JDBMS Report

Names :

Marwan Morsy

Ahmed Reda Amin

Muhammed Essam Khamis

Ahmed Ezzat Elmaghawry

Java Data-Base Connectivity

Abstract:

This Application deals with implementing the java database interface in order to allow java users to interact with the database that we had implemented in the previous assignment through a standard API.

Design:

- This Application is merely an interface to our previous DBMS so many of the tasks are revolved around creating a connection to send the SQL statement to our DBMS and manipulating the results that are returned by it.
- First the connection is established with our driver and the path selected and the relevant user data are extracted from the configuration files.
- Second , the user starts to interact with the JDBC by creating a statement and submitting his Query through it , this is then carried to the driver for processing and it either returns the number of updated records or a data structure interface known as the result set.
- Third , the result set is received by the user for further manipulation and relevant data extraction.
- Finally , the user can choose to close any of the three components of this system to stop any further transactions on it , with the exception of the connection, as closing it would automatically shut down the statements and the result sets.

Design Decisions:

1- Abiding mainly by the java documentation of the JDBC interface.

2- Abiding by SQL standard guidelines [ie. Removing all previous unconventional assumptions].

3- And Some more assumptions were made to appease the Online Tester which were noted test by test, like when to throw an exception Versus returning false or zero.

4- We had to return to our old code and make major changes , including adding 20+ classes and interfaces , we made an interface for most of the old classes , especially the recurring ones [the ones that get called to other classes], and we reorganized the packages and implemented two or three design patterns in the process.

5- We also had to redistribute the classes and interfaces on 11 different packages compared to our previous distribution which was one package.

6- We implemented our own data structures for many of the tasks , ex. Table implementation, DBNodes.

7- All of the processing is done by read and write to file , we don`t handle anything in-cache for synchronizing purposes.

Design Assumptions:

Most of the assumptions here are based on the output from the smoke test and the online tester :

1- We assumed that all the commands and the database/table names are case insensitive.

2- We assumed that an input can be missing fields and the program will have to auto fill the rest of the record with null , instead of previously throwing an exception , this is also in accordance with SQL guidelines.

3- We assumed that there should be a feedback to the user whether positive or negative , negative was previously implemented but not positive .

4- We added all the necessary conditionals using a state machine -like processing procedure.

5- We assumed that some cases will return zero or false for wrong results instead of an exception based on the online tester feedback.

Group division :

Revision and re-organization of old code:

-Ahmed Ezzat

-Ahmed Reda

Introduction of new data types:

-Marwan Morsy

-Muhammed Essam

The new JSON parser:

-Ahmed Ezzat

-Marwan Morsy

The JDBC implementation :

-Ahmed Reda

-Muhammed Essam

Testing:

-Ahmed Ezzat

-Muhammed Essam

Logging:

-Marwan Morsy

Configuration files:

-Ahmed Ezzat

-Marwan Morsy

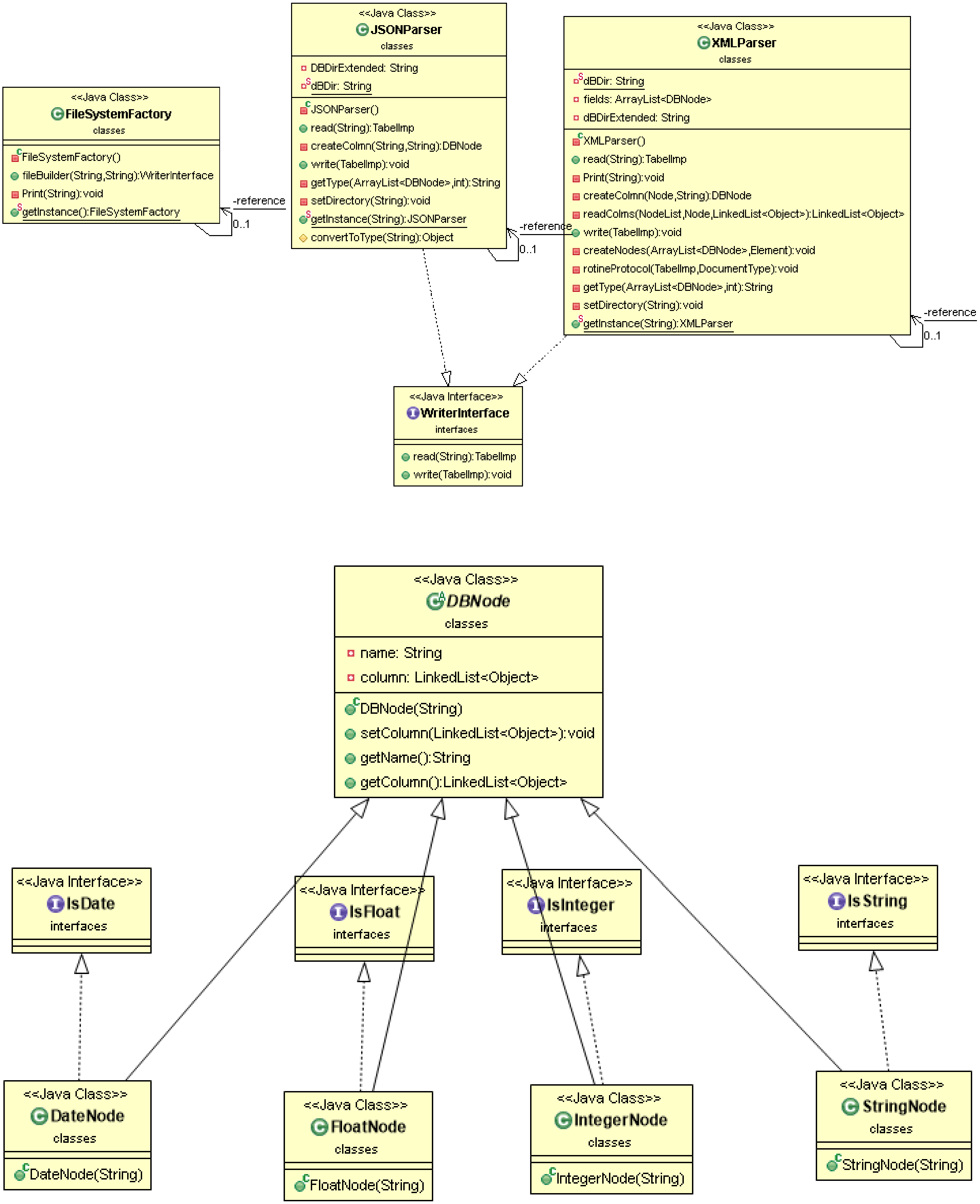
Report:

-Ahmed Reda

Checking on Piazza:

- Everyone

Uml diagram :



<<Java Class>>
FileSystemFactory
classes
FileSystemFactory()
fileBuilder(String,String):WriterInterface
Print(String):void
getInstance():FileSystemFactory
-reference
0..1
<<Java Class>>
JSONParser
classes
DBDirExtended: String
dBDir: String
JSONParser()
read(String):TabelImp
createColmn(String,String):DBNode
write(TabelImp):void
getType(ArrayList<DBNode>,int):String
setDirectory(String):void
getInstance(String):JSONParser
convertToType(String):Object
-reference
0..1
<<Java Class>>
XMLParser
classes
dBDir: String
fields: ArrayList<DBNode>
dBDirExtended: String
XMLParser()
read(String):TabelImp
Print(String):void
createColmn(Node,String):DBNode
readColms(NodeList,Node,LinkedList<Object>):LinkedList<Object>
write(TabelImp):void
createNodes(ArrayList<DBNode>,Element):void
rotineProtocol(TabelImp,DocumentType):void
getType(ArrayList<DBNode>,int):String
setDirectory(String):void
getInstance(String):XMLParser
-reference
0..1
<<Java Interface>>
WriterInterface
interfaces
read(String):TabelImp
write(TabelImp):void
<<Java Class>>
DBNode
classes
name: String
column: LinkedList<Object>
DBNode(String)
setColumn(LinkedList<Object>):void
getName():String
getColumn():LinkedList<Object>
<<Java Interface>>
IsDate
interfaces
<<Java Interface>>
IsFloat
interfaces
<<Java Interface>>
IsInteger
interfaces
<<Java Interface>>
IsString
interfaces
<<Java Class>>
DateNode
classes
DateNode(String)
<<Java Class>>
FloatNode
classes
FloatNode(String)
<<Java Class>>
IntegerNode
classes
IntegerNode(String)
<<Java Class>>
StringNode
classes
StringNode(String)