Assignment 3

Group 30421

Buzan Vlad-Sebastian

1. Scope of the project

The main purpose of this assignment was to build a Java application having the following functionality: parsing commands from a text file(which is given as a command line argument) and executing the commands on a database. The commands are used to manipulate data representing clients, products and orders in a database. The commands required the implementation of all the basic database operations(CRUD).

Another part of the assignment is represented by the generation of PDF reports.

The secondary objectives (which have been derived from the main objective) are:

- Providing the user the possibility to input the path to a text file containing the commands to be executed
- Parsing the text file in order to obtain the commands and their arguments
- Creating representative classes for each object(Order, Client, Product, OrderItem)
- Creating a singleton class for connecting to the database
- Creating data-access-objects for each class interacting with the database
- Implementing the CRUD operations using reflection techniques
- Creating a PDFGenerator class used for outputting reports

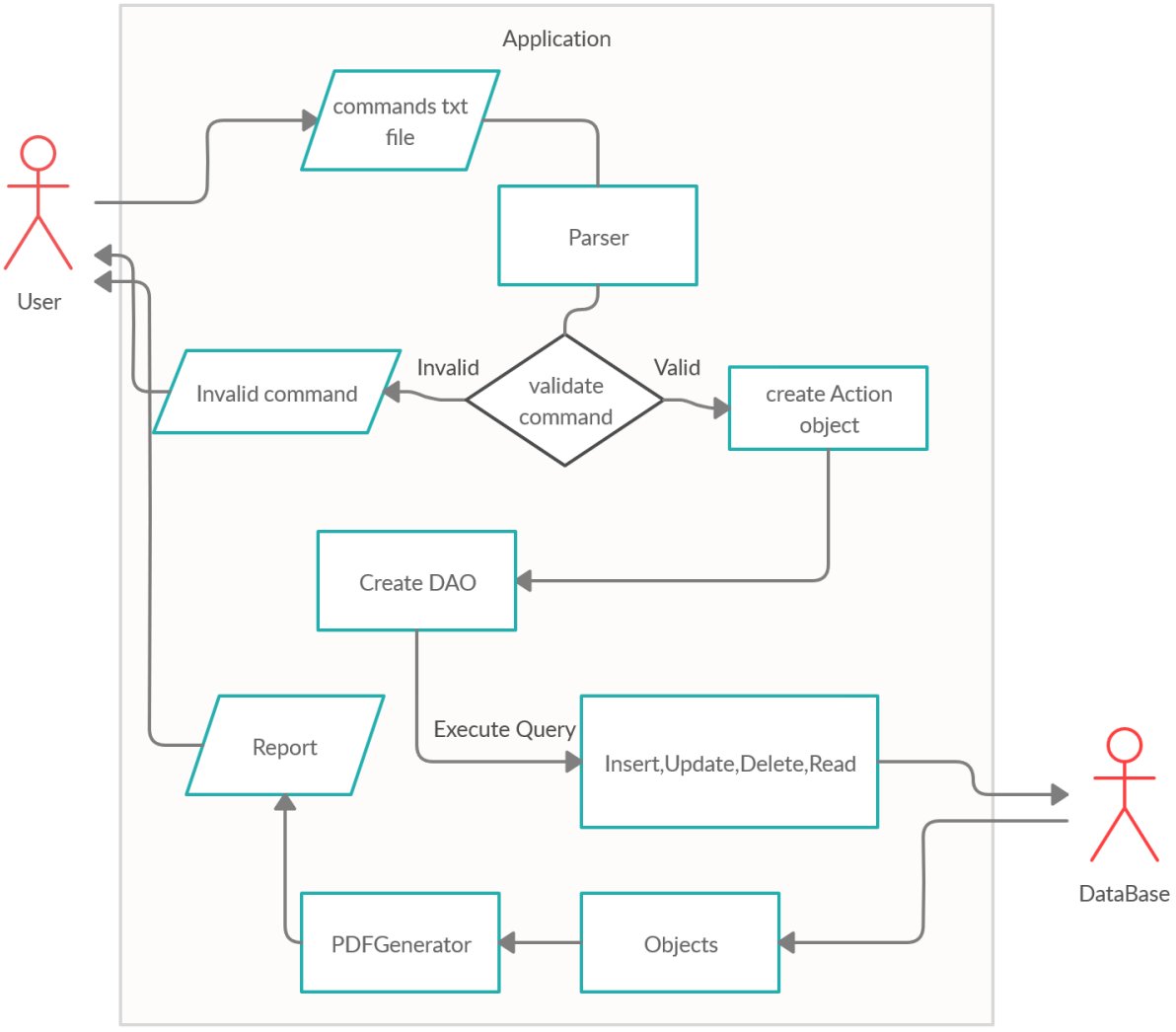
2. Problem analysis, modelling, scenarios, use-cases

Firstly, one can deduce some the classes that need to be designed in order to achieve the main goal. The input is acquired through a txt file which is given as a command line argument, therefore, I decided to create a Parser class and an Action class, the first being used to parse the txt file into instances of the second class. The Action class contains two fields, one being the Operation, the other being the arguments. The values that the operation field could take are represented using an enum. The next step was implementing the connection to the database. This was done using a singleton class called ConnectionFactory. The “singleton” approach was considered, since the application connects only to a single database, there was no need for multiple instances of the same ConnectionFactory class. Another group of classes that needed to be implemented are the objects representing tables in the database, meaning; Client, Orders, Product, OrderItem. For bridging the communication between the database and the acutal objects, there were

several data-access-objects created(ClientDAO, ProductDAO, OrdersDAO, OrderItemDAO), all extending the general DataAcessObject<T> class, where T is the actual class to access the database. Since one can't instantiate the data access object like DataAccessObject<Clinet> = new DataAccessObject<Client> since the T type must be known at compile time, I designed a subclass specific to each type of object. The ClientDAO, ProdutDAO, OrdersDAO and OrderItemDAO provide no use other than that of providing the DataAccessObject the type at compile-time. The DataAcessObject class uses reflective techniques in order to generate the queries that are to be executed on the warehouse database.

When it comes to use-cases, we have the following:

- The user inputs a txt file containing one command on each line



- The parser parses each line and creates an Action object
  - Invalid command => display error message
  - Action object is created successfully

- Data access object is created
- DAO generates a query
    - Query executed unsuccessfully => display error
- If report action was executed, a list of objects is fetched from the database
- PDFGenerator class creates a pdf report of the desired object

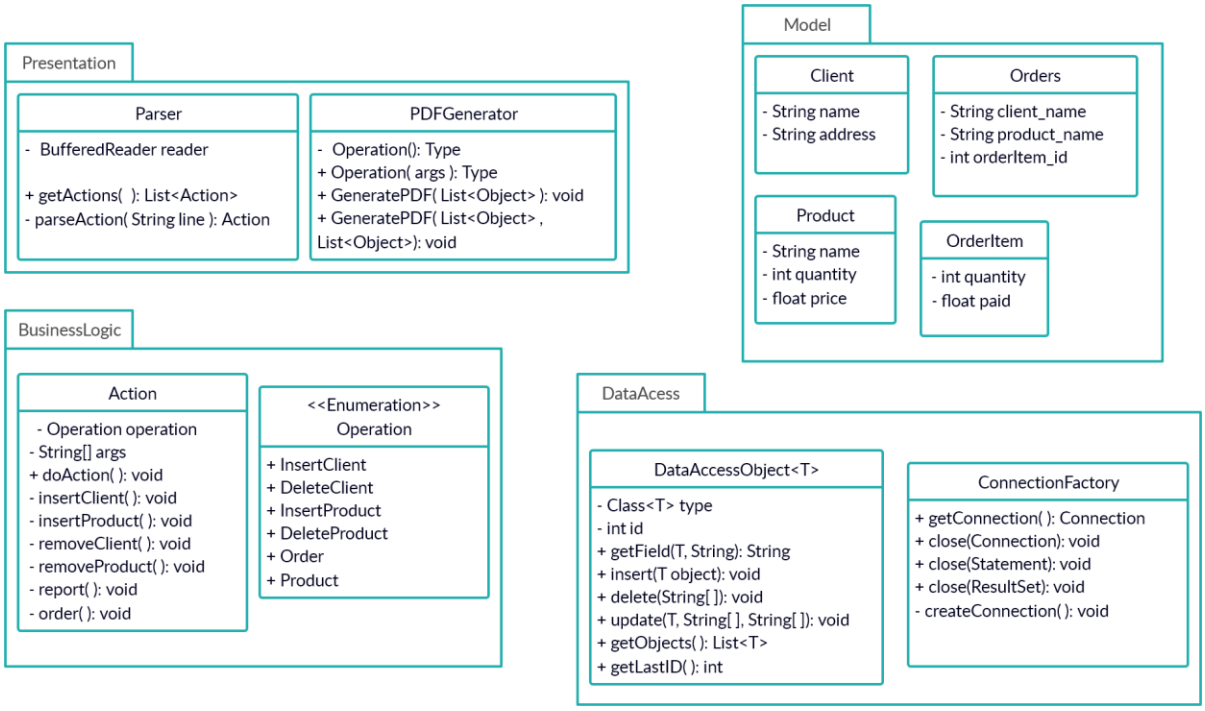
In this case, the primary actor is the user and the execution of the program is highly dependent on the input file provided by the user.

3. Design choices, data structures, class design, packages and algorithms

For this application, I used the "layered architecture" design pattern, as it proved to be a fitting approach for the problem at hand. This design pattern made it easy to design data structures that would make each layer of the application abstract, therefore allowing one to focus on design decisions rather than implementation.

The data structures used to process the information about each object, whether it is a client, order, order item or a product, are mirroring the tables used to store the information in the data base, rendering it possible to generalize the problem, and therefore, to use reflection in a much easier manner. For example, the table used to store information about clients has two fields: the client_name and the address field, therefore, the Java class used to represent a client will have the same fields.

As the design pattern chosen for this application is the layered architecture approach, naturally the app contains four packages / layers: Presentation, Model, BusinessLogic and DataAccess. The Presentation package handles the input / output of the application, meaning it is responsible with parsing the input file and with generating the pdf reports. The Model package contains classes that act as abstractions of real life objects, therefore, since our application handles the orders of a warehouse, there are four classes: Client, Order, Product and OrderItem. The DataAccess layer handles the interaction with the data base. Since the fields of each class correspond to that of a table in the database, reflection is used. The classes in the DataAccess package are: DataAccessObject<T> which handles the interaction with the database of an object of type T and ConnectionFactory which is a singleton class used to connect to the MYSQL database. The other classes in this package, being: ClientDAO, OrderDAO, ProductDAO, OrderItemDAO simply extend the DataAccessObject class, while specifying the T type, so that when instantiated, the DataAccessObject knows what T type is at compile time, this way, avoiding a run time error.



The algorithms that needed to be implemented where those that use reflection in order to access the fields of each object, in order to generate a specific sql query.

For each CRUD operation, the DataAccessObject has a private method that is used to generate the query. The methods use reflection to obtain each field and also the value of each field, therefore the method used to generate a query can be used on any object that respects the fields used in its correspondent in the database.

4. Implementation

The Parser class is used in order to obtain information from the txt file passed as an command line argument. The getActions() method returns a list of parsed commands. This is accomplished by calling the parseAction() method on each line of the txt file. The method returns an Action object which will be discussed later on.

The PDFGenerator class is used for obtaining the reports in the format of a pdf file. All the methods implemented in this class are static methods, since there is no need of an instance of this class(the class has no “state”). The generatePDF() method takes as argument a list of objects and a name, and using reflective techniques, generates a pdf, containing a table whose header is represented by the fields of the class, and whose cells are the values of the fields of each object in the list. The method is overloaded, the second implementation taking as arguments two lists, creating one pdf report with one table, where the header is a combination of the fields of the two classes, same as the values. In order to obtain a unique name for each generated pdf, the name given is concatenated with a String obtained by calling the getTimeString() method. The tables are generated using the writeTable() method, which calls both the addTableHeader() method as well as the addRows() method.

The Client, OrderItem, Orders and Product classes are implemented in such a way that they represent tables in the database. The only methods that these classes implemented are their constructors alongside setters and getters.

The DataAccessObject<T> class is used to interact with the warehouse database. The fields for this class are the type field of the generic parameter T, and the id field which represents the id of the last inserted object in the database. The methods used to create queries are the createSelectQuery(String field) method, which creates a MYSQL SELECT query whose condition is based on the field given as argument, the createInsertStatement() method, which creates an INSERT MYSQL statement, based on the the type of the parameter T, the createDeleteStatement(int nrFields) method, which creates a DELETE statement whose condition is based on a nrFields number of fields., the createUpdateStatement(String[] updateFields, String[] conditionFields) method, which creates a MYSQL UPDATE statement which updates the fields given as arguments (updateFields) and whose condition is based on the conditionFields. The methods used to implement the CRUD operations are: insert(T object) which creates an insert statement using the fields of the object given as parameter, the delete(String[] fields) method which creates a delete statement based on the fields given as arguments, the update(T object, String[] updateFields, String[] conditionFields) which updates the database table corresponding to the T type object, the updated fields being those given as argument, and the condition being based on the fields given as argument in the conditionFields array. The getObjects() method returns a list of the objects that correspond to the database table mirroring the T type object.

The ConnectionFactory class is used as a singleton class in order to obtain a connection to the warehouse database. The getConnection() method is used to obtain a Connection instance, while the close() method can be called having as argument either a ResultSet, Statement or a Connection.

The Action class is used to perform the specific task given as argument in the input txt file. The only public method for this class is the doAction() method, which, based on the Operation field of the Action class, decides which one of the following methods to call: insertClient(), deleteClient(), insertProduct(), deleteProduct(), report(), order(). Each of these methods use the DataAccessObject inherited class(ClientDAO, OrderDAO, ProductDAO, OrderItemDAO) corresponding to the selected method in order to access the database.

5. Results

After running the application multiple times, the results obtained were the expected ones. The generated pdf's represented the state of the database at the time the action was executed and the fields and their respective values were obtained correctly.

6. Conclusions

The approach used(the layered architecture design pattern) deemed useful for this type of application, since one can observe the different layers of the project and their delimitation. This way, all the layers could become abstract constructs, allowing me to focus on one part(layer) of the program at a time.

Bibliography:

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

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