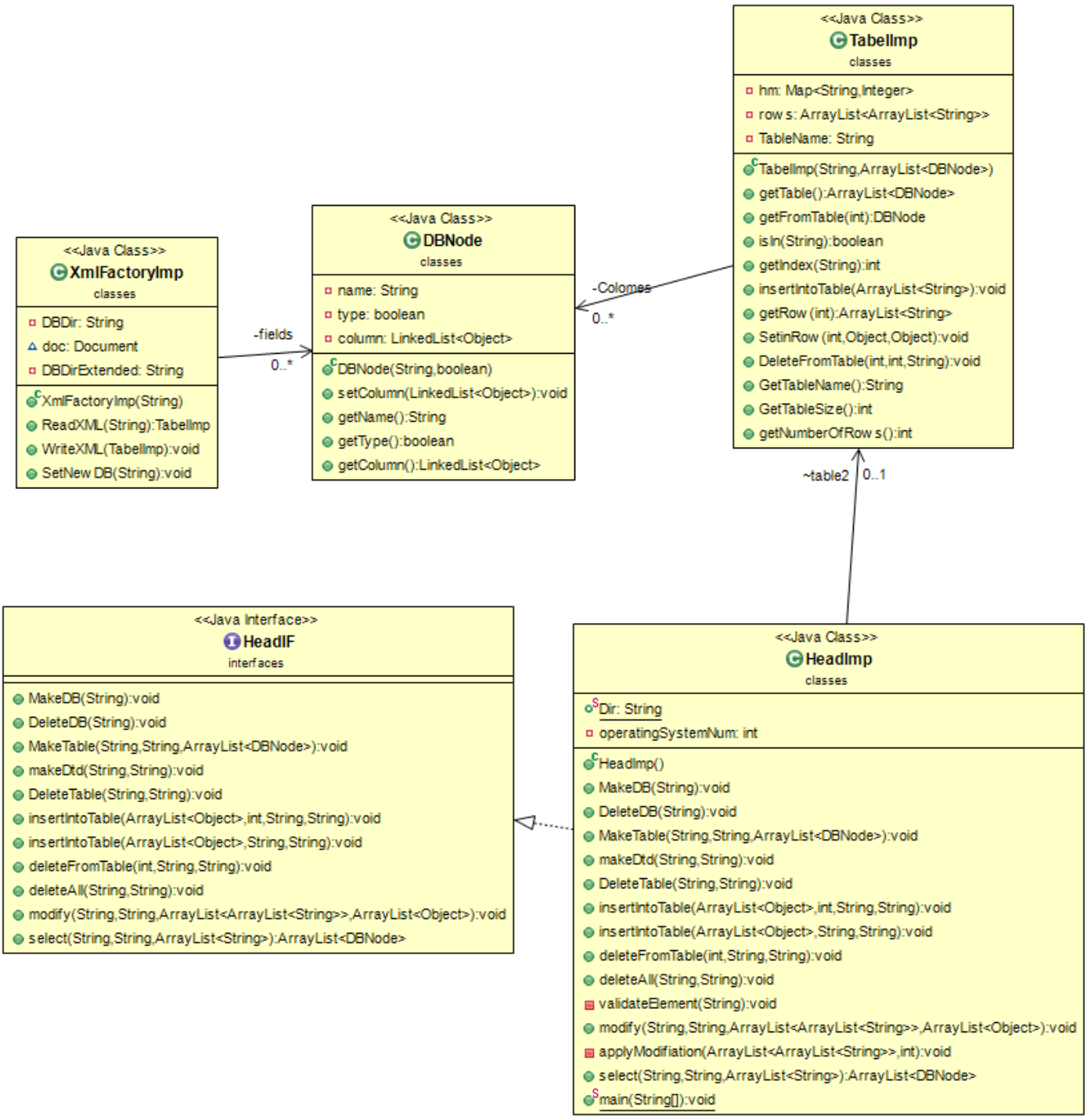
Simple Data Base Management System.

Abstract:

This application aims to help the people to easily manage the data base throw simple instructions and commands to make or delete a data base, select a table from it or even edit it without any constrains or Complexities.

UML Diagram:


<<Java Class>>
TabelImp
classes
hm: Map<String,Integer>
rows: ArrayList<ArrayList<String>>
TableName: String
TabelImp(String,ArrayList<DBNode>)
getTable():ArrayList<DBNode>
getFromTable(int):DBNode
isIn(String):boolean
getIndex(String):int
insertIntoTable(ArrayList<String>):void
getRow(int):ArrayList<String>
SetinRow(int,Object,Object):void
DeleteFromTable(int,int,String):void
GetTableName():String
GetTableSize():int
getNumberOfRows():int
<<Java Class>>
DBNode
classes
name: String
type: boolean
column: LinkedList<Object>
DBNode(String,boolean)
setColumn(LinkedList<Object>):void
getName():String
getType():boolean
getColumn():LinkedList<Object>
-Colomes
0..*
<<Java Class>>
XmlFactoryImp
classes
DBDir: String
doc: Document
DBDirExtended: String
XmlFactoryImp(String)
ReadXML(String):TabelImp
WriteXML(TabelImp):void
SetNewDB(String):void
-fields
0..*
~table2
0..1
<<Java Interface>>
HeadIF
interfaces
MakeDB(String):void
DeleteDB(String):void
MakeTable(String,String,ArrayList<DBNode>):void
makeDtd(String,String):void
DeleteTable(String,String):void
insertIntoTable(ArrayList<Object>,int,String,String):void
insertIntoTable(ArrayList<Object>,String,String):void
deleteFromTable(int,String,String):void
deleteAll(String,String):void
modify(String,String,ArrayList<ArrayList<String>>,ArrayList<Object>):void
select(String,String,ArrayList<String>):ArrayList<DBNode>
<<Java Class>>
HeadImp
classes
Dir: String
operatingSystemNum: int
HeadImp()
MakeDB(String):void
DeleteDB(String):void
MakeTable(String,String,ArrayList<DBNode>):void
makeDtd(String,String):void
DeleteTable(String,String):void
insertIntoTable(ArrayList<Object>,int,String,String):void
insertIntoTable(ArrayList<Object>,String,String):void
deleteFromTable(int,String,String):void
deleteAll(String,String):void
validateElement(String):void
modify(String,String,ArrayList<ArrayList<String>>,ArrayList<Object>):void
applyModifiation(ArrayList<ArrayList<String>>,int):void
select(String,String,ArrayList<String>):ArrayList<DBNode>
main(String[]):void

Design assumptions:

In this application we have made some assumption about how the user is going to interact and use this program.

Those assumptions are:

- We assume that the names of data bases and tables are case sensitive.
- The key words that reserved for the expression that used to make the command such as SELECT are case insensitive.
- If any error happened by the user a message will pop up that tells him there is an error in his input expression.
- In the XML saved file, we saved the table column by column.
- In the condition after WHERE key word, the user can compare between two integers or two strings.
- Every column is saved in a class called DBNode which responsible for saving the type of the content of the column and the content of each cell in the column itself.
- The whole table is saved in class called TableImp which has array list of DBNodes and the name of the table
- The TableImp class also contains many helpful methods which allow us to add any row into the table easily.
- We used external class called XML factory in order to save the new TableImp in its XML file.

Screen shots:

```java
package classes;
import java.util.LinkedList;


public class DBNode {
    private String name ;
    private boolean type;
    private LinkedList<Object> column;

    public DBNode(final String name,final boolean type) {
        // TODO Auto-generated constructor stub
        this.name=name;
        this.type=type;
        column = new LinkedList<Object>();
    }

  public void setColumn(final LinkedList<Object> column){
        this.column = column;
    }

  public String getName (){
    return name;
  }

  public boolean getType (){
        return type;
      }

  public LinkedList<Object> getColumn (){
        return column;
      }
}
```

```
public HeadImp(){
    String operatingSystem = System.getProperty("os.name");
    operatingSystem = operatingSystem.substring(0, 7);
    if(operatingSystem.equals("Windows")) {
        operatingSystemNum = 1;
    Dir = Paths.get("").toAbsolutePath().toString().replace("\\", "\\\\") + "\\\\";
    } else {
        operatingSystemNum = 2;
    Dir = System.getProperty("user.dir")+"/";
    }

}



public void MakeDB(final String Name) {
    File x = new File(Dir + Name);
    x.mkdir();
}

public void DeleteDB(final String Name) {
    File file = new File(Dir + Name);
    File[] files = file.listFiles();
    if (files != null) {
        for (File f : files) {
            if (f.isDirectory())
                file.delete();
            else
                f.delete();
        }
```

```
public void MakeTable(final String Db, final String tableName, final ArrayList<DBNode> s) throws Exception {
    String DirExtended = null;
    switch(operatingSystemNum){
    case 1:
        DirExtended = Dir + Db + "\\";
        break;
    case 2:
        DirExtended = Dir + Db + "/";
    }
    org.w3c.dom.Document doc;
    DocumentBuilderFactory docFact = DocumentBuilderFactory.newInstance();
    DocumentBuilder build = docFact.newDocumentBuilder();
    doc = build.newDocument();
    DOMImplementation domImpl = doc.getImplementation();
    DocumentType doctype = domImpl.createDocumentType(tableName, "SYSTEM", tableName+".dtd");
    doc.appendChild(doctype);
    Element root = doc.createElement(tableName);
    doc.appendChild(root);
    for (int i = 0; i < s.size(); i++) {
        Element DBnode = doc.createElement("DBNode");
        root.appendChild(DBnode);
        Element name = doc.createElement("Name");
        name.appendChild(doc.createTextNode(s.get(i).getName()));
        Element type = doc.createElement("Type");
        String t = s.get(i).getType() ? "integer" : "string";
        type.appendChild(doc.createTextNode(t));
        Element column = doc.createElement("Column");
        for (int j = 0; j < s.get(i).getColumn().size(); j++) {
            Element x = doc.createElement("Row" + "" + j);
            x.appendChild(doc.createTextNode(String.valueOf(s.get(i).getColumn().get(j))));
            column.appendChild(x);
        }
```

```
public void insertIntoTable(final ArrayList<Object> x,final int index, final String dBase, final String tableNa
    XmlFactoryImp aa = new XmlFactoryImp(Dir + dBase);
    ArrayList<DBNode> fields = (aa.ReadXML(tableName)).getTable();
    boolean type = false;
    if (x.size() != fields.size()|| fields.get(0).getColumn().size() < index) {
        throw new RuntimeException("Invalid Input");
    }
    int i = 0;
    for (i = 0; i < x.size(); i++) {
        if (x.get(i) != null)
        {if (x.get(i) instanceof String) {
            type = false;
        } else if (x.get(i) instanceof Integer) {
            type = true;
        } else {
            throw new RuntimeException("Input of Invalid type");
        }
        if (type == fields.get(i).getType()) {
            fields.get(i).getColumn().add(index,x.get(i));
        } else {
            throw new RuntimeException("Invalid Input");
        }}
        else {
            fields.get(i).getColumn().add(index,x.get(i));
        }
    }
    TabelImp xxx = new TabelImp(tableName, fields);
    aa.WriteXML(xxx);
}
```

```
public ArrayList<DBNode> select(final String tableName, final String dBase, final ArrayList<String> wanted) throw
    XmlFactoryImp aa = new XmlFactoryImp(Dir + dBase);
    ArrayList<DBNode> fields = (aa.ReadXML(tableName)).getTable();
    TabelImp table = new TabelImp(tableName, fields);
    ArrayList<DBNode> tmp = new ArrayList<>();
    for (String element : wanted) {
        if (!table.isIn(element)) {
            return null;
        }
        tmp.add(table.getFromTable(table.getIndex(element)));
    }
    return tmp;
}


public ArrayList<ArrayList<String>> SelectMachine() throws Exception {
    // TODO Auto-generated method stub
    Pattern SelecCol = Pattern.compile("(?i)\\s*select\\s+(.+)(\\s*,\\s*(\\w+))*\\s+from\\s+(\\w+).(\\w+)(\
    Matcher matcherCol = SelecCol.matcher(Input);
    if (matcherCol.find()){
        String Columns = matcherCol.group(1);
        String[] Colms = DivideComma(Columns);
        String DBName = matcherCol.group(4);
        String TableName = matcherCol.group(5);
        if (matcherCol.group(6) != null) {
            String ColName = matcherCol.group(7);
            String Condition = matcherCol.group(8);
            String RowName = (matcherCol.group(11) != null)?matcherCol.group(11):matcherCol.group(9);
            ArrayList<Integer> Indexies = getIndexies(ColName, Condition, RowName, DBName, TableName);
            return DoSelectWhere(DBName, TableName, Colms, Condition, Indexies);
        } else {
            return DoSelectSpec(DBName, TableName, Colms);
        }
    } else
        throw new RuntimeException("Invalid Input");
}


private ArrayList<ArrayList<String>> DoSelectWhere(String dBName, String tableName, String[] colms, String condit
    // TODO Auto-generated method stub
    TabelImp Table = new XmlFactoryImp(dBName).ReadXML(tableName);
    if (colms.length == 1 && colms[0].equals("*")) {
        if (condition.equals("=") || condition.equals(">") || condition.equals("<")) {
            ArrayList<ArrayList<String>> x = new ArrayList<>();
        for (int i = 0; i < indexies.size(); i++) {
            ArrayList<String> y = new ArrayList<>();
            for (int j = 0; j < Table.GetTableSize(); j++) {
                y.add((Table.getFromTable(j).getColumn().get(indexies.get(i)) != null)?Table.getFromTable(j).get(
            }
            x.add(y);
        }
        return x;
        } else {
            throw new RuntimeException("Invalid Operator");
        }
        /*else if (){
```

```java
public static void main(String[] args) {
    // TODO Auto-generated method stub
    Scanner s = new Scanner(System.in).useDelimiter(";");
    sqlFactoryImp user = new sqlFactoryImp();
    ArrayList<ArrayList<String>> printFormat;
    while (true) {
        try {
            String input = s.next();
            printFormat = user.execute(input.replaceAll("[\r\n]+", " ") + ";");
            try{
            String[] q = user.Colms();
            for (int i = 0; i < printFormat.size(); i++) {
                for (int j = 0; j < printFormat.get(i).size(); j++) {
                    System.out.print(printFormat.get(i).get(j));
                    for (int k = 0; k < 15 - printFormat.get(i).get(j).length(); k++) {
                        System.out.print(" ");
                    }
                }
                System.out.println("\n------------------------------------------------------------------");
            }
            } catch (Exception e) {

            }
            s.nextLine();
        } catch (Exception e) {
            System.out.println("Error :");
            System.out.println(e.getMessage());
        }
    }

}
```