MINISTRY OF EDUCATION AND RESEARCH
TECHNICAL UNIVERSITY
OF CLUJ-NAPOCA, ROMANIA

FUNDAMENTAL PROGRAMMING TECHNIQUES
ASSIGNMENT 3
ORDER MANAGEMENT

By Mihali Vlad
Group: 30424

Content

1. The objective of this assignment

Consider an application OrderManagement for processing customer orders for a warehouse. Relational databases are used to store the products, the clients and the orders. Furthermore, the application is structured in packages using a layered architecture.

The main goal of this project is to design and implement a order management.
The secondary objectives are:

- parsing the input data; (page 15, Chapter 4.6.1.)
- create database;
- design and implement Model classes; (page 4, Chapter 3. and also page 10, Chapter 4.5.)
- connect to database via application; (page 4, Chapter 3. and also page 7, Chapter 4.2.)
- generates and manages queries ; (page 4, Chapter 3. and also page 7, Chapter 4.3.)
- design and implement logic layer; (page 4, Chapter 3. and also page 5, Chapter 4.1.)
- design and implement output class;(page 16, Chapter 4.6.2.)
- system testing; (page 17, Chapter 5.1.)

Parsing input data is done by extracting strings from file which is given as first command-line argument and convert into processable data. This are than by the class ParseInput.
This are the possible inputs data read from a text file for the application:

Command name	Command Syntax	Description
Add client to the database	Insert client: Ion Popescu, Bucuresti	Insert in the database a new client with name Ion Popescu and address Bucuresti
Delete Client from the database	Delete client: Ion Popescu	Delete from database the client with name Ion Popescu
Add product to the database	Insert product: apple, 20, 1	Add product apple with quantity 20 and price 1
Delete product from the database	Delete product: apple	Delete product apple from database
Create order for client	Order: Ion Popescu, apple, 5	Create order for Ion Popescu, with apple quantity 5. Also update the apple stock to 15. Generate a bill in pdf format with the order and total price of 5
Generate reports	Report client Report order Report product	Generate pdf reports with all clients/orders/products displayed in a tabular form. The reports should contain the information corresponding to the entity for which reports are asked (client, order or product) returned from the database by a SELECT * query, displayed in a table in a PDF file.

Create database using MySQL Workbench. The database will have 4 tabels:

- Client – is data from persons which can buy products
- Product – is data from products that are for sales
- Order – is data from clients that are buying some product
- OrderDetail – is data that represents the detail of the order

In my database implementation I used the next approach and the most used in the industry, do not use the DELETE command at all, but use an extra column, a FLAG that says whether that field in the table is deleted or not. In case you delete a customer with orders, set the deleted flag to those orders, and then set the deleted flag to the customer. If you use this approach, it is very important to keep in mind: ANY SELECT query must verify that the data it takes has the flag deleted = false.

All the other classes work with model reflected classes that are the "mirror" of the database.

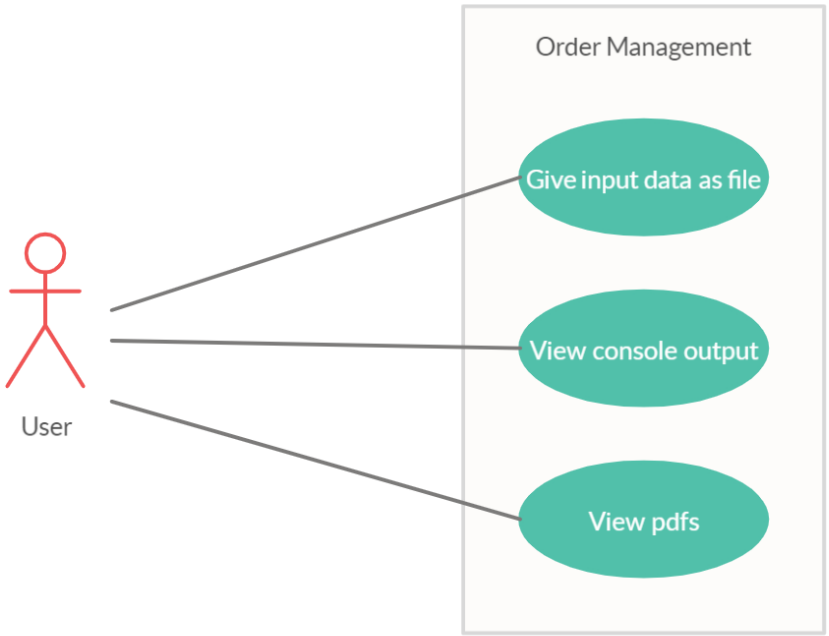
The application use layers so from main => goes to parsing input (ParseInput) => validate and logic the command (one of the Business Layer classes) => apply queries to database => return data from database and print in pdf format (Presentation)

Output class is the Presentation class that is like a user interface. If it is main scenario you will have just pdf outputs else you will have errors printed in console and the application will be close automatically.

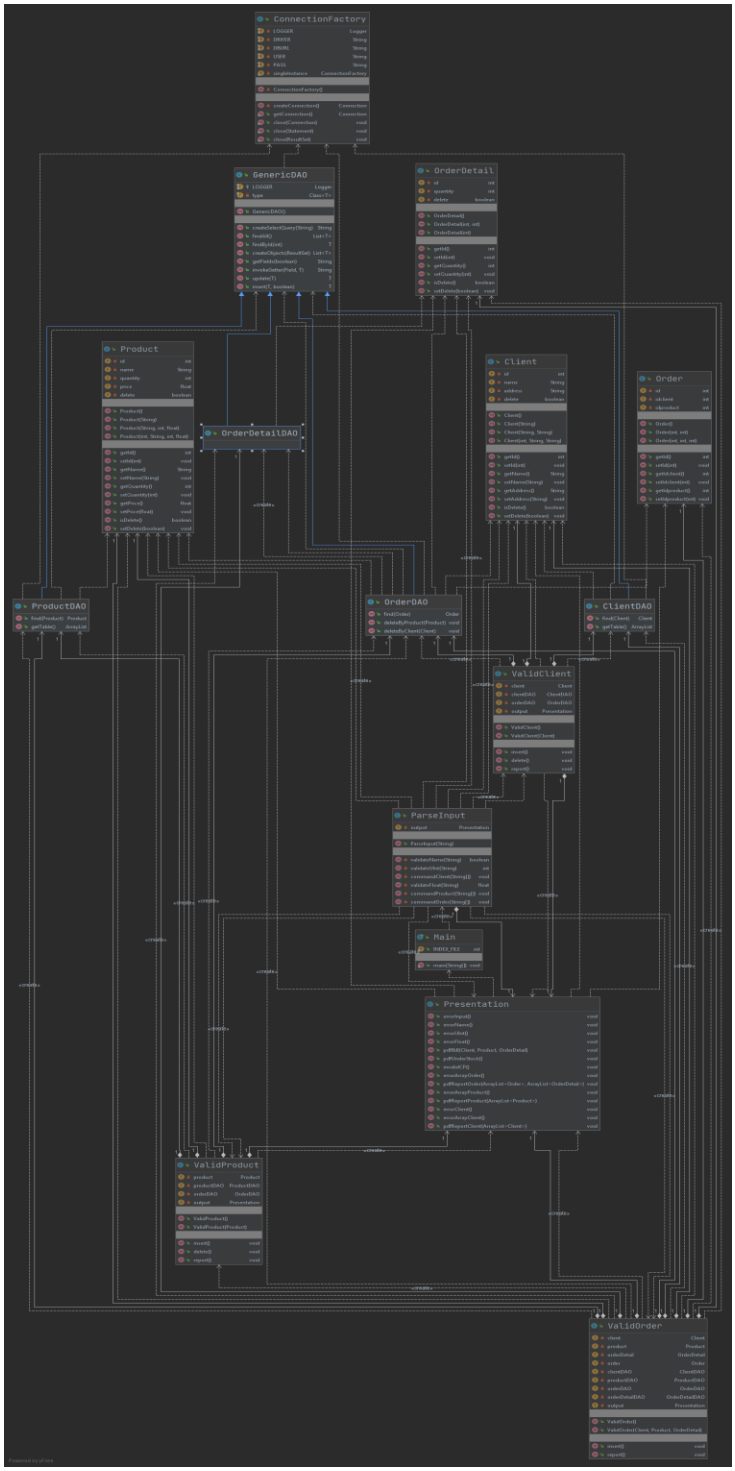
In the system testing we will test the functionality of the application and the response of the input data and the simulation flow.

2. Analysis of the problem

Use case of the application[1]:



System: queue simulation (application)
Actors: The user
Use case is a tool to analyze and divide the problem in small pieces easy to resolve. The main success scenario is the user provide a valid input file and the application is run properly and the user can see the pdf that reflects the database. The alternative (failure) scenario is when the user provides an invalid data or does not connect to database and an error message will be displayed in the console.



3. Design

Class diagram[1]:

In my design I used six packages (tier architecture) : The application is split in different layers. Each layer has a special purpose and calls functions of the layers below it:

- Presentation Layer - contains the classes defining the user interface (ParseInput, Presentation)
- Business Layer – contains the classes that encapsulate the application logic (ValidClient, ValidOrder, ValidProduct)
- Data Access Layer – contains the classes containing the queries and the database connection (ClientDAO, GenericDAO, OrderDAO, OrderDetailDAO, ProductDAO)
- Model – contains classes mapped to the database table (Client, Order, OrderDetail, Product)
- Main – contains the entry point into application(Main)
- Connection – establish connection to database (ConnectionFactory)

Encapsulation is a mechanism of wrapping the variables and code acting on the methods together as a single unit. In encapsulation, the variables of a class will be hidden from other classes and can be accessed only through the methods of their current class. Therefore, it is also known as data hiding. [3]

Reflection is an API which is used to examine or modify the behavior of methods, classes, interfaces at runtime.

- The required classes for reflection are provided under java.lang.reflect package.
- Reflection gives us information about the class to which an object belongs and the methods of that class which can be executed by using the object.
- Through reflection we can invoke methods at runtime irrespective of the access specifier used with them.

The idea of **generics** is to allow type (Integer, String, … etc and user defined types) to be a parameter to methods, classes and interfaces.

To obtain an object-oriented programming design, I used following principles: encapsulation, inheritance, polymorphism.

Data structures used:
ArrayList is used especially in Presentation, ClientDAO and ProductDAO to store dynamically sized collection of Client, Product, Order and OrderDetail objects. Contrary to Arrays that are fixed in size, an ArrayList grows its size automatically when new elements are added to it and are easy to use.

4. Implementation

4.1. businessLayer

4.1.1. ValidClient

public class **ValidClient**
validate and manage a command for the client
private Client client
is private field for the client of the command
private ClientDAO clientDAO
provides data access for the client table
private OrderDAO orderDAO
provides data access for the order table
private Presentation output
is a private presentation which will help the class to output
Constructors
public ValidClient()
is an empty constructor of ValidClient class
public ValidClient(Client client)
is a constructor of the ValidClient class
Parameters:
client - is used to initialize client attribute
Methods
public void insert()
try to insert the client in the client table from database
public void delete()
try to update "delete" row with 1(true) in the client table from database and "deletes" all the orders associated with it
public void report()
try to select all clients and output them in pdf

4.1.2. ValidOrder

public class **ValidOrder**
validate and manage a command for the client
private Client client
is private field for the client of the command
private ClientDAO clientDAO
provides data access for the client table
private Order order
is private field for the order of the command
private OrderDAO orderDAO
provides data access for the order table

private OrderDetail orderDetail
is private field for the orderDetail of the command
private OrderDetailDAO orderDetailDAO
provides data access for the orderDetail table
private Presentation output
is a private presentation which will help the class to output
private Product product
is private field for the product of the command
private ProductDAO productDAO
provides data access for the product table
Constructors
public ValidOrder()
is an empty constructor of ValidOrder class
public ValidOrder(Client client,
Product product,
OrderDetail orderDetail)
is a constructor of the ValidOrder class
Parameters:
client - is used to initialize client attribute
product - is used to initialize product attribute
orderDetail - is used to initialize orderDetail attribute
Methods
public void insert()
try to insert the order in the order table from database and update product stock
public void report()
selects all orders and all the orderDetails and output them in pdf

4.1.3. ValidProduct

public class **ValidProduct**

validate and manage a command for the client
private OrderDAO orderDAO
provides data access for the order table
private Presentation output
is a private presentation which will help the class to output
private Product product
provides data access for the product table
private ProductDAO productDAO
provides data access for the product table
Constructors
public ValidProduct()
is an empty constructor of ValidProduct class
public ValidProduct(Product product)
is a constructor of the ValidProduct class
Parameters:
product - is used to initialize product attribute
Methods
public void insert()
try to insert the client in the client table from database
public void delete()

try to update "delete" row with 1(true) in the product table from database and "deletes" all the orders associated with it
public void report()
try to select all products and output them in pdf

4.2. connection

4.2.1. ConnectionFactory

public class **ConnectionFactory**
Class that connects the app and the database.
private static ConnectionFactory singleInstance
connection is being created only once (single connection)
Methods
public static java.sql.Connection getConnection()
attribute singleInstance will create a connection
Returns:
a new connection
public static void close(java.sql.Connection connection)
trying to lose the connection given as parameter
Parameters:
connection - which it will be close
public static void close(java.sql.Statement statement)
trying to close the statement given as parameter
Parameters:
statement - which it will be close
public static void close(java.sql.ResultSet resultSet)
trying to close the resultSet given as parameter
Parameters:
resultSet - which will be close

4.3. dataAccessLayer

4.3.1. ClientDAO

public class **ClientDAO**
extends GenericDAO<Client>
is a class that inheritance GenericDAO and access the table client in database
private Class<T> type
is a generic private field
Constructors
public ClientDAO()
Methods
public Client find(Client client)
access the client table to find the row by its name
Parameters:
client - is the client witch are searching for
Returns:
the find client if it is else null
public ArrayList getTable()
gets the client table with "delete" = 0
Returns:

array of all clients available

4.3.2. GenericDAO

public class **GenericDAO<T>**
access database using general commands
GenericDAO
public GenericDAO()
is an empty constructor of GenericDAO that extract the general type
Methods
public String createSelectQuery(String field)
automatically create a select query by a condition
Parameters:
field - that is used to select
Returns:
a string with a select query template
public List<T> findAll()
try to connect to database and finds all generic object in the table
Returns:
list of all generic object in the generic table
public T findById(int id)
try to connect to database and find the row by its unique identifier
Parameters:
id - is the primary key we are searching for
Returns:
the object searched if it is possible else null
public List<T> createObjects(java.sql.ResultSet resultSet)
try transform the result of a query into list of objects
Parameters:
resultSet - is a result of a query
Returns:
a list of generic objects
public String getFields(boolean skip)
gets all the fields of the generic object but skip first if it is true
Parameters:
skip - it is set true if you want also the id else false
Returns:
string format of the fields name
public String invokeGetter(reflect.Field field,
T t)
calls getter of the field from the generic class
Parameters:
field - is the field from we are getting the getter
t - is the object from we are getting the getter
Returns:
a string format of the return from the getter
public T update(T t)
try to connect to database and update by the parameter into the generic table
Parameters:
t - is the object that will be update in the database
Returns:
object that is update

public T insert(T t,
boolean skip)
try to connect to database and insert by the parameter into the generic table
Parameters:
t - is the object that will be update in the database
skip - is set true if you want to insert also the id else false
Returns:
object that is inserted

4.3.3. OrderDAO

public class **OrderDAO**
extends GenericDAO<Order>
is a class that inheritance GenericDAO and access the table order in database
Constructors
public OrderDAO()
Methods
public Order find(Order order)
access the order table to find the row by its client and product
Parameters:
order - is the order which are searching for
Returns:
the find order if it is else null
public void deleteByProduct(Product p)
is a method that updates delete entry with true all the rows that have the same id product equal with the parameter
Parameters:
p - is the product that we want to delete its orders
public void deleteByClient(Client c)
is a method that updates delete entry with true all the rows that have the same id client equal with the parameter
Parameters:
c - is the client that we want to delete its orders

4.3.4 OrderDetailDAO

public class **OrderDetailDAO**
extends GenericDAO<OrderDetail>
is a class that inheritance GenericDAO and access the table orderDetail in database

4.3.5 ProductDAO

public class **ProductDAO**
extends GenericDAO<Product>
is a class that inheritance GenericDAO and access the table product in database
Constructors
public ProductDAO()
Methods
public Product find(Product product)
access the product table to find the row by its name
Parameters:
product - is the product which are searching for
Returns:
the find product if it is else null

public ArrayList getTable()
gets the product table with "delete" = 0
Returns:
array of all products available

4.4 main

4.4.1. Main

public class **Main**
Main is the main entity of the application
Fields
public static int INDEX_FILE
this is a public static param which keep track of writing pdf files
Constructors
public Main()
Methods
public static void main(String[] args)
starts the application
Parameters:
args - are arguments of the application

4.5. model

4.5.1. Client

public class **Client**
is a public reflation class for the client table in the database
Fields
private int id
is a private attribute that identifies unique record from Client (primary key)
private String name
is the first name and last name of the client
private String address
is the address of the client
private oolean delete
is a private attribute that takes true if the record client is “deleted”
Constructors
public Client()
is an empty constructor of client class
public Client(String name)
is a constructor of client class
Parameters:
name – is used to initialize name attribute
public Client(String name,
String address)
is a constructor of client class
Parameters:
name – is used to initialize name attribute
address – is used to initialize address attribute
public Client(int id,
String name,

String address)
is a constructor of client class
Parameters:
id – is used to initialize id attribute
name – is used to initialize name attribute
address – is used to initialize address attribute
Methods
public int getId()
is the getter method for id attribute
Returns:
id attribute
public void setId(int id)
is the setter method for id attribute
Parameters:
id – is used to assign id attribute
public String getName()
is the getter method for name attribute
Returns:
name attribute
public void setName(String name)
is the setter method for name attribute
Parameters:
name – is used to assign name attribute
public String getAddress()
is the getter method for address attribute
Returns:
address attribute
public void setAddress(String address)
is the setter method for address attribute
Parameters:
address – is used to assign address attribute
public oolean isDelete()
is the getter method for delete attribute
Returns:
address delete
public void setDelete(oolean delete)
is the setter method for delete attribute
Parameters:
delete – is used to assign address attribute

4.5.2. Order

public class **Order**
is a public reflation class for the order table in the database
Fields
private int id
is a private attribute that identifies unique record from Order (primary key)
private int idclient
is a private attribute that identifies the client (foreign key)
private int idproduct
is a private attribute that identifies the product (foreign key)
Constructors

Order
public Order()
is an empty constructor of order class
public Order(int idclient,
int idproduct)
is a constructor of order class
Parameters:
idclient - is used to initialize idclient attribute
idproduct - is used to initialize idproduct attribute
public Order(int id,
int idclient,
int idproduct)
is a constructor of order class
Parameters:
id - is used to initialize id attribute
idclient - is used to initialize idclient attribute
idproduct - is used to initialize idproduct attribute
Methods
public int getId()
is the getter method for id attribute
Returns:
id attribute
public void setId(int id)
is the setter method for id attribute
Parameters:
id - is used to assign id attribute
public int getIdclient()
is the getter method for idclient attribute
Returns:
idclient attribute
public void setIdclient(int idClient)
is the setter method for idclient attribute
Parameters:
idClient - is used to assign idclient attribute
public int getIdproduct()
is the getter method for idproduct attribute
Returns:
idproduct attribute
public void setIdproduct(int idProduct)
is the setter method for idproduct attribute
Parameters:
idProduct - is used to assign idproduct attribute

4.5.3. OrderDetail

public class **OrderDetail**
is a public reflation class for the orderDetail table in the database
Fields
private int id
is a private attribute that identifies unique record from OrderDetail(primary key)
private int quantity
is a private attribute that represents quantity of the ordered item

private boolean delete
is a private attribute that takes true if the record client is "deleted"
Constructors
public OrderDetail()
is an empty constructor of client class
public OrderDetail(int id,
int quantity)
is a constructor of OrderDetail class
Parameters:
id - is used to initialize id attribute
quantity - is used to initialize quantity attribute
public OrderDetail(int quantity)
is a constructor of OrderDetailclass
Parameters:
quantity - is used to initialize quantity attribute
Methods
public int getId()
is the getter method for id attribute
Returns:
id attribute
public void setId(int id)
is the setter method for id attribute
Parameters:
id - is used to assign id attribute
public int getQuantity()
is the getter method for quantity attribute
Returns:
quantity attribute
public void setQuantity(int quantity)
is the setter method for quantity attribute
Parameters:
quantity - is used to assign quantity attribute
public boolean isDelete()
is the getter method for delete attribute
Returns:
delete attribute
public void setDelete(boolean delete)
is the setter method for delete attribute
Parameters:
delete - is used to assign delete attribute

4.5.4. Product

public class **Product**
is a public reflation class for the product table in the database
private int id
is a private attribute that identifies unique record from Product (primary key)
private String name
is a private attribute that is the name of the product
private int quantity
is a private attribute that represents stock of the product
private float price

is a private attribute that represents the price of the product
private boolean delete
is a private attribute that takes true if the record client is "deleted"
Constructors
public Product()
is an empty constructor of product class
public Product(String name)
is a constructor of Product class
Parameters:
name - is used to initialize name attribute
public Product(String name,
int quantity,
float price)
is a constructor of Product class
Parameters:
name - is used to initialize name attribute
quantity - is used to initialize quantity attribute
price - is used to initialize price attribute
public Product(int id,
String name,
int quantity,
float price)
is a constructor of Product class
Parameters:
id - is used to initialize id attribute
name - is used to initialize name attribute
quantity - is used to initialize quantity attribute
price - is used to initialize price attribute
Methods
public int getId()
is the getter method for id attribute
Returns:
id attribute
public void setId(int id)
is the setter method for id attribute
Parameters:
id - is used to assign id attribute
public String getName()
is the getter method for name attribute
Returns:
name attribute
public void setName(String name)
is the setter method for name attribute
Parameters:
name - is used to assign name attribute
public int getQuantity()
is the getter method for quantity attribute
Returns:
quantity attribute
public void setQuantity(int quantity)
is the setter method for quantity attribute
Parameters:

quantity - is used to assign quantity attribute
public float getPrice()
is the getter method for price attribute
Returns:
price attribute
public void setPrice(float price)
is the setter method for price attribute
Parameters:
price - is used to assign price attribute
public boolean isDelete()
is the getter method for delete attribute
Returns:
delete attribute
public void setDelete(boolean delete)
is the setter method for delete attribute
Parameters:
delete - is used to assign delete attribute

4.6.presentation

4.6.1. ParseInput

public class **ParseInput**
it will parse the input given by argument as String and pass param to other classes
Fields
private Presentation output
is a private presentation which will help the class to output
Constructors
public ParseInput(String fileName)
is a constructor of ParseInput class
Parameters:
fileName - is the name of the input file
Methods
private boolean validateName(String name)
is a private method that will validate a string as a name
Parameters:
name - that will be validate
Returns:
true if it is valid
private int validateUInt(String str)
is a private method that validates a string as unsigned integer and returns it
Parameters:
str - that will be validate
Returns:
extract positive integer if it is posible else -1
private void commandClient(String[] client)
is a private method that is used by the constructor and it will process a command for table client such as insert, delete and report
Parameters:
client - is an array of date representing clients attributes
private float validateFloat(String str)
is a private method that validates a string as positive float number and returns it

Parameters:
str - that will be validate
Returns:
extract positive float number if it is posible else -1
private void commandProduct(String[] product)
is a private method that is used by the constructor and it will process a command for table product such as insert, delete and report
Parameters:
product - is an array of date representing products attributes
private void commandOrder(String[] order)
is a private method that is used by the constructor and it will process a command for table order such as insert, delete and report
Parameters:
order - is an array of date representing orders attributes

4.6.2. Presentation

public class **Presentation**
is the main class for giving the output as pdf or in console
Constructors
public Presentation()
Methods
public void errorInput()
prints and manage an error for application input. Also exit the application
public void errorName()
prints and manage an error for names. Also exit the application
public void errorUInt()
prints and manage an error for unsigned integers. Also exit the application
public void errorFloat()
prints and manage an error for positive float numbers. Also exit the application
public void pdfBill(Client client,
Product product,
OrderDetail orderDetail)
prints as pdf a bill for the client given as parameter
Parameters:
client - represents the person that order
product - represents the product that is order
orderDetail - represents the detail of the order
public void pdfUnderStock()
prints as pdf if the product from the order is under stock
public void invalidCP()
prints and manage an error for the order without a client or product. Also exit the application
public void errorArrayOrder()
prints and manage an error for the order table if it is empty. Also exit the application
public void pdfReportOrder(ArrayList<Order> arrayOrder,
ArrayList<OrderDetail> arrayOrderDetail)
prints as pdf the table order combine with table orderDetail
Parameters:
arrayOrder - is an array of order that we want to print
arrayOrderDetail - is an array of orderDetail that we want to print
public void errorArrayProduct()
prints and manage an error for the product table if it is empty. Also exit the application

public void pdfReportProduct(ArrayList<Product> arrayProduct)
prints as pdf the table product
Parameters:
arrayProduct - is an array of products that we want to print
public void errorClient()
prints and manage an error for the client. Also exit the application
public void errorArrayClient()
prints and manage an error for the client table if it is empty. Also exit the application
pdfReportClient
public void pdfReportClient(ArrayList<Client> arrayClient)
prints as pdf the table client prints
Parameters:
arrayClient - is an array of client that we want to print

5. Results

5.1. System testing

System testing is a level of testing that validates the complete and fully integrated software product.
I tested on my own and analyses the response of main scenario and secondary scenarios.
Main scenario (success scenario):

- Introducing valid input file;
- Type valid commands;
- Verify the result (pdfs) by simulate on your own next step;

Example:

id	name	quantity	price
266	apple	35	1.0
268	orange	40	1.5
269	lemon	65	2.0

Failure scenario:

- Not a valid data;
- Not a valid file;
- Cannot connect to database
- Under stock pdf

I tested and verify multiple times the correctness of the simulation.

6. Conclusions

In conclusion, results of this paper work show fulfilling of the main objective and secondary goals. This application does order management. This assignment approached the following items: Object-oriented programming design; Java naming conventions; Use javadoc for documenting classes and generate the corresponding JavaDoc files; Use relational databases for storing the data for the application; File parser for the basic commands presented in Figure 1 (see page 4); Create a bill for each order as.pdf file; The product stock is decremented after the order is finalized; In case that there are not enough products, the order is not created and the PDF document representing the bill is not generated. In this case, instead of the bill, a PDF document is generated in which an under-stock message is written; Reports as .pdf files, generated for running the text file with commands; jar file - the application runs with the following command; java -jar PT2020_Group_FirstName_LastName_Assignment_3.jar commands.txt; Layered

Architecture (the application contains following packages (dataAccessLayer, businessLayer, model and presentation); Database Structure (four tables); Use reflection techniques.[4]

For future development of this application can be improved in following points: an graphical user interface can be done to see and manage orders in real time; more complex data can be add in clients, orderDetail, product; can create a hole system of ordering stuff.

7. References

[1] https://www.visual-paradigm.com
[4] http://coned.utcluj.ro/~salomie/PT_Lic/4_Lab/Assignment_3/Assignment_3_Indications.pdf
[3] https://www.quora.com/What-is-encapsulation-in-Java
[4] http://coned.utcluj.ro/~salomie/PT_Lic/4_Lab/Assignment_3/Assignment_3.pdf
[5] http://www.mkyong.com/jdbc/how-to-connect-to-mysql-with-jdbc-driver-java/
[6] http://theopentutorials.com/tutorials/java/jdbc/jdbc-mysql-create-database-example/
[7] https://dzone.com/articles/layers-standard-enterprise
[8] http://tutorials.jenkov.com/java-reflection/index.html
[9] https://www.baeldung.com/java-pdf-creation
[10] https://www.baeldung.com/javadoc
[11] https://dev.mysql.com/doc/workbench/en/wb-admin-export-import-management.html
[12] https://dev.mysql.com/doc/refman/5.7/en/using-mysqldump.html