

**FACULTATEA: Automatica si Calculatoare**

**SPECIALIZAREA: Calculatoare si tehnologia informatiei**

**DISCIPLINA: Tehnici de Programare**

**AN UNIVERSITAR: 2019/2020**

**PROIECT: Order Management**

**Realizat de :**

Ghiura Darius

**Grupa:** 30227

**CUPRINS**

1. **Obiectivul temei**
2. **Analiza problemei, modelare, scenarii, cazuri de utilizare**
3. **Proiectare**
4. **Implementare**
5. **Rezultate**
6. **Concluzii**
7. **Bibliografie**
8. Obiectivul temei

Obiectivul temei este proiectarea si implementarea unei aplicatii care proceseaza comenzile unui depozit. Aplicatia trebuie sa poata stoca produse, clienti si comenzi in tabelele unei baze de date relationale.

Aplicatia citeste dintr-un fisier operatiile de efectuat si creeaza fisiere PDF care pot fi : o factura cand este plasata o comanda, un mesaj de under-stock daca stockul unui produs nu este suficient pentru a realiza comanda sau un tabel care afiseaza tabelul din baza de date.

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

Pentru a implementa un sistem de gestionare a unui depozit, aplicatia trebuie sa poata insera, sterge si modifica clienti, produse si comenzi din baza de date.

Aplicatia citeste instructiunile dintr-un fisier, deci are nevoie de un Parser care sa preia acele instructiuni si sa le execute.

Instructiunile din fisier pot fi :

* Insert client: Ion Popescu, Bucuresti : Insereaza clientul cu numele Ion Popescu si adresa Bucuresti in baza de date
* Delete client: Ion Popescu : Sterge clientul cu numele Ion Popescu din baza de date
* Insert product: apple, 20, 1 : Insereaza produsul cu numele „apple”, cantitatea 20 si pretul 1 in baza de date
* Delete product: apple : Sterge produsul cu numele „apple” din baza de date
* Order: Ion Popescu, apple, 5 : Insereaza o comanda in baza de date pentru clientul cu numele Ion Popescu, produsul cu numele „apple” si pretul 5
* Report client : Genereaza un fisier PDF cu tabelul Client din baza de date
* Report order : Genereaza un fisier PDF cu tabelul Order din baza de date
* Report product : Genereaza un fisier PDF cu tabelul Product din baza de date

Pentru fiecare comanda plasata un fisier PDF cu numele „order + id-ul comenzii + „\_bill.pdf” va fi creat. Daca comanda nu se poate efectua din cauza stocului insuficient al unui anumit produs se va crea un fisier PDF cu numele : „order + id\_ul clientului + \_understock.pdf”.

Dupa finalizarea unei comenzi, stocul produselor comandate va fi actualizat.

Baza de date relationala este realizata in MySQL si are numele „ordermanagement” si este formata din 4 tabele :

* Client – date despre clienti
* Produs – date despre produse
* OrderTotal – date despre o comanda
* OrderItem – date despre produsele comandate intr-o comanda

**Utilizare :**

Cand incepe rularea aplicatiei, se vor citi instructiunile din fisier si se vor executa secvential. Output-ul aplicatiei este

Aplicatia poate fi rulata din consola utilizand fisierul jar, folosind comanda :

***java -jar PT2020\_30227\_Darius\_Ghiura\_Assignment\_3.jar commands.txt***

Unde „commands.txt” este numele fisierului cu date de intrare, unde se afla instructiunile de rulat.

1. Proiectare

Proiectul a fost realizat dupa modelul **Layered Architecture** . Acest model presupune impartirea proiectului in 4 pachete principale :

* **dataAccessLayer** – Contine clasa care face conexiunea cu baza de date
* **businessLayer** – Contine clasele ce implementeaza logica aplicatiei si logica instructiunilor pentru fiecare tabel
* **model** – Contine clasele care reprezinta tabele din baza de date cu campurile lor
* **presentation** – Contine clasele care au legatura cu input-ul / output-ul aplicatiei. Adica Parser-ul fisierului de intructiuni, clasele care creeaza fisierele PDF.

Pe langa aceste pachete, mai exista si :

* **main** – Contine clasa care reprezinta startul aplicatie si are metoda main a proiectului
* **dao** (Data Acces Object) – Contine clasele care implementeaza statement-urile in SQL pentru fiecare tabel

Din punct de vedere al structurilor de date, pentru returnarea rezultatelor al statement-urilor de SELECT se folosec liste de tipul modelului tabelului (ex. pentru clasa client List<Client>). Pentru salvarea instructiunilor citite de parser se foloseste un ArrayList<Command>.

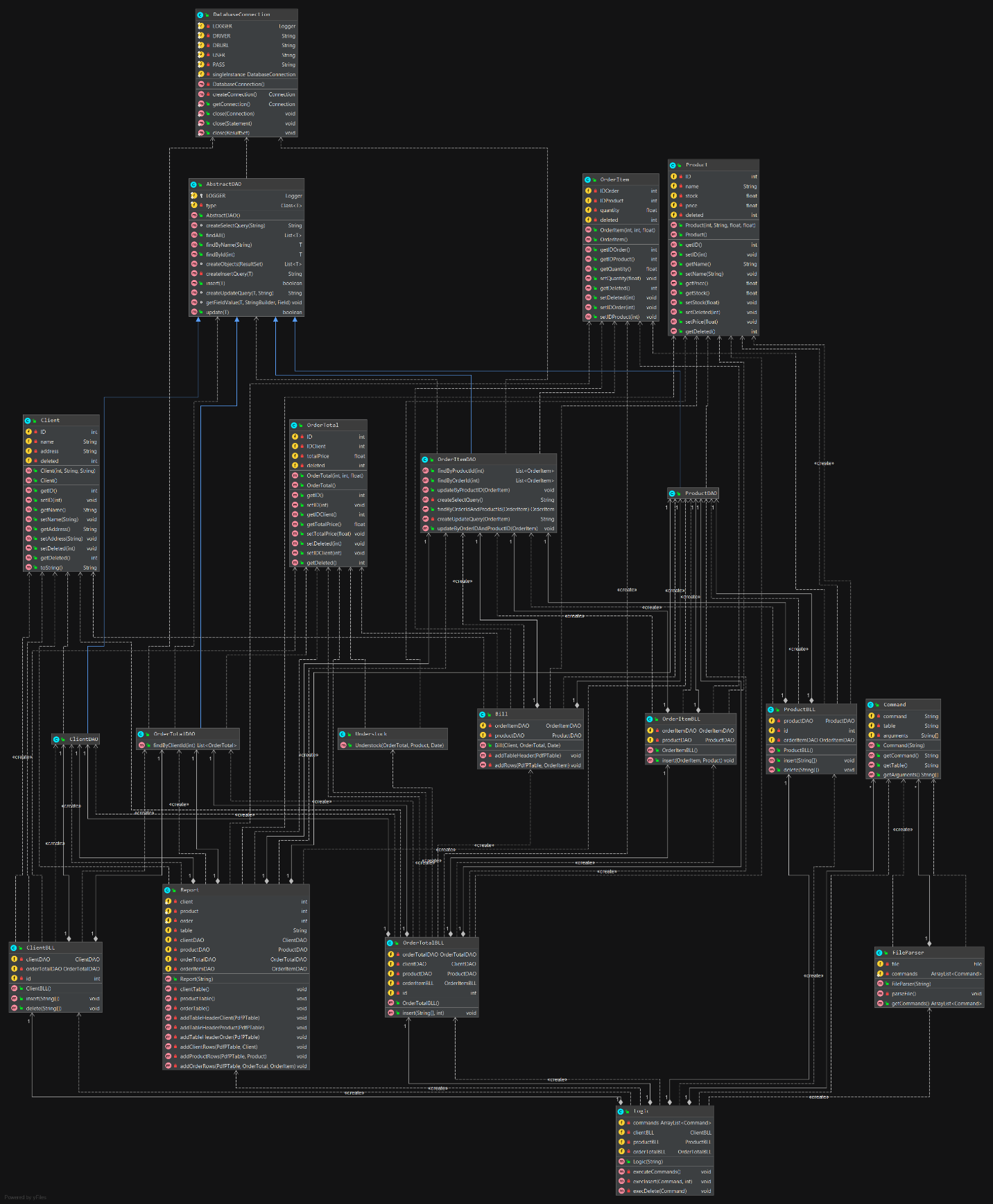
Clasele au fost proiectate respectand conceptul de programare orientata pe obiect de incapsulare. Variabilele instanta sunt declarate cu modificatorul de acces „private” ele fiind accesibile din exteriorul claselor doar prin gettere si settere.

Daca sunt mai multe instructiuni de tipul Order de la acelasi client, se va crea o singura comanda in tabelul OrderTotal si in tabelul OrderItem vor fi inserate sub acelasi idOrder toate produsele asociate comenzii. Factura va fi generat doar dupa ce nu mai avem o instructiune Order de la acelasi client.

Instructiunea delete nu rezulta in executarea unui statement in SQL de DELETE ci se seteaza coloana **deleted** la true.

A fost folosita “reflection technique” care presupune implementarea unei clase generice care contine metode pentru a interactiona cu baza de date : gasirea, inserarea, stergerea, modificarea unui obiect. Query-urile de baza pentru fiecare tabel vor fi generate dinamic prin reflexie.

**Diagrama UML**



1. Implementare

**Clasa : MainClass**

Aceasta clasa contine metoda main a proiectului.

**Clasa : DatabaseConnection**

Variabile instanta :

* **private static final Logger LOGGER** = Logger.getLogger(DatabaseConnection.class.getName());
* **private static final String DRIVER** = "com.mysql.cj.jdbc.Driver" – Driverul pentru conexiunea la baza de date
* **private static final String DBURL** = "jdbc:mysql://localhost:3306/ordermanagement" – Baza de date in MySQL
* **private static final String USER** = "root" – Userul pentru baza de date in MySQL
* **private static final String PASS** = "root"- Parola pentru baza de date in MySQL

**Pachetul model :**

**Clasa : Client**

Variabile instanta :

* **private int ID;**
* **private String name;**
* **private String address;**
* **private int deleted = 0** – 0 daca nu a fost sters din tabel, 1 daca a fost sters

Constructor :

**public Client(int ID, String name, String address)**

Metode :

* **public String getName()** – Returneaza numele cu ghilimele pentru a putea folisit in statement-urile SQL (‘Ion’)
* **public String getAddress()** – Returneaza adresa cu ghilimele pentru a putea folisit in statement-urile SQL (‘Bucuresti’)

**Clasa : Product**

Variabile instanta :

* **private int ID;**
* **private String name;**
* **private float stock;**
* **private float price;**
* **private int deleted = 0 -** 0 daca nu a fost sters din tabel, 1 daca a fost sters

Constructor :

**public Product(int ID, String name, float stock, float price)**

Metode :

* **public String getName()** – Returneaza numele cu ghilimele pentru a putea folisit in statement-urile SQL (‘apple’)

**Clasa : OrderTotal**

Variabile instanta :

* **private int ID;**
* **private int IDClient;**
* **private float totalPrice;**
* **private int deleted = 0 -** 0 daca nu a fost sters din tabel, 1 daca a fost sters

Constructor :

**public OrderTotal(int ID, int IDClient, float totalPrice)**

**Clasa : OrderItem**

Variabile instanta :

* **private int IDOrder;**
* **private int IDProduct;**
* **private float quantity;**
* **private int deleted = 0 -** 0 daca nu a fost sters din tabel, 1 daca a fost sters

Constructor :

**public OrderItem(int IDOrder, int IDProduct, float quantity)**

**Pachetul presentation :**

**Clasa : Bill**

Variabile insanta :

* **private OrderItemDAO orderItemDAO = new OrderItemDAO();**
* **private ProductDAO productDAO = new ProductDAO();**

Constructor :

**public Bill(Client c, OrderTotal ot, Date d)** – Creeaza un document PDF care contine 3 paragrafe cu text despre comanda, client si pret, un tabel care reprezinta produsele comandate. Folosind statement-urile din DAO gaseste OrderItemele care corespund orderului ot si adauga in tabel informatii despre produs : ID, nume, cantitatea comandata, pretul.

Foloseste plugin-ul Itext pentru a crea PDF-ul

Metode :

* **private void addTableHeader(PdfPTable table)** – Adauga header-ul tabelului
* **private void addRows(PdfPTable table, OrderItem i)** – Adauga un rand al tabelului cu datele despre produsul din OrderItem i

**Clasa : Command**

Variabile insanta :

* **private String command;**
* **private String table;**
* **private String[] arguments;**

Constructor :

**public Command(String command)** – Primeste o linie din fisier citita de FileParser si imparte string-ul ca sa formeze o instructiune. Variabila instanta command primeste “Insert” , “Delete” sau “Report”; table primeste numelele tabelului pe care trebuie executata instructiunea; arguments primeste string-uri cu informatii precum : numele clientlui, adresa, numele produsului etc.

Metode :

* **public String getCommand()**
* **public String getTable()**
* **public String[] getArguments()**

**Clasa : FileParser**

Variabile insanta :

* **private File file;**
* **private static ArrayList<Command> commands;**

Constructor :

**public FileParser(String fileName)** – Creeaza un File cu numele primit ca parametru si apeleaza metoda parseFile()

Metode :

* **private void parseFile() throws FileNotFoundException** – Citeste cate o linie din fisier folosind un Scanner si adauga in ArrayList-ul de instructiuni o comanda noua creata de constructorul din clasa Command
* **public ArrayList<Command> getCommands()**

**Clasa : Report**

Foloseste plugin-ul Itext pentru a crea PDF-urile

Variabile insanta :

* **private static int client = 0 –** Variabila folosita pentru crearea numelui PDF-ului
* **private static int product = 0 –** Variabila folosita pentru crearea numelui PDF-ului
* **private static int order = 0 –** Variabila folosita pentru crearea numelui PDF-ului
* **private String table;**

Constructor :

**public Report(String table)** – Primeste un string care reprezinta numele tabelului cu care trebuie executata comanda Report si apeleaza metoda corespunzatoare

Metode :

* **private void clientTable()** –Creeaza un document nou cu numele “reportclient + “\_”client + “.pdf”. Creeaza un tabel pentru clientii care au fost gasiti de metoda findAll() (Care reprezinta Select \*). Apeleaza addTableHeaderClient(table) pentru a adauga header-ul tabelului si addClientRows(table,c) cu fiecare client din lista pentru a adauga randurile tabelului
* **private void productTable()** – Acelasi principiu ca la clientTable()
* **private void orderTable() –** Face un tabel in care afiseaza OrderItemele unui Order
* **private void addTableHeaderClient(PdfPTable table)** – Adauga header-ul tabelului de Client
* **private void addTableHeaderProduct(PdfPTable table)** – Adauga header-ul tabelului de Product
* **private void addTableHeaderOrder(PdfPTable table)** – Adauga header-ul tabelului de Order
* **private void addClientRows(PdfPTable table, Client c)** – Adauga un rand in tabel folosind metoda addCell()
* **private void addProductRows(PdfPTable table, Product p)** – Adauga un rand in tabel folosind metoda addCell()
* **private void addOrderRows(PdfPTable table, OrderTotal o, OrderItem i)** – Adauga un rand in tabel folosind metoda addCell()

**Clasa : Understock**

Foloseste plugin-ul Itext pentru a crea PDF-urile.

Constructor :

**public Understock(OrderTotal o, Product p, Date d)** – Creeaza un PDF in care se afiseaza un mesaj despre anularea comenzii plasata de un client din motive de stoc insuficient.

**Pachetul dao :**

**Clasa : AbstractDAO**

Folosind tehnica reflexiei implementeaza operatiile de baza pentru interactiunea cu baza de date. Metodele gasesc numele atributelor obiectelor folosind metoda getDeclaredFields() apoi cu un PropertyDescriptor gasesc getter-ul sau setter-ul atributului folosind getReadMethod(), respectiv getWriteMethod().

Variabile insanta :

* **protected static final Logger LOGGER = Logger.getLogger(AbstractDAO.class.getName());**
* **private final Class<T> type;**

Constructor :

**public AbstractDAO()**

Metode :

* **public List<T> findAll()**
* **public T findByName(String name)**
* **public T findById(int id)**
* **public boolean insert(T t)**
* **public boolean update(T t)**

**Clasa : ClientDAO**

Extinde clasa generica AbstractDAO cu tipul Client

**Clasa : ProductDAO**

Extinde clasa generica AbstractDAO cu tipul Product

**Clasa : OrderItemDAO**

Extinde clasa generica AbstractDAO cu tipul OrderItem

Folosind acelasi principiu ca in clasa AbstractDAO implementeaza operatiile urmatoare :

Metode :

* **public List<OrderItem> findByProductId(int id)**
* **public List<OrderItem> findByOrderId(int id)**
* **public void updateByProductID(OrderItem t)**
* **public OrderItem findByOrderIdAndProductId(OrderItem i)**
* **public void updateByOrderIDAndProductID(OrderItem i)**

**Clasa : OrderTotalDAO**

Extinde clasa generica AbstractDAO cu tipul OrderTotal

Folosind acelasi principiu ca in clasa AbstractDAO implementeaza operatia urmatoare :

Metoda :

**public List<OrderTotal> findByClientId(int id)**

**Pachetul businessLayer :**

**Clasa : Logic**

Variabile insanta :

* **private ArrayList<Command> commands;**
* **private ClientBLL clientBLL;**
* **private ProductBLL productBLL;**
* **private OrderTotalBLL orderTotalBLL;**

Constructor :

**public Logic(String inputFile)** – Primeste din main numele fisierului de input si apeleaza FileParser(inputFile).getCommands() pentru a primi ArrayList-ul de instructiuni, initializeaza celelalte variabile instanta si apeleaza metoda care proceseaza lista de instructiuni.

Metode :

* **private void executeCommands()** – Parcugand ArrayList-ul de comenzi se verifica pentru fiecare tipul de comanda : daca este “Insert” se apeleaza execInsert(), “Delete” se apeleaza execDelete sau “Report” se creeaza un nou PDF.

Pentru cazul de insert avem o variabila “ok” care este 0 daca urmatoarea instructiune este una de Order de la acelasi clienti astfel factura nu trebuie sa se genereze inca si este 1 daca factura trebuie sa se genereze

* **private void execInsert(Command c, int ok)** – In functie de numele tabelului din instructiune apeleaza metoda de insert din clasa BLL corespunzatoare
* **private void execDelete(Command c)** – Apeleaza functia de delete din clasa BLL corespunzatoare tabelului

**Clasa : ClientBLL**

Variabile insanta :

**private int id = 1** – Se incrementeaza cand se insereaza un client nou

Constructor :

**public ClientBLL()** – Initializeaza variabilele instanta

Metode :

* **public void insert(String[] arguments)** – Creeaza un client nou cu argumentele primite si apeleaza metoda de insert() din clientDAO
* **public void delete(String[] arguments)** – In arguments avem numele clientului care trebuie sters, astfel se apeleaza din clasa clientDAO metoda de findByName(), apoi se seteaza variabila de deleted la 1 clientului gasit si se face apeleaza metoda de update din clientDAO. De asemenea trebuie sterse si comenziile facute de clientul respectiv, asta se realizeaza apeland metoda findByClientId() din clasa orderTotalDAO pentru a gasi comenzile facute de acel client, apoi se seteaza variabila deleted la 1 si se face update tabelei de OrderTotal pentru fiecare order a clientului.

**Clasa : ProductBLL**

Constructor :

**public ProductBLL()**

Metode :

* **public void insert(String[] arguments) –** Folosind metoda findByName() din clasa productDAO stabileste daca exista deja un produs cu numele produsului pe care vrem sa il inseram. In caz afirmativ se face update la tabelul Product la stocul produsului respectiv, altfel se face inser in tabel
* **public void delete(String[] arguments)** – In arguments avem numele produsului care trebuie sters, astfel se apeleaza din clasa productDAO metoda de findByName(), apoi se seteaza variabila de deleted la 1 produsului gasit si se face apeleaza metoda de update din productDAO. De asemenea trebuie sterse si OrderItemele cu produsul respectiv, asta se realizeaza apeland metoda findByProductId() din clasa orderItemDAO pentru a gasi OrderItemele cu acel produs, apoi se seteaza variabila deleted la 1 si se face update tabelei de OrderItem pentru fiecare OrderItem care contine produsul.

**Clasa : OrderItemBLL**

Constructor :

**public OrderItemBLL()**

Metode :

**public void insert(OrderItem i, Product p)** – Insereaza un OrderItem in tabel daca nu a existat un OrderItem care sa aiba acelasi idOrder si idProduct, caz in care ii face update cantitatii acelui element. Altfel face insert in tabel si la final face update stocului produsului.

**Clasa : OrderTotalBLL**

Variabile insanta :

**private int id = 1** – Se incrementeaza cand se insereaza un Order nou

Constructor :

**public OrderTotalBLL()**

Metode :

**public void insert(String[] arguments, int ok)** – Insereaza in tabelele de OrderTotal si OrderItem daca totul e valid. Daca nu e destul stoc apeleaza new Understock() pentru a crea un PDF cu mesajul potrivit. Daca ok = 1 apeleaza new Bill() pentru a crea o factura.

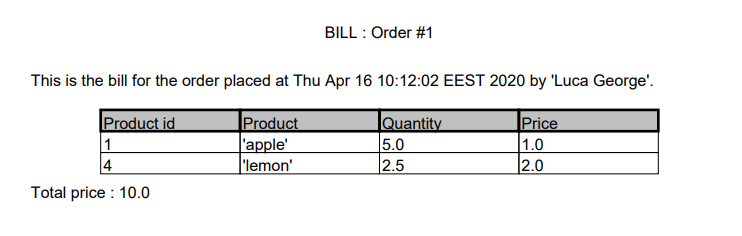
1. Rezultate

Aplicatia a fost testata manual, verificand fisierele PDF create si in comparand cu datele din MySQL.

Asa arata o factura pentru un client care a comandat doua produse dupa executarea instructiunilor :

Order: Luca George, apple, 5

Order: Luca George, lemon, 5



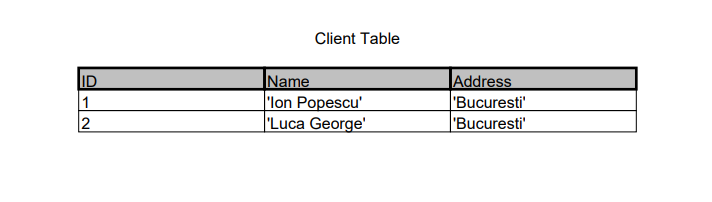
In caz de stoc insuficient un document cu un mesaj asemanator va fi generat :



Report pentru tabelul client dupa executarea instructiunilor :

Insert client: Ion Popescu, Bucuresti

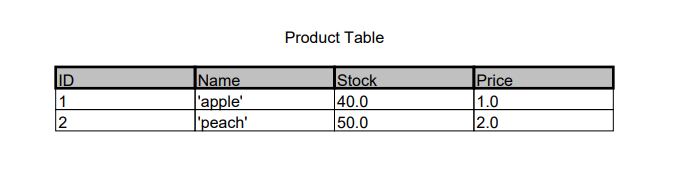
Insert client: Luca George, Bucuresti



Report pentru tabelul product dupa executarea instructiunilor :

Insert product: apple, 20, 1

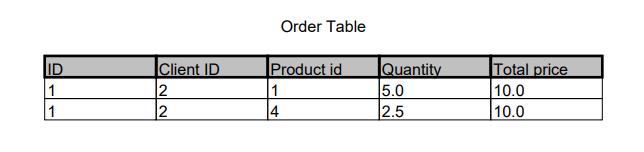
Insert product: peach, 50, 2



Report pentru tabelul Order dupa executarea instructiunilor:

Order: Luca George, apple, 5

Order: Luca George, lemon, 5



1. Concluzii

Aplicatia implementeaza cu succes un sistem de procesare a comenzilor pentru un depozit in care se pot manipula datele unor clienti, produse si comenzi.

Proiectul a fost folositor pentru invatarea tehnicii de Reflection, modelului de design Layered Architecture si pentru lucrul cu baze de date folosind Java

1. Bibliografie

<https://dev.mysql.com/doc/refman/5.7/en/using-mysqldump.html>

<https://www.baeldung.com/javadoc>

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

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

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

<https://bitbucket.org/utcn_dsrl/pt-reflection-example/src/master/src/main/java/dao/AbstractDAO.java>

<https://bitbucket.org/utcn_dsrl/pt-reflection-example/src/master/src/main/java/connection/ConnectionFactory.java>