.println("Failed to make connection!");
			}
		} catch (SQLException e) {
			System.err.format("SQL State: %s\n%s", e.getSQLState(), e.getMessage());
		} catch (Exception e) {
			e.printStackTrace();
		}
		return connection;
	}
```

Clasa DBOperations

- este clasa care contine functiile corespunzatoare interogarilor pentru baza de date

- interogarile folosite in program sunt : select(all sau find dupa un anume camp din tabela) , insert, update si delete

- metodele au fost implementate folosind tehnica de reflexie.

- prin reflexive un program java poate incarca o clasa despre care nu stie nimic, sa afle despre variabilele, metodele si constructorii clasei si apoi sa lucreze cu ele;

-datorita acestei tehnici putem crea o singura metoda de insert (sau orice alta comanda) care sa poata fi executata de oricare din clasele din pachetul model

- pentru reflexie punctul de plecare este un obiect Class care reprezinta o clasa Java pe care o vom inlocui ulterior cu clasa pentru care dorim sa apelam metodele respective

Exemplu de metoda creata folosind reflexia :

- functia findAll care va returna toate datele dintr-o Clasa

- avem deasemenea si functia createFindAll care va fi apelata din functia findAll, fiind functia care creeaza query-ul pentru baza de date

```
private String createFindAll() {
        StringBuilder sb = new StringBuilder();
        sb.append("SELECT ");
        sb.append(" * ");
        sb.append(" FROM ");
        sb.append(type.getSimpleName());
        return sb.toString();
}

public ArrayList<T> findAll() {
        Connection connection = null;
        PreparedStatement statement = null;
        ResultSet resultSet = null;
        String query = createFindAll();
        try {
                connection = ConnectionDB.getConnection();
                statement = connection.prepareStatement(query);
                resultSet = statement.executeQuery();
                return createObjects(resultSet);
        } catch (SQLException e) {
                LOGGER.log(Level.WARNING, type.getName() + "DAO:findAll" + e.getMessage());
        } finally {
                ConnectionDB.close(resultSet);
                ConnectionDB.close(statement);
                ConnectionDB.close(connection);
        }
        return null;
}
```

Clasa ClientDAO, Clasa ProductDAO, Clasa OrderDAO , Clasa ProductDAO

- clase care corespund claselor din pachetul model iar fiecare are rolul de a implementa particularitatile pentru metodele care creeaza query-urile

Exemplu:

```
protected String createInsert(T t) {

	StringBuilder sb = new StringBuilder();
	sb.append("INSERT INTO ");
	sb.append(type.getSimpleName());
	sb.append(" VALUES ");
	return sb.toString();
}
```

```
public String createInsert(Product produs) {
  String s = super.createInsert(produs);
  StringBuilder insert = new StringBuilder();
  insert.append(s);
  insert.append("('" + produs.getNume());
  insert.append("', "+produs.getCantitate() );
  insert.append(", ");
  insert.append(produs.getPret());
  insert.append(")");
  return insert.toString();
}
```

- functia din stanga este functia implementata in clasa DBOperations iar cea din dreapta este functia implementata in clasa ProductDAO
- se poate observa faptul ca in clasa DBOperations am creat ce este comun pentru toate tabelele in ceea ce priveste instructiunea insert iar in fiecare clasa ClientDAO, Clasa ProductDAO, Clasa OrderDAO , Clasa ProductDAO am creat ce este particular pentru fiecare;
- in aceste clase se afla merode pentru a crea query-urile(insert, delete, update si findBy) pentru fiecare tabela din baza de date

Pachetul businessLogicLayer contine clasele : ClientBLL, ProductBLL, OrderBLL si FailedOrderBLL

- in acest pachet sunt clasele in care care vom apela metodele din DBOperations

- daca in DBOperations am creat metodele la modul general folosind metoda reflexiei, in aceste clase vom apela metodele individual pentru fiecare clasa in parte, pentru a particulariza rezultatele returnate de aceste metode in functie de datele care se afla in tabelele respective

```
public class ProductBLL {
	private ProductDAO productDAO;
	public ProductBLL() {
		productDAO = new ProductDAO();
	}
	public void insertProduct(Product product) {
		productDAO.insert(product);
	}
	public void deleteProduct(Product product) {
		productDAO.delete(product);
	}
	public void updateProduct(Product product) {
		productDAO.update(product);
	} . . .
```

Pachetul presentation contine clasele : GenerateReports si Parser.

Clasa GenerateReports

- este clasa in care se genereaza pdf-urile in functie de comanda primita

- se insereaza un tabel in pdf daca comanda este report, urmata de numele tabelei

- daca comanda este order se introduc datele in tabela Order si se genereaza o factura pentru produsul respectiv, totodata se face update la cantitatea de produse in tabela Product (se scade cantitatea comandata)

- daca cantitatea de produse comandata nu se afla in stoc se introduc datele in tabela FailedOrder si se genereaza un pdf cu un mesaj corespunzator

- pentru a insera un client sau un produs se verifica daca acestea nu exista deja; daca exista clientul, acesta nu se mai introduce in tabela, iar in cazul produselor, daca acestea sunt deja se face update la cantitate

Exemplu de functie din clasa GenerateReports:

- functia care insereaza in tabelea Product si verifica daca produsul nu se afla deja acolo, in caz afirmativ face update

```
public void insertProduct(String line) {
	String[] values = line.split("[: ,]+");
	Product produs = new Product();
	produs.setNume(values[2]);
	produs.setCantitate(Integer.parseInt(values[3]));
	produs.setPret(Double.parseDouble(values[4]));
	ProductBLL cd = new ProductBLL();
	Product find = new Product();
	find = cd.findProduct(values[2]);
	if (find.getNume() == null) {
		cd.insertProduct(produs);
	} else {
		Product produs1 = new Product();
		produs1 = produs;
		produs1.setCantitate(produs.getCantitate() + find.getCantitate());
		cd.updateProduct(produs1);
	}
}
```

Clasa Parser

- clasa in care se citesc datele din fisierul text primit ca argument si se extrag comenzile din acesta

- in functie de comanda citita din fisierul text se apeleaza functia corespunzatoare din clasa GenerateReports

Pachetul main contine clasa main.

- in clasa main se creeaza un obiect de tipul parser care va porni executia programului

5 Rezultate

Rezultatele programului vor fi stocate in pdf-uri in functie de comanda primita.

Exemple de comenzi si rezultatul acestora:

- Order: Luca George, lemon, 5

- daca comanda este order se va genera o facture pentru clientul respective;

```
Data/Ora:  2020-04-15 23:49:50.746
Factura numarul: 1
Nume client:  Luca George
Produs cumparat:  apple
Cantitate:  5
Pret total:  5.0
```

- Order: Sandu Vasile, apple, 100

```
Comanda esuata pentru clientul Sandu Vasile. Nu sunt
suficiente produse in stoc.
```

- daca numarul de produse comandate nu este mai mic decat numarul produselor din stoc se va genera un pdf numit FailedOrder cu un mesaj corespunzator;

- Report product

Nume	Cantitate	Pret
apple	35	1.0
lemon	65	2.0
orange	40	1.5
peach	50	2.0

- daca se introduce comanda report + numele unei tabele se vor afisa datele din tabela respective sub forma unui tabel ca cel de mai sus;

6 Concluzii

In concluzie, datorita acestei teme am reusit sa ma familiarizez cu proiectarea unei aplicatii care este conectata la o baza de date, dar si cum sa conectez un program la baza de date. Deasemenea am invatat un nou tip de structurare al unei aplicatii, si anume Layered Architecture.

7.Bibliografie

-https://www.baeldung.com/java-pdf-creation

-http://tutorials.jenkov.com/java-reflection/index.html

-https://dzone.com/articles/layers-standard-enterprise

- https://stackoverflow.com/