Fundamental Programming Techniques

Assignment 3:

ORDER MANAGEMENT

Name: Filimon Adelin

Group: 30424

1.Objective

The objective of this assignment is to design and implement on order management application for processing customer orders for a warehouse.

The secondary objectives are:

1. Design the database
2. Implement model classes
3. Implement database connection
4. Implement data access object classes
5. Implement the business layer
6. Implement the presentation layer

In the first step it is implemented the database structure of the application. The database should have tables for clients, products and orders.

Next it is designed the object relational models such that each row from the database table could be represented like an object in the application.

Database connection is implemented using Singleton pattern and provides access to query the database.

Data access object pattern provides an abstraction model used for querying any type of table from the database using reflection techniques.

Business layer consists on the executor of the application and on some validators.

Presentation layer provides a Controller which interprets the input commands and also provides a report generator capable of generating reports of any table in pdf format.

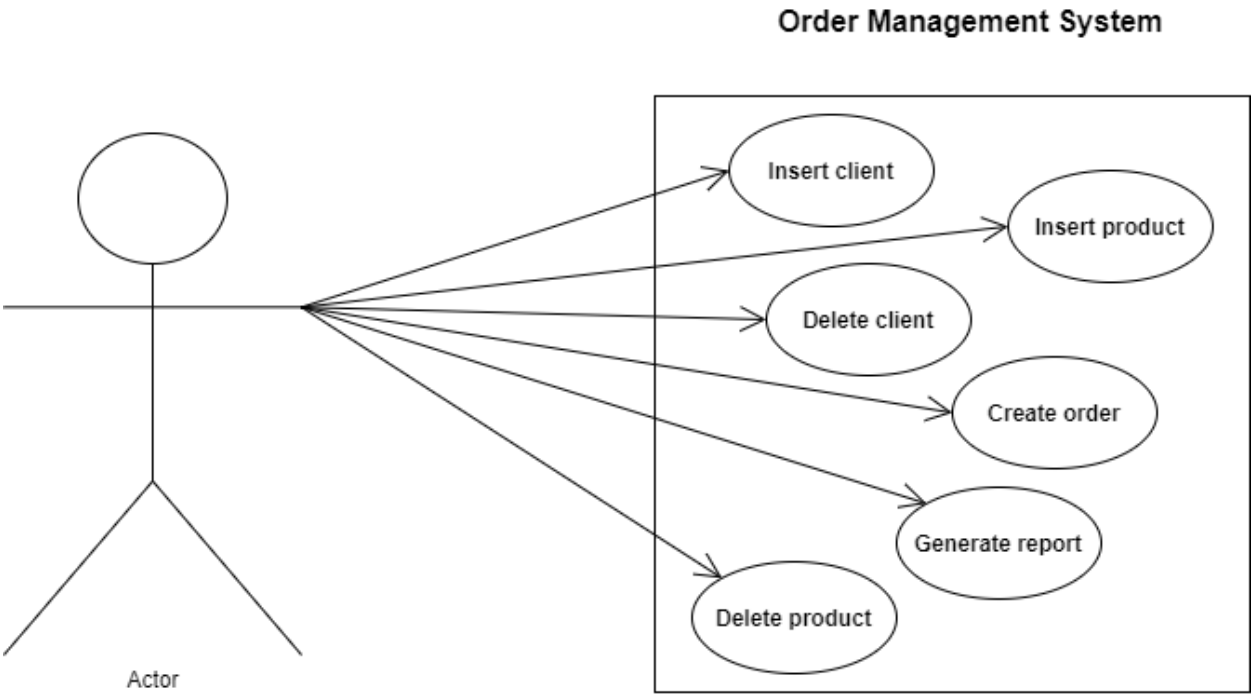
2.Problem analysis

The application should provide to users access to modify a database through commands stored in a text file given as program argument. Such commands imply to select rows from the database, store information in the database, update the database and also to delete rows from the database. The permitted commands are given below:

- Add client
- Delete client
- Add product
- Delete product

- Create order
- Generate reports

The use-case of this application is presented below:



Use case: Insert client

Primary actor: user

Preconditions: The input file given as parameter must be valid

Main success scenario:

1. The next command implies to add a new client
2. The user provided a name and an address for the client
3. The application validates the client
4. The application inserts the client in the database

Alternative sequence:

- Invalid input file: the application shows a message with corresponding issue
- The client is not valid: the client with specified name already exists. The application shows a message with corresponding issue

Use case: Delete product

Primary actor: user

Preconditions: The input file given as parameter must be valid

Main success scenario:

1. The next command implies to delete a client
2. The user provided a name of a client
3. The application deletes the client with the given name

Alternative sequence:

- Invalid input file: the application shows a message with corresponding issue

Use case: Insert product

Primary actor: user

Preconditions: The input file given as parameter must be valid

Main success scenario:

1. The next command implies to insert a product
2. The user provided a name, quantity and price for the product
3. The application validates the product
4. The application inserts the product in the database

Alternative sequence:

- Invalid input file: the application shows a message with corresponding issue
- The product is not valid: the product with specified name and price already exists. The application shows a message with corresponding issue

Use case: Delete product

Primary actor: user

Preconditions: The input file given as parameter must be valid

Main success scenario:

1. The next command implies to delete a product
2. The user provided a name of a product
3. The application deletes the product with the given name

Alternative sequence:

- Invalid input file: the application shows a message with corresponding issue

Use case: Create order

Primary actor: user

Preconditions: The input file given as parameter must be valid

Main success scenario:

1. The next command implies to create an order
2. The user provided a client name, a product name and a quantity
3. The application checks if the product have enough stock
4. The application inserts the order to the database
5. The application generates a bill in pdf format

Alternative sequence:

- The client name/product name are not valid: the application shows a message with corresponding issue

- The stock is not enough to process the command: an under-stock message is generated as pdf file
- Invalid input file: the application shows a message with corresponding issue

Use case: Generate report

Primary actor: user

Preconditions: The input file given as parameter must be valid

Main success scenario:

1. The next command implies to generate a report
2. The user provided a name of the data to be reported
3. The application generates a pdf file with corresponding data

Alternative sequence:

- Invalid input file: the application shows a message with corresponding issue

3.Design of the application

The application is structured in multiple layers: business layer, data access layer, persistence layer and presentation layer.

Business layer includes model classes, validators and executor of the application.

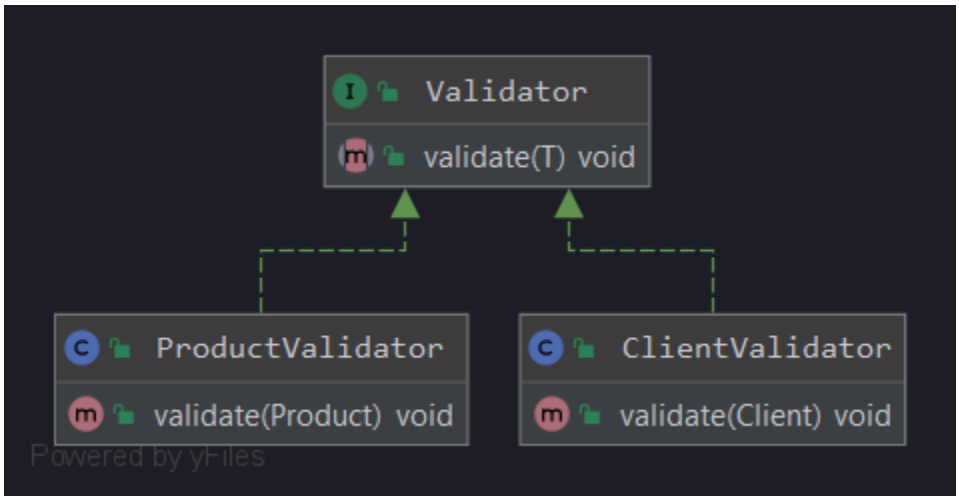
Data access layer provides the access to the database and is represented by a single class, DBConnection, which respects Singleton pattern.

Persistence layer is present through dao package which provides a superclass, AbstractDAO, which implements basic queries for any type of table, and specific classes such as ClientDAO with some extra methods for querying.

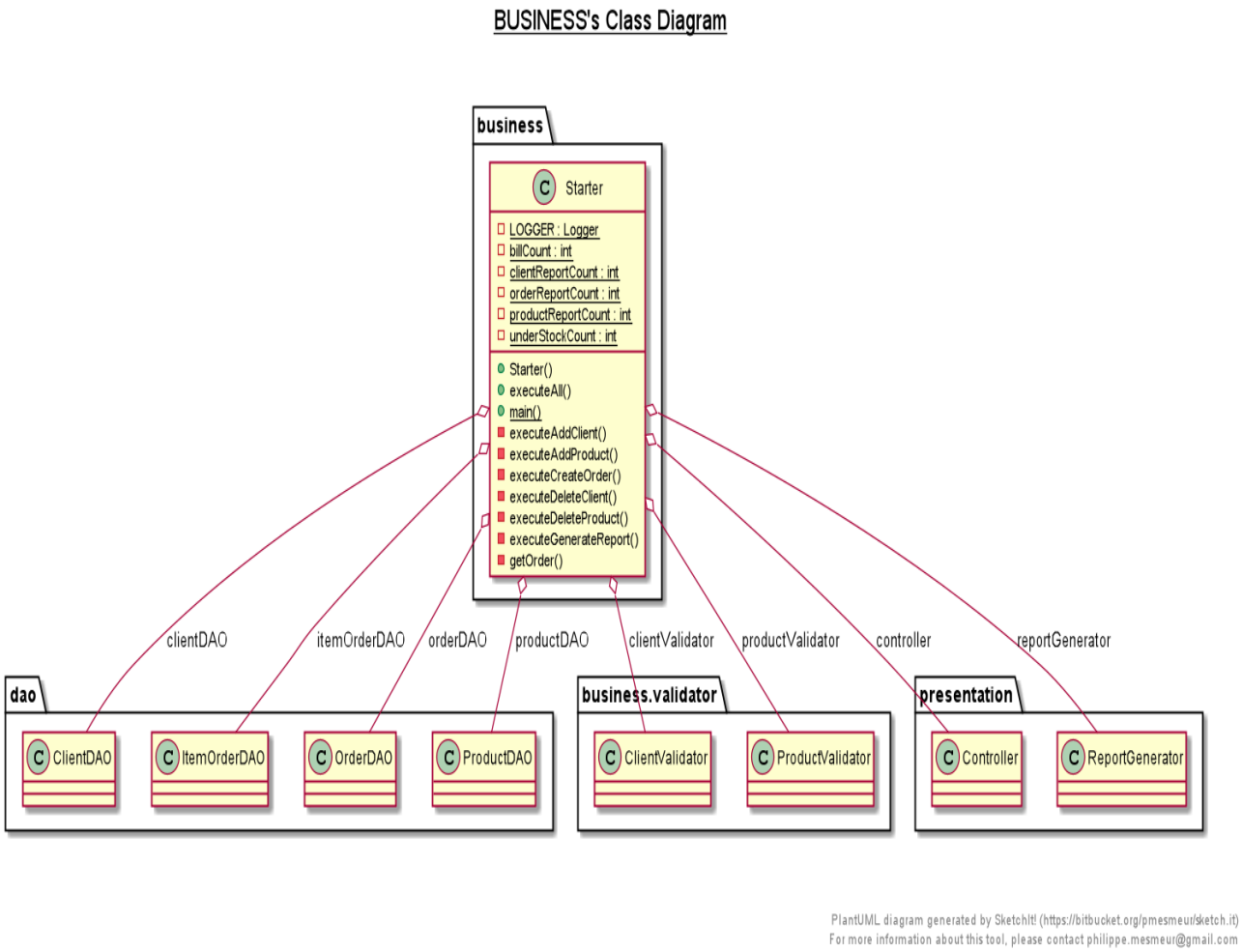
Presentation layer consists on input/output classes such as Controller, which have the job to decode the commands and ReportGenerator capable of generating pdf reports.

Next it is presented the UML diagrams of the project:

business.validator package:

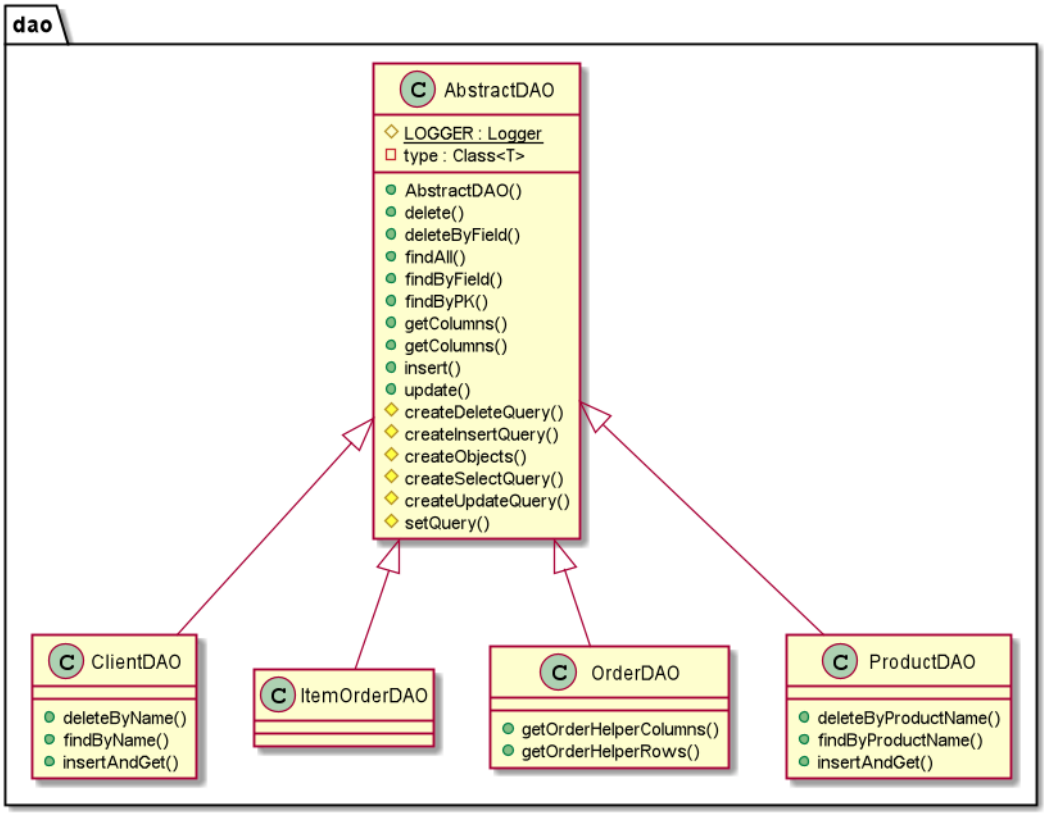


business package:



dao package:

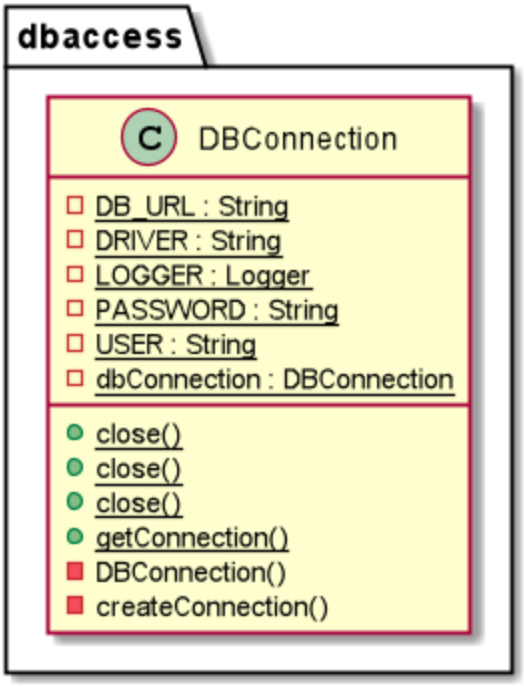
DAO's Class Diagram



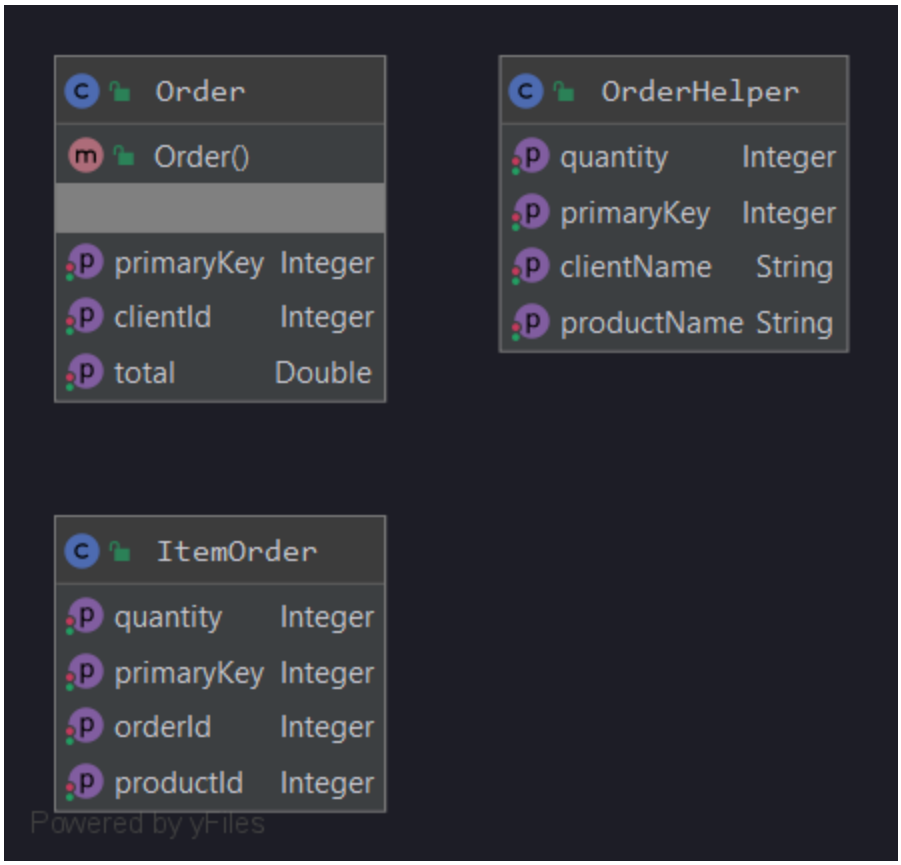
PlantUML diagram generated by SketchIt! (https://bitbucket.org/pmesmeur/sketch.it)
For more information about this tool, please contact philippe.mesmeur@gmail.com

dbaccess package:

DBACCESS's Class Diagram

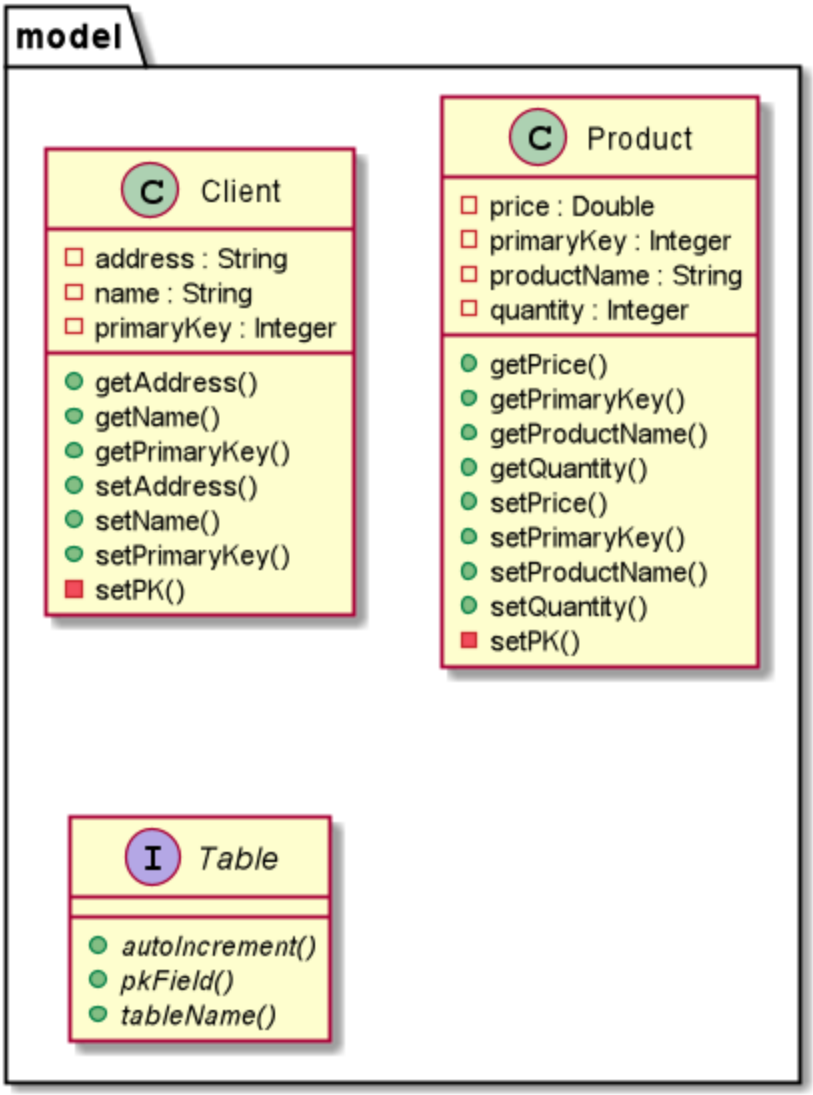


model.order package:



model package:

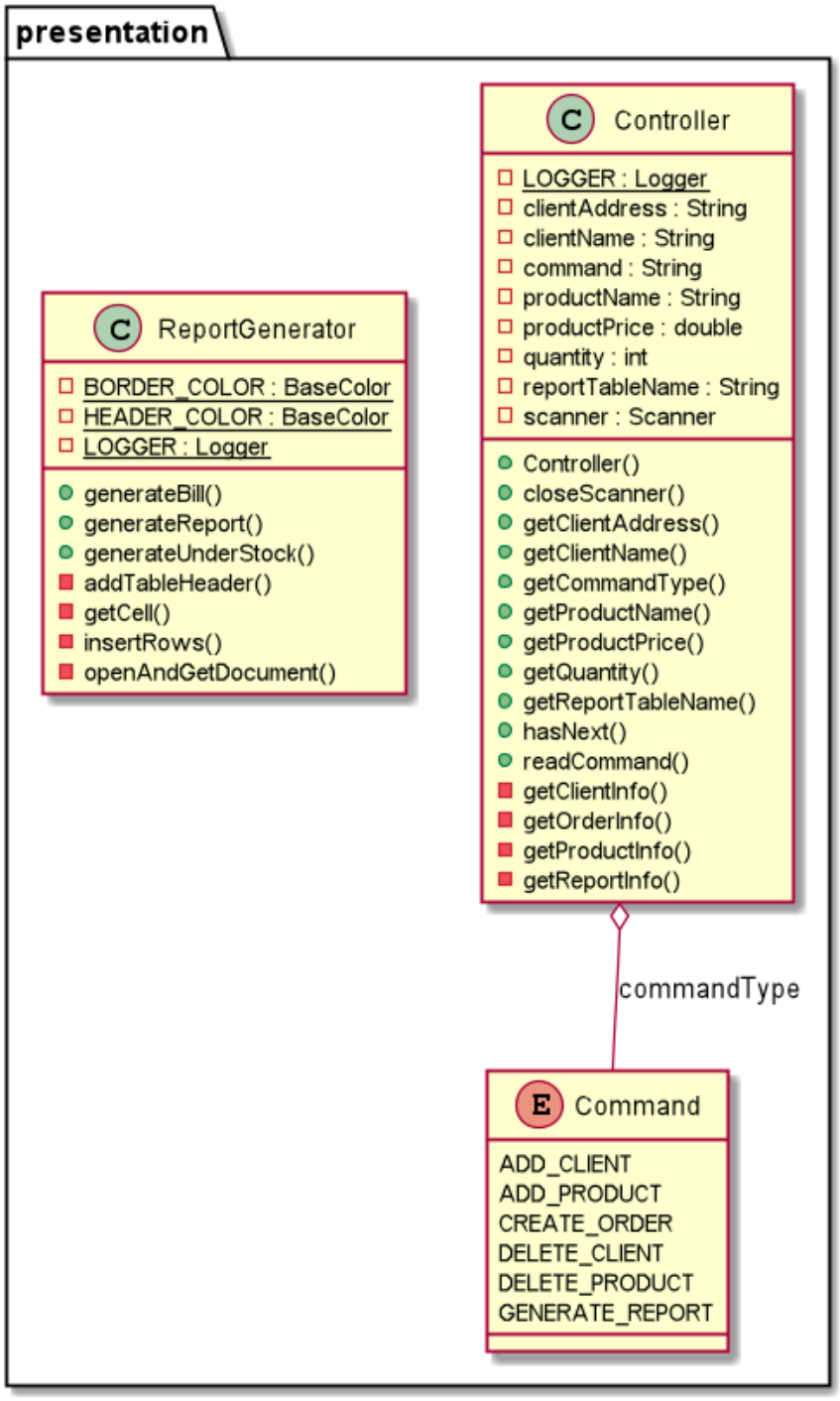
MODEL's Class Diagram



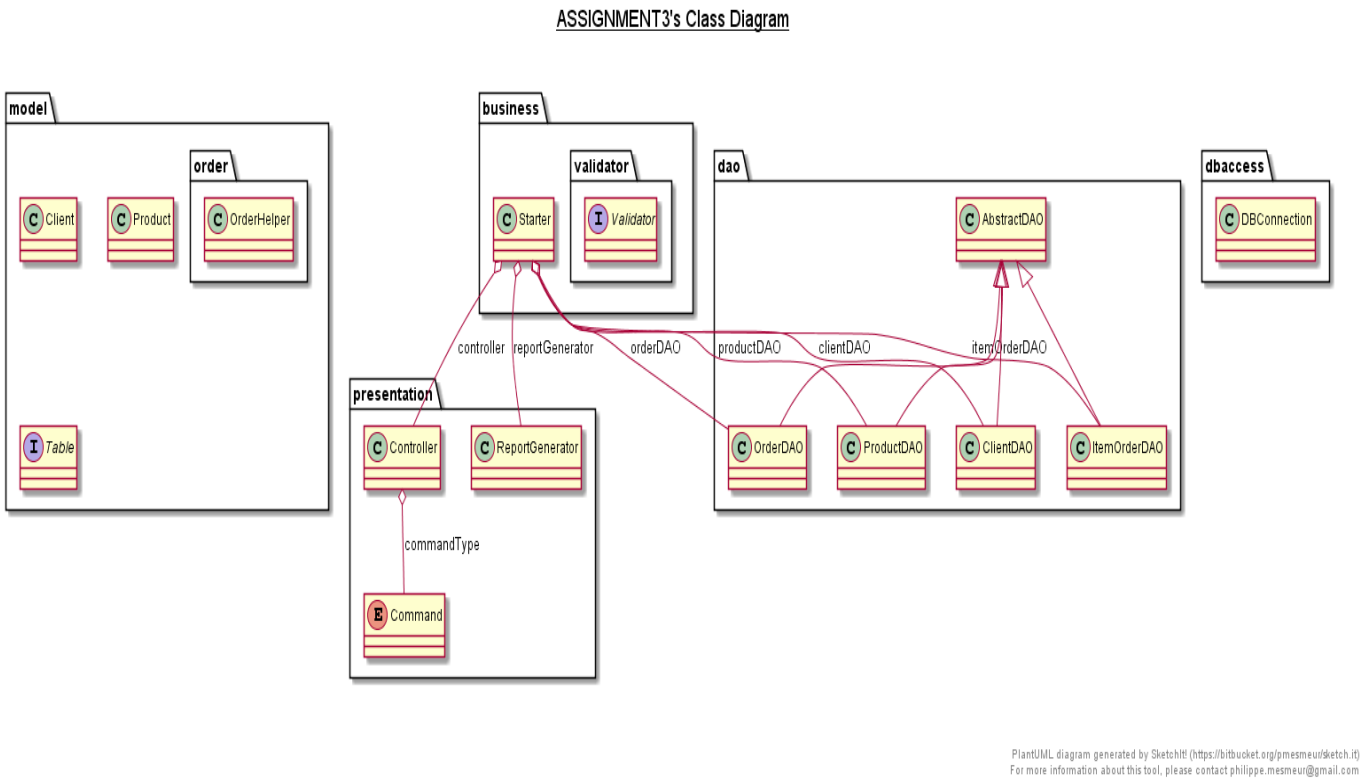
PlantUML diagram generated by SketchIt! (https://bitbucket.org/pmesmeur/sketch.it)
For more information about this tool, please contact philippe.mesmeur@gmail.com

presentation package:

PRESENTATION's Class Diagram



Entire project UML Diagram:



The data structure used is ArrayList and is used in general as a list of models, for example the method findAll of AbstractDAO class returns such a structure. In particular is used for getting the name of the column for reporting purposes.

The application uses metadata of database table stored in a custom annotation, Table which stores the name of the table, the name of the primary key and an auto-increment Boolean type. That annotation must be present on any class which inherits from AbstractDAO.

The Validator represents a defined interface with a single method, validate. That interface must be implemented by any class which validates objects.

The AbstractDAO class make use of reflection technique in such a way that we can use this class with any other ORM class which have all the fields of the table it represents and specifying in the Table annotation information about that table such as the table name, primary key field and auto increment value. Another important aspect a class must ensure is the primary key field and the private method setPK which is used only in AbstractDAO for creating the objects.

Database Connection is constructed using Singleton pattern such that there is ensured a single object of that class which provides methods for opening and closing connections to the database.

From the above UML diagrams its clear that the application is designed in a layered approach. The presentation layer make use only of business and persistence classes. The persistence layer, represented by dao package make use of the database connection for creating the queries. The database connection is the final layer which connects itself to the database.

Important classes are provided with a logger which would cause a warning in case there are thrown exceptions.

4.Implementation

The implementation of the application has been made in a bottom up approach, starting from the implementation of the database in MySQL to the presentation layer of the application.

The database, named shopdb, consists on 4 tables: clients, products, orders and itemorders. All this tables have as primary key an integer id.

clients table: Each row represents a table. The present columns are id, name and address.

products table: Each row represents a product. The present columns are id, productName, quantity and price.

orders table: Each row represents a total order of the client. The present columns are id, clientId and total.

itemOrders table: Each row represents a specific order of a client. The present columns are id, orderId, productId and ordered quantity.

DBConnection class:

The class used for accessing the database. This class have all the necessary fields to connect to the database: the driver, the url which points to the database, the username and the password. This class follows the Singleton pattern for assuring that there can't be created any other instances such that this could lead to synchronization problems with the database.

Methods:

1. The constructor: tries to register the driver. Can throw exceptions if the credentials are not valid or there are problems with the database.
2. createConnection(): private method used for creating a new connection to the database
3. getConnection(): public method, returns a new created connection
4. close (Connection/Statement/ResultSet): public method used for closing connections to the database. Can be used to Connection, Statement or ResultSet instances.

AbstractDAO class:

This class provides general queries applicable to any kind of table. It makes use of reflection and need metainformation about the table obtained from the Table annotation. The generic parameter represents the ORM class to which the queries could be performed.

Fields:

1. Class<T> type: the class object of the generic type.

Methods:

1. The constructor tries to get the class object of the generic type. This constructor is designed in such a way that only classes which inherits from AbstractDAO could manage to correctly call this constructor. In that way we cannot have an instance of AbstractDAO class.
2. Methods used for creating the query: createSelectQuery, createDeleteQuery, createUpdateQuery, createInsertQuery. They return an incomplete query which are processed by setQuery method.
3. findAll method returns all the rows from the table as an ArrayList of generic type objects.
4. createObjects method returns an ArrayList of objects based on the ResultSet given parameter.
5. findByField method is a select query with field and value parameters. It returns an ArrayList of generic type objects.

6. findByPK: an implementation of findByField method specifically for primary key field.
7. deleteByField method is used for deleting rows with the specified value of a given field.
8. Insert method tries to insert a given object as parameter to the table.
9. Update method tries to update the given object in the table.
10. Delete method tries to delete the given object from the table.
11. getColumns method is used for getting the column names of the table. It returns an ArrayList of Strings.

ClientDAO

This class inherits the general methods from the AbstractDAO and also provides some specific methods.

Methods:

1. findByName method will return the client from the table with the specified name.
2. deleteByName method will delete the client from the table with the specified name.
3. insertAndGet method creates a new Client object and inserts it to the table. Returns the created object.

ProductDAO

This class inherits the general methods from the AbstractDAO and also provides some specific methods.

Methods:

1. findByProductName method will return the product from the table with the specified name.
2. deleteByProductName method will delete the product from the table with the specified name.
3. insertAndGet method creates a new Product object and inserts it to the table. Returns the created object.

OrderDAO

This class inherits the general methods from the AbstractDAO and also provides some specific methods.

Methods:

1. getOrderHelperRows: since the order and orderitem tables doesn't provide useful information about bills or orders the OrderHelper class represents an Object which selects the useful data obtained from all tables such as the name of the client corresponded to the order, the name of the ordered product and the ordered quantity. This can be obtained from a select query which looks like this:
   "SELECT itemorders.id as id, `name`, productName, itemorders.quantity FROM clients JOIN orders ON clients.id = orders.clientId JOIN itemorders ON itemorders.orderId = orders.id JOIN products ON itemorders.productId = products.id"
2. getOrderHelperColumns returns the name of the columns used for the order.

ItemOrderDAO

This class inherits the general methods from the AbstractDAO and also provides some specific methods.

The ORM classes:

- Client: getters and setters for fields also a private method setPK used in reflection in AbstractDAO.
- Product: getters and setters for fields, also a private method setPK used in reflection in AbstractDAO.
- ItemOrder: getters and setters for fields, also a private method setPK used in reflection in AbstractDAO.
- Order: getters and setters for fields, also a private method setPK used in reflection in AbstractDAO.
- OrderHelper: a helper class used for printing the resulted order, contains getters and setters for reflection.
- Table annotation represents metadata about the table. Can be used on classes. The fields are tableName, pkField and autoincrement.

The business package provides validator classes and the Starter class which provides the functionality of the application.

ClientValidator: implements the Validator interface and have the job to check if a client already exists in the table. If it is then an exception is thrown.

ProductValidator: implements the Validator interface and have the job to check if product already exists in the table. If it is then an exception is thrown.

Validator: represents an interface with a generic parameter with a single method: validate(<GenericParameter>) used for validation.

Starter class provides methods for processing the given commands. For each command there exist a special method who handles the case.

Fields:

- LOGGER: the logger used for messages
- Controller : the controller who handles the input commands
- clientDAO: the client data access object
- productDAO: the product data access object
- orderDAO: the order data access object
- itemrOderDAO: the item order data access object
- clientValidator: the client validator used
- productValidator: the product validator used
- reportGenerator: the generator of pdf files
- counters for every possible generated pdf files: used for generating the name of the file

Methods:

The constructor initializes the fields.

executeAddClient: this method executes the add client command

executeDeleteClient: this method executes the delete client command

executeAddProduct: this method executes the add product command

executeDeleteProduct: this method executes the delete product command

executeGenerateReport: this method executes the generate report command

getOrder: method used for obtaining the right order, if the client already have an order returns that order otherwise returns a new one.

executeCreateOrder: this method executes the create order command

executeAll: this method executes all commands from the input file

Presentation package contains Command enumeration, ReportGenerator and Controller.

Command enumeration

Enumeration used for maintaining the track of the performed operations.

Controller handles the input file and returns the necessary data for performing the commands.

Fields:

- LOGGER used for possible errors
- commandType represents the type of the current type, is of type Command enumeration
- command: String representing one command from the file
- scanner: The scanner used for getting the information from the file
- clientName: the client name obtained from the command
- clientAddress: the client address obtained from the command
- productName: the product name obtained from the command
- reportTableName: the table name obtained from the command, for generating reports
- quantity: the quantity obtained from the command. Can be a product quantity or an ordered quantity
- productPrice: the price of the product obtained from the command

Methods:
The constructor initializes the scanner object.
Getters for command information fields.
readCommand: method used for reading one command and processing the information obtained
hasNext: used for checking the scanner for possible next commands
close: used for closing the scanner
getProductInfo: method used for obtaining product information from a command
getClientInfo: method used for obtaining client information from a command
getOrderInfo: method used for obtaining order information from a command
getReportInfo: method used for obtaining report information from a command

ReportGenerator: class used for generating reports in pdf format

Fields:

- LOGGER used for possible errors
- HEADER_COLOR: constant used for color of the table header
- BORDER_COLOR: constant used for color of the table borders

Methods:

addTableHeader: method used for adding a designed header to a PDFPTable. It needs an ArrayList of Strings containing the columns of a table

getCell: method used for getting a designed cell used for a table

generateBill: method used for generating bills in pdf format. It needs the order and itemorder for printing the message

generateUnderStock: method used for generating understock messages.

openAndGetDocument: method used for creating a document, opening it and returning that document.

insertRows: method used for inserting the rows of the table using an ArrayList of objects. It uses reflection for getting the data from the objects

generateReport: method used for generating the reports in pdf format.

5.Conclusions

The application could be improved by generalizing the primary key such that any type of primary key could be used. Also, the tables could have more specific data. The presentation could be improved by designing a user interface. From this assignment I learned to organize my project using layers which separates the concerns of the application. Also, I learned how to use reflection techniques to obtain classes which could work on any type of data.

6.Bibliography

https://www.javatpoint.com/java-jdbc

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

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