Text2code Code Listing

v.yesipov.17

May 2019

DirExplorer class

DirExplorer.java - The Java class file to find all of the java source code files from pointed directories and sub-directories.

```
package com.textfromcode;


import java.io.File;


public class DirExplorer {
    public interface Filehandler {
        void handle(int level, String path, File file);
    }

    public interface FileFilter {
        boolean interested(int level, String path, File file);
    }

    private Filehandler filehandler;
    private FileFilter fileFilter;

    public DirExplorer(FileFilter filefilter, Filehandler filehandler) {
        this.fileFilter = filefilter;
        this.filehandler = filehandler;
    }

    public void explore(File root) {
        explore(0, "", root);
    }

    private void explore(int level, String path, File file) {
        if (file.isDirectory()) {
```

```
            for (File subfile : file.listFiles()) {
                explore(level + 1, path + "/" + subfile.getName(), subfile);
            }
        } else {
            if (fileFilter.interested(level, path, file)) {
                filehandler.handle(level, path, file);
            }
        }
    }
}
```

NodeIterator class

NodeIterator.java - The Java class file to iterate over the different `Nodes' created by the JavaParser to parse the source code.

```
package com.textfromcode;


import com.github.javaparser.ast.Node;


public class NodeIterator {
    public interface Nodehandler {
        boolean handle(Node node);
    }

    private Nodehandler node_handler;

    public NodeIterator(Nodehandler nodeHandler) {
        this.node_handler = nodeHandler;
    }

    public void explore(Node node) {
        if (node_handler.handle(node)) {
            for (Node childNode : node.getChildNodes()) {
                explore(childNode);
```

```
        }
      }
   }
}
```

DataModelObj class

DataModelObj.java - The Java class file to generate data objects from the information parsed by JavaParser.

```
package com.textfromcode;


public class DataModelObj{
        // constructor
        int id;
        String classRealName;
        String methodName;
        String [] methodDeclText;


        public DataModelObj (int id, String classRealName, String methodName, String [] methodDeclText) {
                this.id=id;
                this.classRealName=classRealName;
                this.methodName=methodName;
                this.methodDeclText=methodDeclText;
        }


        public int getId() {
                return id;
        }


        public String getClassRealName() {
                return classRealName;
        }
```

```
	public String getMethodName() {
		return methodName;
	}

	public String[] getMethodDeclText() {
		return methodDeclText;
	}

	@Override
	public boolean equals(Object o) {
		// TODO Auto-generated method stub
		if (o == this)
	    return true;
	  if (!(o instanceof DataModelObj))
	    return false;
	  DataModelObj other = (DataModelObj)o;
	  boolean methodNameEquals = (this.methodName == null && other.methodName
== null)
	    || (this.methodName != null && this.methodName.equals(other.methodName));
	  return this.id == other.id && methodNameEquals;
	}

	@Override
	public int hashCode() {
		// TODO Auto-generated method stub
		return super.hashCode();
	}

}
```

TFCmain class

TFCmain.java - The Java main file contains the parser implementation, natural language generator and flowchart generator code. Most of the flowchart generator code is commented out, because the implementation of this part is not finished.

```
package com.textfromcode;


/*
 * @author Vladimir Yesipov
 *
 */


import com.github.javaparser.JavaParser;
//import com.github.javaparser.ParseException;
import com.github.javaparser.ast.body.ClassOrInterfaceDeclaration;
import com.github.javaparser.ast.body.MethodDeclaration;
import com.github.javaparser.ast.expr.MethodCallExpr;
import com.github.javaparser.ast.expr.VariableDeclarationExpr;
import com.github.javaparser.ast.visitor.VoidVisitorAdapter;
import com.google.common.base.Strings;
import com.github.javaparser.ast.Node;
import com.github.javaparser.ast.stmt.Statement;
import com.textfromcode.DirExplorer;
import com.textfromcode.NodeIterator;
import com.textfromcode.DataModelObj;


import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
```

```
public class TFCmain {
	static List<DataModelObj> methodDeclObjList = new ArrayList<DataModelObj>();
	static int classCount = 0;
	static int methodDeclCountProj = 0;
	static int methodCallsCountProj = 0;
	static int variableDeclCountProj = 0;
	static String classRealName = null;

    public static void generateNLunits(File projectDir) {



	//Clear out the output file content
	try{
		PrintWriter writer = new PrintWriter("nlgtext.txt");
		writer.print("");
		writer.close();
	} catch (IOException e) {
		System.out.print("Textfile IO Exception");
	}

	//Start writing into the file
	try(FileWriter fw = new FileWriter("nlgtext.txt", true);
		   BufferedWriter bw = new BufferedWriter(fw);
		   PrintWriter out = new PrintWriter(bw))
		{
		   new DirExplorer((level, path, file) -> path.endsWith(".java"), (level, path, file)
-> {
			out.print("The project Java class file found at location:\r\n" + file + "\r\n");
			out.println(Strings.repeat("=", path.length() + 100));
			classRealName = path.substring(path.lastIndexOf("/")+1,
path.lastIndexOf("."));
```

```
                        classCount ++;
                        //Class visitor
                        try {
                                new VoidVisitorAdapter<Object>() {
                            @Override
                                public void visit(ClassOrInterfaceDeclaration className, Object
arg) {
                                    super.visit(className, arg);
                                    if (classRealName.equals(className.getNameAsString())) {
                                            out.println("The file content is the Java class code. The
Class Name is *** " + className.getName() + " *** ");
                                    }else {
                                            out.println("The Interface '" + className.getName() + "'
declared within the class '" + classRealName + "'");
                                    }
                                    out.println(Strings.repeat("=", path.length()+50));
                                }
                            }.visit(JavaParser.parse(file), null);

                            //Method declaration visitor
                            try {
                                out.println("Methods, declared within the class, are: ");
                                out.println(Strings.repeat("-", path.length()));
                               new VoidVisitorAdapter<Object>() {
                                int methodDeclCountClass = 0;
                                @Override
                                public void visit(MethodDeclaration methodDecl, Object arg) {
                                    super.visit(methodDecl, arg);
                                    methodDeclCountProj +=1;
                                    methodDeclCountClass +=1;
                                    out.println(); // empty line
                                    out.println("-=-<Method declaration #" +
methodDeclCountClass + " within the class " + classRealName + ">-=-");
```

```
out.print(" --- Method declaration lines: " + methodDecl.getBegin() + "-" + methodDecl.getEnd() + "\r\n"+ methodDecl + "\r\n");


/*
 *  NLG for Method declaration
 */
String methodDeclString = methodDecl.toString();
String [] methodDeclText = methodDeclString.split("[\r\n]+");

int methodDecl0LineIndex = 0;
for (int i=0; i < methodDeclText.length; i++) {
    if ((methodDeclText[i].startsWith("//")) || (methodDeclText[i].startsWith("/*")) || (methodDeclText[i].startsWith("*")) || (methodDeclText[i].startsWith(" *")) || (methodDeclText[i].startsWith("* ")) || (methodDeclText[i].startsWith(" * ")) || (methodDeclText[i].startsWith("*/")) || (methodDeclText[i].startsWith("@"))){
        methodDecl0LineIndex ++;
    }
}
String methodDecl0Line = methodDeclText[methodDecl0LineIndex].trim();
String methodName = "";
//System.out.println(methodDecl0Line);
if (methodDecl0Line.contains("(") && methodDecl0Line.contains(")")) {
    int openbr = methodDecl0Line.indexOf('(');
    int closebr = methodDecl0Line.indexOf(')');

    //Method name String declMethodName
    int methodNameStart = 0;
    String methodNameBegin = methodDecl0Line.substring(0, openbr).trim();
    methodNameStart = methodNameBegin.lastIndexOf(" ", methodNameBegin.length())+1;
    //System.out.println(methodNameStart);
```

```
                                        methodName =
methodDecl0Line.substring(methodNameStart, openbr);
                                        //String declMethodName=("The method is declared
with the name "' + methodName + "'. \r\n");


                                        //Method access modifier String methodAccessString
                                        String methodDeclSt = methodDecl0Line.substring(0,
openbr);
                                        String methodAccessString = "";
                                        if (methodDeclSt.contains("private")) {
                                            methodAccessString = ("' is 'private', accessible
within the class. \r\n");
                                        }else if (methodDeclSt.contains("public")) {
                                            methodAccessString = ("' is 'public', accessible
from everywhere. \r\n");
                                        }else if (methodDeclSt.contains("protected")) {
                                            methodAccessString = ("' is 'protected',
accessible within the package and outside the package but through inheritance only. \r\n");
                                        }else {
                                            methodAccessString = ("' is 'default', is
accessible within the package. \r\n");
                                        }


                                        //If method static
                                        String methodStaticString = "";
                                        if (methodDeclSt.contains("static")) {
                                            methodStaticString = ("This method type is
'static'. This method can be invoked without the need for creating an instance of a class. \r\n");
                                        }else {
                                            methodStaticString = ("This method type is not
'static'. \r\n");
                                        }


                                        //Method declaration parameters String paramString
                                        String methodParam =
methodDecl0Line.substring(openbr, closebr+1);
```

```
String paramString = "";
if (methodParam.length() > 2) {
    paramString=("It takes the parameter(s) " + methodParam + " as an input ");
}else {
    paramString=("It takes no parameters as an input ");
}

//Exceptions String methodExcStr
String methodExcStr = "";
try {
        String methodExc = methodDecl0Line.substring(closebr + 2, (methodDecl0Line.length()-1)).trim();

        if (methodExc.length()>2) {
            methodExcStr = ("The method " + methodExc + ". \r\n");
        }else {
            methodExcStr = ("The method throws no exception. \r\n");
        }
}catch(Exception e) {
    methodExcStr = ("The method throws no exception. \r\n");
}
//Method returns
String methodReturnsString = "";
if (methodDeclSt.contains("void")) {
    methodReturnsString = ("and returns no parameter. \r\n");
}else if (methodDeclSt.contains("int")) {
    if ((methodDeclSt.contains("int[]")) || (methodDeclSt.contains("int []"))) {
```

```
                                        methodReturnsString = ("and returns an
array of the integer number variables. \r\n");
                                    }else {
                                        methodReturnsString = ("and returns an
integer number variable. \r\n");
                                    }
                                }else if (methodDeclSt.contains("byte")) {
                                    if ((methodDeclSt.contains("byte[]")) ||
(methodDeclSt.contains("byte []"))) {
                                        methodReturnsString = ("and returns an
array of the byte type variables - the 8-bit signed two's complement integers. \r\n");
                                    }else {
                                        methodReturnsString = ("and returns a
byte type variable - an 8-bit signed two's complement integer. \r\n");
                                    }
                                }else if (methodDeclSt.contains("short")) {
                                    if ((methodDeclSt.contains("short[]")) ||
(methodDeclSt.contains("short []"))) {
                                        methodReturnsString = ("and returns an
array of the short byte type variables - the 16-bit signed two's complement integers. \r\n");
                                    }else {
                                        methodReturnsString = ("and returns a
short byte type variable - a 16-bit signed two's complement integer. \r\n");
                                    }
                                }else if (methodDeclSt.contains("long")) {
                                    if ((methodDeclSt.contains("long[]")) ||
(methodDeclSt.contains("long []"))) {
                                        methodReturnsString = ("and returns an
array of the long byte type variables - the 64-bit signed two's complement integers. \r\n");
                                    }else {
                                        methodReturnsString = ("and returns a
long byte type variable - a 64-bit signed two's complement integer. \r\n");
                                    }
                                }else if (methodDeclSt.contains("float")) {
                                    if ((methodDeclSt.contains("float[]")) ||
(methodDeclSt.contains("float []"))) {
```

```
                                        methodReturnsString = ("and returns an
array of the real, float type number variables. \r\n");
                                    }else {
                                        methodReturnsString = ("and returns a
real number variable of the float type. \r\n");
                                    }
                                }else if (methodDeclSt.contains("double")) {
                                    if      ((methodDeclSt.contains("double[]"))      ||
(methodDeclSt.contains("double []"))) {
                                        methodReturnsString = ("and returns an
array of the real, double type number variables. \r\n");
                                    }else {
                                        methodReturnsString = ("and returns a
real number variable of the double type. \r\n");
                                    }
                                }else if (methodDeclSt.contains("boolean")) {
                                    if      ((methodDeclSt.contains("boolean[]"))      ||
(methodDeclSt.contains("boolean []"))) {
                                        methodReturnsString = ("and returns an
array of the boolean type variables. \r\n");
                                    }else {
                                        methodReturnsString = ("and returns a
variable of the boolean type. \r\n");
                                    }
                                }else if (methodDeclSt.contains("char")) {
                                    if      ((methodDeclSt.contains("char[]"))      ||
(methodDeclSt.contains("char []"))) {
                                        methodReturnsString = ("and returns an
array of the character value variables. \r\n");
                                    }else {
                                        methodReturnsString = ("and returns a
character value variable. \r\n");
                                    }
                                }else if (methodDeclSt.contains("String")) {
                                    if      ((methodDeclSt.contains("String[]"))      ||
(methodDeclSt.contains("String []"))) {
```

```
                                        methodReturnsString = ("and returns an
array of the string value variables. \r\n");
                                    }else {
                                        methodReturnsString = ("and returns a
string value variable. \r\n");
                                    }
                                }else if (methodDeclSt.contains("File")) {
                                    if ((methodDeclSt.contains("File[]")) ||
(methodDeclSt.contains("File []"))) {
                                        methodReturnsString = ("and returns an
array of the files. \r\n");
                                    }else {
                                        methodReturnsString = ("and returns a
file. \r\n");
                                    }
                                }else {
                                    methodReturnsString = (" and the system could
not find what this method returns. \r\n");
                                }

                                //Object generation for FLOWCHART

                                //System.out.println(methodName);
                                //FlowchartObj
(methodName.toString()+methodDeclCountProj.toString())          =          new
FlowchartObj(classRealName, methodName, methodDeclText);
                                methodDeclObjList.add(new
DataModelObj(methodDeclCountProj, classRealName, methodName, methodDeclText));
                                //System.out.println(methodDeclObjList.size());
                                /*Iterator<FlowchartObj>          iterator          =
methodDeclObjList.iterator();
                                while (iterator.hasNext()) {
                                    FlowchartObj methObj = iterator.next();
                                    int methId=methObj.getId();
                                    String
currentMethName=methObj.getMethodName();
```

```
                                        System.out.println(methId + currentMethName);
                                    }*/


                                    //Natural language text generation for method declaration 'section'
                                    out.print("The method '" + methodName + "' from class '" + classRealName + methodAccessString + paramString + methodReturnsString + methodStaticString  + methodExcStr);
                                    }
                                }
                            }.visit(JavaParser.parse(file), null);
                            out.println(); // empty line
                        } catch (IOException e) {
                            new RuntimeException(e);
                        }


                        try {
                            out.println("Method calls: ");
                            out.println(Strings.repeat("-", path.length()));
                            new VoidVisitorAdapter<Object>() {
                                int methodCallsCountClass = 0;
                                @Override
                                public void visit(MethodCallExpr methodCall, Object arg) {
                                    super.visit(methodCall, arg);
                                    methodCallsCountClass ++;
                                    methodCallsCountProj ++;
                                    out.println(); // empty line
                                    out.println("-=-<Method call #" + methodCallsCountClass + " within the class"  + classRealName + ">-=-");
                                    out.print(" Method call lines: " + methodCall.getBegin() + "-" + methodCall.getEnd() + methodCall + "\r\n");
                                }
                            }.visit(JavaParser.parse(file), null);
                            out.println(); // empty line
```

```
                        } catch (IOException e) {
                            new RuntimeException(e);
                        }
                        //Variable declarations visitor
                        try {
                               out.println("Variable declarations: ");
                               out.println(Strings.repeat("-", path.length()));
                            new VoidVisitorAdapter<Object>() {
                               int variableDeclCountClass = 0;
                                @Override
                                public void visit(VariableDeclarationExpr variableDecl, Object
arg) {
                                    super.visit(variableDecl, arg);
                                    variableDeclCountClass +=1;
                                    variableDeclCountProj +=1;
                                    out.println(); // empty line
                                    out.println("-=-<Variable declaration #" +
variableDeclCountClass + " within the class"  + classRealName + ">-=-");
                                    out.println("  Variable declaration line: " +
variableDecl.getBegin() + variableDecl);
                                }
                            }.visit(JavaParser.parse(file), null);
                            out.println(); // empty line
                        } catch (IOException e) {
                            new RuntimeException(e);
                        }

                        //Statements section repeats the content of method declaration
                          new NodeIterator(new NodeIterator.Nodehandler() {
                            @Override
                            public boolean handle(Node node) {
                               if (node instanceof Statement) {
                                    out.println("Statement lines: " + node.getBegin() + " - " +
node.getEnd());
```

```
                    out.println(Strings.repeat("-", path.length()));
                    String nodeString = node.toString();
                    String [] nodeText = nodeString.split("[\r\n]+");

                    for (int i = 0; i < nodeText.length; i++) {
                        String nodeLine = nodeText[i];
                        //out.println(i + " - " + nodeLine);

                        if (i < (nodeText.length -1)) {
                            if ((nodeLine.contains("javax")) && !((nodeText[i+1]).contains("javax"))) {
                                out.println(i + " *** UI generation *** ");

                            }else {
                                if ((nodeLine.contains("javax")) && ((nodeText[i+1]).contains("javax"))) {
                                out.println(i);
                            }else {
                                out.println(i + " - " + nodeLine);
                            }
                            }

                        }else {
                            out.println(i + " - " + nodeLine);
                        }

                    }
                    return false;
                } else {
                    return true;
                }
            }
        }).explore(JavaParser.parse(file));
```

```
                    out.println(Strings.repeat("-", path.length())); // end line


                } catch (IOException e) {
                    new RuntimeException(e);
                }
            }).explore(projectDir);
                //Statistical information
                out.println("-=-<The total amount of classes in the project: " + classCount + ">-=-");
                out.println("-=-<The total amount of methods declared within the project: " + methodDeclCountProj + ">-=-");
                out.println("-=-<The total amount of method calls within the project: " + methodCallsCountProj + ">-=-");
                out.println("-=-<The total amount of variables declared within the project: " + variableDeclCountProj + ">-=-");
                out.close();
                System.out.print("The report text file 'nlgtext.txt' generated and saved");
        } catch (IOException e) {
                System.out.print("Textfile IO Exception");
        }
    }

    public static void generateFlowchart() {


        //find method main and draw the start block
        Iterator<DataModelObj> iterator = methodDeclObjList.iterator();
      while (iterator.hasNext()) {
        DataModelObj methObj = iterator.next();
        //int methId=methObj.getId();
        //String currentMethName=methObj.getMethodName();
        //System.out.println(methId + currentMethName);
        if (methObj.getMethodName().equals("main")) {
                //System.out.println("main found");
                String [] currMethText = methObj.getMethodDeclText();
```

```
                for (int i = 0;i < currMethText.length;i++) {
                        String methStr=currMethText[i];
                        if (methStr.contains("(") && methStr.contains(")") &&
!methStr.contains("main")) {
                                //System.out.println(methStr);
                        }
                }
        }
    }
        /*try {
                for (FlowchartObj curMeth : methodDeclObjList) {
                        String currentMethName=curMeth.getMethodName();
                        int methId=curMeth.getId();
                        //System.out.println(currObj);
                        //String currentMethName = ((FlowchartObj)
currObj).getMethodName();
                        System.out.println(methId + currentMethName);
                        if (currentMethName == "main") {
                                String [] currentMethText = FlowchartObj.getMethodDeclText();
                        }
                }




                }
        } catch (Exception e) {
                System.out.print("FlowchartObj Exception");
    }*/


  }


        public static void main(String[] args) {
                try {
```

```
            File projectDir = new File(args[0]);

                    //File projectDir = new File("D:\\AbUni\\Semester_4\\CS4525_CS4527DissertProject\\Example Projects\\Number Guessing Game in Java");

                    //File projectDir = new File("D:\\AbUni\\Semester_4\\CS4525_CS4527DissertProject\\Example Projects\\sample-java-project-master");

                    //File projectDir = new File("D:\\AbUni\\Semester_4\\CS4525_CS4527DissertProject\\Example Projects\\Airlines Reservation System Java Project");

            //File projectDir = new File("D:\\AbUni\\Semester_4\\CS4525_CS4527DissertProject\\PDF file text extractor");

                    //File projectDir = new File("D:\\eclipse-workspaceIDEforJEE\\TextFromCode");

            generateNLunits(projectDir);

            generateFlowchart();

               }catch (Exception e) {

                    System.out.println("You need to specify the path as an argument! Example: C:\\SomeFolder\\JavaProjectFolder");

               }


        }
}
```