# FUNDAMENTAL PROGRAMMING

# TECHNIQUES

Assignment 3

Order Management

## Name: Leonties Dan Alexandru

Group: 30423

Contents:

1. Objective
2. Analysis
3. Design
4. Implementation
   1. AccessDB
   2. Commands
   3. UserInterf
   4. Client, Orders and Product
   5. Start
5. Results
6. Conclusions
7. Objective

Consider an application Order Management for processing customer orders for a warehouse.

Relational databases are used to store the products, the clients and the orders. Furthermore, the

application should be structured in packages using a layered architecture presented in the support

material and should use (minimally) the following classes:

• Model classes - the data models of the application

• Business Logic classes – implement the application logic

• Presentation classes – implement the user input/output

• Data access classes - implement the access to the database

The secondary objectives are:

* Object-oriented programming design
* Classes with maximum 300 lines
* Methods with maximum 30 lines
* Java naming conventions
* Basic documentation
* Use javadoc for documenting classes and generate the corresponding JavaDoc files.
* Use relational databases for storing the data for the application, minimum three tables: Client, Product and Order.
* A File parser for the basic commands presented in Figure 1 (see page 4).
* Queues should open/close dynamically. Initially all queues are closed. When clients are distributed to the queues, they become open as needed. When a queue becomes empty, it is closed, and the corresponding thread is paused.
* For the create order command:
  + Create a bill for each order as.pdf file
  + The product stock will be decremented after the order is finalized.
  + In case that there are not enough products, the order will not be created and the PDF document representing the bill will not be generated. In this case, instead of the bill, a PDF document will be generated in which an under-stock message will be written.
* Reports as .pdf files, generated for running the text file with commands from Figure 1 (see page 4)
* jar file - the application should permit to be run with the following command:

java -jar PT2020\_Group\_FirstName\_LastName\_Assignment\_3.jar commands.txt

* Quality of the Documentation
* Layered Architecture (the application will contain at least four packages: dataAccessLayer, businessLayer, model and presentation)
* Database Structure (more than three tables)
* Use reflection techniques to create a generic class that contains the methods for accessing the DB: create object, edit object, delete object and find object. The queries for accessing the DB for a specific object that corresponds to a table will be generated dynamically through reflection.

# Analysis

The application should allow processing commands from a text file given as argument, perform the

requested operations, save the data in the database, and generate reports in pdf format. Other classes

and packages can be added to implement the full functionality of the application.

A parser was implemented to read commands in the Presentation layer (instead of the standard graphical

user interface), and a pdf file generator to generate reports, according to the following examples:

*Table 1. Description of the basic commands to be considered for the application*

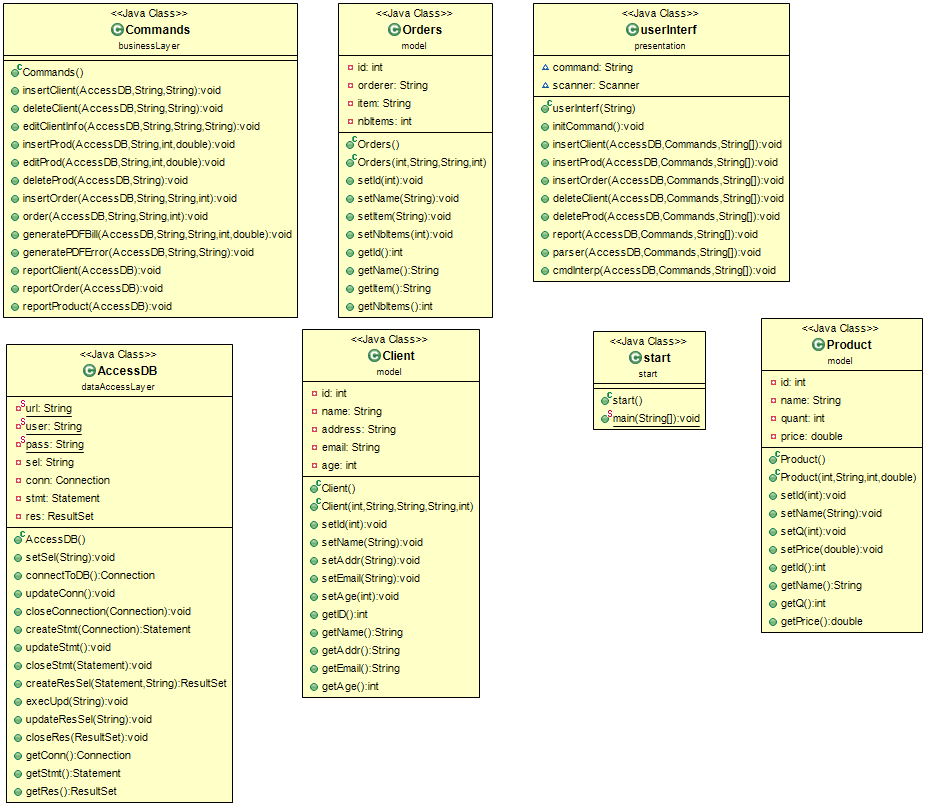
|  |  |  |
| --- | --- | --- |
| **Command name** | **Command Syntax** | **Description** |
| Add client to the database | Insert client: Ion Popescu, Bucuresti | Insert in the database a new client with name Ion Popescu and address Bucuresti |
| Delete Client from the database | Delete client: Ion Popescu | 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. |

Now the user will just need to write on each line of the provided “commands.txt” file, and execute the

program. The changes can and will be observed in the Database.

# Design

The UML diagram of the whole project can be seen here, and next will be described each part of it:



For the design of the project, 5 packages were created :

dataAccessLayer – contains the design of the connection to the Database

businessLayer – contains the commands used to run the application

presentation – contains the implementation of the user input/output

model – contains the 3 objects of the application : Client, Product, Order

start – package which contains the class that starts the application

The dataAccessLayer package contains only one class : AccessDB.

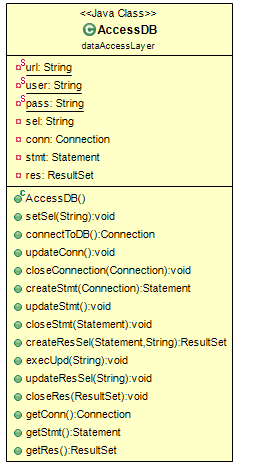
The AccessDB class contains the URL, USERNAME and PASSWORD to the

Database, and a sel String for the select command in SQL, and a Connection, Statement

and ResultSet for the execution of the provided commands. It also contains a

constructor and the needed methods, which I will describe in the next section.

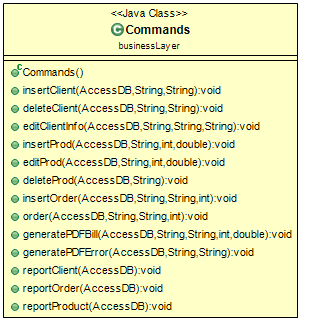
The UML diagram of this Package can be seen here:



The businessLayer package of the project contains only 1 class : Commands

The Commands class contains the needed methods and constructors which I will explain in the next section. This class just creates the needed commands for the project to run.

Here is the UML diagram of this part of the project:



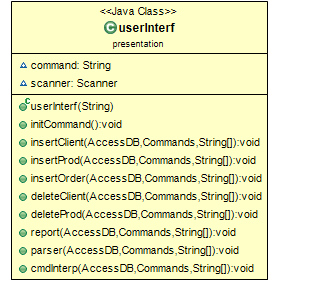
The presentation package of the project contains only 1 class : userInterf

This class contains one String named “command”, which is used to store the current command, and the

Scanner “scanner”, which is used to parse through the “commands.txt” file, and execute the command on

each line. It also contains a constructor and the needed methods, which I will describe in the next section.

Here is the UML diagram of this part of the project:



The model package contains 3 classes : Client, Orders and Product.

These are the 3 Objects that are seen in the Database, all 3 of them have the attributes already present in

the Database :

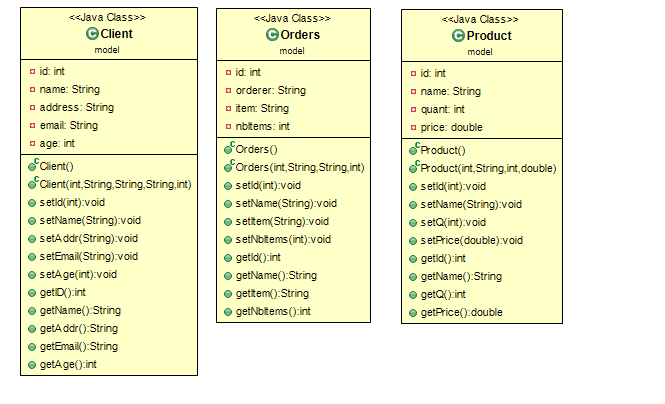
Client : id, name, address, email, age

Product : id, name, quantity, price

Orders : id, orderer, item, number of items

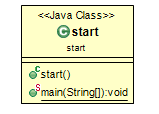
Each one of them also contains the constructors and methods necessary for the project.

This is the UML for this package :



Lastly, we have the package Start which contains only the start class, class used to start the whole project.

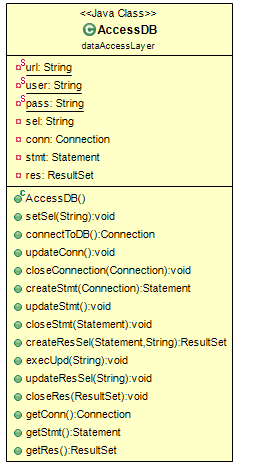
This is the UML for the last package :



Finishing with the Design part of the project, the algorithms that are used, aren’t complicated ones, most of them are used for the insertion, deletion of the data from the Database and for generating PDF files. Overall they are simple, and will be described in the following section of this documentation.

# Implementation

## AccessDB class



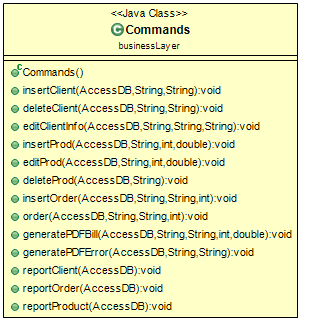
The AccessDB class is composed of 7 attributes, url, user, pass, sel, conn, stmt and res. In terms of constructors, the AccessDB can be created without any parameters needed. The constructor is quite simple, since it doesn’t require any parameters.

The url, user and pass will not be modified, since the Database will run on the default localhost with the port 3306, the username being “root”, and the password being “root”. There are getters and setters for the other 4 attributes : sel, conn, stmt and res. The setSel( String ) is used to set a value to the sel String which is used for the select SQL command. Some methods are used to complete the action of a setter, for example connectToDB () is used with updateConn () to create the connection to the Database. The connectToDB () creates the connection and updateConn () acts as a setter for the conn attribute. We also have a closeConnection () method which closes the connection to the Database. The same Design pattern is used for the “stmt” attribute ( with the methods createStmt ( Connection ) and updateStmt () ) and “res” attribute ( with the methods createResSel ( Statement, String ) and updateResSel ( String ) ), each of them having a closing method ( closeStmt ( Statement ) and closeRes ( ResultSet ) ). Each of the attributes has a getter designed for it. At last there is a method which executes a Query which modifies the contents of the Database ( insert and delete ) : execUpd ( String ), the String containing the needed SQL Query.

This concludes the AccessDB class.

## Commands class

This Class is the most important for the whole project.



This class doesn’t contain any attributes since there isn’t any attribute needed for it.

Starting off, it contains a simple constructor, designed to create the respective object.

The insertClient ( AccessDB, String, String ) method is used to insert a Client into the Database, the parameters being the Object used for connecting to the DB, and the name and address of the Client. The deleteClient ( AccessDB, String, String ) is used to delete a Client from the Database, again the parameters being the Object used for connecting to the DB, and the name and address of the Client. The editClientInfo (AccessDB, String, String, String ) is used to edit a specific Client’s information, the parameters being the Object used for connecting to the DB, the information which needs to be modified, the new information which will replace the old one, and the name of the Client.

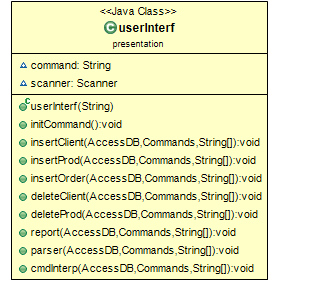
The insertProd ( AccessDB, String, int, double ) method is used to insert a Product into the Database, or if that product already exists, it modifies it, the parameters being the Object used for connecting to the DB, the name of the Product, the quantity and the price. The editProd ( AccessDB, String, int, double ) can is used to edit an existing product from the Database. The deleteProd (AccessDB, String ) searches for a specific product and deletes it from the DB. The parameters are the Object used for connecting to the DB and the name of the Product.

The methods used for creating and order are insertOrder ( AccessDB, String, String, int ) and order ( AccessDB, String, String, int ). The first method is acts as a simple insert, used just like the other 2 inserts for the Client and Product, with the parameters being the Object used for connecting to the DB, the name of the Client, name of the product, and the desired quantity. The other method, is used to verify if the Client exists and the desired quantity is greater than the quantity we have in stock. If the desired quantity is lesser or equal with the one in stock, the order will be created, and a PDF file with the Bill will be created (the method generatePDFBill (AccessDB, String, String, int, double ) will be called). In the other case, the order will not be created, and a PDF file will be created, showing and Under – Stock message (the method generatePDFError ( AccessDB, String, String ) will be called ).

Lastly, there are are 3 report methods, which generate the PDF files with the reports of the Client, Order and Product tables ( reportClient ( AccessDB ), reportOrder ( AccessDB ), reportProduct ( AccessDB ) , for each one of them the parameter is the Object used for connecting to the DB ).

This concludes the Commands class.

## UserInterf



The UserInterf class has 2 attributes, command which contains the file path for the “command.txt” file, and scanner which will be used to parse through the file.

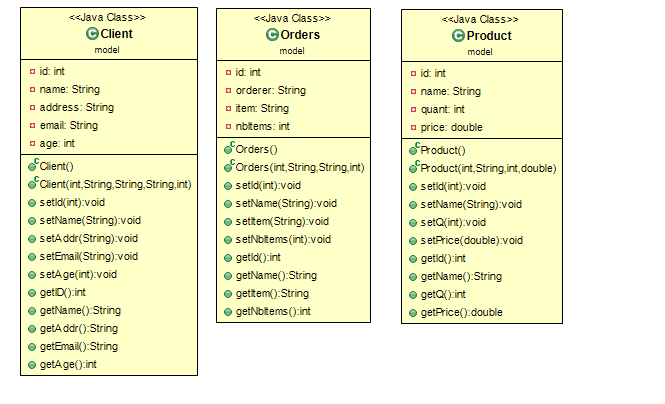
This class contains only contains one construct, which has a single parameter, that one being the file path, written as a String. The methods used, will be described below.

The initCommand () method, is used to initialize the command String, which will be used later as a String of the read commands from the file. The insertClient ( AccessDB, Commands, String[] ), insertProd (AccessDB, Commands, String[] ), insertOrder (AccessDB, Commands, String[] ), deleteClient (AccessDB, Commands, String[] ), deleteProd (AccessDB, Commands, String[] ) and report (AccessDB, Commands, String[] ), perform the actions defined already in the commands class, but this time, the first parameter is the Object used for connecting to the DB, the second parameter is the Object used for performing commands on the DB and the third parameter is a String containing the needed parameters for the function from the Commands class.

The parser ( AccessDB, Commands, String[] ) is used for parsing through the “commands.txt” file, taking a line which contains an instruction, splitting it in multiple parts, and passing it to a String[]. Then the cmdInterp ( AccessDB, Commands, String[] ) is called to interpret the contents and call the required command, based on the interpretation. Both use the same parameters, the first parameter is the Object used for connecting to the DB, the second parameter is the Object used for performing commands on the DB and the third parameter is a String containing the needed parameters for the function from the Commands class.

This concludes the UserInterf class.

## Client, Product and Orders



These three classes are pretty simple to explain, they basically contain all the attributes that are

already present in the Database. The Client with his id, name, address, email and age, the Product, with id,

name, quantity and price, and finally, Orders with id, orderer, item and nbItems. Each of these classes

contains a simple constructor, and more complicated constructors, which contains all the attributes of the

current class. All three classes contain getters and setters for all of their attributes.

## Start

In Start we simply run the project. A String called file with the content “commands.txt” has been created,

which acts as the file path for the file. A String[] called cmds has been created for each command line, and

the objects AccessDB, Commands and UserInterf have been created, with the names acc, com and interf

respectively. The interf.parser ( AccessDB, Commands, String[] ) has been called to start the project.

# Results

In terms of verifying the correctness of the algorithms, the commands from the provided Assignment\_3.pdf

support document have been used.

Insert client: Ion Popescu, Bucuresti Insert client: Luca George, Bucuresti Report client

Insert client: Sandu Vasile, Cluj-Napoca Report client

Delete client: Ion Popescu, Bucuresti Report client

Insert product: apple, 20, 1

Insert product: peach, 50, 2

Insert product: apple, 20, 1 Report product

Delete Product: peach

Insert product: orange, 40, 1.5

Insert product: lemon, 70, 2 Report product

Order: Luca George, apple, 5 Order: Luca George, lemon, 5 Order: Sandu Vasile, apple, 100 Report client

Report order Report product

These are the commands that have been used. The project executes the commands without a problem, and generates the corresponding PDF files.

# Conclusions

The created application is a very easy to use, fast and correct Order Management program.

In my opinion the most important updates that can be brought onto this projects are the addition of more Quality of Life features, such as a Graphical User Interface, which will make the input of a set commands much more easier and more user – friendly and in consequence making the application usable on Android or iOS, which some users may find really useful.

Also I have created the project with the simple idea that a connection to the Database will always be possible, so more memory will be saved. In another implementation, the data will be all saved inside the program, without needing to connect to the Database each time an information is required.

This project helped me learn how to a simulator and also how to create and manage threads. It also helped me understand concepts of structuring both the project as a whole and the smaller, more organized, mini - projects, the uses of encapsulation, and I also learned how to organize my code, so that when I come back to improve this project, I won’t have a hard time understanding what I did in each part.