A Software Tool for Learning Syntax

Andreas Leon Aagaard Moth

Kongens Lyngby 2011
IMM-BSc-2011-11
This report was created as part of a bachelor project, at the Institute of Informatics and Mathematical Modelling at the Technical University of Denmark. The project was created in the five month period from February to June 2011 as a part of the Bachelor degree in Software Technology and corresponds to 15 ECTS points. The project was created with supervision by Jørgen Villadsen and Mordechai Ben-Ari.

As prerequisites for making this project, I have the following courses:

- 02141 Computer Science Modelling.
- 02811 Human Computer Interaction.
- 02152 Concurrent Systems.
- 02105 Algorithms and Data Structures 1.

I also have a part time job in a software company, where I have learned a lot about software development and management along with about 7 years of experience with programming.

Lyngby, June 2011

Andreas Leon Aagaard Moth
Abstract

The purpose of this project is to create a tool that will help students understand the syntax errors they make when they program, such that they can focus on writing code and learning the language instead of spending most of their time trying to fix an error they do not understand. The tool will display the errors graphically using syntax diagrams. These diagrams will show where the error occurred and what was expected instead.
Formålet med dette projekt er at lave et værktøj til at hjælpe studerende med at forstå de syntax fejl de laver mens de skriver programmer, således at de kan fokusere på at skrive kode og lære sproget at kende i stedet for at bruge det meste af deres tid på at fikse en fejl de alligevel ikke forstår. Værktøjet vil grafisk vise fejlene ved brug af syntax diagrammer. Disse diagrammer vil vise hvor fejlen opstod samt hvad der var forventet i stedet.
I would like to thank Jørgen Villadsen for supervising the project and helping me with the design. I also thank my co-supervisor Mordechai Ben-Ari for helping me throughout the project with inputs and ideas regarding the tool and editing the user guide. I also thank him for making a poster and presenting the project at the ITiCSE 2011 conference, which is the 16th Annual Conference on Innovation and Technology in Computer Science Education.
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preface</td>
<td>i</td>
</tr>
<tr>
<td>Abstract</td>
<td>iii</td>
</tr>
<tr>
<td>Resumé</td>
<td>v</td>
</tr>
<tr>
<td>Acknowledgements</td>
<td>vii</td>
</tr>
<tr>
<td>1 Introduction</td>
<td>1</td>
</tr>
<tr>
<td>2 Theory</td>
<td>3</td>
</tr>
<tr>
<td>2.1 Syntax diagrams</td>
<td>4</td>
</tr>
<tr>
<td>2.2 BNF</td>
<td>7</td>
</tr>
<tr>
<td>3 Source code verification</td>
<td>11</td>
</tr>
<tr>
<td>4 BNF compiler implementation</td>
<td>15</td>
</tr>
<tr>
<td>4.1 BNF file</td>
<td>16</td>
</tr>
<tr>
<td>4.2 ANTLR</td>
<td>18</td>
</tr>
<tr>
<td>4.3 Grammar file</td>
<td>20</td>
</tr>
<tr>
<td>4.4 JAVA grammar</td>
<td>21</td>
</tr>
<tr>
<td>5 SyntaxTrain GUI design</td>
<td>23</td>
</tr>
<tr>
<td>5.1 Clapham</td>
<td>26</td>
</tr>
<tr>
<td>6 SyntaxTrain implementation</td>
<td>27</td>
</tr>
<tr>
<td>6.1 Tests</td>
<td>30</td>
</tr>
<tr>
<td>7 Limitations and improvements</td>
<td>33</td>
</tr>
</tbody>
</table>
## CONTENTS

8 SyntaxTrain usage examples 37
  8.1 Example 1 ................................................. 37
  8.2 Example 2 ................................................. 39
  8.3 Example 3 ................................................. 41
  8.4 Comparison: Syntax diagrams vs compiler messages ............ 42

9 Conclusion 47

A BNF Evaluator 49

B Java parser test 53

C Java grammar BNF 57

D Java tests 63
  D.1 Test 1 ...................................................... 63
  D.2 Test 2 ...................................................... 63
  D.3 Test 3 ...................................................... 64
  D.4 Test 4 ...................................................... 64
  D.5 Test 5 ...................................................... 64
  D.6 Test 6 ...................................................... 64
  D.7 Test 7 ...................................................... 65
  D.8 Test 8 ...................................................... 65
  D.9 Test 9 ...................................................... 65
  D.10 Test 10 .................................................... 66

E SyntaxTrain source code 67
  E.1 BnfCompiler .............................................. 67
  E.2 Exceptions ............................................... 90
  E.3 Grammar .................................................. 92
  E.4 GUI ....................................................... 94
  E.5 GuiAPI .................................................... 127
  E.6 Init ....................................................... 128
  E.7 Kernel ..................................................... 130
  E.8 KernelAPI ............................................... 146
  E.9 Library ................................................... 148
  E.10 Xml ...................................................... 151
Chapter 1

Introduction

In the last couple of years more and more students are learning to program. Writing code is not very easy and students still makes lots of errors which takes a lot of their attention away from the actual code. The most common errors are minor ones that can be corrected quickly with the information given by the compiler, but sometimes the error cannot be seen with the naked eye and the explanation given by the compiler does not make sense to the student. In this case the student sometimes spends several minutes or longer trying to understand what is wrong. Often the problem is that the student either forgot how the syntax for the given code looks or reads what he/she believe is written instead of what is actually written (and that way overlooking the actual problem). During the years many approaches have been tried in order to solve this problem, from written materials to giving an understandable error message, but it still remains a big problem [3].

SYNTAXTRAIN is yet another approach at trying to explain the error to the student. The goal of this project is to make a tool that reduces the time students spends on correcting time consuming errors in their code, such that they can focus on learning the language instead. The idea is to make a visual representation of syntax errors using syntax diagrams, which can be used by students when they are learning a new language in order to understand what the problem is and how to correct it. The reason for using syntax diagrams are that they do not require much knowledge to use and are self explanatory.
Syntax diagrams have been used for a long time, they were first used in Pasc­
al; see: K. Jensen and N. Wirth. *PASCAL User Manual and Report*, LNCS 18, Springer-Verlag, 1975. They are often used for showing the structure of a language because of their intuitiveness, but for some reason they almost do not appear in programming textbooks anymore, they are mainly available on sited for technical sites.

The report starts out by introducing the fundamental theory and technical terms used in the report. After this the code verification process is explained along with how the tool is split into two different tools, a BNF compiler and the ac­tual SYNTAXTRAIN. The BNF compiler is then explained in details, followed by the gui and implementation of SYNTAXTRAIN. The limitations and possible improvements are discussed, followed by some usage examples of SYNTAXTRAIN along with a comparison between SYNTAXTRAIN and compiler messages. Fi­nally the report ends with a conclusion to sum up everything.
Chapter 2

Theory

The purpose of this chapter is to explain the technical terms used throughout this report and describe what syntax diagrams and BNF are and how they are used. Additionally the meaning of the following words will be explained: syntax, semantic, decidable, undecidable, token, whitespace, backtrack, syntax diagrams and BNF.

The title of this report already contains one of these words, namely syntax. Syntax describes how a language is structured, like in our language where every sentence is ended with a dot, question mark or similar. Semantic is the opposite of syntax, instead of describing how stuff is arranged it describes what can be written, for the sentence makes sense. As an example you cannot write "it" in a sentence without having a noun which it can refer to.

In this report decidable is used in combination with rules, where it means that the rule only describes one way to get any kind of output. The opposite of decidable is undecidable, which means that the rule describes more than one way to get some input. A simple example of an undecidable rule is one that accepts an optional character followed by another optional character. This means the input "a" can be described by both the first character and the second since they are both optional. If the rule instead accepted an optional character followed by an optional number, then it would be decidable, because it would only be possible to match the rule one way for every input.
When source code is verified it is split up into tiny elements, describing words, spaces, dots etc. (just before being split into individual characters). Each of these elements is a token. Tokens can specify a complex input like any word starting with the letter d or a specific word like "case". An example is the whitespace, which describes space, line feed, new line or tab.

Backtracking is related to rules and means to abandon the current rule that was being matched by going back to the previous rule and try another option. It is used when the current rule can no longer be matched with the given input, such that a different rule can be tried instead. A simple example would be a rule that accepts one of two other rules, where the first accepts "Hello" followed by "world" and the second accepts "Hello" followed by "everyone". Assume the input "Hello everyone" is given. The first rule matches the first token "Hello" so that rule is tested, however the second token "everyone" does not match with "world", so we backtrack to the previous rule and try the second rule instead. Here both "Hello" and "everyone" is matched and the input is accepted.

2.1 Syntax diagrams

Syntax diagrams (also known as railroad diagrams) are used to represent the structure of a language as a set of diagrams. Each diagram represents a part of the language and together they represent the entire language. Each diagram starts in a single point to the far left, splits into different roads which may cross, and finally converge in a final point to the far right. The roads are drawn as arrows, which you must follow in the given direction. If an arrow splits then it is optional which to follow. In order for the source code to be accepted, it must describe a path through the diagrams. Each road consists of one or more terminals and non-terminals. A terminal represent specific input like "case", meaning the word case must appear at this position in the code. Non-terminals are references to other diagrams, which must be accepted before continuing on the current path. Terminals are written in round boxes while non-terminals are written in square boxes. Terminals are normally written without quotes, but that means, the only difference between the two are their shapes, so the student must know what the boxes means. In order to help the student, quotes have been added around terminals, to make it more intuitive that what is written in these boxes should actually be found in the input as raw text. The smart thing about using syntax diagrams like this is that they are self explanatory, meaning the student does not have to learn any technical terms or a completely different language in order to use them. Start from the left and follow the arrows until you reach the right side, it is as simple as that!
The different elements of syntax diagrams include multiple options, sequence, optional and loop, see Figure 2.1, 2.2, 2.3 and 2.4 respectively. Multiple options means that exactly one of the components must be matched, a sequence specifies that all components must be matched in the given order, an optional element is something that may be matched but not required and finally a loop means the component inside this element may occur zero or more times. These elements can be put together in any order, to form a syntax diagram.

**Figure 2.1: Syntax diagram with multiple options.**

**Figure 2.2: Syntax diagram of a sequence.**

**Figure 2.3: Syntax diagram of an optional element.**

### 2.1.1 Examples of syntax diagrams

To understand how syntax diagrams looks, let us look at the simple example in Figure 2.5.

The diagram starts with the option to choose either "I" or "You". Afterwards "parked the car" is required followed by "behind the fence" which is optional and finally "." is required. From this simple diagram four different sentences are accepted. This diagram only contains terminals, a more common diagram would be like the one seen in Figure 2.6, which is the representation of a switch statement in Java.
The switch statement starts by requiring the two terminals "switch" and "(". Next an expression is required, indicating that diagram must be accepted, after which the terminals ")" and "){" are required. Now interestingly it is possible to go through the loop zero or more times, while each time taking one of the three paths: (i) take terminal "case", followed by the non-terminal expression and finally the terminal "):" or; (ii) take the two terminals "default" followed by "):" or; (iii) take the non-terminal statement, which is described in yet another diagram. After looping through this, the sequence must be ended by the terminal "){".

### 2.1.2 Expanding syntax diagrams

As mentioned earlier, syntax diagrams are used in SyntaxTrain to visualize syntax errors, however just having the diagram does not help the student very much unless he/she knows where the error occurred inside the diagram. In order to do this, visual clues has been added, by coloring the terminals and non-terminals in each diagram; their meanings are:
• **Black:** This rule is not relevant for your source code.

• **Blue:** This rule has been correctly matched with your source code.

• **Red:** This rule caused an error when parsing your source code.

• **Yellow:** These rules are legal to write at the position of your error.

An example of this can be seen in Figure 2.7, which is a `switch_statement` in JAVA, like the one shown in Figure 2.6, but with colors.

![Figure 2.7: Colored switch_statement in JAVA.](image)

Here it is instantly clear that the student forgot to write a colon, since the entire path to the colon is marked blue, meaning it has been accepted without any problems, but the colon is not blue, meaning it was not matched, instead it was marked yellow, indicating it is legal to write here. Since this is the only yellow terminal/non-terminal it is the only option, short of removing code.

### 2.2 BNF

BNF stands for Backus-Naur Form, and is a way to describe context free grammars, meaning the syntax of a language and not its semantics. A BNF consists of a set of rules, which each describes a part of the language and together the entire language, just like the syntax diagrams, the only difference is how the rules are represented. In BNF the rules are given as text while syntax diagrams show them visually. It is possible to go from one to the other without data loss.

There are many derivations of BNF but they all build on the same principles. In order to avoid confusion, only the BNF variant used in this project will be explained. This BNF variant that were chosen is based on the one used to describe the JAVA language at [http://www.cui.unige.ch/db-research/Enseignement/analyseinfo/JAVA/BNFindex.html](http://www.cui.unige.ch/db-research/Enseignement/analyseinfo/JAVA/BNFindex.html), which was used to verify code for the JAVA language.
A BNF consists of one or more rules, that are placed one after the other. A rule is described by a rule name, followed by an equal sign, followed by an expression and ended by a dot. The rule name must be a single word, underscores and numbers are allowed. The expression consists of a sequence of terminals and non-terminals, these are separated by one or multiple whitespaces. Terminals describe a text that must occur and are enclosed by quotes, while the non-terminals are names of other rules, which must be accepted at this point. In this sequence other elements may also occur, which describes different behavior; these are:

1. Parenthesis () with an expression inside. This encapsulate a sequence of terminals/non-terminals.
2. Square brackets [] with an expression inside. This indicates that the expression inside is optional.
3. Triangular brackets <> with an expression inside. This indicates that the expression inside may loop zero or more times.
4. Slash / with an expression on each side, meaning one or the other must be taken, but only one of them.

Additionally the BNF cannot contain comments. In order for some input to be accepted by the BNF, it must describe a way to reach the end of the starting rule. This is equivalent to the syntax diagrams, where the input had to describe a path through the diagram [2].

### 2.2.1 BNF example

In order to better understand how a BNF is understood, an example is shown in Figure 2.8.

What this BNF does is to accept input with the exact same syntax as the BNF compiler accepts. This way the example is actually able to accept itself. The first line specifies the start rule, which indicates that there must be at least one rule and that each rule ends with a dot. The second line specifies that a rule consists of an identifier followed by an equal sign and finally an expression. The expression consists of one or more orExpressions, which describes one or more exprBase, that are separated by slashes. Finally the exprBase states that each expression must be a string, an identifier or encapsulated in either parenthesis, squares or triangular brackets.
bnfStartRule = rule "." <rule ".">.  
rule = IDENTIFIER ":=" expression.  
expression = orExpression <orExpression>.  
orExpression = exprBase <"/" exprBase>.  
exprBase = STRING  
/ IDENTIFIER  
/ ( "(" expression ")" )  
/ ( "[" expression "]" )  
/ ( "<" expression ">" ).

Figure 2.8: BNF grammar example.
Chapter 3

Source code verification

This chapter will explain the source code verification process; how is the code verified, what is required to show the syntax diagrams, are there any problems and how are they solved.

The idea of this project is to make a tool that will read the students source code, verify it and visualize any errors found using syntax diagrams. In order to verify source code for different languages, each language must first be specified and a parser and lexer must be created from the specification, which will read the source code and return an error trace if there is any error. In order to describe these languages, BNF was chosen since it is commonly used to specify other languages, and for that reason most languages are already specified in BNF.

Parsers are quite hard to make from scratch and takes up a lot of time, so it was decided to use an external tool for this. There exist many parsers on the internet, some of the most common are ANTLR, Javacc, CUP and SableCC. For SyntaxTrain I decided to go with ANTLR, since it can generate both a parser and a lexer from a single file, it has good documentation and the grammar for ANTLR code is much like BNF. However ANTLR was not written to return a trace of which internal rules that were matched or not but how the code were matched. In order to get a trace of these rules I had to ”cheat” by making an internal stack, where each rule adds itself when started and removes itself when it has been matched. This way the rules in the stack are those that were not
matched. These rules have been added exactly like a stack trace, meaning the first rule added is the starting rule and the last rule is where ANTLR failed to match a token. As a result of doing it this way, ANTLR cannot backtrack, since the current rule has already been added to the stack. This means the grammar has to be decidable. A huge part of programming languages are usually decidable, because of their many parenthesis, curly brackets and squares that encapsulates the different parts of the code, however there is one rule that are undecidable because it consists of two parts and does not require curly brackets which is the if-sentence. This means that you can write the if-sentence seen in Figure 3.1.

```java
1 if( 1 == 1 )
2   if( 2 == 2 )
3     doSomething();
4   else
5     doSomethingElse();
```

Figure 3.1: Java Date class with a syntax error.

Here the else could belong to both the inner and outer if-sentence, there is no way to tell from ANTLR’s point of view. Of course the else belongs to the inner most if-sentence, that is the way it is defined.

Another problem with using ANTLR, is that it generates the parser and lexer in raw Java code, meaning they have to be compiled before they can be used. To solve this problem the product was split into two tools. (i) A command line tool that uses the BNF specification to make an ANTLR file, used to generate the parser and lexer file, which are compiled and saved inside a grammar file. (ii) A graphical user interface called SYNTAXTRAIN, that uses the grammar file to verify source code.

This means that to create a grammar file for a new language the Java development kit is required, but only the standard Java runtime environment is needed for running SYNTAXTRAIN. The smart thing about doing it this way is that a normal student would most likely have Java runtime environment installed, but not the developer version. So for the student it is easy to use, which is important, since they already have a lot on their mind (trying to learn a completely new programming language). The problem of the development kit has been pushed over to the grammar creators, but these people are most likely lectors or similar who knows how to install it, so that should not be a problem. Speed is also gained by the parser and lexer being compiled only one time.

In order to visualize how the tools are used, a use case has been made in Fig-
Figure 3.2, which shows the process of creating a grammar file, verifying code and how they are connected. It is a crude example, but should make the verification process easier to understand. In the diagram the BNF creator first writes the BNF, from which a grammar file is made. This grammar file is then used to verify the students source code, such that errors can be presented in a way that is easy to understand. The error is then fixed, the code is re-verified, all errors are gone and the compiler will say the same (assuming there are only syntax errors).

In order for SyntaxTrain to be able to read any kind of grammar file the command line tool makes, a known structure is required, such that SyntaxTrain knows which functions to call in order to use the parser. This is accomplished by using an interface class which SyntaxTrain knows and the parser generated by the command line tool implements.
Figure 3.2: Use case for creating grammar file and verifying code.
Chapter 4

BNF compiler implementation

This chapter will describe how the BNF compiler is implemented; how does it work, which resources are used and how are they used.

The BNF compiler is a purely command line tool, that uses ANTLR to read a text file containing the grammar of a language in BNF and create a grammar file that can be used by SYNTAXTRAIN. By using BNF it is possible to specify the syntax of almost any language or structure, even XML. In order to compile the source files generated by ANTLR and create a jar file, the BNF compiler uses javac and jar located in the JDK bin directory.

To convert a BNF file into a grammar file, the following steps are performed:

1. The BNF bin folder location is read from the options.xml file.
2. Integrity check, to make sure that all the files needed to create the grammar file exist.
3. Clean up, to make sure nothing unnecessary is put into the grammar file.
4. Use ANTLR to read the BNF file and create Link classes that describes the BNF.
5. Parser and lexer files are generated using ANTLR.
6. The parser and lexer is compiled using javac.
7. Syntax XML file is created (specifies the BNF language).
8. Everything is put into a grammar file using jar.
9. Finally a clean up to remove all temporary files.

In order to illustrate the BNF compilers role, a block diagram has been made to show the overall design structure, see Figure 4.1. The BNF compiler is located to the far left and are marked blue, the components used by the BNF compiler are marked green. From the block diagram it can be seen that both the BNF compiler and SYNTAXTRAIN (its kernel) communicates with ANTLR and the grammar interface. This is because the BNF compiler generates a grammar file that is used by the kernel to verify source code. The interaction between the kernel and the grammar file will be explained in Chapter 6.

![Block Diagram](image_url)

**Figure 4.1:** Block diagram with BNF compiler in focus.

### 4.1 BNF File

The BNF file is a file containing a list of rules, which describe a language using the BNF notation described the theory chapter.

The BNF must abide by the following list of rules, due to implementation issues described in chapter 3.
• The first rule is the starting rule, that is, the rule that must be accepted for the code to be accepted.

• The BNF may not be left recursive, meaning the rules in Figure 4.2 and 4.3 are invalid, even if a rule is not left recursive by itself, then its subrules could make it left recursive.

• The BNF must be decidable, that is, there must be only one way to match a given string.

\[
\text{myRule} = ( \text{myRule} "a" ) | "b". \\
\]

Figure 4.2: Left-recursive rule - 1.

\[
\text{myRule} = ( \text{anotherRule} "a" ) | "b". \\
\text{anotherRule} = \text{myRule} "c". \\
\]

Figure 4.3: Left-recursive rule - 2.

Because the BNF cannot be left recursive, it becomes much harder to write grammars for other languages, however it is possible as demonstrated with the grammar that I have created for JAVA which will be described later, see appendix C. The decidability rule makes some parts of the language impossible to recreate, such as the if sentence described in chapter 3, where the only solution is to require the curly parenthesis. Fortunately this problem is reduced a bit in severity since it is highly recommended to use the curly parenthesis no matter what, because leaving them out is a big source for errors, when the code is edited.

Due to implementation difficulty, it is not possible to write your own complex tokens. For this reason a list of tokens are automatically added to the grammar, which can freely be used in the BNF; they are:

• IDENTIFIER: any string consisting of letters, underscore and numbers, and starting with a letter or underscore.

• INT: integer literals, ex. 532 and -17.

• FLOAT: float literals like 1.3e7 and .6.

• HEX: hex literals, which must start with a number as in 0FE.
• STRING: strings delimited by quotation marks such as "Hello world!".
• HEX_DIGIT: a single-digit hex literal, meaning 0-9 and a-f.

Additionally the following are automatically ignored:

• Comments: delimited by /* and */ or // until end of line.
• Whitespace: new lines, line feed, tabs and spaces.

Even though it is not possible to write complex tokens, it is still possible to write simple ones. A simple token is a specific terminal like "case" or "{", while complex tokens are stuff like a float or an identifier.

### 4.2 ANTLR

ANTLR is used for two things, reading the BNF and creating a parser. In order to read the BNF an ANTLR parser was created, which reads a BNF file and creates a Link for each rule. These Link contains other Link, which are either in sequence, optional, one of multiple or a loop. The source code for this BNF reader can be seen in appendix A. The BNF reader works in much the same way as the example code shown back in Figure 2.8, where a Link contains a sequence of Link that specifies orExpressions. Each orExpression Link contains a Link which contains either a string, an identifier or an expression inside either parenthesis, square brackets or triangular brackets. If one of the three last are taken, then that Link is told that it is either a sequence, optional or a loop respectively.

For an example see the BNF in Figure 4.4, which specifies a for statement in Java, Figure 4.5 then tries to show how it will be recreated in Java using the Link class. In order to simplify the example, the Link have the prefix "sequence", "or" or "optional", in order to tell which type of Link they are. This way all BNF grammars can be represented in an easy to access Java structure.

ANTLR is also used for generating a parser for the language specified by the BNF and returning a stack trace. As mentioned earlier ANTLR was not build to do this and had to be "hacked". Besides from building the stack trace, ANTLR also tries to recover when an error is found by ignoring tokens. This had to be removed since compilers does not recover from errors. Additionally the lexer
for_statement =
  "for" "(" ( variable_declaration / ( expression ";" ) / ";" )
  [ expression ] ";"
  [ expression ] ";"
  ")" statement ;

Figure 4.4: BNF example of a for statement in JAVA.

for_statement =
  sequenceLink(
    Link("for"),
    Link("("),
    orLink(
      Link(variable_declaration),
      sequenceLink(
        Link(expression),
        Link(";"))
    ),
    Link(";")
  ),
  optionalLink(expression),
  Link(";"),
  optionalLink(expression),
  Link("")
);  

Figure 4.5: Example of how Link works.
does not report an error if some part of the input could not be matched to a
token. This is a huge problem, because in this case SyntaxTrain would show
a completely different error than what is actually the problem. To solve this
the nextToken function in the lexer were overwritten with a modified version,
which reports an error if a token could not be matched. A plausible example of
a token that cannot be matched is a string that ends on the second line (there
is a new line between the two quotes). This is illegal in Java and the STRING
token does not match it either. With the new modifications this is reported as
an error and SyntaxTrain will mark the string as being wrong.

The BNF can be converted quite easily to ANTLR, because ANTLR uses a
structure much similar to the BNF, with just a few changes. The parser also
implements an interface, which specifies a function to start the parser, get the
trace and error position.

4.3 Grammar file

The grammar file SyntaxTrain uses is a jar archive that contains the following
files (grammarName is substituted with the name of the grammar):

- BnfParser.class
- grammarName.g
- grammarName.tokens
- grammarName.xml
- grammarNameLexer.class
- grammarNameLexer.java
- grammarNameParser.class
- grammarNameParser.java

In order to verify the source code, parser and lexer classes are required along
with the tokens file. To draw diagrams the xml file is needed, which contains the
same information as the BNF file, but is written in format that is much easier
to read. The grammar file could end there, but the intermediate files are also
included for debugging purposes, such that if anything should be wrong with
the grammar it is easy to go back and find out where the error occurred.
4.4 Java grammar

The grammar file may also contain some additional parser and lexer .class files called something like javagrammarLexer$DFA20.class. These files are created because they are classes inside the parser or lexer files and are used by the parser and lexer to verify the students source code.

4.4 Java grammar

In order to use SYNTAXTRAIN a grammar file is needed. I have for that reason decided to make a BNF specification for the java language and turn it into a grammar. There already exist lots of BNF specifications for JAVA, but all the specifications I found were either decidable or left recursive. For that reason I decided to do it from scratch, but based on an already existing specification<sup>1</sup>. This means some components had to be concatenated, which makes them somewhat harder to read. The expression statement is a perfect example of where I had to concatenate several rules, and probably the hardest part to solve altogether. This is because most of the helper rules it uses immediately refers to expression again, making it left recursive. The only way to avoid that was to centralize the rules and put the optional endings inside a post expression (they can be put at the end of any expression). I think this is acceptable for now when keeping in mind that this project is not about creating an undecidable BNF grammar for JAVA, and it can be modified later to look good by using more time. As an example it is currently possible to split expressionPost into assignment operators, logical operators, testing operators etc., with almost no effort at all. The grammar I created accepts most of the JAVA language, and are even able to verify most of the source code for SYNTAXTRAIN itself. The BNF specification for the JAVA language can be found in appendix C.

Since making this JAVA grammar was not the actual project, there are some limitations to what it accepts, for instance it does not accept for each loops, see Figure 4.6 for an example. But this too would be extremely easy to change in the BNF.

```java
1  for(int myInt : myIntArray ) {
2       doSomething(myInt);
3  }
```

Figure 4.6: For-in JAVA loop.

<sup>1See:</sup> [http://www.cui.unige.ch/db-research/Enseignement/analyseinfo/JAVA/BNFindex.html](http://www.cui.unige.ch/db-research/Enseignement/analyseinfo/JAVA/BNFindex.html)
This chapter will focus on the different parts of the GUI. It starts by explaining what is needed followed by how this is incorporated.

The purpose of the SyntaxTrain GUI is to visualize the students syntax error in his/her source code, such that it is easy to understand, correct and verify. As a result, it must be possible to open and save a file, along with the code being displayed. It must also be possible to edit the code and re-check the syntax. Furthermore the error must be visually displayed in a way that is easy to understand.

The design focus on making the program as intuitive and user friendly as possible. This is very important because a student should not spend time trying to understand how SyntaxTrain works while trying to fix a syntax error, that would kind of defeat the purpose of this tool. This is unlike the BNF compiler, where it is quite complicated to make a grammar, however that tool is not intended for students, but people who already understand programming and BNF. The student should be able to simply start the tool, open the source code and understand what the problem is, then edit the code and revalidate. In order to make the tool user friendly, all unnecessary graphical components should be removed while still looking like and containing what the user expects such as showing the source code and an intuitive way to revalidate the syntax [4]. One thing that I did not implement which the user expects are syntax highlight in
the source code, such as an IDE would do. The reason for this is the complexity of implementing such a feature, in both SyntaxTrain and in the grammar file (since it is different for all lanugages).

In order to implement the features described, the GUI consists of a tool bar and three panes. The tool bar gives access to open, save and validate the syntax, while the tree panes are used to show the source code and diagrams of the syntax errors found. Figure 5.1 shows a screen shot of SyntaxTrain. At the top of the window is a small pane that spans the width of the window; below it, the window is divided into larger left and right panes.

![SyntaxTrain GUI Design](image)

**Figure 5.1: Screen shot of SyntaxTrain.**

The toolbar contains the following buttons:

- **Open** - Open a file (Ctrl+O).
- **Reload file** - Reads the current file again (Ctrl+R).
- **Check Syntax** - Revalidates the code and draws syntax diagrams if a syntax error is found (F5).
- **Save** - Save the source code (Ctrl+S).
- **Show/hide syntax components** - Shows/hides a list of all syntax rules for the current language (F10).
- **Help** - Displays help information.
• **About** - Displays the copyright notice.

When a file is opened its source code is displayed in the left pane. If a syntax error is found, **SyntaxTrain** will highlight the code that did not match the grammar. The code can then be edited and revalidated (F5) without having to be saved.

The right frame shows the syntax diagrams for the rules involved in the incorrect code, starting with the rule that was last used, then the rule which called that rule and so on until the first rule of the language is reached. The diagrams are drawn with colors as described in Chapter 2.1.2. The diagrams are created using an external tool called Clapham. The top pane shows a list of the rules involved in the error, starting on the left with the last one. It also shows a little notification if the source code has been changed since the last verification (the text "modified" are added at the bottom left). If **Show/hide syntax components** is selected, a new pane will appear at the right of the window. The pane contains a complete list of the syntax rules of the current grammar. This can be used to look at specific parts of the language, without having to make it part of an error, in order to make **SyntaxTrain** show it.

Some minor features to make the GUI more user friendly are:

• The open dialog remembers the directory of the last opened file for the current session.

• If the source code has been modified but not saved and **SyntaxTrain** is closed, then a dialog appears, asking whether to save or not.

• The source code is automatically verified when a file is opened.

**SyntaxTrain** is designed to be used when a syntax error is encountered. The student will then open the source code in **SyntaxTrain**, which automatically shows the error. The student then looks at the highlighted code in the left pane and then at the last rule used—the top rule in the right pane. Since the error might have caused the parser to become confused, the student may have to look down through the diagrams until a diagram is found, which the student understands and then go back up, examining the rules on the way, looking for the first rule that does not match what was intended.
5.1 Clapham

One of the main components in the GUI are the syntax diagrams. These diagrams must be highlighted with colors and it must be possible to use different fonts for terminals and non-terminals. In order to do this a tool was needed for drawing the syntax diagrams, however all tools capable of drawing syntax diagrams appeared to have the same two major flaws in comparison to what SyntaxTrain needs:

1. The diagrams are written directly to the disk, with no way of just getting the image as a Java object instead.
2. The diagrams do not support colors.

Clapham are such a tool, however it is open source, written in Java, the diagrams are drawn onto a graphics object, before they are stored on the disk and each rule is stored as a set of nodes, which are accessible when the diagrams are drawn.

The fact that it is open source and written in Java means it is possible to change the code in order to accommodate the needs of SyntaxTrain. The diagrams are drawn onto a graphics element (part of the AWT toolkit), this makes it easy to draw directly to the graphics of a BufferedImage instead and just remove the code that saved the diagrams. The BufferedImage can then easily be drawn onto any panel in Java. Because each rule is stored as a set of nodes, that are accessible during the drawing process, it is possible to modify these nodes, such that they also specify color and font. The only other thing that needs to be changed is where they are drawn, such that it uses these colors and fonts instead of the default. Doing this makes it possible to create any combination of colors and font, such as to make text mono spaced for terminals and normal for non-terminals. Clapham was not a finished product and hence had some errors in it, which had to be fixed as well. The potential to extend Clapham to do all this is the reason it was chosen. Another problem with Clapham is that the terminals are not completely round even though they should be, but it should not be too hard to fix this in the future.
This chapter describes how SYNTAXTRAIN is implemented, how the external tools are used and how it is tested.

SYNTAXTRAIN is a tool that is used while programming. This makes it possible to make it either a plug-in to an existing IDE or an external tool. If SYNTAXTRAIN is made as a plug-in, it is possible to use code coloring from the IDE and automatically show up when a syntax error is found, but on the other side it is locked to that specific IDE. If SYNTAXTRAIN is made as an external tool instead, it would be usable for almost all grammars due to its flexibility. I decided to make SYNTAXTRAIN as an external tool because I have most experience with this and locking it to a single IDE will only make it useful for a handful of languages.

In the implementation of SYNTAXTRAIN, the singleton pattern has been used for most of the classes, in order to avoid dragging references around in the entire program.

In order to figure out what ANTLR is capable of, a tiny part of the JAVA language was hardcoded and trace values were added such that the information available could be used to generate syntax diagrams, see Appendix B. This code was then generalized by making a BNF compiler, which automatically generated files doing the same thing and having the parser extend a known interface as
28 SyntaxTrain implementation

explained earlier. The parser does not use the names for the terminals/non-terminals that were matched, but unique identifiers instead, to avoid name clashing when matching the trace with the BNF. The XML file in the grammar file are used to translate these identifiers to specific components in the BNF.

SyntaxTrain has been implemented with focus on using the well known model-view-controller architecture (the model is henceforth called the kernel). This architecture helps to isolate parts of the program, such that it is easier to understand. In respect to this, an api has been made for both the kernel and the gui, called GuiApi and KernelApi which are used to access the gui and kernel respectively. By doing this it is easier to later extend the program, because changes only has to be made in one place instead of everywhere some specific task is performed. The controller is a very small portion of the code and is closely related to the gui, which is why it does not have an interface. A single controller class manages all gui events, in order to isolate the control part of the tool. There is one exception to this, which is the pane containing the Syntax components. This pane has its own controller because it is so closely related to the gui. The kernel manages the interaction with ANTLR and the modelling part of Clapham; that is, to describe how the diagrams are structured and which components should be highlighted.

To illustrate the structure a block diagram of the system with SyntaxTrain in focus can be seen in Figure 6.1.

When the application starts, the Init component (marked red) is called. This component initializes the kernel and starts the application. The two main com-
ponents, the kernel and the gui are marked blue. As it can be seen the kernel and the gui does not directly communicate with one another, but does it through the GuiApi and KernelApi respectively. The gui also uses the GuiApi, but that is the controller part, such that it does not make direct calls to the gui, however the gui does use the controller directly. From the figure it can also be seen that the kernel uses ANTLR and the grammar file (used to verify source code) along with CLAPHAM, which the GUI also uses to draw syntax diagrams as described above. Hopefully this made the structure of SYNTAXTRAIN more clear. The entire implementation of SYNTAXTRAIN can be seen in appendix E.

The kernel consists of the following classes:

- **GrammarBase**: Reads/writes source code and loads grammar files.
- **GrammarCompiler**: Creates syntax nodes used by CLAPHAM, modelled to match the grammar and colored depending on the stack trace.
- **SourceCodeCompiler**: Verifies the source code using ANTLR parser and lexer files.
- **GrammarInterface**: Interface to the kernel, to ensure that calls are made in right order.
- **Variables**: Various global variables and constants used by the kernel.

The GUI consists of the following classes (graphical components all starts with a lower-case *g*, with the exception of MainScreen):

- **Controller**: Handles all gui events, except those of the Syntax components pane.
- **Dialogs**: Shows about and help dialogs.
- **gErrorTrace**: The top frame that shows a list of rules involved in the error.
- **gGrammarDiagram**: Panel containing the diagrams generated by CLAPHAM.
- **gGrammarOptions**: Panel allowing the student to select which diagrams are shown.
- **gGrammarPanel**: Panel containing both **gGrammarDiagram** and **gGrammarOptions**.
- **gSourceCode**: Text pane for showing and highlighting the source code.
- **gToolbar**: Tool bar at the top (also registers shortcuts).
• **MainScreen**: The main frame.

• **Variables**: Various global variables and constants used by the gui.

Besides from these classes, a few helper classes are also used. These are XmlNode (reads XML), Lock (concurrent lock) and StdLibrary (reads files and escapes strings).

The grammar file contains a parser and a lexer, which are given the source code and returns a stack trace. The grammar file also contains an XML file that is used to draw the syntax diagrams for the language. The stack trace is used to color the relevant diagrams.

During the implementation process a critical race condition was found, which made the application crash if the diagrams were updated while they were being painted. To understand an correct this problem [1] was used.

### 6.1 Tests

Before SyntaxTrain was created, some initial tests were performed on ANTLR and Clapham to make sure the necessary stack trace could be generated correctly and the diagrams looked good and could easily be generated from a BNF. Generating a stack trace appeared to be more troublesome than I first thought, due to the limitations of ANTLR, but it was possible by using a global stack and having every rule adding themselves to it along with the rules that were matched correctly inside them.

SyntaxTrain has not been tested with unit tests, since it mainly consists of the gui. Instead various Java code samples have been used for testing, to verify the stack trace generated by the parser and the diagrams are shown and colored correctly along with making sure the source code was highlighted correctly. The Java samples tests various problems like an error in the start and end of the code or the problem happening inside a loop, an optional node, a sequence and an option. The code examples can be seen in appendix D. The focus of these tests have been to make sure the tool works as it should, not that it accepts the entire Java grammar.

During these tests it was discovered that the source code were not always highlighted correctly, the highlighting were sometimes moved a couple of characters. The problem was because the string object counts carriage return \r as a character but the JTextPane highlight operation does not. Another problem was
also found with the coloring in Clapham. Suddenly a loop would not be col-
ored or an option inside a loop. This was because the coloring modifications
were not implemented in all the drawing functions, which has since been tested
extensively to make sure everything is colored correctly.

There was also a bug in the original Clapham. If an optional node was written
but it only contained one node, then that node would also be added after the
option (required), but if two nodes were inside the optional node instead, then
they would not be added afterwards.
In this chapter the limitations of SyntaxTrain will be described along with an explanation of the possible improvements that could be made. SyntaxTrain is still a new program and hence there are lots of thing that can be improved. This chapter will not discuss the limitations and improvements of the Java grammar that was created but more general about BNF compiler and SyntaxTrain.

The limitations of SyntaxTrain are:

1. The BNF may not be left recursive.
2. The BNF must be decidable.
3. Errors reported by the BNF compiler refers to wrong line numbers.
4. The syntax diagrams are images, which makes it hard to make something happen when the student clicks/hoovers them and hence nothing happens.

The first problem will most likely not be solved, since it is part of the requirements for ANTLR. It might be solved by choosing a different parser, but then the parser would have to be made all over. The second problem is quite troublesome because it makes it harder to make a BNF that are not too bloated, which is essential in order to get syntax diagrams that are easy to use, also
lots of languages have undecidable elements, the most common being the `if` statement. The third problem are an inconvenience, but a quite big one, it means that the BNF writer needs to read the message instead and extrapolate what the problem was (however these messages are quite detailed so it is not too hard). Finally because diagrams are images and not Java objects, it is not possible to click on them. If this was possible then a lot of help could be added such as showing the diagram for that component or explaining what the part means. The three last problems should be possible to fix, but would require a lot of time.

There are lots of possible improvements to SyntaxTrain and for this reason they have been split up into three categories; (i) Important improvements, that makes it hard or impossible to write languages; (ii) Normal improvements, which are very helpful but are not required; (iii) Convenience improvements, stuff that would be nice to have.

The important improvements are:

- Make it possible to write new tokens in the BNF.
- Accept an undecidable BNF.

Right now it is not possible to write your own tokens in the BNF, meaning complex terminals such as the INTEGER, which accepts any integer value and not just a specific one. A list of the most common tokens are already made such that they can be used inside the BNF, however these may not be good enough for all languages, which is why this is important. As explained in the limitations part, a lot of languages are not completely decidable, which is why it is important to support these languages. This should not be impossible, since ANTLR currently only reports warnings, but it would have to be tested carefully.

Improvements with normal priority:

- Show correct line numbers for errors reported by the BNF compiler.
- Graphically show why the grammar is left recursive or undecidable.
- Use highlighting tools to highlight the source code.
- Undo and redo support.

The errors reported by the BNF compiler refers to wrong line numbers, fortunately the errors are usually explained quite well (if you understand what it
says) so it is not crucial, but is still a bit problem.
It can sometimes be quite hard to understand why a grammar is left recursive
or undecidable, because it may involve five or more rules. Therefore it would be
very useful with a graphical representation of the problem. The ANTLR IDE
is able to show this using graphs, so it should not be too difficult if this can be
used, otherwise it might be a project in itself.
Normal syntax highlighting would be a big help for novices who already have
problems with reading code.
Undo and redo sounds like being a convenience problem, but if you by accident
delete something that should be there while not having saved for ten minutes,
it would be extremely troublesome, so this should absolutely be implemented.

There are numerous convenience problems, but since their not very important
I will not go into details but just list them here:

• Graphical interface to the BNF compiler.
• Comments in the BNF.
• Comments to rules which can be seen in SYNTAXTRAIN, such that the
author can add help for known problems.
• Implement SYNTAXTRAIN into one or more IDE for convenience.
• Make terminals drawn by CLAPHAM completely round.
• Remember the last opened directory in the last session.
• When a component inside a syntax diagram is clicked, go to that diagram.
• When a component inside a syntax diagram is hoovered, highlight the
associated source code.
• Let the student choose the color and font for the source code and compo-
nents in the syntax diagrams.
• Drop files into SYNTAXTRAIN (drag and drop).
• Make a BNF file from a grammar file, such that it can be updated in case
the grammar file becomes obsolete.
• Automatically check if the file has been edited on the disk and ask whether
to re-open it if it is.
• Grammars knows which file extensions they can verify.
  – Filter files in open dialog.
– Load multiple grammar files as long as their extensions does not clash.
– Verify entire directories, using the grammar file matching each file.

These improvements would make SYNTAXTRAIN easier to use, but are hardly any requirement for it to work.

Even though so much can be optimized, most of the optimizations are either normal or low priority, and the important optimizations can be avoided/accepted in most cases.
The best way to understand how SyntaxTrain works is with illustrations. In this chapter, three examples will be shown with different syntax errors a student might make. Afterwards SyntaxTrain will be compared to compiler messages, which are arguably the most commonly used tool for helping students understanding their errors.

8.1 Example 1

Consider the Java code in Figure 8.1 which computes the number of days in a month (ignoring leap years). Figure 8.2 shows a screen shot of SyntaxTrain after opening this file.

From the highlighted source code it is obvious that there is a problem with the code that follows the keyword case. The right pane shows that the error occurred when parsing expressionMain as part of Expression. The yellow elements in the first diagram show what the parser was expecting, but there does not seem to be a problem with any expression, so we look down to the syntax diagram for the switch_statement, shown magnified in Figure 8.3.
class Dates {
    int dayInMonth(int month) {
        switch (month) {
            case 1: case 3: case 5: case 7: case 8:
                case 10: case 12:
                    return 31;
            case 4: case 6: case 9: case 11:
                return 30;
            case 2
                return 28;
            default:
                return 0;
        }
    }
}

Figure 8.1: Example 1.

Figure 8.2: The Date class syntax error displayed by SyntaxTrain.
8.2 Example 2

Trace through this diagram; every element is blue until the element after the keyword case is reached. Clearly, the parser was expecting an expression, but the colon was written before the expression (the integer literal 2) instead of after it. By modifying the source code and revalidating the error is gone!

8.2 Example 2

Look at the code in Figure 8.4 which writes a string to a file.

```java
1. class FileWriterHelper {
2.     import java.io.*;
3.     
4.     public void writeTextToFile( String text, File file ) {
5.         try {
6.             BufferedWriter out = new BufferedWriter(new FileWriter(file));
7.             out.write(text);
8.             out.flush();
9.         }
10.        catch (IOException e) {
11.            System.out.println("Text was not written to file!");
12.        }
13.    }
14. }
```

Figure 8.4: Example 2.
Figure 8.5 shows a screen shot of the code opened in SyntaxTrain. The left pane shows that the error occurred while parsing the import statement.

Figure 8.5: FileWriterHelper opened in SyntaxTrain.

In the right pane the top diagram is class_declaration (Figure 8.6). This diagram shows that only \} and field_declaration are legal to write at this point. The former is not relevant since the parsing has not reached the end of the class. However field_declaration must be either a method, a constructor, a variable declaration or a static initializer. If you do not remember this, select Show/hide syntax components (F10) and click the field_declaration component. It follows that import is illegal at this point. Looking down through the diagrams, we see that compilation_unit can accept an import_statement, but before the type_declaration, which is the class. Moving the import statement before the class declaration solves the problem.
8.3 Example 3

Figure 8.7 shows a fragment of a calculator class, including a method for the operation square root. The method stores the result in the variable sum. This code contains no syntax errors.

```java
class Calculator {
    int sum;

    public void sqrt() {
        int s = 2;
        while ((s)*(s) <= sum) {
            s = s + 1;
        }
        sum = s - 1;
    }

    public Calculator(int initialValue) {
        sum = initialValue;
    }
}
```

Figure 8.7: Example 3.

Suppose the student forgot the semicolon at the end of line 7. Figure 8.8 shows a screen shot of SYNTAXTRAIN; it is clear that the semicolon is missing, since expression was matched correctly and semicolon is now the only legal option.

Suppose the student instead forgets to write the left brace { for the while statement at line 6. Figure 8.9 shows this error displayed in SYNTAXTRAIN. From the highlighted source code in the left pane, it is seen that the error occurred after the assignment to the variable sum. The right pane shows that the error occurred while parsing field_declaration, which means that the compiler expects a method, constructor, variable declaration or static initializer at this point. What the student intended to write at that point is neither of those, but an expression. Therefore we go on to the next diagram, in order to find something familiar and then backtrack. The next diagram is class_declaration, see Figure 8.10. This is where the Calculator class is defined. Tracing through this diagram we see that the error occurs while inside the brace and parsing field_declaration, which we already knew. Since the error occurred at the level of the field_declaration, and not inside the method, that means that for some reason we are outside the function. A careful inspection of the code will show that the brace that should have closed the while statement actually
closed the function instead.

8.4  Comparison: Syntax diagrams vs compiler messages

For SyntaxTrain to be usable, is would have to be better to explain syntax errors than compiler messages, otherwise these would just be used instead. To see if this is the case, SyntaxTrain is compared to compiler messages, based on the three examples that were just shown. The compiler messages chosen are those created by Eclipse, since it is a widely used IDE for writing Java code.

8.4.1  Example 1

The compiler message for the Date example, that was shown in Figure 8.1 are:

- Syntax error on tokens, delete these tokens.
8.4 Comparison: Syntax diagrams vs compiler messages

Figure 8.9: Screenshot of a syntax error where the \{ is missing.

Figure 8.10: The class declaration statement shown by SYNTAXTRAIN.
The error spans the entire case line, indicating the entire line should be removed. This message is not helping the student, but is in fact counter productive since it wants him/her to remove the case they intended to write. The student will most likely not remove the line because he/she knows that the case is supposed to be there, which is correct. The student can of course also just continue and go back to the problem later, but they seldom do that (speaking from my own experience), and are therefore stuck for several minutes, until some break through happens. What is even worse is if the error occurred in the very first case the student made. if this happened the student would get the error "Syntax error on tokens, SwitchLabels expected instead" which uses technical terminology, making it extremely difficult for a novice to understand.

In the example SyntaxTrain was quite effective to explain this error, so it is clear that here it is much better than compiler messages.

8.4.2 Example 2

If the FileWriterHelper example is written in Eclipse, then the following error messages will be shown:

1. Line 24: Syntax error on token ”}”", delete this token.
2. Line 6: Syntax error on token ”File”, = expected after this token.
3. Line 6: Syntax error on token ”String”, = expected after this token.
4. Line 6: Syntax error on token ”void”, @ expected.
5. Line 4: Syntax error on token(s), misplaced construct(s).
6. Line 6: Syntax error, insert ”enum Identifier” to complete EnumHeader.

Let us go through the error messages one at a time. The first message indicates that the last curly parenthesis should be removed, even though we know it should be there. This will only confuse the student, but not lead to any solution, so let us move on. The second and third error are similar, the compiler thinks these are variables and has to be assigned, hence the expected equal sign, no help here either. The fourth message says that a @ should be placed instead of the void, I am not exactly sure why and how this should help, so moving on. The fifth error states that the class is a misplaced construct. This error hits the nail on the head, if the student is able to understand it. The class is misplaced because there is an import statement after it, but this requires the
student knows the import statement should be placed at the top, which he/she does not know since it was put inside the class in the first place, so this will most likely not help. Finally the sixth error message indicates that the compiler thinks this is an enum, which it is not, this message does not help either.

In this example SyntaxTrain does not explain the problem extremely well, but if the student sees the import statement in the last diagram it should not be hard to correct, while the error message most likely will not help the student at all. So here SyntaxTrain is best, since it does explain the problem while the compiler messages require the student knows where the import statement should be placed which is the problem itself!

8.4.3 Example 3

This example consists of two parts, one where the semicolon is missing and one where the curly parenthesis \{ after \texttt{while} is missing. The source code can be seen back in Figure 8.7. In the case with the missing semicolon, the compiler message is:

- Syntax error, insert ",;" to complete BlockStatement.

The error message points to the line which the semicolon was removed from. This makes it quite easy for the student to fix even though they have no idea what "BlockStatement" is referring to. This message is even easier to understand than SyntaxTrain, since SyntaxTrain points to the first token on the next line, as it is here the error occurs. So here compiler messages are best, but SyntaxTrain are not bad either.

In the second case, where the curly parenthesis \{ is missing, the compiler reports the following two error messages:

1. Line 9: Syntax error on token "sum", VariableDeclaratorId expected after this token.

2. Line 10: Syntax error on token "\}" delete this token.

Neither of these messages helps the student. The first message requires the student understands what VariableDeclaratorId means, which is highly unlikely. But in case they do, VariableDeclaratorId refers to either a variable type or variable name (I am not sure which), meaning the compiler thinks it is a variable
declaration. If the person reading this error message is very skilled, he/she might try to understand why this error message was reported and not another, but a student will most certainly not and would just be confused why the compiler wishes to make a variable declaration here.

The second error tells the student to remove the curly parenthesis } at line 10. If the student does this, then that error will disappear, but the other will still remain, so no help there. The student will most likely wonder why the compiler wishes to remove the curly parenthesis }, and might click on the right side of it. In this case the parenthesis for the class will be slightly marked, and if the student sees this, he/she might understand what the problem is, but if they do not do this then they will probably stare at the code for a long time without becoming any wiser. The SYNTAXTRAIN diagrams are not optimal for this error either, but it does show that the error occurs outside the function and once the student sees that, it should be visible what the problem is.
Chapter 9

Conclusion

The goal of this project is to make a tool that is able to read a students source code and explain the syntax errors using syntax diagrams, such that the student understands what the problem is and how to solve it. This was done by highlighting the source code that could not be matched and drawing syntax diagrams to show the construct that failed. In each syntax diagram the components are colored depending on what were matched, not matched, failed while trying to match and what is possible to write at the given position highlighted in the code.

The examples in this report shows that SyntaxTrain is able to help the student understand the error in their code and how to fix it. Some errors are not displayed very well by SyntaxTrain, but it is still possible to understand them without prior knowledge of the language syntax.

It is still too early to tell if SyntaxTrain will actually be able to help students with understanding their errors since no field studies or similar has been made. Instead the chosen examples are ones that a student might make (some were even found on the internet, posted by students) and in these examples it does look promising.

When comparing SyntaxTrain to compiler messages, it is clear that some compiler messages are almost complete rubbish to a novice, where Syntax-
Conclusion

Train is able to explain the problem in a more understandable way. There are also examples where compiler messages are better to explain the problem than SyntaxTrain is. This means that SyntaxTrain is probably not going to be used instead of compiler messages, but as a supplement.

A BNF compiler was also made, that is able to take a BNF that accepts a language and make a grammar file from it, which can be used by SyntaxTrain to validate source code for another language. This means that SyntaxTrain is not limited to one language only. The downside of the tool is that it must be run as an external tool instead of a part of the IDE, which makes it more inconvenient.

All in all, I believe SyntaxTrain has a future as an educational tool for helping students to correct errors in their code faster, such that they can focus on the code instead of the errors.
Appendix A

BNF Evaluator

1 grammar BnfEvaluator;
2
3 options
4 {
5     output=AST;
6     ASTLabelType=CommonTree;
7 }
8
9 @header
10 {
11 package BnfCompiler;
12
13 import java.util.HashMap;
14 }
15
16 @lexer::header
17 {
18 package BnfCompiler;
19 }
20
21 @members
22 {
23     public String startRule = " ";
HashMap<String, Link> ruleNameToLink = new
    HashMap<String, Link>();

public void displayRecognitionError(String[]
  tokenNames, RecognitionException e) throws
    RuntimeException
{
    throw new RuntimeException();
}

bnf : startRule '.' (rule '.')* EOF;

startRule
  : ruleName '=' expression
  {startRule = $ruleName.value; ruleNameToLink.
    put($ruleName.value, $expression.value);};

rule : ruleName '=' expression
  {ruleNameToLink.put($ruleName.value,
    $expression.value);};

ruleName returns [String value]
  : NONTERMINAL {$value=$NONTERMINAL.text;};

expression returns [Link value]
  : ( expr {link.add($expr.value);} )+

expr returns [Link value]
  : (Link orLink = null;)
  start=exprBase {Link link = $start.value;
    $value = link;}
  ( '/ ' base=exprBase
  { if(orLink == null)
    { orLink = new Link(link);
    orLink.oneOfMultiple();
    $value = orLink;
    } orLink.add($base.value);
  })*;

exprBase returns [Link value]
TERMINAL {
$value = new Link($TERMINAL.text);
};

| NONTERMINAL {
$value = new Link($NONTERMINAL.text);
};
| (' expression ') {
$value = $expression.value;
};
| [' expression '] {
$value = $expression.value; $value.optional();
};
| '<' expression '>' {
$value = $expression.value; $value.loop();
};

TERMINAL :
" " (~ '" ') * '" ';

NONTERMINAL : ( 'a'..'z'| 'A'..'Z'| '0'..'9'| '_') +;

WS :
( " " |
'\t' |
'\r' |
'\n'
) {$channel=HIDDEN;}

;
Appendix B

Java parser test

```
1 grammar Java;
2
3 options
4 {
5     output=AST;
6     ASTLabelType=CommonTree;
7 }
8
9 @header
10 {
11    package Test1;
12
13    import java.util.Stack;
14 }
15
16 @members
17 {
18    public Stack<Stack<String>> trace = new Stack<
19                  Stack<String>>();
20    public void displayRecognitionError(String[]
21                  tokenNames, RecognitionException e) throws
22                  RuntimeException
23    {
24```
throw new RuntimeException();
}

compilationUnit
: {Stack<String> tr = new Stack<String>();
   trace.push(tr); tr.push("compilationUnit");}
   (typeDeclaration )*
{trace.pop();}
;

typeDeclaration
: {Stack<String> tr = new Stack<String>(); trace .push(tr); tr.push("typeDeclaration");}
classDeclaration {trace.pop();};

classDeclaration
: {Stack<String> tr = new Stack<String>(); trace .push(tr); tr.push("classDeclaration");}
  'class' {tr.push("'class'");}
  IDENTIFIER {tr.push("IDENTIFIER");}
  '{' {tr.push("'{'");}
  (fieldDeclaration {tr.push("fieldDeclaration");}
   )*
  '}
{trace.pop();};

fieldDeclaration
: {Stack<String> tr = new Stack<String>(); trace .push(tr); tr.push("fieldDeclaration");}
  methodDeclaration
{trace.pop();};

methodDeclaration
: {Stack<String> tr = new Stack<String>(); trace .push(tr); tr.push("methodDeclaration");}
  type {tr.push("type");}
  IDENTIFIER {tr.push("IDENTIFIER");}
  '(' {tr.push("'(");}
  ')' {tr.push("')'");}
  ';' {tr.push("'\n'");}
{trace.pop();};
type : {Stack<String> tr = new Stack<String>();
    trace.push(tr); tr.push("type");}
    typeSpecifier {tr.push("typeSpecifier");
    ( [ ] )* {trace.pop();};
    typeSpecifier : {Stack<String> tr = new Stack<String>(); trace.
    .push(tr); tr.push("typeSpecifier");} ( 'boolean'
    | 'byte'
    | 'char'
    | 'short'
    | 'int'
    | 'float'
    | 'long'
    | 'double'
    | packageName ) {trace.pop();};
    packageName : {Stack<String> tr = new Stack<String>(); trace.
    .push(tr); tr.push("packageName");}
    IDENTIFIER {tr.push("IDENTIFIER");
    '.' )* {tr.push("'.'");}
    IDENTIFIER
    {trace.pop();};
    IDENTIFIER : ( 'a'..'z'| 'A'..'Z'| '_' ) ( 'a'..'z'| 'A'
    '..Z'| '0'..'9'| '_' )* ;
    WS : ( ,
    | \t,
    | \r,
    | \n,
    ) {$channel=HIDDEN;}
Appendix C

Java grammar BNF

```
1 compilation_unit =
2   [ package_statement ]
3   < import_statement >
4   < type_declaration > .
5
6 package_statement =
7   "package" package_name ";" .
8
9 import_statement =
10  "import" ( ( package_name "." "*" ";" )
11    / ( class_name ) ) ";" .
12
13 type_declaration =
14  class_declaration / interface_declaration .
15
16 class_declaration =
17  < modifier > "class" IDENTIFIER
18    [ "extends" class_name ]
19    [ "implements" interface_name < ",",
20     interface_name > ]
21  "{" < field_declaration > "}" .
22
23 interface_declaration =
```
< modifier > "interface" IDENTIFIER
[ "extends" interface_name < "," interface_name > ]
"{" < field_declaration > "}"

field_declaration =
( method_declaration
 / constructor_declaration
 / variable_declaration )
 / ";" .

method_declaration =
< modifier > type IDENTIFIER
"(" [ parameter_list ] ")" < "[" "]" >
( statement_block / ";" ) .

constructor_declaration =
< modifier > IDENTIFIER "(" [ parameter_list ] ")"

statement_block .

statement_block = "{" < statement > "}"

variable_declaration =
< modifier > type variable_declarator
< "," variable_declarator > ";" .

variable_declarator =
IDENTIFIER < "[" "]" > [ "="
 variable_initializer ] .

variable_initializer =
expression
/ ( "{" [ variable_initializer
 < "," variable_initializer > [ "," ] ] "}"
 ) .

static_initializer =
"static" statement_block .

parameter_list =
parameter < "," parameter > .
parameter  =
type IDENTIFIER < "[" "]" >  .

statement  =
variable_declaration
/ ( expression ";;" )
/ ( statement_block )
/ ( if_statement )
/ ( do_statement )
/ ( while_statement )
/ ( for_statement )
/ ( try_statement )
/ ( switch_statement )
/ ( "synchronized" "(" expression ")" statement )
/ ( "return" [ expression ] ";;" )
/ ( "throw" expression ";;" )
/ ( "break" [ IDENTIFIER ] ";;" )
/ ( "continue" [ IDENTIFIER ] ";;" )
/ ( ";;" )  .

if_statement  =
"if" "(" expression ")" statement_block
[ "else" statement ]  .
do_statement  =
"do" statement_block "while" "(" expression ")" ";;"  .

while_statement  =
"while" "(" expression ")" statement  .

for_statement  =
"for" "(" ( variable_declaration / ( expression ";;" ) / ";;" )
[ expression ] ";;"
[ expression ]
")" statement  .

try_statement  =
"try" statement_block
< "catch" "(" parameter ")" statement_block >
[ "finally" statement_block ]  .
```plaintext
switch_statement =
  "switch" "(" expression ")" 
  "{" < ( "case" expression ":" ) / ( "default" ":" ) / statement > 
  "}" .

expression =
  [expressionPrefix]
  expressionMain
  < "." expressionMain >
  [expressionPost].

expressionMain =
  "null"
  / "this"
  / ( "super" [ "(" [ arglist ] ")" ] )
  / ( IDENTIFIER [ "(" [ arglist ] ")" ] < "[" expression "]" ] > )
  / ( "(" expression ")" )
  / "true"
  / "false"
  / integer_literal
  / FLOAT
  / STRING
  / ( "new" ( class_name "(" [arglist] ")" )
   / ( type_specifier "[" [ expression ] "]" ] < 
   "[" "]" ] > )
   / ( "(" expression ")" )
  ).

expressionPrefix =
  "-"
  / "++"
  / "--"
  / "!!"
  / "~" .

expressionPost =
  "++"
  / "--"
```
arglist =
expression < "," expression > .
type =
type_specifier < "[ [ ] ]" > .
type_specifier =
"boolean"
/ "byte"
/ "char"
Java grammar BNF

```
/ "short"
/ "int"
/ "float"
/ "long"
/ "double"
/ classOrInterface_name .

modifier =
"public"
/ "private"
/ "protected"
/ "static"
/ "final"
/ "native"
/ "synchronized"
/ "abstract"
/ "threads"afe"
/ "transient" .

package_name = <IDENTIFIER "." > IDENTIFIER .
class_name = <IDENTIFIER "." > IDENTIFIER .
interface_name = <IDENTIFIER "." > IDENTIFIER .
classOrInterface_name = <IDENTIFIER "." > IDENTIFIER .

integerLiteral =
( INT [ "l" ] )
/ ( "0x" HEX <HEX> ).
```
Appendix D

Java tests

D.1 Test 1

```java
class MyClass {
    String longString = "This is first half of a long string ;
    void f() {
        System.out.println(longString);
    }
}
```

D.2 Test 2

```java
class MyClass {
    void f() {
        if (i > j) // Error, extra parenthesis
            max = i;
    }
}
```
D.3 Test 3

```java
1 class MyClass {
2     void f() {
3         if (i > j) {
4             max = 1;
5         }
6     }
```}

D.4 Test 4

```java
1 cclass MyClass {
2     public MyClass() {
3     }
4 }
5 }
```}

D.5 Test 5

```java
1 class MyClass {
2     public MyClass() {
3     }
4 }
5 }
```}

D.6 Test 6

```java
1 class MyClass {
2     public MyClass(int initialValue) {
3         int myVal = ;
4     }
5 }
```}
D.7 Test 7

```java
class MyClass {
    public static boolean testFunc(int value) {
        if (value == 2) {
            
        } else {
            for (int i = 0; i < 5; i++) {
                
            }
        }
    }
}
```

D.8 Test 8

```java
class MyClass {
    public static boolean testFunc(int value) {
        while (true) {
            someOtherClass.dosomething();
        }
    }
}
```

D.9 Test 9

```java
class MyClass {
    void f() {
        int n = 10;
        // Error, closing brace is missing
        void g() {
            int m = 20;
        }
    }
}
```
D.10 Test 10

```java
class MyClass {
    void f() {
        if (i > j) // Error, unbalanced parentheses
            max = i // Error, missing semicolon
        }
    else
        max = j;
    }
}
```
E.1 BnfCompiler

E.1.1 CommandLineTool.java

```java
package BnfCompiler;

import java.io.BufferedWriter;
import java.io.ByteArrayInputStream;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStream;
import java.io.UnsupportedEncodingException;
import org.antlr.Tool;
import org.antlr.runtime.ANTLRInputStream;
import org.antlr.runtime.CommonTokenStream;
import org.antlr.runtime.RecognitionException;
import org.antlr.tool.ErrorManager;
```
import Xml.XmlNode;
import BnfCompiler.Link.RepeatType;
import Exceptions.XMLLoadException;
import Exceptions.XMLAttributeDoesNotExist;
import Exceptions.XMLNodeDoesNotExist;
import Kernel.Variables;
import Library.StdLibrary;

public class CommandLineTool
{
    private static String jdkBinFolder = "";
    private static final String optionsXmlFile = "options.xml";

    public static void printUsage()
    {
        System.out.println("Usage: java -jar BnfCompiler.jar BnfFile");
        System.out.println();
        System.out.println("BnfFile:	Reference to the file containing the language to be compiled, written in BNF.");
    }

    public static void main(String[] args)
    {
        if( args.length != 1 )
        {
            printUsage();
            System.exit(0);
        }
        readOptions();
        String filename = args[0];
        String filenamePre = filename.substring(0, filename.length() - 4);
        integrityCheck(filename);
        cleanup(filenamePre);
        String Bnf = StdLibrary.readFileAsString(filename);
        if( Bnf == null )
        {
System.out.println("There was a problem reading file "+ filename + ".");
System.exit(0);
}

try
{
    System.out.println("Parsing bnf");
    InputStream is = new ByteArrayInputStream(Bnf.getBytes("UTF-8"));
    ANTLRInputStream input = new ANTLRInputStream(is);
    BnfEvaluatorLexer lexer = new BnfEvaluatorLexer(input);
    CommonTokenStream tokens = new CommonTokenStream(lexer);
    BnfEvaluatorParser parser = new BnfEvaluatorParser(tokens);

    try
    {
        parser.bnf();
        if( !(
            createAntlrFiles(filenamePre, parser.startRule, parser.ruleNameToLink) &
            compileAntlrFiles("Grammar\" + filenamePre) &
            createSyntaxXml("Grammar\" + filenamePre, parser.ruleNameToLink) &
            createJarFile(filenamePre, "Grammar\"")
        ) )
        {
            System.out.println("Failed to create grammar file!");
            return;
        }
    }
    cleanup(filenamePre);
}
File jarFile = new File(filenamePre + ".jar");
System.out.println("Jar file created at: "+ jarFile.getCanonicalPath());
    return;
}
catch (RecognitionException e)
{
}
catch (RuntimeException e)
{
    //TODO: the bnf wasn’t written correctly
    System.out.println("Error creating jar file!");
}
}
catch (UnsupportedEncodingException e)
{
    System.out.println("Unknown Error!");
}
catch (IOException e)
{
    System.out.println("Unknown Error!");
}
System.exit(1);

private static void readOptions()
{
    String optionsXml = StdLibrary.readFileAsString(optionsXmlFile);
    if( optionsXml == null )
    {
        System.out.println("There was a problem reading file "+ optionsXmlFile + ".");
        System.exit(0);
    }
    try
    {
        XmlNode compilerOptions = new XmlNode(optionsXml, "1.0");
        jdkBinFolder = compilerOptions.getChildNode("BnfCompiler").getChildNode("path")
            .getAttribute("jdkBinFolder") + \\
            ;
return;
}
catch (XMLLoadException e)
{
    System.out.println( optionsXmlFile + " contains invalid xml.");
}
catch (XMLAttributeDoesNotExist e)
{
    System.out.println("Invalid xml structure in " + optionsXmlFile);
}
catch (XMLNodeDoesNotExist e)
{
    System.out.println("Invalid xml structure in " + optionsXmlFile);
}
System.exit(1);
}

private static boolean createJarFile(String filenamePre, String folder)
{
    try
    {
        System.out.println("Creating jar file...");
        Process jarProcess = Runtime.getRuntime().exec( jdkBinFolder + "jar.exe cf " + filenamePre + ".jar " + folder );
        Long start = System.currentTimeMillis();
        while( true )
        {
            Thread.sleep(500);
            try
            {
                int exitCode1 = jarProcess.exitValue();
                if( exitCode1 != 0 )
                {
                    System.out.println("Error, jar file not created.");
                    return false;
                }
            }
        }
    }
}
break;
}
catch (IllegalThreadStateException e)
{
    if ( System.currentTimeMillis() -
            start > 30000 )
    {
        //over 30 sec passed
        System.out.println("Jar file
            creation took too long.");
        jarProcess.destroy();
        return false;
    }
}
return true;
}
catch (IOException e)
{
    // TODO Auto-generated catch block
    e.printStackTrace();
}
catch (InterruptedException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
return false;
}
private static void cleanup( String filenamePre )
{
    File directory = new File("Grammar\\");
    File[] files = directory.listFiles();
    for( File file : files )
    {
        String name = file.getName();
        if( name.equalsIgnoreCase("bnfparser.class") ||
            name.equalsIgnoreCase(filenamePre + ".jar") ||
            file.isDirectory() )
        {
            continue;
        }
        file.delete();
private static void integrityCheck(String filename)
{
    //TODO: checks that grammar folder exists (otherwise it creates it), that given jdk bin folder exists and contains javac and jar.
    //also verifies that antlr.jar and grammar/BnfParser.class exists.
    File antlr = new File("antlr.jar");
    File bnfparser = new File("Grammar\bnfparser.class");
    File javac = new File(jdkBinFolder + "javac.exe");
    File jar = new File(jdkBinFolder + "jar.exe");
    File bnfFile = new File(filename);

    if( !antlr.exists() )
    {
        System.out.println("Missing file: antlr.jar");
        System.exit(1);
    }
    if( !bnfparser.exists() )
    {
        System.out.println("Missing file Grammar\bnfparser.class");
        System.exit(1);
    }
    if( !javac.exists() )
    {
        System.out.println("Missing or wrongly set up java compilation file: " + jdkBinFolder + "javac.exe");
        System.exit(1);
    }
    if( !jar.exists() )
    {
        System.out.println("Missing or wrongly set up jar compilation file: " + jdkBinFolder + "jar.exe");
        System.exit(1);
    }
if( !bnfFile.exists() )
{
    System.out.println("Missing file: " + filename);
    System.exit(1);
}

private static boolean compileAntlrFiles( String filenamePre )
{
    try
    {
        System.out.println("Compiling...");
        Process compilation1 = Runtime.getRuntime().
                .exec( jdkBinFolder + "javac.exe -cp antlr.jar;. " + filenamePre + "Lexer. java" );
        Process compilation2 = Runtime.getRuntime().
                .exec( jdkBinFolder + "javac.exe -cp antlr.jar;. " + filenamePre + "Parser. java" );
        Long start = System.currentTimeMillis();
        while( true )
        {
            Thread.sleep(500);
            try
            {
                int exitCode1 = compilation1.exitValue();
                int exitCode2 = compilation2.exitValue();
                if( exitCode1 != 0 || exitCode2 != 0 )
                {
                    System.out.println("Failed to compile code.");
                    return false;
                }
                break;
            }
            catch (IllegalThreadStateException e)
            {
            }
        }
    }
if (System.currentTimeMillis() - start > 30000 ) {
    // over 30 sec passed
    System.out.println("Compilation error, took too long time to compile.");
    compilation1.destroy();
    compilation2.destroy();
    return false;
}

private static boolean createSyntaxXml( String filenamePre, HashMap<String, Link> ruleNameToLink ) {
    System.out.println("Creating syntax xml file");
    String filename = filenamePre + ".xml";
    File file = new File(filename);
    if(file.exists()) {
        // just to be sure
        file.delete();
    }
    try {
        BufferedWriter out = new BufferedWriter(new FileWriter(file));
        
    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    } catch (InterruptedException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
    return false;
}
out.write("<SyntaxTrain version=" + Variables.xmlVersion + ">
");  
for( String ruleName : ruleNameToLink.keySet() )  
{  
    Link link = ruleNameToLink.get(ruleName);  
    out.write("\t<Rule ID=" + ruleName + "">
");  
    writeXmlRule(link , out , 2);  
    out.write("\t</Rule>\n\n");  
    }  
out.write("</SyntaxTrain>\n");  
out.close();  
return true;  
catch (IOException e)  
{  
    System.out.println("Could not write to xml file!!!!");  
}  
return false;  
}  
private static void writeTabs(int tabs ,  
BufferedWriter out) throws IOException  
{  
for(int i=0;i<tabs;i++)  
{  
    out.write("\t");  
}  
}  
private static void writeXmlRule( Link rule ,  
BufferedWriter out , int tabs) throws IOException  
{  
    int numTabs = tabs;  
    RepeatType repeating = rule.getRepeat();  
    String id = rule.getId();  
    String UUID = rule.getUUID();  
    Stack<Link> ids = rule.getIds();  
    if( id != null )  
    {  
        writeTabs(tabs , out);  
    }  
}
out.write("<Rule ID=" + StdLibrary.xmlEscapeString( id ) + " UUID=" + UUID + "/>");
}
else
{
    if( repeating == RepeatType.oneOfMultiple )
    {
        //special case
        writeTabs(tabs, out);
        out.write("<or>\n");

        for( Link subRule : ids )
        {
            writeTabs(tabs+1, out);
            out.write("<option>\n");
            writeXmlRule(subRule, out, numTabs+1) ;
            writeTabs(tabs+1, out);
            out.write("</option>\n");
        }
        out.write("</or>\n");
    }
    else
    {
        switch( repeating )
        {
            case atleastOnce:
                writeTabs(tabs, out);
                out.write( "<OnePlus>\n" );
                break;
            case loop:
                writeTabs(tabs, out);
                out.write( "<Repeat>\n" );
                break;
            case optional:
                writeTabs(tabs, out);
                out.write( "<Optional>\n" );
                break;
            case sequence:
                //the next items stay on the same line
                numTabs--;
                break;
        }
    }
}
numTabs++;
for( Link subRule : ids )
{
    writeXmlRule(subRule, out, numTabs);
}
switch( repeating )
{
case atleastOnce:
    writeTabs(tabs, out);
    out.write( "</OnePlus>\n" );
    break;
case loop:
    writeTabs(tabs, out);
    out.write( "</Repeat>\n" );
    break;
case optional:
    writeTabs(tabs, out);
    out.write( "</Optional>\n" );
    break;
case sequence:
    break;
}
}

private static boolean createAntlrFiles( String filenamePre , String startRule , HashMap<String, Link> ruleNameToLink )
{
    System.out.println("Creating parser and lexer files");
    String filename = filenamePre + ".g";
    File file = new File(filename);
    if(file.exists())
    {
        //just to be sure
        file.delete();
    }
    try
    {

BufferedWriter out = new BufferedWriter(new FileWriter(file));
out.write("grammar " + filenamePre + ";
" +
"options
" +
"{\n" +
" superClass=BnfParser;\n" +
"}
" +
"@header\n" +
"{\n" +
"package Grammar;\n" +
"\n" +
"import java.util.Stack;\n" +
"})
" +
"\n" +
"@lexer::header\n" +
"{\n" +
"package Grammar;\n" +
"} \n" +
"\n" +
"@parser::members\n" +
"{\n" +
" public void bnf() throws RecognitionException
" +
"{\n" +
" + startRule + "();\n" +
" }
" +
"@lexer::members \n" +
" public Token nextToken() {\n" +
" while (true) {\n" +
" state.token = null;\n" +
" state.channel = Token. DEFAULT_CHANNEL;\n" +
" state.tokenStartCharIndex = input.index();\n" +
" state.
" tokenStartCharPositionInLine = input.getCharPositionInLine();\n" +
" state.tokenStartLine = input.
" getLine();\n" +
" state."
```java
state.text = null;
if ( input.LA(1) == CharStream.EOF ) {
    return Token.EOF_TOKEN;
} 
try {
    mTokens();
    if ( state.token == null ) {
       emit();
    } else if ( state.token == Token.SKIP_TOKEN ) {
       continue;
    } 
    return state.token;
} catch (RecognitionException re) {
    emit();
    return state.token;
}
} }
```

writeBnfRule( startRule, ruleNameToLink.get (startRule), out, true );
for( String ruleName : ruleNameToLink. keySet() )
{
    if(ruleName.equalsIgnoreCase(startRule))
    {
        //rule is already written
        continue;
    }
    writeBnfRule( ruleName, ruleNameToLink. get(ruleName), out, false );
}
```

```
out.write("INT : '0 '.. '9 '+
";
" \n" +
" ");
```

```
writeBnfRule( startRule, ruleNameToLink.get (startRule), out, true );
for( String ruleName : ruleNameToLink. keySet() )
{
    if(ruleName.equalsIgnoreCase(startRule))
    {
        //rule is already written
        continue;
    }
    writeBnfRule( ruleName, ruleNameToLink. get(ruleName), out, false );
}
```

```
out.write("INT : '0 '.. '9 '+\n";
" \n" +
" ");
```
"FLOAT\n" +
"  : (\'0\'..\'9\')\'+\', (\'0\'..\'9\')*\n   \n"  | \'.\' (\'0\'..\'9\')+ EXPONENT?\n"  +
"  | (\'0\'..\'9\')+ EXPONENT\n" +
" ;\n" +
"\n" +
"IDENTIFIER : ('a'..'z'|'A'..'Z'|'_' +
  ) ('a'..'z'|'A'..'Z'|'0'..'9'|'_' )* ;\n" +
"\n" +
"HEX\n" +
"  : HEX_DIGIT+;\n" +
"\n" +
"STRING\n" +
"  : '\"' ( ESC_SEQ | 
   "\"\"\"|'\"'|'\\n') \"\n" +
" ;\n" +
"\n" +
"fragment\n" +
"EXponent : ('e'|'E') ('+'|'-')?\n  (\'0\'..\'9\')+ ;\n" +
"\n" +
"fragment\n" +
"HEX_DIGIT : (\'0\'..\'9\'|'a'..'f'|'A'..'F') ;\n" +
"\n" +
"fragment\n" +
"ESC_SEQ\n"+
"  : '\\"'( 'b'|'t'|'n'|'f'|'r 
   | '\\"'|'\\'|'\\\\'|'\\\"')\n" +
" | UNICODE_ESC\n" +
" | OCTAL_ESC\n" +
" ;\n" +
"\n" +
"fragment\n" +
"OCTAL_ESC\n"+
"  : '\\\"'( '0'..'3' ) ('0'..'7')\n   (\'0\'..'7')\n" +
" | '\\\"'( '0'..'7') ('0'..'7')\n" +
" | '\\\"'( '0'..'7')\n" +
" ;\n" +
"\n" +
"fragment\n" +
"UNICODE_ESC\n" +
" : '\\\\' 'u' HEX_DIGIT
  HEX_DIGIT HEX_DIGIT\n" +
" ;\n" +
"\n" +
"COMMENT\n" +
" : '//' ~( '\n'| '\r')* '\r'?
  '\n' {$channel=HIDDEN;}\n" +
" | '/*' ( options {greedy=false
  ;} : . )* '*/' {$channel=HIDDEN
  ;}\n" +
" ;\n" +
"\n" +
"WS : ( ', ', '\n' +
  " | '\t','\n' +
  " | '\r','\n' +
  " | '\n','\n' +
  " ) {$channel=HIDDEN;}\n" +
" ;\n"
);
out.close();
ErrorListener listener = new ErrorListener()
();
ErrorManager.setErrorListener(listener);
Tool antlrTool = new Tool();
antlrTool.addGrammarFile(filename);
antlrTool.process();

if( listener.numErrorsAndWarnings > 0 )
{
    //Error reporting has been moved to the
    //listener since it is not certain
    //that the antlr tool will stop
    //in some cases it loops forever and
    //then no error message will be
    return false;
}
//move the files
File grammarFile = new File(filename);
File tokenFile = new File(filenamePre + "tokens");
File lexerFile = new File(filenamePre + "Lexer.java");
File parserFile = new File(filenamePre + "Parser.java");

File GrammarDir = new File("Grammar\");

if( !( 
    grammarFile.renameTo(new File(
        GrammarDir, grammarFile.getName() ) ) &&
    tokenFile.renameTo(new File(
        GrammarDir, tokenFile.getName()))
    &&
    lexerFile.renameTo(new File(
        GrammarDir, lexerFile.getName()))
    &&
    parserFile.renameTo(new File(
        GrammarDir, parserFile.getName())))

){
    System.out.println("Could not rename antlr files!");
    return false;
}
return true;
}
catch (IOException e)
{
    System.out.println("Could not write to file !!!!");
}
return false;
}
private static void writeBnfRule(String ruleName, Link rule, BufferedWriter out, boolean isStartRule) throws IOException
{
    out.write( ruleName + " :\n");
    out.write("\t\t{Stack<String> stack = new 
Stack<String>(); trace.push(stack); stack.


push("" + ruleName + ",\\n")
for( Link subRule : rule.getIds() )
{
    writeBnfSubRule( subRule, out );
}
if( isStartRule )
{
    out.write("EOF");
}
out.write("\n\t\t\t{trace.pop();};\n\n");
private static void writeBnfSubRule(Link rule,
    BufferedWriter out) throws IOException
{
    RepeatType repeating = rule.getRepeat();
    String id = rule.getId();
    String UUID = rule.getUUID();
    Stack<Link> ids = rule.getIds();
    if( id != null )
    {
        out.write( "({stack.push(" + UUID + ",\\n    popLast = true;})");
        out.write( "" + id + " {popLast = false;}}");
    }
    else
    {
        if( repeating == RepeatType.oneOfMultiple )
        {
            //special case
            out.write("(");
            for( int i=0;i<ids.size()-1;i++)
            {
                writeBnfSubRule(ids.get(i), out);
                out.write("|");
            }
            writeBnfSubRule(ids.get(ids.size()-1),
                out);
            out.write("")
        }
        else
        {
            //special case
            out.write("(");
            for( int i=0;i<ids.size()-1;i++)
            {
                writeBnfSubRule(ids.get(i), out);
                out.write("|");
            }
            writeBnfSubRule(ids.get(ids.size()-1),
                out);
            out.write("")
        }
    }
}
E.1 BnfCompiler

```java
{ 
    switch( repeating )
    {
    case atleastOnce:
        out.write( "(" );
        break;
    case loop:
        out.write( "(" );
        break;
    case optional:
        out.write( "(" );
        break;
    case sequence:
        out.write( "" );
        break;
    }
    for( Link subRule : ids )
    {
        writeBnfSubRule(subRule, out);
    }
    switch( repeating )
    {
    case atleastOnce:
        out.write( "+" );
        break;
    case loop:
        out.write( "*" );
        break;
    case optional:
        out.write( "?" );
        break;
    case sequence:
        out.write( "" );
        break;
    }
```

```
package BnfCompiler;

import org.antlr.tool.ANTLRErrorListener;
import org.antlr.tool.Message;
import org.antlr.tool.ToolMessage;

public class ErrorListener implements ANTLRErrorListener
{
    public int numErrorsAndWarnings = 0;

    private void addError(String error)
    {
        if (numErrorsAndWarnings == 0)
        {
            // first error
            System.out.println("Bnf error.");
            System.out.println("Make sure there's no left recursion or multiple alternatives for rules.");
            System.out.println("List of errors (line numbers does not match with the given BNF): ");
        }
        numErrorsAndWarnings++;
        System.out.println(error);
    }

    @Override
    public void error(Message msg)
    {
        addError(msg.toString());
    }

    @Override
    public void error(ToolMessage tmsg)
    {
        addError(tmsg.toString());
    }

    @Override
    public void info(String info)
System.out.println("Info: " + info);
}

@Override
public void warning(Message warning)
{
    addError(warning.toString());
}

E.1.3  Link.java

package BnfCompiler;
import java.util.Stack;

public class Link
{
    public enum RepeatType {sequence, optional, 
        atleastOnce, loop, oneOfMultiple}
    private String id;
    private Stack<Link> ids;
    private RepeatType repeating;
    private String UUID;
    private static int currentID = 0;

    public Link()
    {
        createLink( null, null );
    }

    public Link( Link link )
    {
        createLink( null, link );
    }

    //when this option is used, no functions can be used!
    public Link( String identifier )
    {
        //Replace " with '
        //Replace " with '
String id = identifier.replace("\\", ",");
createLink(id, null);

private void createLink(String identifier, Link link) {
    ids = new Stack<Link>();
    if (link != null) {
        ids.push(link);
    }
    id = identifier;
    if (id != null) {
        UUID = "id" + Integer.toString(currentID++);
    }
    //by default all Links are sequences
    repeating = RepeatType.sequence;
}

public String getUUID() {
    return UUID;
}

private void setRepeat(RepeatType repeat) {
    this.repeating = repeat;
}

public RepeatType getRepeat() {
    return repeating;
}

public String getId() {
    return id;
}

public Stack<Link> getIds() {

return ids;

/**
 * Adds another link to the sequence
 * @param link
 */
public void add( Link link )
{
    ids.add(link);
}

/**
 * This link requires just one of the ids, ex. "abc" | someRule | myrule | hehehe.
 */
public void oneOfMultiple()
{
    setRepeat(RepeatType.oneOfMultiple);
}

/**
 * This link is optional, ex. 
  * ["abc" someRule]
 */
public void optional()
{
    integrityCheck();
    setRepeat(RepeatType.optional);
}

/**
 * This link can loop from 0 to infinitely many times, ex. ("abc" someRule)* or <"abc" someRule>
 */
public void loop()
{
    integrityCheck();
    setRepeat(RepeatType.loop);
}

/**
 * This link loops atleast once, ex. ("abc" someRule)+
 */
public void loopAtleastOnce()
{
    integrityCheck();
    setRepeat(RepeatType.atleastOnce);
}

private void integrityCheck()
{
    if (id != null )
    {
        throw new Error("Integrity check error in
Link, id is defined and trying to add other objects!");
    }
}

E.2 Exceptions

E.2.1 UnknownXMLFormat.java

package Exceptions;

public class UnknownXMLFormat extends Exception
{
    private static final long serialVersionUID = -2212083386200335853L;

    public UnknownXMLFormat()
    {
    }
}

E.2.2 XMLATTRIBUTEDoesNotExist.java

package Exceptions;
E.2 Exceptions

```java
public class XMLattributeDoesNotExist extends Exception {
    private static final long serialVersionUID = -1349699666130356843L;
    String attribute;

    public XMLattributeDoesNotExist(String attribute) {
        System.out.println("XML attribute does not exist: " + attribute);
    }
}
```

E.2.3 XMLLoadException.java

```java
package Exceptions;

public class XMLLoadException extends Exception {
    private static final long serialVersionUID = -2522214346015679085L;
    private String error;

    public XMLLoadException( String error ) {
        this.error = error;
    }

    public String toString() {
        return error;
    }
}
```

E.2.4 XMLnodeDoesNotExist.java

```java
package Exceptions;
```
public class XMLnodeDoesNotExist extends Exception
{
    private static final long serialVersionUID =
        2952107033994886886L;
    private String error;

    public XMLnodeDoesNotExist( String node )
    {
        error = "Missing node: " + node;
    }

    public String toString()
    {
        return error;
    }
}

package Exceptions;

public class XMLTextDoesNotExist extends Exception
{
    private static final long serialVersionUID =
        2663518752007485741L;

    public XMLTextDoesNotExist()
    {
    }
}

package Grammar;

public class BnfParser.java
{
    private static final long serialVersionUID =
        2663518752007485741L;

    public BnfParser()
    {
    }
}
import java.util.Stack;
import org.antlr.runtime.Parser;
import org.antlr.runtime.RecognitionException;
import org.antlr.runtime.RecognizerSharedState;
import org.antlr.runtime.TokenStream;

public abstract class BnfParser extends Parser
{
    public BnfParser(TokenStream input)
    {
        super(input);
    }

    public BnfParser(TokenStream input,
            RecognizerSharedState state)
    {
        super(input, state);
    }

    public int errorLine = -1,
            errorCharPositionInLine = -1;
    public boolean popLast = false;
    public Stack<Stack<String>> trace = new Stack<Stack<String>>();
    public void displayRecognitionError(String[]
            tokenNames, RecognitionException e) throws
            RecognitionException
    {
        errorLine = e.line;
        errorCharPositionInLine = e.
                charPositionInLine;
        throw new RecognitionException();
    }

    //the default call to check syntax
    public abstract void bnf() throws
            RecognitionException;
}
E.4 GUI

E.4.1 Controller.java

```java
package GUI;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowEvent;
import java.awt.event.WindowListener;
import java.io.File;
import java.io.IOException;
import javax.swing.JFileChooser;
import javax.swing.JOptionPane;
import javax.swing.event.DocumentEvent;
import javax.swing.event.DocumentListener;
import GuiAPI.GuiApi;
import KernelAPI.KernelApi;

public class Controller implements ActionListener,
    WindowListener, DocumentListener {
    private static Controller instance;

    private Controller() {
    }

    public void actionPerformed(ActionEvent action) {
        String command = action.getActionCommand();
        if (command.equals(Variables.OPEN_SOURCE_FILE)) {
            final JFileChooser fileChooser = new JFileChooser(Variables.
                lastOpenedDirectory);
            int retVal = fileChooser.showOpenDialog(
                MainScreen.getInstance());
            if (retVal == JFileChooser.APPROVE_OPTION ) {
                final JFileChooser fileChooser = new JFileChooser(Variables.
                    lastOpenedDirectory);
                int retVal = fileChooser.showOpenDialog(
                    MainScreen.getInstance());
                if (retVal == JFileChooser.APPROVE_OPTION ) {
```

```java
    File selectedFile = fileChooser.getSelectedFile();
    KernelApi.readSourceFile(selectedFile);
    Variables.lastOpenedDirectory = selectedFile.getParent();
}

else if (command.equals(Variables.RELOAD_SOURCE_FILE))
{
    KernelApi.reloadSourceCode();
}
else if (command.equals(Variables.CHECK_SYNTAX))
{
    KernelApi.setSourceCode(gSourceCode.getInstance().getSourceCode());
    GuiApi.updateDiagrams();
}
else if (command.equals(Variables.SAVE_SOURCE_FILE))
{
    saveSourceCode();
}
else if (command.equals(Variables.SHOW_HIDE_BNF_GRAMMARS))
{
    gGrammarPanel.getInstance().swapBetweenShowAndHideGrammarOptions();
}
else if (command.equals(Variables.HELP))
{
    Dialogs.showHelpDialog();
}
else if (command.equals(Variables.ABOUT))
{
    Dialogs.showAboutDialog();
}
else
{
    System.out.println("Unknown command: " + command);
}
```
private boolean saveSourceCode() {
    try {
        KernelApi.setSourceCode(gSourceCode.getInstance().getSourceCode());
        KernelApi.saveSourceCode();
        Variables.setCodeChanged(false);
        return true;
    }
    catch (IOException e) {
        JOptionPane.showMessageDialog(MainScreen.getInstance(), "Failed to save file!");
    }
    return false;
}

@Override
public void windowActivated(WindowEvent arg0) {
}

@Override
public void windowClosed(WindowEvent arg0) {
}

@Override
public void windowClosing(WindowEvent arg0) {
    if( Variables.isCodeChanged() ) {
        String filename = MainScreen.getInstance().getFilename();
        if( filename == null ) {
            System.exit(0);
        }
        int result = JOptionPane.showConfirmDialog(
            MainScreen.getInstance(), "Save file " + filename + ", " + "Close", JOptionPane.
           ......}
YES_NO_CANCEL_OPTION, JOptionPane.
QUESTION_MESSAGE);

switch( result )
{
    case JOptionPane.YES_OPTION:
        if( saveSourceCode() )
        {
            System.exit(0);
        }
        break;
    case JOptionPane.NO_OPTION:
        System.exit(0);
        break;
}
else
{
    System.exit(0);
}

@Override
public void windowDeactivated(WindowEvent arg0)
{
}

@Override
public void windowDeiconified(WindowEvent arg0)
{
}

@Override
public void windowIconified(WindowEvent arg0)
{
}

@Override
public void windowOpened(WindowEvent arg0)
{
}

@Override
public void changedUpdate(DocumentEvent e)
E.4.2 Dialogs.java

```java
package GUI;

import java.awt.SystemColor;
import javax.swing.JOptionPane;
import javax.swing.JTextArea;

public class Dialogs {
    public static Controller getInstance()
    {
        if (instance == null)
        {
            instance = new Controller();
        }
        return instance;
    }
}
```
public static void showHelpDialog()
{
    final JTextArea helpTextArea = new JTextArea("In order to use this program, first open a source file, which is accepted by the current grammar file. 
" +
    "If there’s any error in your code, SyntaxTrain will show syntax diagrams to describe the error, just like a stack trace. For an example you can try using the java grammar included and open example.java.\n\n" +
    "If a node in the diagram is marked blue then it means that node was correctly matched (no error here)\n" +
    "If instead the node is colored red, it means that the error occurred here, and above you will see the appropriate diagram for that node. Finally if the node is marked orange, that means this is what you can write at that point (one or multiple nodes may be colored orange).\n" +
    "In the top there’s a diagram showing where in your code the error is located.\n" +
    "Ex. it could be inside an if sentence inside your function inside your class.");
    helpTextArea.setEditable(false);
    helpTextArea.setBackground(SystemColor.control);
    JOptionPane.showMessageDialog(MainScreen.getInstance(), helpTextArea, "SyntaxTrain - Help", JOptionPane.PLAIN_MESSAGE);
}

public static void showAboutDialog()
{
E.4.3  gErrorTrace.java

1 package GUI;
2
3 final JTextArea aboutTextArea = new JTextArea("SyntaxTrain - Syntax Diagrams for Java
  Programs\n" +
  "Version 1.0\n" +
  "Copyright 2011 by Andreas Leon Aagaard
  Moth.\n\n" +
  "This program is free software: you can
 redistribute it and/or modify\n" +
  "it under the terms of the GNU General
 Public License as published by\n" +
  "the Free Software Foundation, either
 version 3 of the License, or\n" +
  "(at your option) any later version.\n" +
  "\n" +
  "This program is distributed in the hope
 that it will be useful,\n" +
  "but WITHOUT ANY WARRANTY; without even
 the implied warranty of\n" +
  "MERCHANTABILITY or FITNESS FOR A
 PARTICULAR PURPOSE. See the\n" +
  "GNU General Public License for more
 details.\n" +
  "\n" +
  "You should have received a copy of the
 GNU General Public License\n" +
  "along with this program. If not, see <
 http://www.gnu.org/licenses/>.\n");
4 aboutTextArea.setEditable(false);
5 aboutTextArea.setBackground(SystemColor.control);
6 JOptionPane.showMessageDialog(MainScreen.getInstance(), aboutTextArea, "About SyntaxTrain - version 1.0", JOptionPane.PLAIN_MESSAGE);
7  
8 }
9 }
}

100
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Font;
import java.awt.Graphics;
import java.awt.geom.Rectangle2D;
import java.util.ArrayList;
import java.util.Stack;
import javax.swing.JPanel;
import KernelAPI.KernelApi;

/**
 * Displays a rail-road diagram of the source code.
 * (top)
 */
public class gErrorTrace extends JPanel
{
    private static final int SPACE_BETWEEN_BOXES = 10, SPACE_ON_EACH_SIDE_INSIDE_BOXES = 2;
    private static final long serialVersionUID = 7253097565513080782L;
    private static gErrorTrace instance = null;
    private ArrayList<String> errorTrace;
    private Dimension size;
    private boolean outOfSync;

    private gErrorTrace()
    {
        setBackground(Color.WHITE);
        updateDiagram();
        outOfSync = false;
    }

    public Dimension getPreferredSize()
    {
        if( size != null )
            return size;
        return new Dimension(400, 100);
    }

    public void updateSyncStatus()
    {
        outOfSync = Variables.isDiagramsOutOfSync();
    }
}
getParent().repaint();

public void paint(Graphics g)
{
    super.paint(g);
    Font header_font = new Font( "Serif", Font.PLAIN, 20 );
    Font box_font = new Font( "Serif", Font.PLAIN, 14 );
    int posX = 15, posY = 25;
    g.setFont(header_font);
    String textBuffer = "No syntax errors";
    if( errorTrace.size() > 0 )
    {
        textBuffer = "First syntax error: ";
    }
    Rectangle2D bounds = g.getFontMetrics().getStringBounds(textBuffer, g);
    posY = (int)bounds.getHeight() + 5;
    g.drawString(textBuffer, posX, posY);
    if( outOfSync )
    {
        textBuffer = "modified";
        g.setFont(new Font( "Serif", Font.PLAIN, 12 ));
        int modHeight = (int)g.getFontMetrics().getStringBounds(textBuffer, g).getHeight();
        g.drawString(textBuffer, posX, posY + modHeight + 3);
    }
    posX += bounds.getWidth();
    g.setFont(box_font);
    for( int i=errorTrace.size()-1;i>0;i-- )
    {
        //text
        g.setColor(Color.BLACK);
        textBuffer = errorTrace.get(i);
        g.drawString(textBuffer, posX, posY);
E.4 GUI

81     bounds = g.getFontMetrics().getStringBounds
82         (textBuffer, g);
83
84     //rectangle
85     g.setColor(Color.RED);
86     g.drawRect(posX -
87             SPACE_ON_EACH_SIDE_INSIDE_BOXES, (int)
88             (posY - bounds.getHeight()), (int) (  
89             bounds.getWidth() +
90             SPACE_ON_EACH_SIDE_INSIDE_BOXES * 2), (  
91             int)bounds.getHeight() * 3 / 2);
92     posX += SPACE_BETWEEN_BOXES;
93     posX += bounds.getWidth();
94 {
95     size = new Dimension(posX, posY);
96     getParent().doLayout();
97 }
98
99     public void updateDiagram()
100 {
101         errorTrace = new ArrayList<String>();
102         //get grammar
103         Stack<Stack<String>> errorTrace = KernelApi.
104             getErrorTrace();
105         if( errorTrace == null )
106             return;
107         for( Stack<String> ruleTrace : errorTrace )
108             this.errorTrace.add(ruleTrace.firstElement()
109                 ());
110         }
111 }
112
113     public static synchronized gErrorTrace
114         getInstance()
115         {
116             if( instance == null )
117             {
118             }
instance = new gErrorTrace();
}
return instance;
}
}

E.4.4 gGrammarDiagram.java

package GUI;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.image.BufferedImage;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Stack;
import javax.swing.JPanel;
import KernelAPI.KernelApi;
import Library.Lock;
import net.hydromatic.clapham.Clapham;
import net.hydromatic.clapham.graph.Chart;
import net.hydromatic.clapham.graph.Grammar;
import net.hydromatic.clapham.graph.Symbol;
import net.hydromatic.clapham.parser.ProductionNode;
/**
 * Displays a rail-road diagram of the source code (bottom middle)
 */
public class gGrammarDiagram extends JPanel {
    private static final long serialVersionUID = 1334930201861271821L;
    private static gGrammarDiagram instance = null;
    private BufferedImage[] grammarDiagrams;
    private HashMap<String, Integer> grammarToId;
    private static final long serialVersionUID = 1333493020186127182L;
    private static gGrammarDiagram instance = null;
    private BufferedImage[] grammarDiagrams;
    private HashMap<String, Integer> grammarToId;
    private static final long serialVersionUID = 1334930201861271821L;
    private static gGrammarDiagram instance = null;
    private BufferedImage[] grammarDiagrams;
    private HashMap<String, Integer> grammarToId;
    private static final long serialVersionUID = 1334930201861271821L;
    private static gGrammarDiagram instance = null;
    private BufferedImage[] grammarDiagrams;
    private HashMap<String, Integer> grammarToId;
    private static final long serialVersionUID = 1334930201861271821L;
    private static gGrammarDiagram instance = null;
    private BufferedImage[] grammarDiagrams;
    private HashMap<String, Integer> grammarToId;
    private static final long serialVersionUID = 1334930201861271821L;
    private static gGrammarDiagram instance = null;
    private BufferedImage[] grammarDiagrams;
    private HashMap<String, Integer> grammarToId;
    private static final long serialVersionUID = 1334930201861271821L;
    private static gGrammarDiagram instance = null;
    private BufferedImage[] grammarDiagrams;
    private HashMap<String, Integer> grammarToId;
    private static final long serialVersionUID = 1334930201861271821L;
    private static gGrammarDiagram instance = null;
    private BufferedImage[] grammarDiagrams;
    private HashMap<String, Integer> grammarToId;
    private static final long serialVersionUID = 1334930201861271821L;
    private static gGrammarDiagram instance = null;
    private BufferedImage[] grammarDiagrams;
    private HashMap<String, Integer> grammarToId;
    private static final long serialVersionUID = 1334930201861271821L;
    private static gGrammarDiagram instance = null;
    private BufferedImage[] grammarDiagrams;
    private HashMap<String, Integer> grammarToId;
    private static final long serialVersionUID = 1334930201861271821L;
    private static gGrammarDiagram instance = null;
    private BufferedImage[] grammarDiagrams;
    private HashMap<String, Integer> grammarToId;
    private static final long serialVersionUID = 1334930201861271821L;
    private static gGrammarDiagram instance = null;
    private BufferedImage[] grammarDiagrams;
    private HashMap<String, Integer> grammarToId;
    private static final long serialVersionUID = 1334930201861271821L;
    private static gGrammarDiagram instance = null;
    private BufferedImage[] grammarDiagrams;
    private HashMap<String, Integer> grammarToId;
    private static final long serialVersionUID = 1334930201861271821L;
    private static gGrammarDiagram instance = null;
    private BufferedImage[] grammarDiagrams;
    private HashMap<String, Integer> grammarToId;
    private static final long serialVersionUID = 1334930201861271821L;
    private static gGrammarDiagram instance = null;
    private BufferedImage[] grammarDiagrams;
    private HashMap<String, Integer> grammarToId;
    private static final long serialVersionUID = 1334930201861271821L;
    private static gGrammarDiagram instance = null;
    private BufferedImage[] grammarDiagrams;
    private HashMap<String, Integer> grammarToId;
    private static final long serialVersionUID = 1334930201861271821L;
    private static gGrammarDiagram instance = null;
    private BufferedImage[] grammarDiagrams;
    private HashMap<String, Integer> grammarToId;
    private static final long serialVersionUID = 1334930201861271821L;
    private static gGrammarDiagram instance = null;
    private BufferedImage[] grammarDiagrams;
    private HashMap<String, Integer> grammarToId;
    private static final long serialVersionUID = 1334930201861271821L;
    private static gGrammarDiagram instance = null;
    private BufferedImage[] grammarDiagrams;
    private HashMap<String, Integer> grammarToId;
    private static final long serialVersionUID = 1334930201861271821L;
    private static gGrammarDiagram instance = null;
    private BufferedImage[] grammarDiagrams;
    private HashMap<String, Integer> grammarToId;
    private static final long serialVersionUID = 1334930201861271821L;
    private static gGrammarDiagram instance = null;
    private BufferedImage[] grammarDiagrams;
    private HashMap<String, Integer> grammarToId;
    private static final long serialVersionUID = 1334930201861271821L;
    private static gGrammarDiagram instance = null;
    private BufferedImage[] grammarDiagrams;
    private HashMap<String, Integer> grammarToId;
    private static final long serialVersionUID = 1334930201861271821L;
    private static gGrammarDiagram instance = null;
    private BufferedImage[] grammarDiagrams;
    private HashMap<String, Integer> grammarToId;
    private static final long serialVersionUID = 1334930201861271821L;
    private static gGrammarDiagram instance = null;
    private BufferedImage[] grammarDiagrams;
    private HashMap<String, Integer> grammarToId;
    private static final long serialVersionUID = 1334930201861271821L;
    private static gGrammarDiagram instance = null;
    private BufferedImage[] grammarDiagrams;
    private HashMap<String, Integer> grammarToId;
    private static final long serialVersionUID = 1334930201861271821L;
    private static gGrammarDiagram instance = null;
    private BufferedImage[] grammarDiagrams;
    private HashMap<String, Integer> grammarToId;
    private static final long serialVersionUID = 1334930201861271821L;
    private static gGrammarDiagram instance = null;
    private BufferedImage[] grammarDiagrams;
    private HashMap<String, Integer> grammarToId;
    private static final long serialVersionUID = 1334930201861271821L;
    private static gGrammarDiagram instance = null;
    private BufferedImage[] grammarDiagrams;
    private HashMap<String, Integer> grammarToId;
    private static final long serialVersionUID = 1334930201861271821L;
    private static gGrammarDiagram instance = null;
    private BufferedImage[] grammarDiagrams;
    private HashMap<String, Integer> grammarToId;
    private static final long serialVersionUID = 1334930201861271821L;
    private static gGrammarDiagram instance = null;
    private BufferedImage[] grammarDiagrams;
    private HashMap<String, Integer> grammarToId;
    private static final long serialVersionUID = 1334930201861271821L;
    private static gGrammarDiagram instance = null;
    private BufferedImage[] grammarDiagrams;
    private HashMap<String, Integer> grammarToId;
    private static final long serialVersionUID = 1334930201861271821L;
    private static gGrammarDiagram instance = null;
    private BufferedImage[] grammarDiagrams;
    private HashMap<String, Integer> grammarToId;
private String[] grammars;
private boolean[] showGrammar;
private Lock grammarDiagramsLock;

private gGrammarDiagram()
{
    grammarDiagramsLock = new Lock();
    setBackground(Color.WHITE);
    updateDiagram();
}

synchronized public void updateDiagram()
{
    ArrayList<String> grammarNames = KernelApi.getGrammars();
    if (grammarNames == null)
    {
        return;
    }
    grammarDiagramsLock.P();
    //initialize grammar varaibles
    int grammarId = 0;
    grammars = new String[grammarNames.size()];
    showGrammar = new boolean[grammarNames.size()];
    grammarToId = new HashMap<String, Integer>();
    grammarDiagrams = new BufferedImage[grammarNames.size()];

    if (KernelApi.getErrorTrace() != null) //== null if no error
    {
        @SuppressWarnings("unchecked")
        Stack<Stack<String>> errorTrace = (Stack<
            Stack<String>>) KernelApi.getErrorTrace().clone();
        while (!errorTrace.isEmpty())
        {
            String ruleName = errorTrace.pop().firstElement();
            if (grammarToId.containsKey(ruleName))
                continue;
            grammars[grammarId] = ruleName;
            showGrammar[grammarId] = true;
        }
    }
}
grammarToId.put(ruleName, grammarId);
    grammarId++;
}
}

if( grammarNames != null )
{
    for ( String grammarName : grammarNames )
    {
        if( grammarToId.containsKey(grammarName) )
            continue;

        grammars[grammarId] = grammarName;
        showGrammar[grammarId] = false;
        grammarToId.put(grammarName, grammarId);
        grammarId++;
    }
}

//build grammar diagrams
List<ProductionNode> productionNodes =
    KernelApi.getGrammarProductionNodes();
if( productionNodes == null )
{
    return;
}
Grammar grammar = Clapham.buildGrammar(
    productionNodes);
List<String> nameList = new ArrayList<String>();
nameList.clear();
nameList.addAll(grammar.symbolMap.keySet());
for( String grammarName : nameList )
{
    BufferedImage image = drawNode(grammarName, grammar);
    int id = grammarToId.get(grammarName);
    grammarDiagrams[id] = image;
}
grammarDiagramsLock.V();

//show default grammars
public void setGrammarVisible( String grammar, boolean visible )
{
    int id = grammarToId.get(grammar);
    showGrammar[id] = visible;

    updateDimensions();
    repaint();
}

private void updateDimensions()
{
    int width=0, height=0;

    for( int i=0;i<grammars.length;i++ )
    {
        if(showGrammar[i])
        {
            BufferedImage image = grammarDiagrams[i];

            width = Math.max(image.getWidth(), width);
            height += image.getHeight();
        }
    }

    Dimension dim = new Dimension(width, height);
    this.setSize(dim);
    this.setPreferredSize(dim);
}

public void paint(Graphics g)
{
    super.paint(g);

    if( grammars == null )
    {
        return;
    }

    grammarDiagramsLock.P();
for( int i=0;i<grammars.length;i++ )
{
    if(showGrammar[i])
    {
        BufferedImage image = grammarDiagrams[i];
        g.drawImage(image, 0, 0, null);
        g.translate(0, image.getHeight());
    }
}
grammarDiagramsLock.V();

private BufferedImage drawNode(String symbolName,
                                Grammar grammar)
{
    //temporary image to draw on
    BufferedImage tempImg = new BufferedImage(1,
                                                  1, BufferedImage.TYPE_INT_RGB);
    Graphics2D graphics = tempImg.createGraphics();

    Symbol symbol = grammar.symbolMap.get(symbolName);
    if (symbol.graph == null)
    {
        throw new RuntimeException("Symbol \\
                                    "+symbolName+", not found");
    }

    Chart chart = new Chart(grammar, (Graphics2D) graphics);
    chart.calcDrawing();
    chart.drawComponent(symbol);

    //draw the final image
    Dimension dim = chart.getDimension();
    BufferedImage finalDrawing = new BufferedImage((int)dim.getWidth(), (int)dim.getHeight() + 5, BufferedImage.TYPE_INT_RGB);
    graphics = finalDrawing.createGraphics();
chart = new Chart(grammar, (Graphics2D) graphics);
chart.calcDrawing();
chart.drawComponent(symbol);
return finalDrawing;
}

public static synchronized gGrammarDiagram getInstance()
{
    if( instance == null )
    {
        instance = new gGrammarDiagram();
    }
    return instance;
}

E.4.5  gGrammarOptions.java

package GUI;

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Component;
import java.awt.Dimension;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.HashSet;
import java.util.Stack;
import javax.swing.BorderFactory;
import javax.swing.JCheckBox;
import javax.swing.JLabel;
import javax.swing.JList;
import javax.swing.JPanel;
import javax.swing.ListCellRenderer;
import javax.swing.event.ListSelectionEvent;
import javax.swing.event.ListSelectionListener;
import KernelAPI.KernelApi;

import java.util.List;
import java.util.LinkedList;
import java.util.ListIterator;
import java.util.List;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.HashSet;
import java.util.Stack;
import javax.swing.BorderFactory;
import javax.swing.JCheckBox;
import javax.swing.JLabel;
import javax.swing.JList;
import javax.swing.JPanel;
import javax.swing.ListCellRenderer;
import javax.swing.event.ListSelectionEvent;
import javax.swing.event.ListSelectionListener;
import KernelAPI.KernelApi;

...
public class gGrammarOptions extends JList
    implements ListCellRenderer, ListSelectionListener
{
    private static final long serialVersionUID =
        4419173956236695445L;

    private static gGrammarOptions instance = null;

    private HashMap<String, Boolean> isChecked;
    private HashSet<String> errorTraceComponents;
    // used by getListCellRendererComponent to avoid
    // allocating a new checkbox all the time
    private JCheckBox checkBox;
    private JLabel label;
    private JPanel panel;
    private String[] listData;
    private int numChecked;

    public gGrammarOptions()
    {
        clearCheckedRules();
        setCellRenderer( this );
        addListSelectionListener( this );

        checkBox = new JCheckBox();
        label = new JLabel();
        panel = new JPanel( new BorderLayout() );
        panel.add( BorderLayout.WEST , label );
        panel.add( BorderLayout.EAST , checkBox );
        panel.setBorder(BorderFactory.createLineBorder(Color.black));

        // set default grammars
        updateGrammars();
    }

    public Dimension getPreferredSize()
    {
        Dimension dim = super.getPreferredSize();
        dim.setSize(dim.getWidth()+20 , dim.getHeight());
        return dim;
    }
}
private void clearCheckedRules() {
    isChecked = new HashMap<String, Boolean>();
    errorTraceComponents = new HashSet<String>();
    ArrayList<String> grammarNames = KernelApi.getGrammars();
    if( grammarNames != null ) {
        listData = new String[grammarNames.size() + 1];
        listData[0] = "Syntax Components:";
        for ( int i=0; i<grammarNames.size(); i++ ) {
            String grammarName = grammarNames.get(i);
            listData[i+1] = grammarName;
            isChecked.put( grammarName, false );
        }
        setListData( listData );
        numChecked = 0;
    }
}

class public void updateGrammars() {
    clearCheckedRules();
    showErrorTrace();
}

private void showErrorTrace() {
    if( KernelApi.getErrorTrace() != null ) {
        for(Stack<String> ruleTrace : KernelApi.getErrorTrace()) {
            String component = ruleTrace.firstElement();
            isChecked.put(component, true);
            errorTraceComponents.add(component);
        }
    }
public Component getListCellRendererComponent(
    JList list,
    Object value,
    int index,
    boolean isSelected,
    boolean cellHasFocus )
{
    if ( !(value instanceof String) )
    {
        return new JLabel( "Error: " + value.
            toString() );
    }
    String name = ( String ) value;
    if( name.equalsIgnoreCase("Syntax Components :") )
    {
        //top
        checkBox.setSelected(false);
        label.setText(name);
        panel.setComponentOrientation( list.
            getComponentOrientation() );
        panel.setBackground( new Color(200 ,200 ,200) );
        checkBox.setBackground( new Color(200 ,200 ,200) );
        checkBox.setSelected( numChecked > 0 );
        panel.setForeground( list.getForeground
            ( ) );
    }
    else
    {
        checkBox.setSelected( isChecked.get( name
            ) );
        label.setText( name );
        panel.setComponentOrientation( list.
            getComponentOrientation() );
        if ( isChecked.get( name )
            )
        {
        }
    }
}
panel.setBackground( list.getSelectionBackground() );
checkBox.setBackground( list.getSelectionBackground() );
panel.setForeground( list.getSelectionForeground() );
}
else {
    panel.setBackground( list.getBackground() );
    checkBox.setBackground( list.getBackground() );
    panel.setForeground( list.getForeground() );
}
}
return panel;
}
public void valueChanged( ListSelectionEvent e )
{
    //make sure we aren’t dragging anything into the list
    if ( !e.getValueIsAdjusting() )
    {
        Object value = getSelectedValue();
        if ( value instanceof String )
        {
            String name = ( String ) value;
            if( name.equalsIgnoreCase("Syntax Components:"))
            {
                // header clicked
                if( numChecked > 0 )
                {
                    numChecked = 0;
                    for ( int i=1;i<listData.length; i++ )
                    {
                        String grammarName = listData[i];
                        boolean showGrammar = false;
        }
// make sure to still show the error trace
if ( errorTraceComponents.
    contains(grammarName) ) {
    showGrammar = true;
}
isChecked.put( grammarName, showGrammar );
gGrammarDiagram.getInstance().
    setGrammarVisible( grammarName, showGrammar);

else {
    numChecked = listData.length -
        1 - errorTraceComponents.
    size();
    for ( int i=1;i<listData.length
        ;i++ ) {
        String grammarName =
            listData[i];
        isChecked.put( grammarName, true );
        gGrammarDiagram.
            getInstance().
            setGrammarVisible( grammarName, true);
    }
}
else {
    boolean isSelected = isChecked.get
        ( name );
    if ( errorTraceComponents.contains( name ) ) {
        // selection is part of the trace, so if it's turned off, then we move one step away from default (0),
otherwise we move one closer.
numChecked += isSelected ? 1 :
-1;
}
else {
  // selection is not part of the error trace, if it's turned off then move one step closer to default (0), otherwise move one step away.
  numChecked += isSelected ? -1 : 1;
}
isChecked.put(name, !isSelected);
gGrammarDiagram.getInstance().
setGrammarVisible(name, !isSelected);
}
removeSelectionInterval(0,
isChecked.size());
}
}

public static synchronized gGrammarOptions getInstance()
{
  if(instance == null)
  {
    instance = new gGrammarOptions();
  }
  return instance;
}
import java.awt.BorderLayout;
import java.awt.Graphics;

import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JSplitPane;

public class gGrammarPanel extends JPanel {
    private static final long serialVersionUID = 1452823813888634187L;
    private static gGrammarPanel instance = null;
    private boolean isGrammarOptionsVisible;
    private JScrollPane scrollGrammarOptions;
    private JSplitPane diagramsOptionsSplitPane;

    private gGrammarPanel()
    {
        super(new BorderLayout());
        //Grammar diagrams
        JScrollPane scrollGrammarDiagram = new JScrollPane(gGrammarDiagram.getInstance());
        scrollGrammarDiagram.getVerticalScrollBar().setUnitIncrement(16);
        //Grammar options
        scrollGrammarOptions = new JScrollPane(gGrammarOptions.getInstance());
        //by default the grammar options are hidden
        isGrammarOptionsVisible = false;
        //Split pane containing the diagram pane and the options pane
        diagramsOptionsSplitPane = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT,
                                                   scrollGrammarDiagram, scrollGrammarOptions);
        diagramsOptionsSplitPane.setEnabled(false);
        diagramsOptionsSplitPane.setDividerSize(0);
        diagramsOptionsSplitPane.setDividerLocation(0); //to avoid drawing a line at startup
        add(BorderLayout.CENTER, diagramsOptionsSplitPane);
    }
}
public void paint(Graphics g) {
    if( isGrammarOptionsVisible ) {
        diagramsOptionsSplitPane.setDividerLocation((int) (getWidth() - scrollGrammarOptions.getPreferredSize().getWidth()) - 1);
    } else {
        // hide grammar options
        diagramsOptionsSplitPane.setDividerLocation(1.0);
    }
    super.paint(g);
}

public void swapBetweenShowAndHideGrammarOptions () {
    isGrammarOptionsVisible = !isGrammarOptionsVisible;
    repaint();
}

public static synchronized gGrammarPanel getInstance() {
    if(instance == null) {
        instance = new gGrammarPanel();
    }
    return instance;
}

E.4.7 gSourceCode.java

package GUI;
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Font;
import java.awt.FontMetrics;
import java.awt.Point;
import javax.swing.JPanel;
import javax.swing.JTextPane;
import javax.swing.text.MutableAttributeSet;
import javax.swing.text.SimpleAttributeSet;
import javax.swing.text.StyleConstants;
import javax.swing.text.StyledDocument;
import javax.swing.text.TabStop;
import KernelAPI.KernelApi;

public class gSourceCode extends JPanel {
  private static final long serialVersionUID = 4279528648857232158L;
  private static gSourceCode instance = null;
  private JTextPane textPane;

  private gSourceCode() {
    this.setLayout(new BorderLayout());
    textPane = new JTextPane();
    textPane.setFont(new Font("Courier New", Font.PLAIN, 14));
    setTabs(textPane, 4);
    textPane.getDocument().addDocumentListener(Controller.getInstance());
    add(BorderLayout.CENTER, textPane);
    updateErrorPosition();
  }

  public void updateErrorPosition() {
    MutableAttributeSet errorAttrib = new SimpleAttributeSet();
    MutableAttributeSet normAttrib = new SimpleAttributeSet();
  }
}
StyleConstants.setBackground(errorAttrib, new Color(255, 150, 50));

StyledDocument document = textPane.getStyledDocument();

//clear formatting
document.setCharacterAttributes(0, textPane.getText().length(), normAttrib, true);

int lineNumber = KernelApi.getErrorLine() - 1;
int charPositionInLine = KernelApi.getErrorCharPositionInLine();
if (lineNumber >= 0)
{
    Point errorOffsets = getLineStartEndOffset(lineNumber);
    if (errorOffsets != null)
    {
        document.setCharacterAttributes(
            errorOffsets.x + charPositionInLine,
            errorOffsets.y - errorOffsets.x - charPositionInLine, errorAttrib, true);
    }
}

private Point getLineStartEndOffset(int line)
{
    String text = textPane.getText().replace("\r", "");  //For some reason indexOf counts \r
    but document doesn’t, so just remove all
    of them :)
    int start = 0, end;
    while (line > 0)
    {
        start = text.indexOf("\n", start) + 1;
        line--;
    }

    if (start == -1)
    {
        //just in case of any errors, shouldn’t be
        any since the line info is based on
        this string
    }
```java
start = 0;
}
end = text.indexOf("\n", start + 1);
if( end == -1 )
{
    end = textPane.getText().length();
}
return new Point(start, end);
}

public String getSourceCode()
{
    return textPane.getText();
}

public void updateSourceCode()
{
    textPane.setText(KernelApi.getSourceCode());
}

private void setTabs( JTextPane textPane , int charactersPerTab)
{
    FontMetrics fm = textPane.getFontMetrics(
            textPane.getFont() );
    int charWidth = fm.charWidth( ' ' );
    int tabWidth = charWidth * charactersPerTab;
    TabStop[] tabs = new TabStop[10];
    for (int j = 0; j < tabs.length; j++)
    {
        tabs[j] = new TabStop( (j+1) * tabWidth );
    }
    TabSet tabSet = new TabSet(tabs);
    SimpleAttributeSet attributes = new SimpleAttributeSet();
    StyleConstants.setTabSet(attributes, tabSet);
    int length = textPane.getDocument().getLength();
    textPane.getStyledDocument().setParagraphAttributes(0 , length ,
            attributes , false);
```
E.4.8  gToolbar.java

```java
package GUI;

import javax.swing.JButton;
import javax.swing.JComponent;
import javax.swing.JToolBar;
import javax.swing.KeyStroke;

public class gToolbar {

    static gToolbar instance = null;
    private JToolBar toolbar;

    private gToolbar() {
        toolbar = createToolBar( Variables.TOOBAR_ITEMS, Variables.SHORTCUTS );
    }

    private JToolBar createToolBar( String[] items, KeyStroke[] keystrokes ) {
        JToolBar toolbar = new JToolBar();
        toolbar.setFloatable(false);
        for( int i = 0; i < items.length; i++ ) {
            String item = items[i];
            KeyStroke key = keystrokes[i];
            JButton itemButton = new JButton(item);
            if( key != null )
```
E.4.9 MainScreen.java

```java
package GUI;

import java.awt.BorderLayout;
import java.awt.Graphics;

import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JSplitPane;

public class MainScreen extends JFrame {
    ...
```
private static final long serialVersionUID = -5921202176825848947L;
private static MainScreen instance = null;
private String filename;
private boolean isDividerLocationInitialized;
private JSplitPane BottomPanes, topBottomPane;

private MainScreen()
{
    super("SyntaxTrain");
    filename = null;
    isDividerLocationInitialized = false;
    //initialize frame
    setLayout(new BorderLayout());
    setDefaultCloseOperation( JFrame.DO_NOTHING_ON_CLOSE );
    addWindowListener(Controller.getInstance());
    setBounds(300, 50, 1000, 700);
    setExtendedState(MAXIMIZED_BOTH);

    //add toolbar
    add( BorderLayout.PAGE_START, gToolbar.getInstance().getToolBar() );

    //center panel
    JPanel mainPanel = new JPanel(new BorderLayout ());
    JScrollPane sourceScrollPane = new JScrollPane( gSourceCode.getInstance() );
    sourceScrollPane.getVerticalScrollBar().setUnitIncrement(16);
    BottomPanes = new JSplitPane( JSplitPane.HORIZONTAL_SPLIT , sourceScrollPane ,
        gGrammarPanel.getInstance() ) ;
    BottomPanes.setOneTouchExpandable(true);
    JScrollPane scrollSourceDiagram = new JScrollPane(gErrorTrace.getInstance());
    topBottomPane = new JSplitPane(JSplitPane.VERTICAL_SPLIT , scrollSourceDiagram , BottomPanes) ;
    topBottomPane.setDividerSize(0);
mainPanel.add(topBottomPane);
add( BorderLayout.CENTER, mainPanel );

// when using int the divider location can be
// set before showing the frame
    topBottomPane.setDividerLocation(80);

// This is to ensure enough room is given to
// the source code
    BottomPanes.setDividerLocation(400);

    // show frame
    setVisible(true);
doLayout();
}

public void paint(Graphics g)
{
    super.paint(g);
    if( ! isDividerLocationInitialized )
    {
        BottomPanes.setDividerLocation(0.3); // when
        using double the divider must be set
        after showing the frame

    isDividerLocationInitialized = true;
    repaint();
    }
}

public void setFilename( String filename )
{
    this.filename = filename;
    setTitle("SyntaxTrain - " + filename);
}

public String getFilename()
{
    return filename;
}
E.4 GUI

```java
85     public static MainScreen getInstance()
86     {
87         if(instance == null)
88             {  
89                 instance = new MainScreen();
90             }
91         return instance;
92     }
93 }
```

E.4.10 Variables.java

```java
1 package GUI;
2
3 import java.awt.event.KeyEvent;
4
5 import javax.swing.KeyStroke;
6
7 public class Variables
8 {
9     public static String lastOpenedDirectory = ".";
10     //Toolbar strings
11     public final static String[] TOOLBAR_ITEMS =
12         {
13             Variables.OPEN_SOURCE_FILE,
14             Variables.RELOAD_SOURCE_FILE,
15             Variables.CHECK_SYNTAX,
16             Variables.SHOW_HIDE_BNF_GRAMMARS,
17             Variables.SAVE_SOURCE_FILE,
18             Variables.HELP,
19             Variables.ABOUT};
20
21     public final static KeyStroke[] SHORTCUTS =
22         {
23             KeyStroke.getKeyStroke(KeyEvent.VK_O, KeyEvent
24                     .CTRL_DOWN_MASK),
25             KeyStroke.getKeyStroke(KeyEvent.VK_R, KeyEvent
26                     .CTRL_DOWN_MASK),
27             KeyStroke.getKeyStroke(KeyEvent.VK_F5, 0),
28             KeyStroke.getKeyStroke(KeyEvent.VK_F10, 0),
29             KeyStroke.getKeyStroke(KeyEvent.VK_S, KeyEvent
30                     .CTRL_DOWN_MASK),
```
private static boolean codeChanged = false;
public static boolean isCodeChanged()
{
    return codeChanged;
}
public static void setCodeChanged(boolean codeChanged)
{
    Variables.codeChanged = codeChanged;
}
private static boolean diagramsOutOfSync = false;
public static boolean isDiagramsOutOfSync()
{
    return diagramsOutOfSync;
}
public static void setDiagramsOutOfSync(boolean diagramsOutOfSync)
{
    Variables.diagramsOutOfSync = diagramsOutOfSync;
}
E.5 GuiAPI

E.5.1 GuiApi.java

```java
package GuiAPI;

import javax.swing.JOptionPane;
import GUI.MainScreen;
import GUI.Variables;
import GUI.gGrammarDiagram;
import GUI.gGrammarOptions;
import GUI.gSourceCode;
import GUI.gErrorTrace;

public class GuiApi {

    public static void updateDiagrams() {
        boolean wasCodeChanged = Variables.isCodeChanged();
        gGrammarOptions.getInstance().updateGrammars();
        gGrammarDiagram.getInstance().updateDiagram();
        gErrorTrace.getInstance().updateDiagram();
        gSourceCode.getInstance().updateErrorPosition();
        Variables.setDiagramsOutOfSynch(false);
        gErrorTrace.getInstance().updateSyncStatus();
        Variables.setCodeChanged(wasCodeChanged);
    }

    public static void updateSourceCode(String fileOpened) {
        MainScreen.getInstance().setFilename(fileOpened);
        gSourceCode.getInstance().updateSourceCode();
        Variables.setCodeChanged(false);
    }
}
```
private static final String optionsXmlFile = "options.xml";

public static void main(String[] args)
{
    Chart.titleColor = Color.BLACK;
}
Chart.titleFont = new Font("Serif", Font.PLAIN, 18);
readOptions();
MainScreen.getInstance();
if( !GrammarInterface.getInstance().loadGrammar("Grammar/" + Kernel.Variables.grammarName + ".xml") )
{
    GuiApi.showMessage("An error occoured while loading grammar file: " + Kernel.Variables.grammarName + ".jar.");
    System.exit(0);
}
GrammarInterface.getInstance().compile(); //to initialize kernel
GuiApi.updateDiagrams(); //to initialize gui
Variables.setCodeChanged(false);
Variables.setDiagramsOutOfSynch(false);

private static void readOptions()
{
    String optionsXml = StdLibrary.readFileAsString(optionsXmlFile);
    if( optionsXml == null )
    {
        JOptionPane.showMessageDialog( null , "There was a problem reading file " + optionsXmlFile + ".");
        System.exit(0);
    }
    try
    {
        XmlNode options = new XmlNode(optionsXml , "1.0");
        Kernel.Variables.grammarName = options.getNode("GUI").getNode("path ").getAttribute("grammarFile");
        return;
    }
SyntaxTrain source code

```java
60     }
61     catch (XMLLoadException e)
62     {
63         }
64     catch (XMLattributeDoesNotExist e)
65     {
66         }
67     catch (XMLnodeDoesNotExist e)
68     {
69         }
70     JOptionPane.showMessageDialog( null, "Invalid xml in option file " + optionsXmlFile + ".jar.");
71     System.exit(1);
72 }
73 }

E.7  Kernel

E.7.1  GrammarBase.java

```
protected File currentFile;

protected GrammarBase()
{
    boolean success = false;
    String jarFileName = Variables.grammarName + 
        ".jar";
    sourceCode = "";
    /*
     * Load jar file
     */
    try
    {
        File jarFile = new File(jarFileName);
        if( jarFile.exists())
        {
            URL[] urls = {jarFile.toURI().toURL()};
            classLoader = new URLClassLoader(urls,
                ClassLoader.getSystemClassLoader());
            success = true;
        }
    }
    catch (MalformedURLException e)
    {
    }
    if( !success )
    {
        JOptionPane.showMessageDialog( null , "Jar 
            file could not be read: " + jarFileName 
        );
        System.exit(1);
    }
}

public void reloadSourceCode()
{
    readSourceCode(currentFile);
}

public void readSourceCode( File file )
{
    currentFile = file;
    if( file == null )
    {

return;
}
sourceCode = StdLibrary.readFileAsString( file );
GuiApi.updateSourceCode( file.getName() );
}

public boolean loadGrammar( String grammarFile )
{
    xmlGrammar = StdLibrary.readFileAsString( classLoader.getResourceAsStream( grammarFile ) );
    return xmlGrammar == null ? false : true;
}

public String getSourceCode()
{
    return sourceCode;
}

public void setSourceCode( String code )
{
    sourceCode = code;
}

public void saveSourceCode() throws IOException
{
    if( sourceCode == null || currentFile == null )
    {
        throw new IOException();
    }
    FileOutputStream stream = new FileOutputStream( currentFile );
    stream.write(sourceCode.getBytes());
    stream.close();
}

E.7.2 GrammarCompiler.java

package Kernel;

public boolean loadGrammar( String grammarFile )
{
    xmlGrammar = StdLibrary.readFileAsString( classLoader.getResourceAsStream( grammarFile ) );
    return xmlGrammar == null ? false : true;
}

public String getSourceCode()
{
    return sourceCode;
}

public void setSourceCode( String code )
{
    sourceCode = code;
}

public void saveSourceCode() throws IOException
{
    if( sourceCode == null || currentFile == null )
    {
        throw new IOException();
    }
    FileOutputStream stream = new FileOutputStream( currentFile );
    stream.write(sourceCode.getBytes());
    stream.close();
}
import java.awt.Color;
import java.awt.Font;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.Stack;
import Exceptions.XMLLoadException;
import Exceptions.XMLattributeDoesNotExist;
import Exceptions.XMLnodeDoesNotExist;
import Xml.XmlNode;
import net.hydromatic.clapham.parser.AlternateNode;
import net.hydromatic.clapham.parser.EbnfNode;
import net.hydromatic.clapham.parser.IdentifierNode;
import net.hydromatic.clapham.parser.LiteralNode;
import net.hydromatic.clapham.parser.OptionNode;
import net.hydromatic.clapham.parser.ProductionNode;
import net.hydromatic.clapham.parser.RepeatNode;
import net.hydromatic.clapham.parser.SequenceNode;

public class GrammarCompiler extends SourceCodeCompiler
{
    protected ArrayList<ProductionNode> productionNodes;
    protected ArrayList<String> grammars;
    private boolean lastHighlightConsumed;

    protected GrammarCompiler()
    {
        super();
        lastHighlightConsumed = false;
    }

    public ArrayList<String> getGrammars()
    {
        return grammars;
    }

    public List<ProductionNode> getProductionNodes()
    {
        return productionNodes;
    }
}
protected void createBnfComponents() {
    productionNodes = new ArrayList<ProductionNode>();
    grammars = new ArrayList<String>();

    if( errorTrace == null || xmlGrammar == null ) {
        return;
    }
    try {
        XmlNode xml = new XmlNode(xmlGrammar,
            Variables.xmlVersion);
        mainForLoop:
        for(XmlNode rule : xml.getChildNodes("rule")) {
            String ruleName = rule.getAttribute("ID");
            grammars.add(ruleName);
            //checks if there's a trace for the given rule
            for( int i=errorTrace.size() - 1; i>=0; i-- ) {
                Stack<String> trace = errorTrace.get(i);
                if( trace.firstElement().equals( ruleName ) ) {
                    @SuppressWarnings("unchecked")
                    Stack<String> clone = (Stack<String>) trace.clone();
                    clone.remove(0); //remove the first element indicating the rule name
                    if( i == errorTrace.size() - 1 ) {
                        if( popLast ) {
                            clone.pop(); //hack, the very last element in the
error trace is wrong
and has to be removed.

productionNodes.add(
    createProductionNodeFromRule
    ( rule, clone, null, true )
);

else
{
    productionNodes.add(
        createProductionNodeFromRule
        ( rule, clone, clone.
            lastElement(), false )
    );
}
    continue mainForLoop;
}

productionNodes.add(
    createProductionNodeFromRule( rule,
        new Stack<String>(), null, false )
);

Collections.sort(grammars);
}
catch (XMLLoadException e)
{
    e.printStackTrace();
    System.out.println("XmlLoadException: " + e
        .toString());
    System.exit(1);
}
catch (XMLnodeDoesNotExist e)
{
    e.printStackTrace();
    System.out.println("XMLnodeDoesNotExist: "
        + e.toString());
    System.exit(1);
}
catch (XMLattributeDoesNotExist e)
{
    e.printStackTrace();
    System.out.println("XMLattributeDoesNotExist: "
        + e.toString());
System.exit(1);
}

private ProductionNode createProductionNodeFromRule(XmlNode rule, Stack<String> highlights, String errorNode, boolean markNextTokens) {
    try {
        if (!rule.getName().equalsIgnoreCase("rule ")) {
            System.out.println("GrammarCompiler returns null - 1");
            return null;
        }
        String id = rule.getAttribute("ID");
        boolean markFirst = highlights.size() == 0;
        SequenceNode seq = createSequenceFromXml(rule, highlights, errorNode, markNextTokens, markFirst);
        return new ProductionNode(new IdentifierNode(id), seq);
    }
    catch (XMLattributeDoesNotExist e) {
        e.printStackTrace();
    }
    catch (XMLnodeDoesNotExist e) {
        e.printStackTrace();
    }
    System.out.println("GrammarCompiler returns null - 2");
    return null;
private SequenceNode createSequenceFromXml(
    XmlNode xml, Stack<String> highlights, String
    errorNode, boolean markNextTokens, boolean
    markThisOne ) throws XMLnodeDoesNotExist,
    XMLattributeDoesNotExist
{
    List<EbnfNode> nodes = new ArrayList<EbnfNode
    >();
    boolean markThis = markThisOne;
    boolean highLightNext = false;

    for(XmlNode child : xml.getAllChildNodes())
    {
        highLightNext = false;
        if( lastHighlightConsumed || markThis )
        {
            lastHighlightConsumed = false;
            markThis = false;
            highLightNext = true;
        }
        nodes.add(createEbnfNodeFromXml(child ,
            highlights, errorNode, markNextTokens,
            highLightNext));
    }
    return new SequenceNode(nodes);
}

private EbnfNode createEbnfNodeFromXml( XmlNode
    xml, Stack<String> highlights, String
    errorNode, boolean markNextTokens, boolean
    markThisOne ) throws XMLnodeDoesNotExist,
    XMLattributeDoesNotExist
{
    String nodeName = xml.getName();
    boolean highLightNext = false;

    if( lastHighlightConsumed || markThisOne )
    {
        lastHighlightConsumed = false;
        highLightNext = true;
    }
    if( nodeName.equalsIgnoreCase("rule") )
    {
    }
String id = xml.getAttribute("ID");
String uuid = xml.getAttribute("UUID");
Font font;
if( id.startsWith("'") )
{
    font = new Font( "Courier New", Font.PLAIN, 14 );

    if( uuid.equals(errorNode) )
        return new LiteralNode(id, Color.RED, Color.BLACK, font);
    else if( (highlightNext || markThisOne)
        && markNextTokens )
        return new LiteralNode(id, Variables.highlightColor, Variables.highlightColor, font);
    else if( highlights.remove( uuid ) )
    {
        if( highlights.isEmpty() )
        {
            lastHighlightConsumed = true;
        }
        return new LiteralNode(id, Color.BLUE, Color.BLUE, font);
    }
    else
        return new LiteralNode(id, Color.BLACK, Color.BLACK, font);
}
else
{
    font = new Font( "Serif", Font.PLAIN, 14 );

    if( uuid.equals(errorNode) )
        return new IdentifierNode(id, Color.RED, Color.BLACK, font);
    else if( (highlightNext || markThisOne)
        && markNextTokens )
        return new IdentifierNode(id, Variables.highlightColor, Variables.highlightColor, font);
    else if( highlights.remove( uuid ) )
    {
        font = new Font( "Courier New", Font.PLAIN, 14 );

        if( uuid.equals(errorNode) )
            return new IdentifierNode(id, Color.RED, Color.BLACK, font);
        else if( (highlightNext || markThisOne)
            && markNextTokens )
            return new IdentifierNode(id, Variables.highlightColor, Variables.highlightColor, font);
        else if( highlights.remove( uuid ) )
        {
if( highlights.isEmpty() )
{
    lastHighlightConsumed = true;
}
return new IdentifierNode(id, Color.BLUE, Color.BLUE, font);
else
    return new IdentifierNode(id, Color.BLACK, Color.BLACK, font);

else if( nodeName.equalsIgnoreCase("repeat"))
{
    SequenceNode repeatSequence = createSequenceFromXml(xml, highlights, errorNode, markNextTokens, highLightNext);
    lastHighlightConsumed = highLightNext; // the next token should also be highlighted
    return new RepeatNode(repeatSequence);
}
else if( nodeName.equalsIgnoreCase("or"))
{
    List<EbnfNode> nodes = new ArrayList<EbnfNode>();
    for(XmlNode child : xml.getChildNodes("option"))
    {
        nodes.add(createEbnfNodeFromXml(child, highlights, errorNode, markNextTokens, highLightNext));
    }
    return new AlternateNode(nodes);
}
else if( nodeName.equalsIgnoreCase("option"))
{
    //this section is called from the "or" node above
    return createSequenceFromXml(xml, highlights, errorNode, markNextTokens, highLightNext);
}
else if (nodeName.equalsIgnoreCase("optional")

SequenceNode optionalSequence =
    createSequenceFromXml(xml, highlights, errorNode, markNextTokens, highLightNext);
lastHighlightConsumed = highLightNext; // the next token should also be highlighted
return new OptionNode(optionalSequence);
}
throw new XMLnodeDoesNotExist("Unknown xml node.");

E.7.3 GrammarInterface.java

```java
class GrammarInterface extends GrammarCompiler
{
    private static GrammarInterface instance = null;
    private GrammarInterface()
    {
        super();
    }
    public void compile()
    {
        compileSourceCode();
        createBnfComponents();
        GuiApi.updateDiagrams();
    }
    public static synchronized GrammarInterface getInstance()
    {
        return instance;
    }
}
```
```java
{
    if (instance == null)
    {
        instance = new GrammarInterface();
    }
    return instance;
}
```

### E.7.4 SourceCodeCompiler.java

```java
package Kernel;

import java.io.ByteArrayInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.UnsupportedEncodingException;
import java.lang.reflect.Constructor;
import java.lang.reflect.InvocationTargetException;
import java.util.Stack;
import org.antlr.runtime.ANTLRInputStream;
import org.antlr.runtime.CharStream;
import org.antlr.runtime.CommonTokenStream;
import org.antlr.runtime.Lexer;
import org.antlr.runtime.RecognitionException;
import org.antlr.runtime.TokenStream;
import Grammar.BnfParser;

public class SourceCodeCompiler extends GrammarBase {
    /**
     * Stack of rules which are in the error (top down).
     * Each rule is a stack trace indicating which steps in these rules have been taken (starting with the rule name as the first element)
     *
     * Example:
     * compilationUnit
     */
```
typeDeclaration
classDeclaration 'class' IDENTIFIER '{'
fieldDeclaration
*
* This means the last '}' is missing (if
fieldDeclaration weren't there, the error
would be in that.
*/
protected Stack<Stack<String>> errorTrace;
protected int errorLine, errorCharPositionInLine;
protected boolean popLast;
private Constructor<Lexer> LexerConstructor;
private Constructor<BnfParser> ParserConstructor;
protected SourceCodeCompiler()
{
    super();
    boolean success = false;
    errorTrace = null;
    errorLine = -1;
    errorCharPositionInLine = -1;
    popLast = false;

    /*
    * Load classes
    */
    try
    {
        @SuppressWarnings("unchecked")
        Class<Lexer> CLexer = (Class<Lexer>) Class.forName("Grammar." + Variables.grammarName + "Lexer", false, classLoader);
        @SuppressWarnings("unchecked")
        Class<BnfParser> CParser = (Class<BnfParser>) Class.forName("Grammar." + Variables.grammarName + "Parser", false, classLoader);
        LexerConstructor = CLexer.getConstructor(new Class[]{CharStream.class});
        ParserConstructor = CParser.getConstructor(new Class[]{TokenStream.class});
    }
//A little check to make sure files can be loaded
InputStream is = new ByteArrayInputStream((new String()).getBytes("UTF-8"));
ANTLRInputStream input = new ANTLRInputStream(is);
Lexer lexer = LexerConstructor.newInstance(new Object[]{input});
CommonTokenStream tokens = new CommonTokenStream(lexer);
@SuppressWarnings("unused")
BNFParser parser = ParserConstructor.newInstance(new Object[]{tokens});

success = true;
}
catch (SecurityException e)
{
  e.printStackTrace();
}
catch (ClassNotFoundException e)
{
  e.printStackTrace();
}
catch (NoSuchMethodException e)
{
  e.printStackTrace();
}
catch (UnsupportedEncodingException e)
{
  e.printStackTrace();
}
catch (IllegalArgumentException e)
{
  e.printStackTrace();
}
catch (IOException e)
{
  e.printStackTrace();
}
catch (InstantiationException e)
{  
e.printStackTrace();  
}
catch (IllegalAccessException e)  
{  
e.printStackTrace();  
}  
catch (InvocationTargetException e)  
{  
e.printStackTrace();  
}
if( !success )  
{  
    System.out.println("Invalid Jar file!");  
    System.exit(1);  
}

public Stack<Stack<String>> getErrorTrace()  
{  
    return errorTrace;  
}

public int getErrorLine()  
{  
    return errorLine;  
}

public int getErrorCharPositionInLine()  
{  
    return errorCharPositionInLine;  
}

protected void compileSourceCode()  
{  
    try  
    {  
        if( getSourceCode() == null )  
            return;  
        InputStream is = new ByteArrayInputStream(  
            getSourceCode().getBytes("UTF-8"));  
        ANTLRInputStream input = new  
            ANTLRInputStream(is);  
    }  
}
Lexer lexer = LexerConstructor.newInstance(
    new Object[]{input});

CommonTokenStream tokens = new
    CommonTokenStream(lexer);

BnfParser parser = ParserConstructor.
    newInstance(new Object[]{tokens});

try
{
    parser.bnf();
    //System.out.println("No errors :)");
}
catch(RuntimeException e)
{
    //System.out.println("Failed, errors in
code :(");
}
errorTrace = parser.trace;
errorLine = parser.errorLine;
errorCharPositionInLine = parser.
    errorCharPositionInLine;
popLast = parser.popLast;

return;
}
catch (UnsupportedEncodingException e)
{
}
}
catch (IOException e)
{
}
catch (RecognitionException e)
{
}
catch (InstantiationException e1)
{
    System.out.println("Invalid Jar file!");
    System.exit(1);
}
catch (IllegalAccessException e1) {
    System.out.println("Invalid Jar file!");
    System.exit(1);
catch (InvocationTargetException e1) {
    System.out.println("Invalid Jar file!");
    System.exit(1);
}
errorTrace = null;
}

E.7.5 Variables.java

package Kernel;
import java.awt.Color;
import java.util.Hashtable;
public class Variables {
    public static final String xmlVersion = "1.0";
    public static String grammarName = "invalid";
    public static Hashtable <String , String> idToVariable;
    public static final Color highlightColor = new Color(255,150,50);
}

E.8 KernelAPI

E.8.1 KernelApi.java

package KernelAPI;
import java.io.File;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import java.util.Stack;
import net.hydromatic.clapham.parser.ProductionNode;
import Kernel.GrammarInterface;

public class KernelApi {

    public static String getSourceCode() {
        return GrammarInterface.getInstance().getSourceCode();
    }

    public static Stack<Stack<String>> getErrorTrace() {
        return GrammarInterface.getInstance().getErrorTrace();
    }

    public static ArrayList<String> getGrammars() {
        return GrammarInterface.getInstance().getGrammars();
    }

    public static List<ProductionNode> getGrammarProductionNodes() {
        return GrammarInterface.getInstance().getProductionNodes();
    }

    public static int getErrorLine() {
        return GrammarInterface.getInstance().getErrorLine();
    }

    public static int getErrorCharPositionInLine() {
    }
}
return GrammarInterface.getInstance().getErrorCharPositionInLine();
}

public static void saveSourceCode() throws IOException {
    GrammarInterface.getInstance().saveSourceCode();
}

public static void setSourceCode(String code) {
    GrammarInterface.getInstance().setSourceCode(code);
    GrammarInterface.getInstance().compile();
}

public static void readSourceFile(File file) {
    GrammarInterface.getInstance().readSourceCode(file);
    GrammarInterface.getInstance().compile();
}

public static void reloadSourceCode() {
    GrammarInterface.getInstance().reloadSourceCode();
    GrammarInterface.getInstance().compile();
}

E.9 Library

E.9.1 Lock.java

1 package Library;
2
3 public class Lock {
4     private boolean isTaken;
5 }
E.9 Library

```java
public Lock()
{
    isTaken = false;
}

public synchronized void P()
{
    while( isTaken )
    {
        try {
            wait();
        } catch (InterruptedException e)
        {
        }
    }
    isTaken = true;
}

public synchronized void V()
{
    isTaken = false;
    notify();
}
}
```

E.9.2 StdLibrary.java

```java
package Library;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.Reader;

public class StdLibrary
{
    public static String xmlEscapeString( String str
```
{ return str.replace("\"", "\"\"").replace("\", "\&\"").replace("\", "\&apos\"").replace("\", "\&apos\"").replace("\", "\>\""); }

public static String xmlUnEscapeString( String str )
{
    return str.replace("\"", "\"\"").replace("\", "\&\"").replace("\", "\&apos\"").replace("\", "\&apos\"").replace("\", "\>\"");
}

public static String readFileAsString(InputStream inputStream)
{
    if( inputStream == null )
    {
        return null;
    }
    return readFileAsString(new InputStreamReader(inputStream));
}

public static String readFileAsString(File file)
{
    try
    {
        return readFileAsString(new FileInputStream(file));
    }
    catch (FileNotFoundException e)
    {
        return null;
    }
}

/**
 * Reads a file and returns it’s content
 * @param filePath path of the file to read
 * @return Content of the file specified, returns null if file does not exist or there’s an IOException.
 */
public static String readFileAsString(String filePath)
{
    try
    {
        return readFileAsString(new FileReader(filePath));
    }
    catch (FileNotFoundException e)
    {
        return null;
    }
}

public static String readFileAsString(Reader reader)
{
    try{
        StringBuffer fileData = new StringBuffer(1000);
        char[] buf = new char[1024];
        int numRead=0;
        while((numRead=reader.read(buf)) != -1){
            String readData = String.valueOf(buf, 0, numRead);
            fileData.append(readData);
            buf = new char[1024];
        }
        reader.close();
        return fileData.toString();
    }
    catch (IOException e) {}
    return null;
}

E.10 Xml

E.10.1 XmlNode.java
package Xml;

import java.io.StringReader;
import java.util.ArrayList;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;

import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.NamedNodeMap;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.xml.sax.InputSource;

import Exceptions.XMLLoadException;
import Exceptions.XMLTextDoesNotExist;
import Exceptions.XMLattributeDoesNotExist;
import Exceptions.XMLnodeDoesNotExist;

public class XmlNode
{
    private Node node;
    private Node currentNodeGiven;

    public XmlNode(String xml , String version) throws XMLLoadException
    {
        String xmlVersion;
        currentNodeGiven = null;
        try
        {
            DocumentBuilderFactory dbf =
            DocumentBuilderFactory.newInstance();
            DocumentBuilder db = dbf.newDocumentBuilder();
            Document doc = db.parse(new InputSource(new StringReader(xml)));
            node = doc.getDocumentElement();
            xmlVersion = getAttribute("version");
        }
        catch (Exception e)
        {
            throw new XMLLoadException("XmlNode -> the string given isn't xml: " + xml);
        }
    }
}
40 }  
41 // This has to be outside the try catch, otherwise the catch will just catch this exception too and it’s message won’t be sent on!  
42 if(!xmlVersion.equals(version))  
43 throw new XMLLoadException("XmlNode -> version expected: " + version + " got: " + xmlVersion");

44 }
45 private XmlNode(Node node)
46 {
47   this.node = node;
48 }
49
50 public XmlNode getNextNode() throws XMLnodeDoesNotExist
51 {
52   if(currentNodeGiven == null)
53   {
54     currentNodeGiven = node.getFirstChild();
55     if(currentNodeGiven.getNodeType() != Node.ELEMENT_NODE)
56     return getNextNode();
57     return new XmlNode(currentNodeGiven);
58   }
59   currentNodeGiven = currentNodeGiven.getNextSibling();
60   if(currentNodeGiven == null)
61     throw new XMLnodeDoesNotExist("<Next node >");
62   if(currentNodeGiven.getNodeType() != Node.ELEMENT_NODE)
63     return getNextNode();
64   return new XmlNode(currentNodeGiven);
65 }
66 public boolean hasNextNode()
67 {
68   Node temp = currentNodeGiven;
69   if(currentNodeGiven == null)
70     if(node.hasChildNodes())
71       temp = node.getFirstChild();
72     else
73       return false;
Node sibling = temp.getNextSibling();
while(sibling != null)
{
    if(sibling.getNodeType() == Node.ELEMENT_NODE)
        return true;
    sibling = sibling.getNextSibling();
}
return false;

public String getName()
{
    return node.getNodeName();
}

public ArrayList<XmlNode> getAllChildNodes()
throws XMLnodeDoesNotExist
{
    ArrayList<XmlNode> childNodes = new ArrayList<XmlNode>();
    NodeList nodes = node.getChildNodes();
    for(int i=0; i<nodes.getLength(); i++)
    {
        Node node = nodes.item(i);
        if (node.getNodeType() == Node.ELEMENT_NODE)
            childNodes.add(new XmlNode(node));
    }
    return childNodes;
}

public ArrayList<XmlNode> getChildNodes(String
nodeName) throws XMLnodeDoesNotExist
{
    ArrayList<XmlNode> childNodes = new ArrayList<XmlNode>();
    NodeList nodes = node.getChildNodes();
    for(int i=0; i<nodes.getLength(); i++)
    {
        Node node = nodes.item(i);
        if (node.getNodeType() == Node.ELEMENT_NODE)
            if(((Element) node).getTagName().
equalsIgnoreCase(nodeName))
                childNodes.add(new XmlNode(node));
    }
}
public XmlNode getChildNode(String nodeName, int skipAmount) throws XMLnodeDoesNotExist
{
    NodeList nodes = node.getChildNodes();
    for(int i=0; i<nodes.getLength(); i++)
    {
        Node node = nodes.item(i);
        if (node.getNodeType() == Node.ELEMENT_NODE)
            if(((Element) node).getTagName().equalsIgnoreCase(nodeName))
                if(skipAmount-- > 0)
                    continue;
                else
                    return new XmlNode(node);
    }
    throw new XMLnodeDoesNotExist( nodeName );
}

public XmlNode getChildNode(String nodeName) throws XMLnodeDoesNotExist
{
    return getChildNode(nodeName , 0);
}

public int getIntAttribute(String attName) throws XMLattributeDoesNotExist
{
    return Integer.parseInt(getAttribute(attName));
}

public boolean getBooleanAttribute( String attName ) throws XMLattributeDoesNotExist
{
    return Boolean.parseBoolean( getAttribute( attName ) );
}

public String getAttribute(String attName) throws XMLattributeDoesNotExist
{
    NamedNodeMap nodeMap = node.getAttributes();
    for(int i=0;i<nodeMap.getLength();i++)
    {
        Node node = nodeMap.item(i);
if (node.getNodeType() != Node.ATTRIBUTE_NODE)
{
    System.out.println("XmlNode -
            getAttribute: Node: " + node + " is
            not an attribute! :S");
}
if (node.getNodeName().equalsIgnoreCase(attName))
{
    return node.getNodeValue();
}
throw new XMLattributeDoesNotExist(attName);
}
public String getText() throws XMLTextDoesNotExist
{
    NodeList nodes = node.getChildNodes();
    for (int i = 0; i < nodes.getLength(); i++)
    {
        Node node = nodes.item(i);
        if (node.getNodeType() == Node.TEXT_NODE)
            return node.getNodeValue();
    }
    throw new XMLTextDoesNotExist();
}
Bibliography


