A Software Tool for Learning Syntax

Andreas Leon Aagaard Moth

Kongens Lyngby 2011
IMM-BSc-2011-11

Technical University of Denmark
Informatics and Mathematical Modelling
Building 321, DK-2800 Kongens Lyngby, Denmark
Phone +45 45253351, Fax +45 45882673
reception@imm.dtu.dk
www.imm.dtu.dk

IMM-BSc: ISSN 0909-3192

Preface

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.

Resumé

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.

Acknowledgements

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

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 Pascal; 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 bacck 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 maches 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 represents 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.

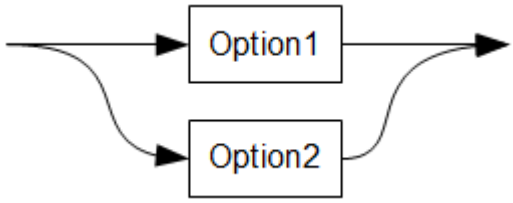

Figure 2.1: Syntax diagram with multiple options.

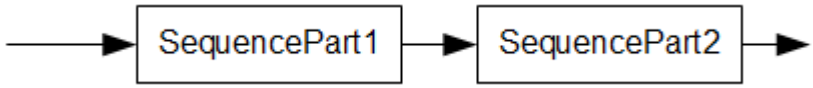

Figure 2.2: Syntax diagram of a sequence.

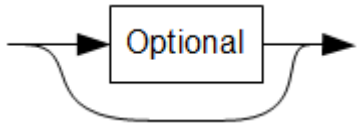

Figure 2.3: Syntax diagram of an optional element.

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.

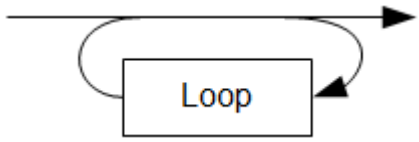

Figure 2.4: Syntax diagram containing a loop.

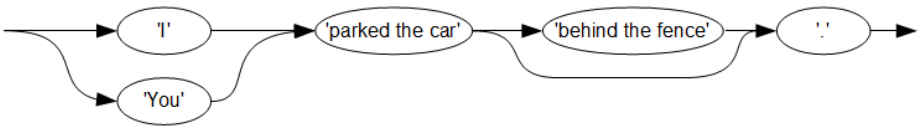

Figure 2.5: Simple syntax diagram.

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:

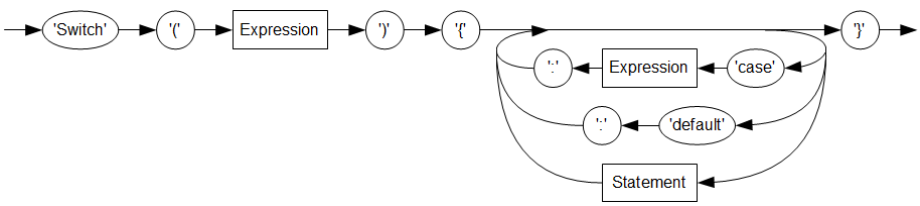

Figure 2.6: Classical `switch_statement` in Java.

- **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_satement` in JAVA, like the one shown in Figure 2.6, but with colors.

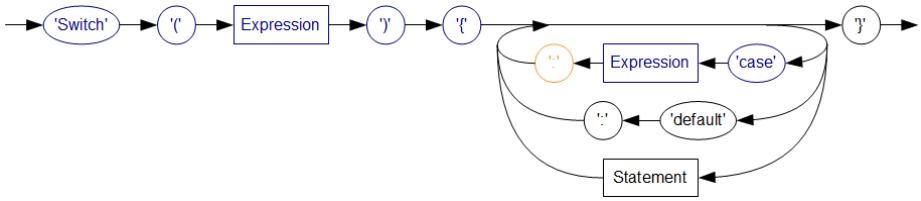

Figure 2.7: Colored `switch_statement` in JAVA.

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 shows 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 langauge at 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.

```
1 bnfStartRule = rule "." <rule ".">.
2 rule = IDENTIFIER "=" expression.
3 expression = orExpression <orExpression>.
4 orExpression = exprBase < "/" exprBase >.
5 exprBase = STRING
6       / IDENTIFIER
7       / ( "(" expression ")" )
8       / ( "[" expression "]" )
9       / ( "<" expression ">" ).
```

Figure 2.8: BNF grammar example.

Chapter 3

Source code verification

This chapter will explains 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.

```
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-

ure 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.

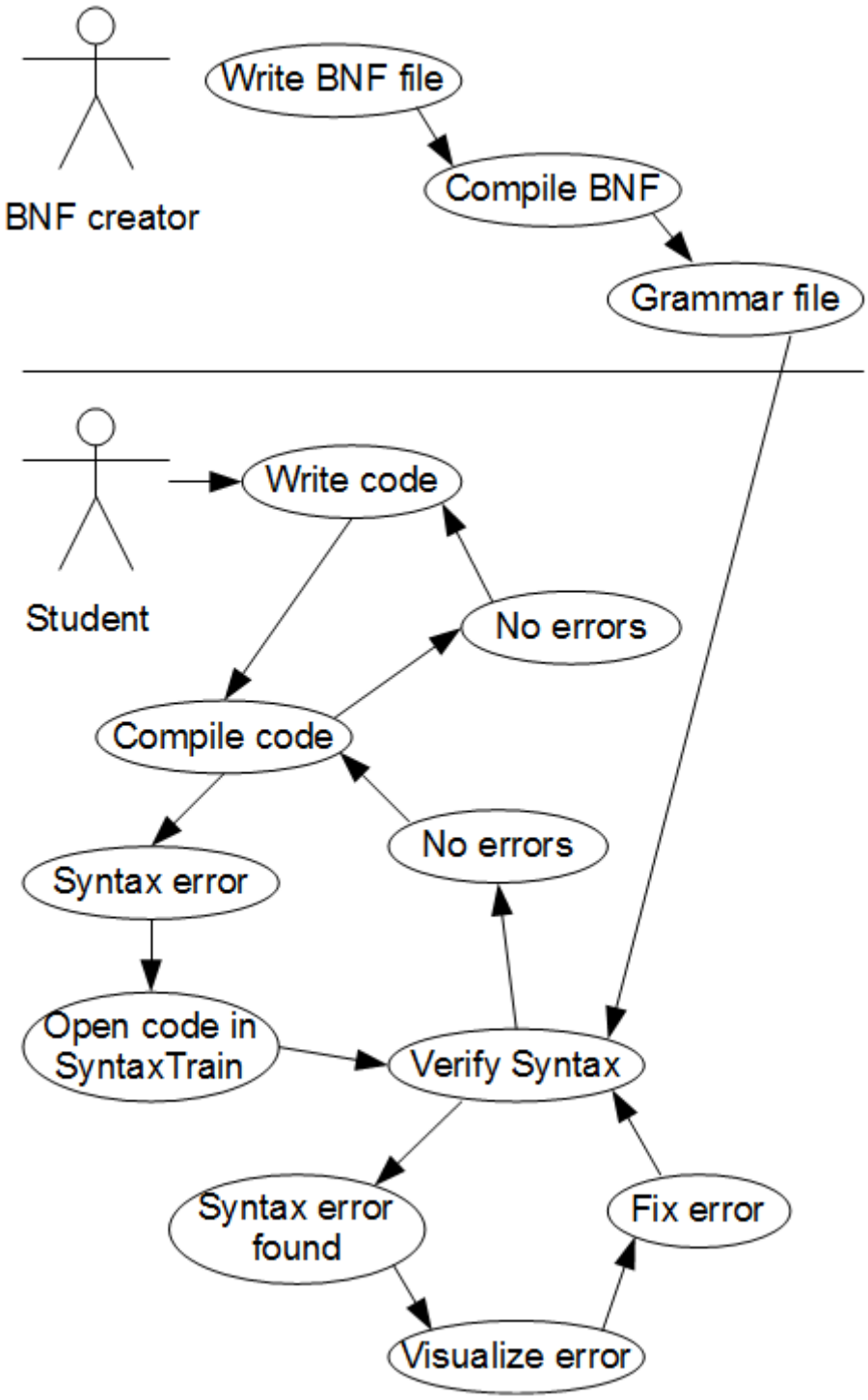

Figure 3.2: Use case for creating grammar file and verifying code.

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.

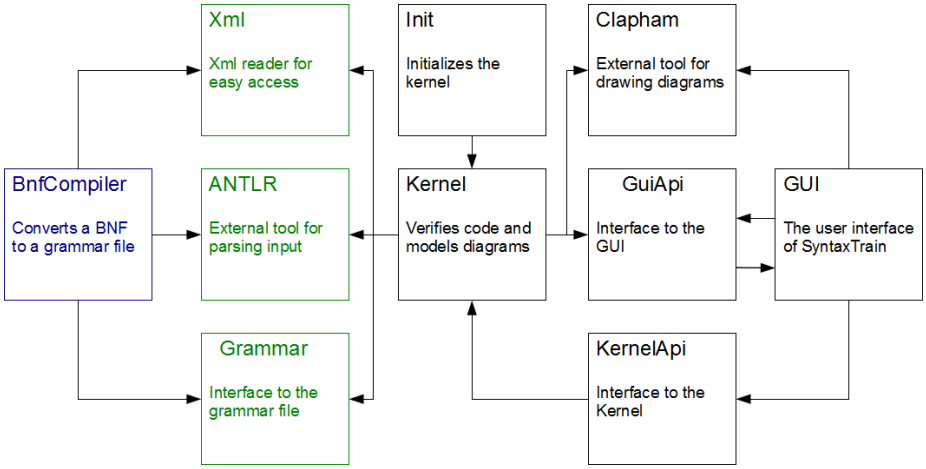

Figure 4.1: Block diagram with BNF compiler in focus.

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.

```
myRule = ( myRule "a" ) | "b".
```

Figure 4.2: Left-recursive rule - 1.

```
myRule = ( anotherRule "a" ) | "b".
anotherRule = 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

```
1 for_statement =
2    "for"  "("  ( variable_declaration  /  (
        expression  ";"  )  /  ";"  )
3       [ expression ]  ";"
4       [ expression ]
5    ")" statement  .
```

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

```
1 for_statement =
2    sequenceLink(
3       Link("for"),
4       Link("("),
5       orLink(
6          Link(variable_declaration),
7          sequeneLink(
8             Link(expression),
9             Link(";")
10         ),
11         Link(";")
12       ),
13      optionalLink(expression),
14      Link(";"),
15      optionalLink(expression),
16      Link(")"),
17      Link(statement)
18   );
```

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.

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[1]. 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.

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

Figure 4.6: For-in Java loop.

[1] See: http://www.cui.unige.ch/db-research/Enseignement/analyseinfo/JAVA/BNFindex.html

Chapter 5

SyntaxTrain GUI design

This chapter will focus on the different parts of the GUI. It starts by explaining what is needed followed by how this in 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 understands 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 contining 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.

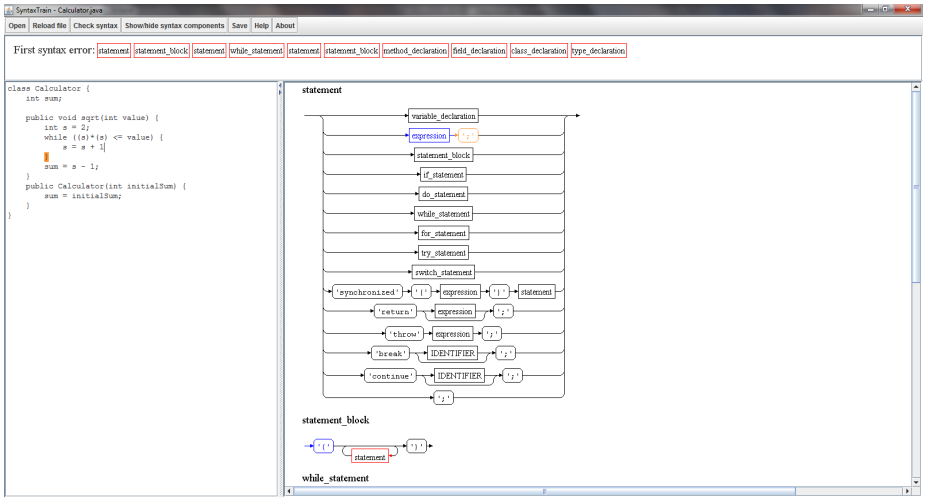

Figure 5.1: Screen shot of SyntaxTrain.

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 undrstands 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.

Chapter 6

SyntaxTrain implementation

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

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.

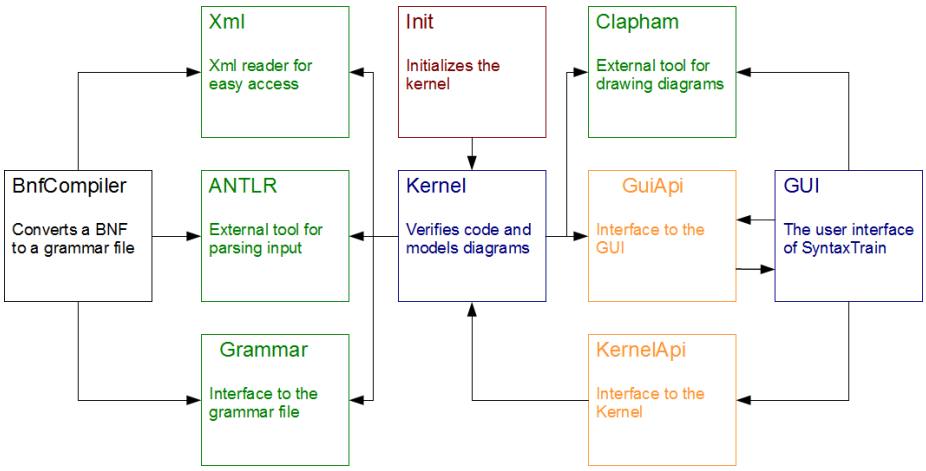

Figure 6.1: Block diagram with SyntaxTrain in focus.

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 colored 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.

Chapter 7

Limitations and improvements

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 defficult 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 components 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.

Chapter 8

SyntaxTrain usage examples

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.

```
1  class Dates {
2     int dayInMonth(int month) {
3        switch (month) {
4           case 1: case 3: case 5: case 7: case 8:
                case 10: case 12:
5              return 31;
6           case 4: case 6: case 9: case 11:
7              return 30;
8           case: 2
9              return 28;
10          default:
11             return 0;
12       }
13    }
14 }
```

Figure 8.1: Example 1.

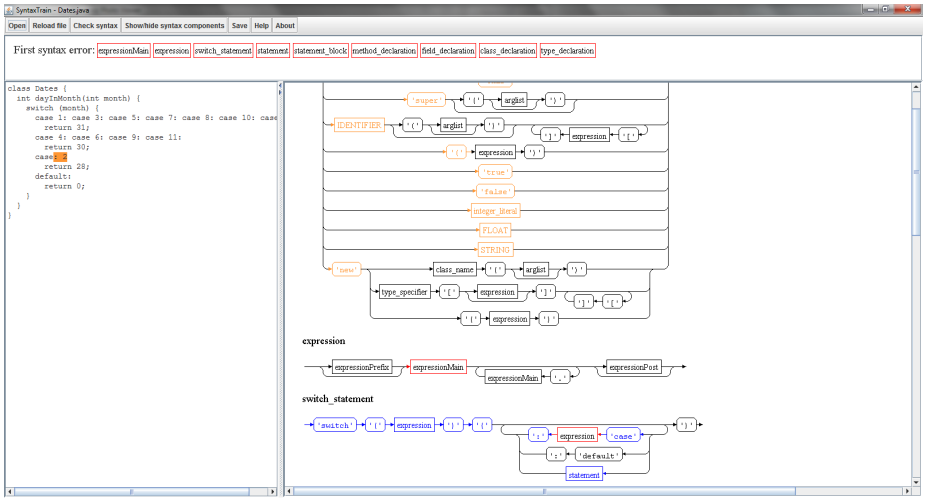

Figure 8.2: The Date class syntax error displayed by SyntaxTrain.

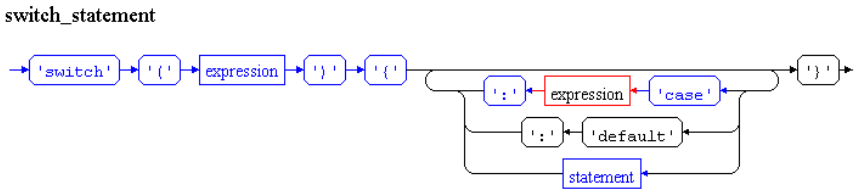

Figure 8.3: The rule for the `switch` statement.

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.

```
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.

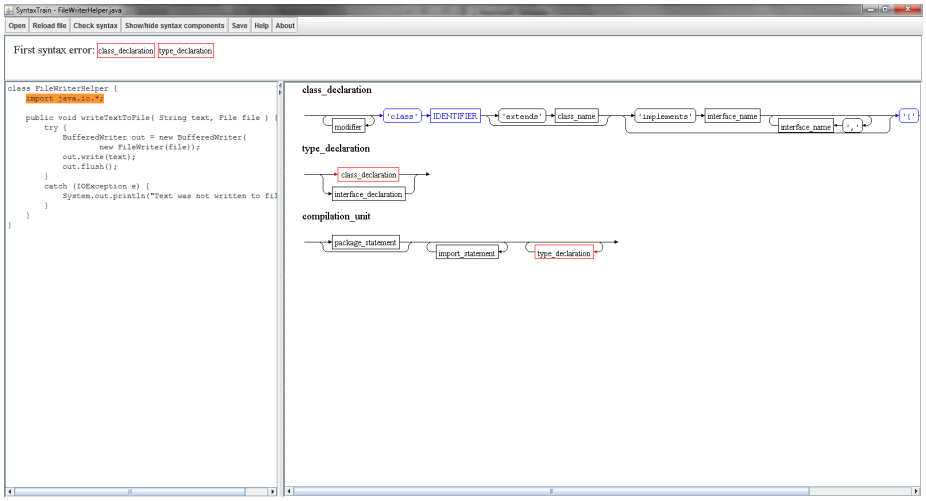

Figure 8.5: FileWriterHelper opened in SyntaxTrain.

Figure 8.6: The `class_declaration` statement shown by 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.

```
1 class Calculator {
2    int sum;
3
4    public void sqrt() {
5       int s = 2;
6       while ((s)*(s) <= sum) {
7          s = s + 1;
8       }
9       sum = s - 1;
10   }
11   public Calculator(int initialValue) {
12      sum = initialValue;
13   }
14 }
```

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

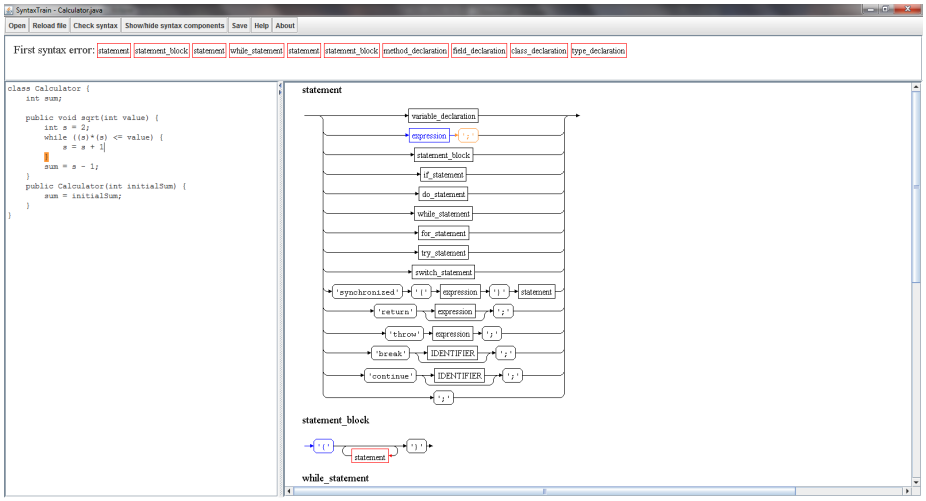

Figure 8.8: Screenshot of a syntax error where a semicolon is missing.

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.

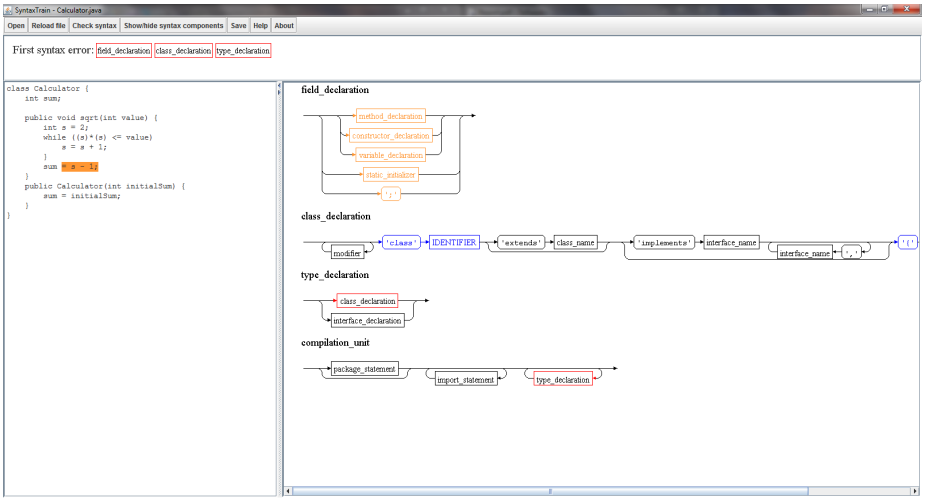

Figure 8.9: Screenshot of a syntax error where the **{** is missing.

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 `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-

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 = "";
```

```
24    HashMap<String, Link> ruleNameToLink = new
          HashMap<String, Link>();
25
26    public void displayRecognitionError(String[]
          tokenNames, RecognitionException e) throws
          RuntimeException
27        {
28        throw new RuntimeException();
29        }
30 }
31
32 bnf    :  startRule '.' (rule '.')* EOF;
33 startRule
34    :  ruleName '=' expression
35       {startRule = $ruleName.value; ruleNameToLink.
             put($ruleName.value, $expression.value);};
36
37 rule   :  ruleName '=' expression
38       {ruleNameToLink.put($ruleName.value,
             $expression.value);};
39
40 ruleName returns [String value]
41    :  NONTERMINAL {$value=$NONTERMINAL.text;};
42
43 expression returns [Link value]
44    : {Link link = new Link(); $value = link;}
45      ( expr {link.add($expr.value);} )+;
46
47 expr returns [Link value]
48    : {Link orLink = null;}
49      start=exprBase {Link link = $start.value;
            $value = link;}
50      ( '/' base=exprBase
51      {
52       if(orLink == null)
53       {
54          orLink = new Link(link);
55          orLink.oneOfMultiple();
56          $value = orLink;
57       }
58       orLink.add($base.value);
59      })*;
60
61 exprBase returns [Link value]
```

```
62     :  TERMINAL {$value = new Link ($TERMINAL.text )
          ;}
63        | NONTERMINAL {$value = new Link($NONTERMINAL.
             text);}
64        | '(' expression ')' {$value = $expression.
             value;}
65        | '[' expression ']' {$value = $expression.
             value; $value.optional();}
66        | '<' expression '>' {$value = $expression.
             value; $value.loop();};
67
68 TERMINAL
69     :  '"' (~'"')* '"';
70 NONTERMINAL  : ('a'..'z'|'A'..'Z'|'0'..'9'|'_')+;
71
72 WS  :   ( ' '
73          | '\t'
74          | '\r'
75          | '\n'
76          ) {$channel=HIDDEN;}
77     ;
```

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<
          Stack<String>>();
19    public void displayRecognitionError(String[]
          tokenNames, RecognitionException e) throws
          RuntimeException
20       {
```

```
21        throw new RuntimeException();
22        }
23 }
24
25 compilationUnit
26     :    {Stack<String> tr = new Stack<String>();
           trace.push(tr); tr.push("compilationUnit");}
27     (typeDeclaration )*
28     {trace.pop();}
29     ;
30
31 typeDeclaration
32     :  {Stack<String> tr = new Stack<String>(); trace
          .push(tr); tr.push("typeDeclaration");}
          classDeclaration {trace.pop();};
33
34 classDeclaration
35     : {Stack<String> tr = new Stack<String>(); trace
           .push(tr); tr.push("classDeclaration");}
36     'class' {tr.push("'class'");}
37     IDENTIFIER {tr.push("IDENTIFIER");}
38     '{' {tr.push("'{'");}
39     (fieldDeclaration {tr.push("fieldDeclaration");}
           )*
40     '}'
41     {trace.pop();};
42
43 fieldDeclaration
44     :  {Stack<String> tr = new Stack<String>(); trace
          .push(tr); tr.push("fieldDeclaration");}
45    methodDeclaration
46    {trace.pop();};
47
48 methodDeclaration
49     :  {Stack<String> tr = new Stack<String>(); trace
          .push(tr); tr.push("methodDeclaration");}
50    type {tr.push("type");}
51    IDENTIFIER {tr.push("IDENTIFIER");}
52    '(' {tr.push("'('");}
53    ')' {tr.push("')'");}
54    ';'
55    {trace.pop();};
56
```

```
57 type  :  {Stack<String> tr = new Stack<String>();
       trace.push(tr); tr.push("type");}
58     typeSpecifier {tr.push("typeSpecifier");}
59     (  '[' {tr.push("'['");}
60     ']' )*
61     {trace.pop();};
62 typeSpecifier
63     :  {Stack<String> tr = new Stack<String>(); trace
          .push(tr); tr.push("typeSpecifier");} (
64        'boolean'
65           | 'byte'
66           | 'char'
67           | 'short'
68           | 'int'
69           | 'float'
70           | 'long'
71           | 'double'
72           | packageName ) {trace.pop();};
73
74 packageName
75     :  {Stack<String> tr = new Stack<String>(); trace
          .push(tr); tr.push("packageName");}
76     ( IDENTIFIER {tr.push("IDENTIFIER");}
77     '.' )* {tr.push("'.'");}
78     IDENTIFIER
79     {trace.pop();};
80
81
82
83
84 IDENTIFIER  :  ('a'..'z'|'A'..'Z'|'_') ('a'..'z'|'A
       '..'Z'|'0'..'9'|'_')* ;
85
86
87 WS  :   ( ' '
88           | '\t'
89           | '\r'
90           | '\n'
91           ) {$channel=HIDDEN;}
92      ;
```

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  <  ","
       interface_name >  ]
20  "{"  < field_declaration >  "}"  .
21
22 interface_declaration  =
```

```
23  < modifier >  "interface" IDENTIFIER
24  [  "extends" interface_name  <  "," interface_name
       >  ]
25  "{"  < field_declaration >  "}"  .
26
27 field_declaration  =
28  (  method_declaration
29  / constructor_declaration
30  / variable_declaration )
31  / static_initializer
32  /  ";"  .
33
34 method_declaration  =
35  < modifier > type IDENTIFIER
36  "("  [ parameter_list ]  ")"  <  "["  "]"  >
37  ( statement_block  /  ";"  )  .
38
39 constructor_declaration  =
40  < modifier > IDENTIFIER  "("  [ parameter_list ]
       ")"
41 statement_block  .
42
43 statement_block  =  "{"  < statement >  "}"  .
44
45 variable_declaration  =
46  < modifier > type variable_declarator
47  <  "," variable_declarator  >  ";"  .
48
49 variable_declarator  =
50  IDENTIFIER  <  "["  "]"  >  [  "="
       variable_initializer ]  .
51
52 variable_initializer  =
53 expression
54  /  (  "{"  [ variable_initializer
55  <  "," variable_initializer  >  [  ","  ]  ]  "}"
       )  .
56
57 static_initializer  =
58  "static" statement_block  .
59
60 parameter_list  =
61 parameter  <  "," parameter >  .
62
```

```
63 parameter  =
64 type IDENTIFIER  <  "["  "]"  >  .
65
66 statement  =
67 variable_declaration
68  /  ( expression  ";"  )
69  /  ( statement_block )
70  /  ( if_statement )
71  /  ( do_statement )
72  /  ( while_statement )
73  /  ( for_statement )
74  /  ( try_statement )
75  /  ( switch_statement )
76  /  (  "synchronized"  "(" expression  ")" statement
        )
77  /  (  "return"  [ expression ]  ";"  )
78  /  (  "throw" expression ";"  )
79  /  (  "break"  [ IDENTIFIER ]  ";"  )
80  /  (  "continue"  [ IDENTIFIER ]  ";"  )
81  /  (  ";"  )  .
82
83 if_statement =
84  "if"  "(" expression  ")" statement_block
85  [  "else" statement ]  .
86
87 do_statement =
88  "do" statement_block  "while"  "(" expression  ")"
       ";"  .
89
90 while_statement =
91  "while"  "(" expression  ")" statement  .
92
93 for_statement =
94  "for"  "("  ( variable_declaration  /  ( expression
       ";"  )  /  ";"  )
95  [ expression ]  ";"
96  [ expression ]
97  ")" statement  .
98
99 try_statement =
100  "try" statement_block
101  <  "catch"  "(" parameter  ")" statement_block >
102  [  "finally" statement_block ]  .
103
```

```
104 switch_statement =
105  "switch"  "(" expression  ")"  "{"
106  <  (  "case" expression  ":"  )
107  /  (  "default"  ":"  )
108  / statement >
109  "}"  .
110
111 expression  =
112    [expressionPrefix]
113       expressionMain
114       < "." expressionMain >
115    [expressionPost].
116
117 expressionMain =
118      "null"
119    / "this"
120    / ( "super" [ "(" [ arglist ] ")" ] )
121    / ( IDENTIFIER [ "(" [ arglist ] ")" ] < "["
          expression "]" > )
122    / ( "(" expression ")" )
123    / "true"
124    / "false"
125    / integer_literal
126    / FLOAT
127    / STRING
128    / (
129       "new" (
130            ( class_name "(" [arglist] ")" )
131          / ( type_specifier "[" [ expression ] "]" <
                "[" "]" > )
132          / ( "(" expression ")" )
133       )
134      ).
135
136 expressionPrefix =
137    "-"
138  / "++"
139  / "--"
140  / "!"
141  / "~".
142
143 expressionPost =
144    "++"
145  / "--"
```

```
146  / ( (
147        "="
148     / "+"
149       / "+="
150       / "-"
151       / "-="
152       / "*"
153       / "*="
154       / "/"
155       / "/="
156       / "%"
157       / "%="
158     /   ">"
159       / "<"
160       / ">="
161       / "<="
162       / "=="
163       / "!="
164     / "&"
165       / "&="
166       / "|"
167       / "|="
168       / "^"
169       / "^="
170       / "&&"
171       / "||="
172     / ("?" expression ":")
173     / ">>="
174     / "<<"
175     / ">>"
176     / ">>>"
177     ) expression )
178     .
179
180 arglist  =
181 expression  <  "," expression  >  .
182
183 type  =
184 type_specifier  <  "["  "]"  >  .
185
186 type_specifier =
187  "boolean"
188  /  "byte"
189  /  "char"
```

```
190  /  "short"
191  /  "int"
192  /  "float"
193  /  "long"
194  /  "double"
195  / classOrInterface_name .
196
197 modifier =
198  "public"
199  /  "private"
200  /  "protected"
201  /  "static"
202  /  "final"
203  /  "native"
204  /  "synchronized"
205  /  "abstract"
206  /  "threadsafe"
207  /  "transient"  .
208
209 package_name = <IDENTIFIER "." > IDENTIFIER  .
210
211 class_name  = <IDENTIFIER "." > IDENTIFIER  .
212
213 interface_name = <IDENTIFIER "." > IDENTIFIER  .
214
215 classOrInterface_name = <IDENTIFIER "." > IDENTIFIER
        .
216
217 integer_literal =
218  ( INT [ "l" ] )
219  / ( "0x" HEX <HEX> ).
```

Appendix D

Java tests

D.1 Test 1

```
1 class MyClass {
2    String longString =
3       "This is first half of a long string ;
4    void f() {
5       System.out.println(longString);
6    }
7 }
```

D.2 Test 2

```
1 class MyClass {
2    void f() {
3       if (i > j) )       // Error, extra parenthesis
4          max = i;
5    }
6 }
```

D.3 Test 3

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

D.4 Test 4

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

D.5 Test 5

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

D.6 Test 6

```
1 class MyClass {
2    public MyClass(int initialValue) {
3       int myVal = ;
4    }
5 }
```

D.7 Test 7

```
1 class MyClass {
2    public static boolean testFunc(int value) {
3       if( value == 2 ) {
4
5       }
6       else {
7          for( int i=0;i<5;i+ ) {
8
9          }
10      }
11   }
12 }
```

D.8 Test 8

```
1 class MyClass {
2    public static boolean testFunc(int value) {
3       while( true ) {
4          someOtherClass.dosomething();
5       }
6    }
7 }
```

D.9 Test 9

```
1 class MyClass {
2    void f() {
3       int n = 10;
4                    // Error, closing brace is missing
5    void g() {
6       int m = 20;
7    }
8 }
```

D.10 Test 10

```
1 class MyClass {
2    void f() {
3       if (i > j   // Error, unbalanced parentheses
4       {
5          max = i  // Error, missing semicolon
6       }
7       else
8          max = j;
9    }
10 }
```

Appendix E

SyntaxTrain source code

E.1 BnfCompiler

E.1.1 CommandLineTool.java

```
1 package BnfCompiler;
2
3 import java.io.BufferedWriter;
4 import java.io.ByteArrayInputStream;
5 import java.io.File;
6 import java.io.FileWriter;
7 import java.io.IOException;
8 import java.io.InputStream;
9 import java.io.UnsupportedEncodingException;
10 import java.util.HashMap;
11 import java.util.Stack;
12
13 import org.antlr.Tool;
14 import org.antlr.runtime.ANTLRInputStream;
15 import org.antlr.runtime.CommonTokenStream;
16 import org.antlr.runtime.RecognitionException;
17 import org.antlr.tool.ErrorManager;
```

```
18
19 import Xml.XmlNode;
20
21 import BnfCompiler.Link.RepeatType;
22 import Exceptions.XMLLoadException;
23 import Exceptions.XMLattributeDoesNotExist;
24 import Exceptions.XMLnodeDoesNotExist;
25 import Kernel.Variables;
26 import Library.StdLibrary;
27
28 public class CommandLineTool
29 {
30    private static String jdkBinFolder = "";
31    private static final String optionsXmlFile = "
         options.xml";
32
33    public static void printUsage()
34    {
35       System.out.println("Usage: java -jar
            BnfCompiler.jar BnfFile");
36       System.out.println();
37       System.out.println("BnfFile:\tReference to the
             file containing the language to be
            compiled, written in BNF.");
38    }
39
40    public static void main(String[] args)
41    {
42       if( args.length != 1 )
43       {
44          printUsage();
45          System.exit(0);
46       }
47       readOptions();
48       String filename = args[0];
49       String filenamePre = filename.substring(0,
            filename.length() - 4);
50       integrityCheck(filename);
51       cleanup(filenamePre);
52
53       String Bnf = StdLibrary.readFileAsString(
            filename);
54       if( Bnf == null )
55       {
```

```
56          System.out.println("There was a problem
               reading file " + filename + ".");
57          System.exit(0);
58       }
59
60       try
61       {
62          System.out.println("Parsing bnf");
63
64          InputStream is = new ByteArrayInputStream(
               Bnf.getBytes("UTF-8"));
65          ANTLRInputStream input = new
               ANTLRInputStream(is);
66          BnfEvaluatorLexer lexer = new
               BnfEvaluatorLexer(input);
67          CommonTokenStream tokens = new
               CommonTokenStream(lexer);
68          BnfEvaluatorParser parser = new
               BnfEvaluatorParser(tokens);
69
70          try
71          {
72             parser.bnf();
73             if( !(
74                   createAntlrFiles(filenamePre,
                         parser.startRule, parser.
                         ruleNameToLink) &&
75                   compileAntlrFiles("Grammar\\" +
                         filenamePre) &&
76                   createSyntaxXml( "Grammar\\" +
                         filenamePre, parser.
                         ruleNameToLink) &&
77                   createJarFile( filenamePre, "
                         Grammar\\" )
78                ) )
79             {
80                System.out.println("Failed to create
                      grammar file!");
81                return;
82             }
83             cleanup(filenamePre);
84             File jarFile = new File(filenamePre + ".
                   jar");
```

```
85              System.out.println("Jar file created at:
                    " + jarFile.getCanonicalPath());
86              return;
87           }
88           catch (RecognitionException e)
89           {
90
91           }
92           catch(RuntimeException e)
93           {
94              //TODO: the bnf wasn't written correctly
95              System.out.println("Error creating jar
                   file!");
96           }
97        }
98        catch (UnsupportedEncodingException e)
99        {
100          System.out.println("Unknown Error!");
101       }
102       catch (IOException e)
103       {
104          System.out.println("Unknown Error!");
105       }
106       System.exit(1);
107    }
108
109    private static void readOptions()
110    {
111       String optionsXml = StdLibrary.
             readFileAsString(optionsXmlFile);
112       if( optionsXml == null )
113       {
114          System.out.println("There was a problem
              reading file " + optionsXmlFile + ".");
115          System.exit(0);
116       }
117
118       try
119       {
120          XmlNode compilerOptions = new XmlNode(
              optionsXml, "1.0");
121          jdkBinFolder = compilerOptions.getChildNode
              ("BnfCompiler").getChildNode("path").
              getAttribute("jdkBinFolder") + "\\";
```

```
122             return;
123         }
124         catch (XMLLoadException e)
125         {
126             System.out.println( optionsXmlFile + "
                    contains invalid xml.");
127         }
128         catch (XMLattributeDoesNotExist e)
129         {
130             System.out.println("Invalid xml structure
                    in " + optionsXmlFile);
131         }
132         catch (XMLnodeDoesNotExist e)
133         {
134             System.out.println("Invalid xml structure
                    in " + optionsXmlFile);
135         }
136         System.exit(1);
137     }
138
139     private static boolean createJarFile(String
            filenamePre, String folder)
140     {
141         try
142         {
143             System.out.println("Creating jar file...");
144             Process jarProcess = Runtime.getRuntime().
                    exec( jdkBinFolder + "jar.exe cf " +
                    filenamePre + ".jar " + folder );
145             Long start = System.currentTimeMillis();
146             while( true )
147             {
148                 Thread.sleep(500);
149                 try
150                 {
151                     int exitCode1 = jarProcess.exitValue
                            ();
152                     if( exitCode1 != 0 )
153                     {
154                         System.out.println("Error, jar
                                file not created.");
155                         return false;
156                     }
157
```

```
158                 break;
159              }
160              catch (IllegalThreadStateException e)
161              {
162                 if( System.currentTimeMillis() -
                       start > 30000 )
163                 {
164                    //over 30 sec passed
165                    System.out.println("Jar file
                          creation took too long.");
166                    jarProcess.destroy();
167                    return false;
168                 }
169              }
170           }
171           return true;
172        }
173        catch (IOException e)
174        {
175           // TODO Auto-generated catch block
176           e.printStackTrace();
177        }
178        catch (InterruptedException e) {
179           // TODO Auto-generated catch block
180           e.printStackTrace();
181        }
182        return false;
183     }
184     private static void cleanup( String filenamePre )
185     {
186        File directory = new File("Grammar\\");
187        File[] files = directory.listFiles();
188        for( File file : files )
189        {
190           String name = file.getName();
191           if( name.equalsIgnoreCase("bnfparser.class
                 ") ||
192              name.equalsIgnoreCase(filenamePre + ".
                    jar") ||
193              file.isDirectory() )
194           {
195              continue;
196           }
197           file.delete();
```

```
198         }
199     }
200     private static void integrityCheck(String
            filename)
201     {
202         //TODO: checks that grammar folder exists (
                otherwise it creates it), that given jdk
                bin folder exists and contains javac and
                jar.
203         //also verifies that antlr.jar and grammar/
                BnfParser.class exists.
204         File antlr = new File("antlr.jar");
205         File bnfparser = new File("Grammar\\bnfparser.
                class");
206         File javac = new File(jdkBinFolder + "javac.
                exe");
207         File jar = new File(jdkBinFolder + "jar.exe");
208         File bnfFile = new File(filename);
209 
210         if( !antlr.exists() )
211         {
212             System.out.println("Missing file: antlr.jar
                    ");
213             System.exit(1);
214         }
215         if( !bnfparser.exists() )
216         {
217             System.out.println("Missing file Grammar\\
                    bnfparser.class");
218             System.exit(1);
219         }
220         if( !javac.exists() )
221         {
222             System.out.println("Missing or wrongly set
                    up java compilation file: " +
                    jdkBinFolder + "javac.exe");
223             System.exit(1);
224         }
225         if( !jar.exists() )
226         {
227             System.out.println("Missing or wrongly set
                    up jar compilation file: " +
                    jdkBinFolder + "jar.exe");
228             System.exit(1);
```

```
229         }
230         if( !bnfFile.exists() )
231         {
232             System.out.println("Missing file: " +
                filename);
233             System.exit(1);
234         }
235     }
236     private static boolean compileAntlrFiles( String
        filenamePre )
237     {
238         try
239         {
240             System.out.println("Compiling...");
241             Process compilation1 = Runtime.getRuntime()
                .exec( jdkBinFolder + "javac.exe -cp
                antlr.jar;. " + filenamePre + "Lexer.
                java" );
242             Process compilation2 = Runtime.getRuntime()
                .exec( jdkBinFolder + "javac.exe -cp
                antlr.jar;. " + filenamePre + "Parser.
                java" );
243             Long start = System.currentTimeMillis();
244             while( true )
245             {
246                 Thread.sleep(500);
247                 try
248                 {
249                     int exitCode1 = compilation1.
                        exitValue();
250                     int exitCode2 = compilation2.
                        exitValue();
251                     if( exitCode1 != 0 || exitCode2 != 0
                        )
252                     {
253                         System.out.println("Failed to
                            compile code.");
254                         return false;
255                     }
256
257                     break;
258                 }
259                 catch (IllegalThreadStateException e)
260                 {
```

```
261                 if( System.currentTimeMillis() -
                        start > 30000 )
262                 {
263                    //over 30 sec passed
264                    System.out.println("Compilation
                          error, took too long time to
                          compile.");
265                    compilation1.destroy();
266                    compilation2.destroy();
267                    return false;
268                 }
269              }
270           }
271           return true;
272        }
273        catch (IOException e)
274        {
275           // TODO Auto-generated catch block
276           e.printStackTrace();
277        }
278        catch (InterruptedException e) {
279           // TODO Auto-generated catch block
280           e.printStackTrace();
281        }
282        return false;
283     }
284     private static boolean createSyntaxXml( String
           filenamePre, HashMap<String, Link>
           ruleNameToLink )
285     {
286        System.out.println("Creating syntax xml file")
              ;
287        String filename = filenamePre + ".xml";
288        File file = new File(filename);
289        if(file.exists())
290        {
291           //just to be sure
292           file.delete();
293        }
294        try
295        {
296           BufferedWriter out = new BufferedWriter(new
                 FileWriter(file));
```

```
297          out.write("<SyntaxTrain version=\"" +
                 Variables.xmlVersion + "\">\n");
298          for( String ruleName : ruleNameToLink.
                 keySet() )
299          {
300             Link link = ruleNameToLink.get(ruleName)
                    ;
301             out.write("\t<Rule ID=\"" + ruleName +
                    "\">\n");
302             writeXmlRule(link, out, 2);
303             out.write("\t</Rule>\n\n");
304          }
305          out.write("</SyntaxTrain>\n");
306          out.close();
307          return true;
308       }
309       catch (IOException e)
310       {
311          System.out.println("Could not write to xml
                 file!!!!");
312       }
313       return false;
314    }
315    private static void writeTabs(int tabs,
          BufferedWriter out) throws IOException
316    {
317       for(int i=0;i<tabs;i++)
318       {
319          out.write("\t");
320       }
321    }
322    private static void writeXmlRule( Link rule,
          BufferedWriter out, int tabs) throws
          IOException
323    {
324       int numTabs = tabs;
325       RepeatType repeating = rule.getRepeat();
326       String id = rule.getId();
327       String UUID = rule.getUUID();
328       Stack<Link> ids = rule.getIds();
329
330       if( id != null )
331       {
332          writeTabs(tabs, out);
```

```
333             out.write("\t<Rule ID=\"" + StdLibrary.
                    xmlEscapeString( id ) + "\" UUID=\"" +
                    UUID + "\" />\n");
334         }
335         else
336         {
337             if( repeating == RepeatType.oneOfMultiple )
338             {
339                 //special case
340                 writeTabs(tabs, out);
341                 out.write("<or>\n");
342 
343                 for( Link subRule : ids )
344                 {
345                     writeTabs(tabs+1, out);
346                     out.write("<option>\n");
347                     writeXmlRule(subRule, out, numTabs+1)
                            ;
348                     writeTabs(tabs+1, out);
349                     out.write("</option>\n");
350                 }
351                 out.write("</or>\n");
352             }
353             else
354             {
355                 switch( repeating )
356                 {
357                 case atleastOnce:
358                     writeTabs(tabs, out);
359                     out.write( "<OnePlus>\n" );
360                     break;
361                 case loop:
362                     writeTabs(tabs, out);
363                     out.write( "<Repeat>\n" );
364                     break;
365                 case optional:
366                     writeTabs(tabs, out);
367                     out.write( "<Optional>\n" );
368                     break;
369                 case sequence:
370                     //the next items stay on the same
                            line
371                     numTabs--;
372                     break;
```

```
373             }
374             numTabs++;
375             for( Link subRule : ids )
376             {
377                writeXmlRule(subRule, out, numTabs);
378             }
379             switch( repeating )
380             {
381             case atleastOnce:
382                writeTabs(tabs, out);
383                out.write( "</OnePlus>\n" );
384                break;
385             case loop:
386                writeTabs(tabs, out);
387                out.write( "</Repeat>\n" );
388                break;
389             case optional:
390                writeTabs(tabs, out);
391                out.write( "</Optional>\n" );
392                break;
393             case sequence:
394
395                break;
396             }
397          }
398       }
399    }
400
401    private static boolean createAntlrFiles( String
          filenamePre, String startRule, HashMap<String
          , Link> ruleNameToLink )
402    {
403       System.out.println("Creating parser and lexer
             files");
404       String filename = filenamePre + ".g";
405       File file = new File(filename);
406       if(file.exists())
407       {
408          //just to be sure
409          file.delete();
410       }
411       try
412       {
```

```
413           BufferedWriter out = new BufferedWriter(new
                  FileWriter(file));
414           out.write(
415                 "grammar " + filenamePre + ";\n" +
416                 "\n" +
417                 "options\n" +
418                 "{\n" +
419                 "  superClass=BnfParser;\n" +
420                  "}\n" +
421                 "\n" +
422                 "@header\n" +
423                 "{\n" +
424                 "package Grammar;\n" +
425                 "\n" +
426                 "import java.util.Stack;\n" +
427                 "}\n" +
428                 "\n" +
429                 "@lexer::header\n" +
430                 "{\n" +
431                 "package Grammar;\n" +
432                 "}\n" +
433                 "\n" +
434                 "@parser::members\n" +
435                 "{\n" +
436                 "  public void bnf() throws
                       RecognitionException\n" +
437                 "  {\n" +
438                 "     " + startRule + "();\n" +
439                 "  }\n" +
440                 "}\n" +
441                 "@lexer::members {\n" +
442                 "    public Token nextToken() {\n" +
443                 "  while (true) {\n" +
444                 "     state.token = null;\n" +
445                 "     state.channel = Token.
                       DEFAULT_CHANNEL;\n" +
446                 "     state.tokenStartCharIndex =
                       input.index();\n" +
447                 "     state.
                       tokenStartCharPositionInLine =
                       input.getCharPositionInLine();\n"
                        +
448                 "     state.tokenStartLine = input.
                       getLine();\n" +
```

```
449                 "     state.text = null;\n" +
450                 "     if ( input.LA(1)==CharStream.
                        EOF ) {\n" +
451                 "        return Token.EOF_TOKEN;\n" +
452                 "     }\n" +
453                 "     try {\n" +
454                 "        mTokens();\n" +
455                 "        if ( state.token==null ) {\n
                        " +
456                 "           emit();\n" +
457                 "        }\n" +
458                 "        else if ( state.token==Token
                        .SKIP_TOKEN ) {\n" +
459                 "           continue;\n" +
460                 "        }\n" +
461                 "        return state.token;\n" +
462                 "     }\n" +
463                 "     catch (RecognitionException re)
                         {\n" +
464                 "           emit();\n" +
465                 "        return state.token;\n" +
466                 "     }\n" +
467                 "  }\n" +
468                 "}\n" +
469                 "}\n" +
470                 "\n"
471                 );
472           writeBnfRule( startRule, ruleNameToLink.get
                  (startRule), out, true );
473           for( String ruleName : ruleNameToLink.
                  keySet() )
474           {
475              if(ruleName.equalsIgnoreCase(startRule))
476              {
477                 //rule is already written
478                 continue;
479              }
480              writeBnfRule( ruleName, ruleNameToLink.
                    get(ruleName), out, false );
481           }
482           out.write(
483                 "INT :   '0'..'9'+\n" +
484                 "    ;\n" +
485                 "\n" +
```

```
486                 "FLOAT\n" +
487                 "    :   ('0'..'9')+ '.' ('0'..'9')*
                        EXPONENT?\n" +
488                 "    |   '.' ('0'..'9')+ EXPONENT?\n"
                        +
489                 "    |   ('0'..'9')+ EXPONENT\n" +
490                 "    ;\n" +
491                 "\n" +
492                 "IDENTIFIER  : ('a'..'z'|'A'..'Z'|'_
                        ') ('a'..'z'|'A'..'Z'|'0'..'9'|'_
                        ')* ;\n" +
493                 "\n" +
494                 "HEX\n" +
495                 "  :  HEX_DIGIT+;\n" +
496                 "\n" +
497                 "STRING\n" +
498                 ":  '\"' ( ESC_SEQ |
                        ~('\\\\'|'\"'|'\\n') )* '\"'\n"
                        +
499                 ";\n" +
500                 "\n" +
501                 "fragment\n" +
502                 "EXPONENT : ('e'|'E') ('+'|'-')?
                        ('0'..'9')+ ;\n" +
503                 "\n" +
504                 "fragment\n" +
505                 "HEX_DIGIT : ('0'..'9'|'a'..'f'|'A
                        '..'F') ;\n" +
506                 "\n" +
507                 "fragment\n" +
508                 "ESC_SEQ\n" +
509                 "    :   '\\\\' ('b'|'t'|'n'|'f'|'r
                        '|'\\\"'|'\\''|'\\\\')\n" +
510                 "    |   UNICODE_ESC\n" +
511                 "    |   OCTAL_ESC\n" +
512                 "    ;\n" +
513                 "\n" +
514                 "fragment\n" +
515                 "OCTAL_ESC\n" +
516                 "    :   '\\\\' ('0'..'3') ('0'..'7')
                        ('0'..'7')\n" +
517                 "    |   '\\\\' ('0'..'7') ('0'..'7')
                        \n" +
518                 "    |   '\\\\' ('0'..'7')\n" +
```

```
519                 "    ;\n" +
520                 "\n" +
521                 "fragment\n" +
522                 "UNICODE_ESC\n" +
523                 "    :   '\\\\' 'u' HEX_DIGIT
                        HEX_DIGIT HEX_DIGIT HEX_DIGIT\n"
                        +
524                 "    ;\n" +
525                 "\n" +
526                 "COMMENT\n" +
527                 "    :   '//' ~('\\n'|'\\r')* '\\r'?
                        '\\n' {$channel=HIDDEN;}\n" +
528                 "    |   '/*' ( options {greedy=false
                        ;} : . )* '*/' {$channel=HIDDEN
                        ;}\n" +
529                 "    ;\n" +
530                 "\n" +
531                 "WS  :   ( ' '\n" +
532                 "        | '\\t'\n" +
533                 "        | '\\r'\n" +
534                 "        | '\\n'\n" +
535                 "        ) {$channel=HIDDEN;}\n" +
536                 "    ;\n"
537             );
538             out.close();
539             ErrorListener listener = new ErrorListener
                    ();
540             ErrorManager.setErrorListener(listener);
541             Tool antlrTool = new Tool();
542             antlrTool.addGrammarFile(filename);
543             antlrTool.process();
544
545             if( listener.numErrorsAndWarnings > 0 )
546             {
547                 //Error reporting has been moved to the
                        listener since it is not certain
                        that the antlr tool will stop
548                 //in some cases it loops forever and
                        then no error message will be
                        reported :(
549                 return false;
550             }
551             //move the files
552             File grammarFile = new File(filename);
```

```
553          File tokenFile = new File(filenamePre + ".
                 tokens");
554          File lexerFile = new File(filenamePre + "
                 Lexer.java");
555          File parserFile = new File(filenamePre + "
                 Parser.java");
556
557          File GrammarDir = new File("Grammar\\");
558
559          if( !(
560                grammarFile.renameTo(new File(
                      GrammarDir, grammarFile.getName()
                      )) &&
561                tokenFile.renameTo(new File(
                      GrammarDir, tokenFile.getName()))
                       &&
562                lexerFile.renameTo(new File(
                      GrammarDir, lexerFile.getName()))
                       &&
563                parserFile.renameTo(new File(
                      GrammarDir, parserFile.getName())
                      )
564          ))
565          {
566             System.out.println("Could not rename
                    antlr files!");
567             return false;
568          }
569          return true;
570       }
571       catch (IOException e)
572       {
573          System.out.println("Could not write to file
                 !!!!");
574       }
575       return false;
576    }
577    private static void writeBnfRule(String ruleName,
           Link rule, BufferedWriter out, boolean
          isStartRule) throws IOException
578    {
579       out.write( ruleName + " :\n" );
580       out.write( "\t\t{Stack<String> stack = new
             Stack<String>(); trace.push(stack); stack.
```

```
            push(\"" + ruleName + "\"); popLast =
            false;}\n" );
581         for( Link subRule : rule.getIds() )
582         {
583            writeBnfSubRule( subRule, out );
584         }
585         if( isStartRule )
586         {
587            out.write("EOF");
588         }
589         out.write("\n\t\t{trace.pop();};\n\n");
590      }
591      private static void writeBnfSubRule(Link rule,
            BufferedWriter out) throws IOException
592      {
593         RepeatType repeating = rule.getRepeat();
594         String id = rule.getId();
595         String UUID = rule.getUUID();
596         Stack<Link> ids = rule.getIds();
597
598         if( id != null )
599         {
600            out.write( "({stack.push(\"" + UUID + "\");
                popLast = true;}");
601            out.write( " " + id + " {popLast = false;})
                " );
602
603         }
604         else
605         {
606            if( repeating == RepeatType.oneOfMultiple )
607            {
608               //special case
609               out.write("(");
610               for( int i=0;i<ids.size()-1;i++)
611               {
612                  writeBnfSubRule(ids.get(i), out);
613                  out.write("|");
614               }
615               writeBnfSubRule(ids.get(ids.size()-1),
                     out);
616               out.write(")");
617            }
618            else
```

```
619             {
620                 switch( repeating )
621                 {
622                 case atleastOnce:
623                     out.write( "(" );
624                     break;
625                 case loop:
626                     out.write( "(" );
627                     break;
628                 case optional:
629                     out.write( "(" );
630                     break;
631                 case sequence:
632                     out.write( "" );
633                     break;
634                 }
635                 for( Link subRule : ids )
636                 {
637                     writeBnfSubRule(subRule, out);
638                 }
639                 switch( repeating )
640                 {
641                 case atleastOnce:
642                     out.write( ")+" );
643                     break;
644                 case loop:
645                     out.write( ")*" );
646                     break;
647                 case optional:
648                     out.write( ")?" );
649                     break;
650                 case sequence:
651                     out.write( "" );
652                     break;
653                 }
654             }
655         }
656     }
657
658 }
```

E.1.2 ErrorListener.java

```
1  package BnfCompiler;
2
3  import org.antlr.tool.ANTLRErrorListener;
4  import org.antlr.tool.Message;
5  import org.antlr.tool.ToolMessage;
6
7  public class ErrorListener implements
       ANTLRErrorListener
8  {
9     public int numErrorsAndWarnings = 0;
10
11    private void addError( String error )
12    {
13       if( numErrorsAndWarnings == 0 )
14       {
15          //first error
16          System.out.println("Bnf error.");
17          System.out.println("Make sure theres no
                left recursion or multiple alternatives
                 for rules.");
18          System.out.println("List of errors (line
                numbers does not match with the given
                BNF):");
19       }
20       numErrorsAndWarnings++;
21       System.out.println( error );
22    }
23
24    @Override
25    public void error(Message msg)
26    {
27       addError(msg.toString());
28    }
29
30    @Override
31    public void error(ToolMessage tmsg)
32    {
33       addError(tmsg.toString());
34    }
35
36    @Override
37    public void info(String info)
```

```
38     {
39        System.out.println("Info: " + info);
40     }
41
42     @Override
43     public void warning(Message warning)
44     {
45        addError(warning.toString());
46     }
47 }
```

E.1.3 Link.java

```
1  package BnfCompiler;
2
3  import java.util.Stack;
4
5  public class Link
6  {
7     public enum RepeatType {sequence, optional,
          atleastOnce, loop, oneOfMultiple}
8     private String id;
9     private Stack<Link> ids;
10    private RepeatType repeating;
11    private String UUID;
12    private static int currentID = 0;
13
14    public Link()
15    {
16       createLink( null, null );
17    }
18
19    public Link( Link link )
20    {
21       createLink( null, link );
22    }
23
24    //when this option is used, no functions can be
          used!
25    public Link( String identifier )
26    {
27       //Replace " with '
```

```
28          String id = identifier.replace("\"", "'");
29          createLink( id, null );
30      }
31
32      private void createLink( String identifier, Link
            link )
33      {
34          ids = new Stack<Link>();
35          if( link != null )
36          {
37              ids.push(link);
38          }
39          id = identifier;
40          if( id != null )
41          {
42              UUID = "id" + Integer.toString(currentID++)
                    ;
43          }
44          //by default all Links are sequences
45          repeating = RepeatType.sequence;
46      }
47
48      public String getUUID()
49      {
50          return UUID;
51      }
52
53      private void setRepeat( RepeatType repeat )
54      {
55          this.repeating = repeat;
56      }
57
58      public RepeatType getRepeat()
59      {
60          return repeating;
61      }
62
63      public String getId()
64      {
65          return id;
66      }
67
68      public Stack<Link> getIds()
69      {
```

```
70          return ids;
71      }
72
73      /**
74       * Adds another link to the sequence
75       * @param link
76       */
77      public void add( Link link )
78      {
79          ids.add(link);
80      }
81
82      /**
83       * This link requires just one of the ids, ex. "
             abc" | someRule | myrule | hehehe.
84       */
85      public void oneOfMultiple()
86      {
87          setRepeat(RepeatType.oneOfMultiple);
88      }
89
90      /**
91       * This link is optional, ex. ["abc" someRule]
92       */
93      public void optional()
94      {
95          integrityCheck();
96          setRepeat(RepeatType.optional);
97      }
98
99      /**
100      * This link can loop from 0 to infinitely many
             times, ex. ("abc" someRule)* or <"abc"
             someRule>
101      */
102     public void loop()
103     {
104         integrityCheck();
105         setRepeat(RepeatType.loop);
106     }
107
108     /**
109      * This link loops atleast once, ex. ("abc"
             someRule)+
```

```
110      */
111     public void loopAtleastOnce()
112     {
113        integrityCheck();
114        setRepeat(RepeatType.atleastOnce);
115     }
116
117     private void integrityCheck()
118     {
119        if( id != null )
120        {
121           throw new Error("Integrity check error in
                  Link, id is defined and trying to add
                  other objects!");
122        }
123     }
124 }
```

E.2 Exceptions

E.2.1 UnknownXMLFormat.java

```
1  package Exceptions;
2
3  public class UnknownXMLFormat extends Exception
4  {
5     private static final long serialVersionUID =
          -2212083386200335853L;
6
7     public UnknownXMLFormat()
8     {
9
10    }
11 }
```

E.2.2 XMLattributeDoesNotExist.java

```
1  package Exceptions;
```

```
2
3 public class XMLattributeDoesNotExist extends
      Exception
4 {
5     private static final long serialVersionUID =
          -1349699666130356843L;
6     String attribute;
7
8     public XMLattributeDoesNotExist(String attribute)
9     {
10        System.out.println("XML attribute does not
              exist: " + attribute);
11    }
12 }
```

E.2.3 XMLLoadException.java

```
1 package Exceptions;
2
3 public class XMLLoadException extends Exception
4 {
5     private static final long serialVersionUID =
          -2522214346015679085L;
6     private String error;
7
8     public XMLLoadException( String error )
9     {
10        this.error = error;
11    }
12
13    public String toString()
14    {
15        return error;
16    }
17 }
```

E.2.4 XMLnodeDoesNotExist.java

```
1 package Exceptions;
```

```
2
3 public class XMLnodeDoesNotExist extends Exception
4 {
5    private static final long serialVersionUID =
        2952107033994886886L;
6    private String error;
7
8    public XMLnodeDoesNotExist( String node )
9    {
10      error = "Missing node: " + node;
11   }
12
13   public String toString()
14   {
15      return error;
16   }
17 }
```

E.2.5 XMLTextDoesNotExist.java

```
1 package Exceptions;
2
3 public class XMLTextDoesNotExist extends Exception
4 {
5    private static final long serialVersionUID =
        2663518752007485741L;
6
7    public XMLTextDoesNotExist()
8    {
9
10   }
11 }
```

E.3 Grammar

E.3.1 BnfParser.java

```
1 package Grammar;
```

```
2
3 import java.util.Stack;
4
5 import org.antlr.runtime.Parser;
6 import org.antlr.runtime.RecognitionException;
7 import org.antlr.runtime.RecognizerSharedState;
8 import org.antlr.runtime.TokenStream;
9
10 public abstract class BnfParser extends Parser
11 {
12    public BnfParser(TokenStream input)
13    {
14       super(input);
15    }
16
17    public BnfParser(TokenStream input,
         RecognizerSharedState state)
18    {
19       super(input, state);
20    }
21    public int errorLine = -1,
         errorCharPositionInLine = -1;
22    public boolean popLast = false;
23    public Stack<Stack<String>> trace = new Stack<
         Stack<String>>();
24    public void displayRecognitionError(String[]
         tokenNames, RecognitionException e) throws
         RuntimeException
25    {
26          errorLine = e.line;
27          errorCharPositionInLine = e.
               charPositionInLine;
28       throw new RuntimeException();
29    }
30    //the default call to check syntax
31    public abstract void bnf() throws
         RecognitionException;
32 }
```

E.4 GUI

E.4.1 Controller.java

```
1 package GUI;
2
3 import java.awt.event.ActionEvent;
4 import java.awt.event.ActionListener;
5 import java.awt.event.WindowEvent;
6 import java.awt.event.WindowListener;
7 import java.io.File;
8 import java.io.IOException;
9
10 import javax.swing.JFileChooser;
11 import javax.swing.JOptionPane;
12 import javax.swing.event.DocumentEvent;
13 import javax.swing.event.DocumentListener;
14
15 import GuiAPI.GuiApi;
16 import KernelAPI.KernelApi;
17
18 public class Controller implements ActionListener,
     WindowListener, DocumentListener
19 {
20    private static Controller instance;
21
22    private Controller()
23    {
24    }
25
26    public void actionPerformed(ActionEvent action)
27    {
28       String command = action.getActionCommand();
29       if( command.equals(Variables.OPEN_SOURCE_FILE)
             )
30       {
31
32          final JFileChooser fileChooser = new
               JFileChooser(Variables.
               lastOpenedDirectory);
33          int retVal = fileChooser.showOpenDialog(
               MainScreen.getInstance());
34          if( retVal == JFileChooser.APPROVE_OPTION )
```

```
35          {
36              File selectedFile = fileChooser.
                    getSelectedFile();
37              KernelApi.readSourceFile(selectedFile);
38              Variables.lastOpenedDirectory =
                    selectedFile.getParent();
39          }
40      }
41      else if( command.equals(Variables.
            RELOAD_SOURCE_FILE) )
42      {
43          KernelApi.reloadSourceCode();
44      }
45      else if( command.equals(Variables.CHECK_SYNTAX
            ) )
46      {
47          KernelApi.setSourceCode(gSourceCode.
                getInstance().getSourceCode());
48          GuiApi.updateDiagrams();
49      }
50      else if( command.equals(Variables.
            SAVE_SOURCE_FILE))
51      {
52          saveSourceCode();
53      }
54      else if( command.equals(Variables.
            SHOW_HIDE_BNF_GRAMMARS))
55      {
56          gGrammarPanel.getInstance().
                swapBetweenShowAndHideGrammarOptions();
57      }
58      else if( command.equals(Variables.HELP))
59      {
60          Dialogs.showHelpDialog();
61      }
62      else if( command.equals(Variables.ABOUT))
63      {
64          Dialogs.showAboutDialog();
65      }
66      else
67      {
68          System.out.println("Unknown command: " +
                command);
69      }
```

```
70    }
71
72    private boolean saveSourceCode()
73    {
74       try
75       {
76          KernelApi.setSourceCode(gSourceCode.
               getInstance().getSourceCode());
77          KernelApi.saveSourceCode();
78          Variables.setCodeChanged(false);
79          return true;
80       }
81       catch (IOException e)
82       {
83          JOptionPane.showMessageDialog(MainScreen.
               getInstance(), "Failed to save file!");
84       }
85       return false;
86    }
87
88    @Override
89    public void windowActivated(WindowEvent arg0)
90    {
91    }
92
93    @Override
94    public void windowClosed(WindowEvent arg0)
95    {
96    }
97
98    @Override
99    public void windowClosing(WindowEvent arg0)
100   {
101      if( Variables.isCodeChanged() )
102      {
103         String filename = MainScreen.getInstance().
               getFilename();
104         if( filename == null )
105         {
106            System.exit(0);
107         }
108         int result = JOptionPane.showConfirmDialog(
               MainScreen.getInstance(), "Save file "
               + filename + "?", "Close", JOptionPane.
```

```
                YES_NO_CANCEL_OPTION, JOptionPane.
                QUESTION_MESSAGE);
109          switch( result )
110          {
111          case JOptionPane.YES_OPTION:
112             if( saveSourceCode() )
113             {
114                System.exit(0);
115             }
116             break;
117          case JOptionPane.NO_OPTION:
118             System.exit(0);
119             break;
120          }
121       }
122       else
123       {
124          System.exit(0);
125       }
126    }
127
128    @Override
129    public void windowDeactivated(WindowEvent arg0)
130    {
131    }
132
133    @Override
134    public void windowDeiconified(WindowEvent arg0)
135    {
136    }
137
138    @Override
139    public void windowIconified(WindowEvent arg0)
140    {
141    }
142
143    @Override
144    public void windowOpened(WindowEvent arg0)
145    {
146    }
147
148    @Override
149    public void changedUpdate(DocumentEvent e)
150    {
```

```
151         sourceChanged();
152      }
153
154      @Override
155      public void insertUpdate(DocumentEvent e)
156      {
157         sourceChanged();
158      }
159
160      @Override
161      public void removeUpdate(DocumentEvent e)
162      {
163         sourceChanged();
164      }
165      public void sourceChanged()
166      {
167         Variables.setCodeChanged(true);
168         Variables.setDiagramsOutOfSynch(true);
169         gErrorTrace.getInstance().updateSyncStatus();
170      }
171
172      public static Controller getInstance()
173      {
174         if( instance == null )
175         {
176            instance = new Controller();
177         }
178         return instance;
179      }
180   }
```

E.4.2 Dialogs.java

```
1  package GUI;
2
3  import java.awt.SystemColor;
4
5  import javax.swing.JOptionPane;
6  import javax.swing.JTextArea;
7
8  public class Dialogs
9  {
```

```
10    public static void showHelpDialog()
11    {
12       final JTextArea helpTextArea = new JTextArea(
13             "In order to use this program, first
                  open a source file, which is
                  accepted by the current grammar file
                  .\n" +
14             "If there's any error in your code,
                  SyntaxTrain will show syntax
                  diagrams to describe the error, just
                  \n" +
15             "like a stack trace. For an example you
                  can try using the java grammar
                  included and open example.java.\n\n"
                   +
16             "If a node in the diagram is marked blue
                   then it means that node was
                  correctly matched (no error here)\n"
                   +
17             "If instead the node is colored red, it
                  means that the error occured here,
                  and above you will see the\n" +
18             "appropriate diagram for that node.
                  Finally if the node is marked orange
                  , that means this is what you\n" +
19             "can write at that point (one or
                  multiple nodes may be colored orange
                  ).\n" +
20             "In the top there's a diagram showing
                  where in your code the error is
                  located.\n" +
21             "Ex. it could be inside an if sentence
                  inside your function inside your
                  class.");
22       helpTextArea.setEditable(false);
23       helpTextArea.setBackground(SystemColor.control
             );
24       JOptionPane.showMessageDialog(MainScreen.
            getInstance(), helpTextArea,"SyntaxTrain -
             Help",JOptionPane.PLAIN_MESSAGE);
25    }
26
27    public static void showAboutDialog()
28    {
```

```
29        final JTextArea aboutTextArea = new JTextArea(
30              "SyntaxTrain - Syntax Diagrams for Java
                   Programs\n" +
31              "Version 1.0\n" +
32              "Copyright 2011 by Andreas Leon Aagaard
                   Moth.\n\n" +
33              "This program is free software: you can
                   redistribute it and/or modify\n" +
34              "it under the terms of the GNU General
                   Public License as published by\n" +
35              "the Free Software Foundation, either
                   version 3 of the License, or\n" +
36              "(at your option) any later version.\n"
                   +
37              "\n" +
38              "This program is distributed in the hope
                    that it will be useful,\n" +
39              "but WITHOUT ANY WARRANTY; without even
                   the implied warranty of\n" +
40              "MERCHANTABILITY or FITNESS FOR A
                   PARTICULAR PURPOSE.  See the\n" +
41              "GNU General Public License for more
                   details.\n" +
42              "\n" +
43              "You should have received a copy of the
                   GNU General Public License\n" +
44              "along with this program.  If not, see <
                   http://www.gnu.org/licenses/>.");
45        aboutTextArea.setEditable(false);
46        aboutTextArea.setBackground(SystemColor.
             control);
47        JOptionPane.showMessageDialog(MainScreen.
             getInstance(), aboutTextArea,"About
             SyntaxTrain - version 1.0",JOptionPane.
             PLAIN_MESSAGE);
48    }
49 }
```

E.4.3 gErrorTrace.java

```
1 package GUI;
2
```

```
3 import java.awt.Color;
4 import java.awt.Dimension;
5 import java.awt.Font;
6 import java.awt.Graphics;
7 import java.awt.geom.Rectangle2D;
8 import java.util.ArrayList;
9 import java.util.Stack;
10
11 import javax.swing.JPanel;
12
13 import KernelAPI.KernelApi;
14
15 /**
16  * Displays a rail-road diagram of the source code.
      (top)
17  */
18 public class gErrorTrace extends JPanel
19 {
20    private static final int SPACE_BETWEEN_BOXES =
         10, SPACE_ON_EACH_SIDE_INSIDE_BOXES = 2;
21    private static final long serialVersionUID =
         7253097565513080782L;
22    private static gErrorTrace instance = null;
23    private ArrayList<String> errorTrace;
24    private Dimension size;
25    private boolean outOfSync;
26
27    private gErrorTrace()
28    {
29       setBackground(Color.WHITE);
30       updateDiagram();
31       outOfSync = false;
32    }
33
34    public Dimension getPreferredSize()
35    {
36       if( size != null )
37          return size;
38       return new Dimension(400, 100);
39    }
40
41    public void updateSyncStatus()
42    {
43       outOfSync = Variables.isDiagramsOutOfSync();
```

```
44          getParent().repaint();
45      }
46
47      public void paint(Graphics g)
48      {
49          super.paint(g);
50          Font header_font = new Font( "Serif", Font.
                PLAIN, 20 );
51          Font box_font = new Font( "Serif", Font.PLAIN,
                14 );
52          int posX = 15, posY = 25;
53
54          g.setFont(header_font);
55          String textBuffer = "No syntax errors";
56          if( errorTrace.size() > 0 )
57          {
58              textBuffer = "First syntax error: ";
59          }
60          Rectangle2D bounds = g.getFontMetrics().
                getStringBounds(textBuffer, g);
61          posY = (int)bounds.getHeight() + 5;
62          g.drawString(textBuffer, posX, posY);
63
64          if( outOfSync )
65          {
66              textBuffer = "modified";
67              g.setFont(new Font( "Serif", Font.PLAIN,
                    12));
68              int modHeight = (int)g.getFontMetrics().
                    getStringBounds(textBuffer, g).
                    getHeight();
69              g.drawString(textBuffer, posX, posY +
                    modHeight + 3);
70          }
71
72          posX += bounds.getWidth();
73
74          g.setFont(box_font);
75          for( int i=errorTrace.size()-1;i>0;i-- )
76          {
77              //text
78              g.setColor(Color.BLACK);
79              textBuffer = errorTrace.get(i);
80              g.drawString(textBuffer, posX, posY);
```

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

```
116             instance = new gErrorTrace();
117         }
118         return instance;
119     }
120 }
```

E.4.4 gGrammarDiagram.java

```
 1 package GUI;
 2
 3 import java.awt.Color;
 4 import java.awt.Dimension;
 5 import java.awt.Graphics;
 6 import java.awt.Graphics2D;
 7 import java.awt.image.BufferedImage;
 8 import java.util.ArrayList;
 9 import java.util.HashMap;
10 import java.util.List;
11 import java.util.Stack;
12
13 import javax.swing.JPanel;
14
15 import KernelAPI.KernelApi;
16 import Library.Lock;
17
18 import net.hydromatic.clapham.Clapham;
19 import net.hydromatic.clapham.graph.Chart;
20 import net.hydromatic.clapham.graph.Grammar;
21 import net.hydromatic.clapham.graph.Symbol;
22 import net.hydromatic.clapham.parser.ProductionNode;
23
24 /**
25  * Displays a rail-road diagram of the source code (
       bottom middle)
26  */
27 public class gGrammarDiagram extends JPanel
28 {
29     private static final long serialVersionUID =
          1333493020186127182L;
30     private static gGrammarDiagram instance = null;
31     private BufferedImage[] grammarDiagrams;
32     private HashMap<String, Integer> grammarToId;
```

```
33    private String[] grammars;
34    private boolean[] showGrammar;
35    private Lock grammarDiagramsLock;
36
37    private gGrammarDiagram()
38    {
39       grammarDiagramsLock = new Lock();
40       setBackground(Color.WHITE);
41       updateDiagram();
42    }
43
44    synchronized public void updateDiagram()
45    {
46       ArrayList<String> grammarNames = KernelApi.
            getGrammars();
47       if( grammarNames == null )
48       {
49          return;
50       }
51       grammarDiagramsLock.P();
52       //initialize grammar varaibles
53       int grammarId = 0;
54         grammars = new String[grammarNames.size()];
55       showGrammar = new boolean[grammarNames.size()
            ];
56       grammarToId = new HashMap<String, Integer>();
57       grammarDiagrams = new BufferedImage[
            grammarNames.size()];
58
59       if( KernelApi.getErrorTrace() != null ) //==
            null if no error
60       {
61          @SuppressWarnings("unchecked")
62          Stack<Stack<String>> errorTrace = (Stack<
               Stack<String>>) KernelApi.getErrorTrace
               ().clone();
63          while(!errorTrace.isEmpty())
64          {
65             String ruleName = errorTrace.pop().
                  firstElement();
66             if( grammarToId.containsKey(ruleName) )
67                continue;
68             grammars[grammarId] = ruleName;
69             showGrammar[grammarId] = true;
```

```
70              grammarToId.put(ruleName, grammarId);
71              grammarId++;
72          }
73      }
74
75      if( grammarNames != null )
76        {
77           for ( String grammarName : grammarNames )
78           {
79            if( grammarToId.containsKey(grammarName)
                  )
80               continue;
81
82            grammars[grammarId] = grammarName;
83            showGrammar[grammarId] = false;
84            grammarToId.put(grammarName, grammarId);
85            grammarId++;
86           }
87        }
88
89      //build grammar diagrams
90      List<ProductionNode> productionNodes =
           KernelApi.getGrammarProductionNodes();
91      if( productionNodes == null )
92      {
93         return;
94      }
95      Grammar grammar = Clapham.buildGrammar(
           productionNodes);
96      List<String> nameList = new ArrayList<String
           >();
97      nameList.clear();
98        nameList.addAll(grammar.symbolMap.keySet());
99
100     for( String grammarName : nameList )
101     {
102        BufferedImage image = drawNode(grammarName,
              grammar);
103        int id = grammarToId.get(grammarName);
104        grammarDiagrams[id] = image;
105     }
106     grammarDiagramsLock.V();
107
108     //show default grammars
```

```
109         updateDimensions();
110         repaint();
111     }
112
113     public void setGrammarVisible( String grammar,
            boolean visible )
114     {
115         int id = grammarToId.get(grammar);
116         showGrammar[id] = visible;
117
118         updateDimensions();
119         repaint();
120     }
121
122     private void updateDimensions()
123     {
124         int width=0, height=0;
125
126         for( int i=0;i<grammars.length;i++ )
127         {
128             if(showGrammar[i])
129             {
130                 BufferedImage image = grammarDiagrams[i
                        ];
131
132                 width = Math.max(image.getWidth(), width
                        );
133                 height += image.getHeight();
134             }
135         }
136         Dimension dim = new Dimension(width, height);
137           this.setSize(dim);
138           this.setPreferredSize(dim);
139     }
140
141     public void paint(Graphics g)
142     {
143         super.paint(g);
144
145         if( grammars == null )
146         {
147             return;
148         }
149         grammarDiagramsLock.P();
```

```
150         for( int i=0;i<grammars.length;i++ )
151         {
152             if(showGrammar[i])
153             {
154                 BufferedImage image = grammarDiagrams[i
                        ];
155                 g.drawImage(image, 0, 0, null);
156                 g.translate(0, image.getHeight());
157             }
158         }
159         grammarDiagramsLock.V();
160     }
161 
162     private BufferedImage drawNode(String symbolName,
            Grammar grammar)
163     {
164         //temporary image to draw on
165         BufferedImage tempImg = new BufferedImage(1,
                1, BufferedImage.TYPE_INT_RGB);
166         Graphics2D graphics = tempImg.createGraphics()
                ;
167 
168         Symbol symbol = grammar.symbolMap.get(
                symbolName);
169         if (symbol.graph == null)
170         {
171             throw new RuntimeException(
172                 "Symbol '" + symbolName + "' not found
                        ");
173         }
174 
175         Chart chart = new Chart(grammar, (Graphics2D)
                graphics);
176         chart.calcDrawing();
177         chart.drawComponent(symbol);
178 
179         //draw the final image
180         Dimension dim = chart.getDimension();
181 
182         BufferedImage finalDrawing = new BufferedImage
                ((int)dim.getWidth(), (int)dim.getHeight()
                    + 5, BufferedImage.TYPE_INT_RGB);
183         graphics = finalDrawing.createGraphics();
184 
```

```
185         chart = new Chart(grammar, (Graphics2D)
                graphics);
186         chart.calcDrawing();
187         chart.drawComponent(symbol);
188         return finalDrawing;
189     }
190 
191     public static synchronized gGrammarDiagram
            getInstance()
192     {
193         if( instance == null )
194         {
195             instance = new gGrammarDiagram();
196         }
197         return instance;
198     }
199 }
```

E.4.5 gGrammarOptions.java

```
 1 package GUI;
 2 
 3 import java.awt.BorderLayout;
 4 import java.awt.Color;
 5 import java.awt.Component;
 6 import java.awt.Dimension;
 7 import java.util.ArrayList;
 8 import java.util.HashMap;
 9 import java.util.HashSet;
10 import java.util.Stack;
11 
12 import javax.swing.BorderFactory;
13 import javax.swing.JCheckBox;
14 import javax.swing.JLabel;
15 import javax.swing.JList;
16 import javax.swing.JPanel;
17 import javax.swing.ListCellRenderer;
18 import javax.swing.event.ListSelectionEvent;
19 import javax.swing.event.ListSelectionListener;
20 
21 import KernelAPI.KernelApi;
22 
```

```
23 public class gGrammarOptions extends JList
       implements ListCellRenderer,
       ListSelectionListener
24 {
25     private static final long serialVersionUID =
           4419173956236695445L;
26
27     private static gGrammarOptions instance = null;
28
29     private HashMap<String, Boolean> isChecked;
30     private HashSet<String> errorTraceComponents;
31      //used by getListCellRendererComponent to avoid
            allocating a new checkbox all the time
32      private JCheckBox checkBox;
33      private JLabel label;
34      private JPanel panel;
35      private String[] listData;
36      private int numChecked;
37
38     public gGrammarOptions()
39     {
40        clearCheckedRules();
41          setCellRenderer( this );
42          addListSelectionListener( this );
43
44          checkBox = new JCheckBox();
45          label = new JLabel();
46          panel = new JPanel( new BorderLayout() );
47          panel.add( BorderLayout.WEST, label );
48          panel.add( BorderLayout.EAST, checkBox );
49          panel.setBorder(BorderFactory.
                createLineBorder(Color.black));
50
51          //set default grammars
52          updateGrammars();
53     }
54
55     public Dimension getPreferredSize()
56     {
57        Dimension dim = super.getPreferredSize();
58        dim.setSize(dim.getWidth()+20, dim.getHeight()
              );
59        return dim;
60     }
```

```
61
62    private void clearCheckedRules()
63    {
64       isChecked = new HashMap<String, Boolean>();
65       errorTraceComponents = new HashSet<String>();
66       ArrayList<String> grammarNames = KernelApi.
            getGrammars();
67         if( grammarNames != null )
68         {
69          listData = new String[grammarNames.size()
               +1];
70          listData[0] = "Syntax Components:";
71            for ( int i=0;i<grammarNames.size();i++ )
72            {
73             String grammarName = grammarNames.get(i)
                  ;
74             listData[i+1] = grammarName;
75                isChecked.put( grammarName, false );
76            }
77
78            setListData( listData );
79         }
80
81         numChecked = 0;
82    }
83
84    public void updateGrammars()
85    {
86       clearCheckedRules();
87       showErrorTrace();
88    }
89
90    private void showErrorTrace()
91    {
92       if( KernelApi.getErrorTrace() != null )
93       {
94          for(Stack<String> ruleTrace : KernelApi.
               getErrorTrace())
95          {
96             String component = ruleTrace.
                  firstElement();
97             isChecked.put(component, true);
98             errorTraceComponents.add(component);
99          }
```

```
100        }
101    }
102
103    public Component getListCellRendererComponent(
104              JList list,
105              Object value,
106              int index,
107              boolean isSelected,
108              boolean cellHasFocus )
109     {
110          if ( !(value instanceof String) )
111          {
112              return new JLabel( "Error: " + value.
                     toString() );
113          }
114          String name = ( String ) value;
115          if( name.equalsIgnoreCase("Syntax Components
                 :") )
116          {
117           //top
118           checkBox.setSelected(false);
119           label.setText(name);
120           panel.setComponentOrientation( list.
                 getComponentOrientation() );
121           panel.setBackground( new Color(200,200,200)
                  );
122           checkBox.setBackground( new Color
                 (200,200,200) );
123           checkBox.setSelected( numChecked > 0 );
124              panel.setForeground( list.getForeground
                     () );
125
126          }
127          else
128          {
129             checkBox.setSelected( isChecked.get( name
                     ) );
130             label.setText( name );
131
132             panel.setComponentOrientation( list.
                    getComponentOrientation() );
133             if ( isChecked.get( name ) )
134             {
```

```
135                 panel.setBackground( list.
                       getSelectionBackground() );
136                 checkBox.setBackground( list.
                       getSelectionBackground() );
137                 panel.setForeground( list.
                       getSelectionForeground() );
138             }
139             else
140             {
141                 panel.setBackground( list.
                       getBackground() );
142                 checkBox.setBackground( list.
                       getBackground() );
143                 panel.setForeground( list.
                       getForeground() );
144             }
145         }
146
147         return panel;
148     }
149
150     public void valueChanged( ListSelectionEvent e )
151     {
152         //make sure we aren't dragging anything into
                the list
153         if ( !e.getValueIsAdjusting() )
154         {
155             Object value = getSelectedValue();
156             if ( value instanceof String )
157             {
158                 String name = ( String ) value;
159                 if( name.equalsIgnoreCase("Syntax
                       Components:") )
160                 {
161                     // header clicked
162                     if( numChecked > 0 )
163                     {
164                         numChecked = 0;
165                         for ( int i=1;i<listData.length
                             ;i++ )
166                           {
167                             String grammarName =
                                 listData[i];
168                             boolean showGrammar = false;
```

```
169                     //make sure to still show
                            the error trace
170                     if( errorTraceComponents.
                            contains(grammarName) )
171                     {
172                        showGrammar = true;
173                     }
174                     isChecked.put( grammarName,
                            showGrammar );
175                  gGrammarDiagram.getInstance().
                        setGrammarVisible(
                        grammarName, showGrammar);
176                   }
177               }
178               else
179               {
180                  numChecked = listData.length -
                        1 - errorTraceComponents.
                        size();
181                  for ( int i=1;i<listData.length
                        ;i++ )
182                    {
183                     String grammarName =
                           listData[i];
184                        isChecked.put(
                              grammarName, true );
185                        gGrammarDiagram.
                              getInstance().
                              setGrammarVisible(
                              grammarName, true);
186                    }
187               }
188            }
189            else
190            {
191               boolean isSelected = isChecked.get
                     ( name );
192               if( errorTraceComponents.contains(
                      name ) )
193               {
194                  // 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.
195                     numChecked += isSelected ? 1 :
                        -1;
196                  }
197                  else
198                  {
199                     //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.
200                     numChecked += isSelected ? -1 :
                         1;
201                  }
202                   isChecked.put( name, !isSelected
                      );
203                   gGrammarDiagram.getInstance().
                      setGrammarVisible(name, !
                      isSelected);
204               }
205                removeSelectionInterval( 0,
                   isChecked.size() );
206            }
207         }
208      }
209
210   public static synchronized gGrammarOptions
         getInstance()
211   {
212      if(instance == null)
213      {
214         instance = new gGrammarOptions();
215      }
216      return instance;
217   }
218 }
```

E.4.6 gGrammarPanel.java

```
1 package GUI;
```

```
2
3 import java.awt.BorderLayout;
4 import java.awt.Graphics;
5
6 import javax.swing.JPanel;
7 import javax.swing.JScrollPane;
8 import javax.swing.JSplitPane;
9
10 public class gGrammarPanel extends JPanel
11 {
12    private static final long serialVersionUID =
          1452823813888634187L;
13    private static gGrammarPanel instance = null;
14    private boolean isGrammarOptionsVisible;
15    private JScrollPane scrollGrammarOptions;
16    private JSplitPane diagramsOptionsSplitPane;
17
18    private gGrammarPanel()
19    {
20       super(new BorderLayout());
21       //Grammar diagrams
22       JScrollPane scrollGrammarDiagram = new
             JScrollPane(gGrammarDiagram.getInstance())
             ;
23       scrollGrammarDiagram.getVerticalScrollBar().
             setUnitIncrement(16);
24       //Grammar options
25       scrollGrammarOptions = new JScrollPane(
             gGrammarOptions.getInstance());
26       //by default the grammar options are hidden
27       isGrammarOptionsVisible = false;
28
29       //Split pane containing the diagram pane and
             the options pane
30       diagramsOptionsSplitPane = new JSplitPane(
             JSplitPane.HORIZONTAL_SPLIT,
             scrollGrammarDiagram, scrollGrammarOptions
             );
31       diagramsOptionsSplitPane.setEnabled(false);
32       diagramsOptionsSplitPane.setDividerSize(0);
33       diagramsOptionsSplitPane.setDividerLocation(0)
             ; //to avoid drawing a line at startup
34       add(BorderLayout.CENTER,
             diagramsOptionsSplitPane);
```

```
35    }
36
37    public void paint(Graphics g)
38    {
39       if( isGrammarOptionsVisible )
40       {
41          diagramsOptionsSplitPane.setDividerLocation
               ((int) (getWidth() -
               scrollGrammarOptions.getPreferredSize()
               .getWidth()) - 1);
42       }
43       else
44       {
45          // hide grammar options
46          diagramsOptionsSplitPane.setDividerLocation
               ( 1.0 );
47       }
48       super.paint(g);
49    }
50
51    public void swapBetweenShowAndHideGrammarOptions
          ()
52    {
53       isGrammarOptionsVisible = !
            isGrammarOptionsVisible;
54       repaint();
55    }
56
57    public static synchronized gGrammarPanel
          getInstance()
58    {
59       if(instance == null)
60       {
61          instance = new gGrammarPanel();
62       }
63       return instance;
64    }
65 }
```

E.4.7 gSourceCode.java

```
1 package GUI;
```

```
2
3  import java.awt.BorderLayout;
4  import java.awt.Color;
5  import java.awt.Font;
6  import java.awt.FontMetrics;
7  import java.awt.Point;
8
9  import javax.swing.JPanel;
10 import javax.swing.JTextPane;
11 import javax.swing.text.MutableAttributeSet;
12 import javax.swing.text.SimpleAttributeSet;
13 import javax.swing.text.StyleConstants;
14 import javax.swing.text.StyledDocument;
15 import javax.swing.text.TabSet;
16 import javax.swing.text.TabStop;
17
18 import KernelAPI.KernelApi;
19
20 public class gSourceCode extends JPanel
21 {
22    private static final long serialVersionUID =
         4279528648857232158L;
23    private static gSourceCode instance = null;
24    private JTextPane textPane;
25
26    private gSourceCode()
27    {
28       this.setLayout(new BorderLayout());
29       textPane = new JTextPane();
30       textPane.setFont(new Font("Courier New",Font.
            PLAIN, 14 ));
31       setTabs(textPane, 4);
32       textPane.getDocument().addDocumentListener(
            Controller.getInstance());
33       add( BorderLayout.CENTER, textPane );
34       updateErrorPosition();
35    }
36
37    public void updateErrorPosition()
38    {
39       MutableAttributeSet errorAttrib = new
            SimpleAttributeSet();
40       MutableAttributeSet normAttrib = new
            SimpleAttributeSet();
```

```
41          StyleConstants.setBackground(errorAttrib, new
               Color(255,150,50));
42          StyledDocument document = textPane.
               getStyledDocument();
43
44          //clear formatting
45          document.setCharacterAttributes(0, textPane.
               getText().length(), normAttrib, true);
46
47          int lineNumber = KernelApi.getErrorLine() - 1;
48          int charPositionInLine = KernelApi.
               getErrorCharPositionInLine();
49          if( lineNumber >= 0 )
50          {
51             Point errorOffsets = getLineStartEndOffset(
                   lineNumber );
52             if( errorOffsets != null )
53             {
54                document.setCharacterAttributes(
                      errorOffsets.x + charPositionInLine,
                       errorOffsets.y - errorOffsets.x -
                      charPositionInLine, errorAttrib,
                      true);
55             }
56          }
57       }
58
59       private Point getLineStartEndOffset(int line)
60       {
61          String text = textPane.getText().replace("\r",
                """); //For some reason indexOf counts \r
               but document doesn't, so just remove all
               of them :)
62          int start = 0, end;
63          while (line>0)
64          {
65             start = text.indexOf("\n", start) + 1;
66             line--;
67          }
68          if( start == -1 )
69          {
70             //just in case of any errors, shouldn't be
                  any since the line info is based on
                  this string
```

```
71          start = 0;
72       }
73       end = text.indexOf("\n", start + 1);
74       if( end == -1 )
75       {
76          end = textPane.getText().length();
77       }
78
79       return new Point(start,end);
80    }
81
82    public String getSourceCode()
83    {
84       return textPane.getText();
85    }
86
87    public void updateSourceCode()
88    {
89       textPane.setText(KernelApi.getSourceCode());
90    }
91
92    private void setTabs( JTextPane textPane, int
         charactersPerTab)
93    {
94       FontMetrics fm = textPane.getFontMetrics(
            textPane.getFont() );
95       int charWidth = fm.charWidth( ' ' );
96       int tabWidth = charWidth * charactersPerTab;
97       TabStop[] tabs = new TabStop[10];
98       for (int j = 0; j < tabs.length; j++)
99       {
100         tabs[j] = new TabStop( (j+1) * tabWidth );
101      }
102      TabSet tabSet = new TabSet(tabs);
103      SimpleAttributeSet attributes = new
            SimpleAttributeSet();
104      StyleConstants.setTabSet(attributes, tabSet);
105      int length = textPane.getDocument().getLength
            ();
106      textPane.getStyledDocument().
            setParagraphAttributes(0, length,
            attributes, false);
107   }
108
```

```
109     public static synchronized gSourceCode
            getInstance()
110     {
111        if( instance == null )
112        {
113           instance = new gSourceCode();
114        }
115        return instance;
116     }
117 }
```

E.4.8 gToolbar.java

```
 1 package GUI;
 2
 3 import javax.swing.JButton;
 4 import javax.swing.JComponent;
 5 import javax.swing.JToolBar;
 6 import javax.swing.KeyStroke;
 7
 8 public class gToolbar
 9 {
10    private static gToolbar instance = null;
11    private JToolBar toolbar;
12
13    private gToolbar()
14    {
15       toolbar = createToolBar( Variables.
             TOOLBAR_ITEMS, Variables.SHORTCUTS );
16    }
17
18    private JToolBar createToolBar( String[] items,
          KeyStroke[] keystrokes )
19    {
20       JToolBar toolbar = new JToolBar();
21       toolbar.setFloatable(false);
22       for( int i = 0;i<items.length;i++ )
23       {
24          String item = items[i];
25          KeyStroke key = keystrokes[i];
26          JButton itemButton = new JButton(item);
27          if( key != null)
```

```
28          {
29             itemButton.registerKeyboardAction(
                  Controller.getInstance(), item, key,
                   JComponent.WHEN_IN_FOCUSED_WINDOW);
30             itemButton.setToolTipText(item);
31          }
32          itemButton.addActionListener(Controller.
               getInstance());
33          toolbar.add( itemButton );
34       }
35       return toolbar;
36    }
37
38    public JToolBar getToolBar()
39    {
40       return toolbar;
41    }
42
43    public static synchronized gToolbar getInstance()
44    {
45       if( instance == null )
46       {
47          instance = new gToolbar();
48       }
49       return instance;
50    }
51 }
```

E.4.9 MainScreen.java

```
1 package GUI;
2
3 import java.awt.BorderLayout;
4 import java.awt.Graphics;
5
6 import javax.swing.JFrame;
7 import javax.swing.JPanel;
8 import javax.swing.JScrollPane;
9 import javax.swing.JSplitPane;
10
11 public class MainScreen extends JFrame
12 {
```

```
13    private static final long serialVersionUID =
          -5921202176825848947L;
14    private static MainScreen instance = null;
15    private String filename;
16    private boolean isDividerLocationInitialized;
17    private JSplitPane BottomPanes, topBottomPane;
18
19    private MainScreen()
20    {
21       super("SyntaxTrain");
22       filename = null;
23       isDividerLocationInitialized = false;
24       //initialize frame
25       setLayout(new BorderLayout());
26       setDefaultCloseOperation( JFrame.
             DO_NOTHING_ON_CLOSE );
27       addWindowListener(Controller.getInstance());
28       setBounds(300, 50, 1000, 700);
29       setExtendedState(MAXIMIZED_BOTH);
30
31       //add toolbar
32       add( BorderLayout.PAGE_START, gToolbar.
             getInstance().getToolBar() );
33
34       //center panel
35       JPanel mainPanel = new JPanel(new BorderLayout
             ());
36          JScrollPane sourceScrollPane = new
               JScrollPane( gSourceCode.getInstance()
               );
37          sourceScrollPane.getVerticalScrollBar().
               setUnitIncrement(16);
38          BottomPanes = new JSplitPane(
39                JSplitPane.HORIZONTAL_SPLIT,
40                sourceScrollPane,
41                gGrammarPanel.getInstance()
42                );
43          BottomPanes.setOneTouchExpandable(true);
44          JScrollPane scrollSourceDiagram = new
               JScrollPane(gErrorTrace.getInstance());
45          topBottomPane = new JSplitPane(JSplitPane.
               VERTICAL_SPLIT, scrollSourceDiagram,
               BottomPanes);
46          topBottomPane.setDividerSize(0);
```

```
47          mainPanel.add(topBottomPane);
48       add( BorderLayout.CENTER, mainPanel );
49
50       //when using int the divider location can be
            set before showing the frame
51       topBottomPane.setDividerLocation(80);
52
53       //This is to ensure enough room is given to
            the source code
54       //(sometimes java doesn't update the frame
            after the 30% division is set in the paint
             function below)
55       BottomPanes.setDividerLocation(400);
56
57       //show frame
58       setVisible(true);
59       doLayout();
60    }
61
62    public void paint(Graphics g)
63    {
64       super.paint(g);
65       if( ! isDividerLocationInitialized )
66       {
67          BottomPanes.setDividerLocation(0.3); //when
               using double the divider must be set
              after showing the frame
68
69          isDividerLocationInitialized = true;
70          repaint();
71       }
72    }
73
74    public void setFilename( String filename )
75    {
76       this.filename = filename;
77       setTitle("SyntaxTrain - " + filename);
78    }
79
80    public String getFilename()
81    {
82       return filename;
83    }
84
```

```
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

```
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
             .CTRL_DOWN_MASK),
24       KeyStroke.getKeyStroke(KeyEvent.VK_R, KeyEvent
             .CTRL_DOWN_MASK),
25       KeyStroke.getKeyStroke(KeyEvent.VK_F5, 0 ),
26       KeyStroke.getKeyStroke(KeyEvent.VK_F10, 0 ),
27       KeyStroke.getKeyStroke(KeyEvent.VK_S, KeyEvent
             .CTRL_DOWN_MASK),
```

```
28          null,
29          null
30      };
31
32      public final static String SEPERATOR = "seperator
            ";
33      public final static String OPEN_SOURCE_FILE = "
            Open";
34      public final static String RELOAD_SOURCE_FILE = "
            Reload file";
35      public final static String SAVE_SOURCE_FILE = "
            Save";
36      public final static String CHECK_SYNTAX = "Check
            syntax";
37      public final static String SHOW_HIDE_BNF_GRAMMARS
            = "Show/hide syntax components";
38      public final static String HELP = "Help";
39      public final static String ABOUT = "About";
40
41      private static boolean codeChanged = false;
42      public static boolean isCodeChanged()
43      {
44          return codeChanged;
45      }
46      public static void setCodeChanged(boolean
            codeChanged)
47      {
48          Variables.codeChanged = codeChanged;
49      }
50      private static boolean diagramsOutOfSync = false;
51      public static boolean isDiagramsOutOfSync()
52      {
53          return diagramsOutOfSync;
54      }
55      public static void setDiagramsOutOfSynch(boolean
            diagramsOutOfSynch)
56      {
57          Variables.diagramsOutOfSync =
                diagramsOutOfSynch;
58      }
59  }
```

E.5 GuiAPI

E.5.1 GuiApi.java

```
1 package GuiAPI;
2
3 import javax.swing.JOptionPane;
4
5 import GUI.MainScreen;
6 import GUI.Variables;
7 import GUI.gGrammarDiagram;
8 import GUI.gGrammarOptions;
9 import GUI.gSourceCode;
10 import GUI.gErrorTrace;
11
12 public class GuiApi
13 {
14    public static void updateDiagrams()
15    {
16       boolean wasCodeChanged = Variables.
             isCodeChanged();
17       gGrammarOptions.getInstance().updateGrammars()
             ;
18       gGrammarDiagram.getInstance().updateDiagram();
19       gErrorTrace.getInstance().updateDiagram();
20       gSourceCode.getInstance().updateErrorPosition
             ();
21
22       Variables.setDiagramsOutOfSynch(false);
23       gErrorTrace.getInstance().updateSyncStatus();
24       Variables.setCodeChanged(wasCodeChanged);
25    }
26
27    public static void updateSourceCode(String
          fileOpened)
28    {
29       MainScreen.getInstance().setFilename(
             fileOpened);
30       gSourceCode.getInstance().updateSourceCode();
31
32       Variables.setCodeChanged(false);
33    }
34
```

```
35    public static void showMessage( String message )
36    {
37       JOptionPane.showMessageDialog(MainScreen.
            getInstance(), message);
38    }
39 }
```

E.6 Init

E.6.1 Init.java

```
1  package Init;
2
3  import java.awt.Color;
4  import java.awt.Font;
5
6  import javax.swing.JOptionPane;
7
8  import Xml.XmlNode;
9
10 import net.hydromatic.clapham.graph.Chart;
11
12 import Exceptions.XMLLoadException;
13 import Exceptions.XMLattributeDoesNotExist;
14 import Exceptions.XMLnodeDoesNotExist;
15 import GUI.MainScreen;
16 import GUI.Variables;
17 import GuiAPI.GuiApi;
18 import Kernel.GrammarInterface;
19 import Library.StdLibrary;
20
21 public class Init
22 {
23    private static final String optionsXmlFile = "
         options.xml";
24
25    public static void main(String[] args)
26    {
27       Chart.titleColor = Color.BLACK;
```

```
28        Chart.titleFont = new Font("Serif", Font.PLAIN
              , 18);
29
30        readOptions();
31
32        MainScreen.getInstance();
33
34        if( ! GrammarInterface.getInstance().
              loadGrammar("Grammar/" + Kernel.Variables.
              grammarName + ".xml") )
35        {
36           GuiApi.showMessage("An error occured while
                 loading grammar file: " + Kernel.
                 Variables.grammarName + ".jar.");
37           System.exit(0);
38        }
39        GrammarInterface.getInstance().compile(); //to
               initialize kernel
40        GuiApi.updateDiagrams(); //to initialize gui
41
42        Variables.setCodeChanged(false);
43        Variables.setDiagramsOutOfSynch(false);
44     }
45
46     private static void readOptions()
47     {
48        String optionsXml = StdLibrary.
              readFileAsString(optionsXmlFile);
49        if( optionsXml == null )
50        {
51           JOptionPane.showMessageDialog( null, "There
                was a problem reading file " +
                 optionsXmlFile + ".");
52           System.exit(0);
53        }
54
55        try
56        {
57           XmlNode options = new XmlNode(optionsXml,
                 "1.0");
58           Kernel.Variables.grammarName = options.
                 getChildNode("GUI").getChildNode("path
                 ").getAttribute("grammarFile");
59           return;
```

```
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

```
1  package Kernel;
2
3  import java.io.File;
4  import java.io.FileOutputStream;
5  import java.io.IOException;
6  import java.net.MalformedURLException;
7  import java.net.URL;
8  import java.net.URLClassLoader;
9
10 import javax.swing.JOptionPane;
11
12 import GuiAPI.GuiApi;
13 import Library.StdLibrary;
14
15 public class GrammarBase
16 {
17    protected String sourceCode;
18    protected String xmlGrammar;
19    protected URLClassLoader classLoader;
```

```
20    protected File currentFile;
21
22    protected GrammarBase()
23    {
24       boolean success = false;
25       String jarFileName = Variables.grammarName +
            ".jar";
26       sourceCode = "";
27       /*
28        * Load jar file
29        */
30       try
31       {
32          File jarFile = new File(jarFileName);
33          if( jarFile.exists())
34          {
35             URL[] urls = {jarFile.toURI().toURL()};
36             classLoader = new URLClassLoader(urls,
                  ClassLoader.getSystemClassLoader());
37             success = true;
38          }
39       }
40       catch (MalformedURLException e)
41       {
42       }
43       if( !success )
44       {
45          JOptionPane.showMessageDialog( null, "Jar
               file could not be read: " + jarFileName
               );
46          System.exit(1);
47       }
48    }
49
50    public void reloadSourceCode()
51    {
52       readSourceCode(currentFile);
53    }
54
55    public void readSourceCode( File file )
56    {
57       currentFile = file;
58       if( file == null )
59       {
```

```
60            return;
61         }
62         sourceCode = StdLibrary.readFileAsString( file
               );
63         GuiApi.updateSourceCode( file.getName() );
64      }
65
66      public boolean loadGrammar( String grammarFile )
67      {
68         xmlGrammar = StdLibrary.readFileAsString(
               classLoader.getResourceAsStream(
               grammarFile ) );
69         return xmlGrammar == null ? false : true;
70      }
71
72      public String getSourceCode()
73      {
74         return sourceCode;
75      }
76      public void setSourceCode( String code )
77      {
78         sourceCode = code;
79      }
80
81      public void saveSourceCode() throws IOException
82      {
83         if( sourceCode == null || currentFile == null
               )
84         {
85            throw new IOException();
86         }
87         FileOutputStream stream = new FileOutputStream
               (currentFile);
88         stream.write(sourceCode.getBytes());
89         stream.close();
90      }
91  }
```

E.7.2 GrammarCompiler.java

```
1  package Kernel;
2
```

```
3 import java.awt.Color;
4 import java.awt.Font;
5 import java.util.ArrayList;
6 import java.util.Collections;
7 import java.util.List;
8 import java.util.Stack;
9
10 import Exceptions.XMLLoadException;
11 import Exceptions.XMLattributeDoesNotExist;
12 import Exceptions.XMLnodeDoesNotExist;
13 import Xml.XmlNode;
14
15 import net.hydromatic.clapham.parser.AlternateNode;
16 import net.hydromatic.clapham.parser.EbnfNode;
17 import net.hydromatic.clapham.parser.IdentifierNode;
18 import net.hydromatic.clapham.parser.LiteralNode;
19 import net.hydromatic.clapham.parser.OptionNode;
20 import net.hydromatic.clapham.parser.ProductionNode;
21 import net.hydromatic.clapham.parser.RepeatNode;
22 import net.hydromatic.clapham.parser.SequenceNode;
23
24 public class GrammarCompiler extends
       SourceCodeCompiler
25 {
26    protected ArrayList<ProductionNode>
          productionNodes;
27    protected ArrayList<String> grammars;
28    private boolean lastHighlightConsumed;
29
30    protected GrammarCompiler()
31    {
32       super();
33       lastHighlightConsumed = false;
34    }
35
36    public ArrayList<String> getGrammars()
37    {
38       return grammars;
39    }
40
41    public List<ProductionNode> getProductionNodes()
42    {
43       return productionNodes;
44    }
```

```
45
46    protected void createBnfComponents()
47    {
48       productionNodes = new ArrayList<ProductionNode
          >();
49       grammars = new ArrayList<String>();
50
51       if( errorTrace == null || xmlGrammar == null )
52       {
53          return;
54       }
55       try
56       {
57          XmlNode xml = new XmlNode(xmlGrammar,
             Variables.xmlVersion);
58          mainForLoop:
59          for(XmlNode rule : xml.getChildNodes("rule
             "))
60          {
61             String ruleName = rule.getAttribute("ID
                ");
62             grammars.add(ruleName);
63             //checks if there's a trace for the
                given rule
64             for( int i=errorTrace.size() - 1;i>=0;i
                -- )
65             {
66                Stack<String> trace = errorTrace.get(
                   i);
67                if( trace.firstElement().equals(
                   ruleName ))
68                {
69                   @SuppressWarnings("unchecked")
70                   Stack<String> clone = (Stack<
                      String>) trace.clone();
71                   clone.remove(0); //remove the
                      first element indicating the
                      rule name
72                   if( i == errorTrace.size() - 1 )
73                   {
74                      if( popLast )
75                      {
76                         clone.pop(); //hack, the
                            very last element in the
```

```
                                error trace is wrong
                               and has to be removed.
77                         }
78                         productionNodes.add(
                               createProductionNodeFromRule
                               ( rule, clone, null, true )
                                );
79                     }
80                     else
81                     {
82                         productionNodes.add(
                               createProductionNodeFromRule
                               ( rule, clone, clone.
                               lastElement(), false ) );
83                     }
84                     continue mainForLoop;
85                 }
86             }
87             productionNodes.add(
                   createProductionNodeFromRule( rule,
                   new Stack<String>(), null, false ) )
                   ;
88         }
89         Collections.sort(grammars);
90     }
91     catch (XMLLoadException e)
92     {
93         e.printStackTrace();
94         System.out.println("XmlLoadException: " + e
               .toString());
95         System.exit(1);
96     }
97     catch (XMLnodeDoesNotExist e)
98     {
99         e.printStackTrace();
100        System.out.println("XMLnodeDoesNotExist: "
               + e.toString());
101        System.exit(1);
102    }
103    catch (XMLattributeDoesNotExist e)
104    {
105        e.printStackTrace();
106        System.out.println("XMLnodeDoesNotExist: "
               + e.toString());
```

```
107             System.exit(1);
108         }
109     }
110 
111     //Returns a production node for the given rule,
            all rules can then be put together into a
            list
112     private ProductionNode
            createProductionNodeFromRule( XmlNode rule,
            Stack<String> highlights, String errorNode,
            boolean markNextTokens )
113     {
114         try
115         {
116 
117 
118             if( !rule.getName().equalsIgnoreCase("rule
                ") )
119             {
120                 System.out.println("GrammarCompiler
                    returns null - 1");
121                 return null;
122             }
123 
124             String id = rule.getAttribute("ID");
125             boolean markFirst = highlights.size() == 0;
126             SequenceNode seq = createSequenceFromXml(
                    rule, highlights, errorNode,
                    markNextTokens, markFirst);
127             return new ProductionNode(new
                    IdentifierNode(id), seq );
128         }
129         catch (XMLattributeDoesNotExist e)
130         {
131             e.printStackTrace();
132         }
133         catch (XMLnodeDoesNotExist e)
134         {
135             e.printStackTrace();
136         }
137         System.out.println("GrammarCompiler returns
                null - 2");
138         return null;
139     }
```

```
140
141     private SequenceNode createSequenceFromXml(
            XmlNode xml, Stack<String> highlights, String
             errorNode, boolean markNextTokens, boolean
            markThisOne ) throws XMLnodeDoesNotExist,
            XMLattributeDoesNotExist
142     {
143         List<EbnfNode> nodes = new ArrayList<EbnfNode
                >();
144         boolean markThis = markThisOne;
145         boolean highLightNext = false;
146
147         for(XmlNode child : xml.getAllChildNodes())
148         {
149             highLightNext = false;
150             if( lastHighlightConsumed || markThis )
151             {
152                 lastHighlightConsumed = false;
153                 markThis = false;
154                 highLightNext = true;
155             }
156             nodes.add(createEbnfNodeFromXml(child,
                    highlights, errorNode, markNextTokens,
                    highLightNext));
157         }
158         return new SequenceNode(nodes);
159     }
160
161     private EbnfNode createEbnfNodeFromXml( XmlNode
            xml, Stack<String> highlights, String
            errorNode, boolean markNextTokens, boolean
            markThisOne ) throws XMLnodeDoesNotExist,
            XMLattributeDoesNotExist
162     {
163         String nodeName = xml.getName();
164         boolean highLightNext = false;
165
166         if( lastHighlightConsumed || markThisOne )
167         {
168             lastHighlightConsumed = false;
169             highLightNext = true;
170         }
171         if( nodeName.equalsIgnoreCase("rule") )
172         {
```

```
173          String id = xml.getAttribute("ID");
174          String uuid = xml.getAttribute("UUID");
175          Font font;
176          if( id.startsWith("'") )
177          {
178             font = new Font( "Courier New", Font.
                PLAIN, 14 );
179
180             if( uuid.equals(errorNode) )
181                return new LiteralNode(id, Color.RED,
                    Color.BLACK, font);
182             else if( (highLightNext || markThisOne)
                && markNextTokens )
183                return new LiteralNode(id, Variables.
                   highlightColor, Variables.
                   highlightColor, font);
184             else if( highlights.remove( uuid ) )
185             {
186                if( highlights.isEmpty() )
187                {
188                   lastHighlightConsumed = true;
189                }
190                return new LiteralNode(id, Color.BLUE
                   , Color.BLUE, font);
191             }
192             else
193                return new LiteralNode(id, Color.
                   BLACK, Color.BLACK, font);
194          }
195          else
196          {
197             font = new Font( "Serif", Font.PLAIN, 14
                );
198
199             if( uuid.equals(errorNode) )
200                return new IdentifierNode(id, Color.
                   RED, Color.BLACK, font);
201             else if( (highLightNext || markThisOne)
                && markNextTokens )
202                return new IdentifierNode(id,
                   Variables.highlightColor,
                   Variables.highlightColor, font);
203             else if( highlights.remove( uuid ) )
204             {
```

```
205                 if( highlights.isEmpty() )
206                 {
207                    lastHighlightConsumed = true;
208                 }
209                 return new IdentifierNode(id, Color.
                       BLUE, Color.BLUE, font);
210              }
211              else
212                 return new IdentifierNode(id, Color.
                       BLACK, Color.BLACK, font);
213           }
214        }
215        else if( nodeName.equalsIgnoreCase("repeat"))
216        {
217           SequenceNode repeatSequence =
                createSequenceFromXml(xml, highlights,
                errorNode, markNextTokens,
                highLightNext);
218           lastHighlightConsumed = highLightNext; //
                the next token should also be
                highlighted
219           return new RepeatNode(repeatSequence);
220        }
221        else if( nodeName.equalsIgnoreCase("or"))
222        {
223           List<EbnfNode> nodes = new ArrayList<
                EbnfNode>();
224           for(XmlNode child : xml.getChildNodes("
                option"))
225           {
226              nodes.add(createEbnfNodeFromXml(child,
                    highlights, errorNode,
                    markNextTokens, highLightNext));
227           }
228           return new AlternateNode(nodes);
229        }
230        else if( nodeName.equalsIgnoreCase("option"))
231        {
232           //this section is called from the "or" node
                above
233           return createSequenceFromXml(xml,
                highlights, errorNode, markNextTokens,
                highLightNext);
234        }
```

```
235         else if( nodeName.equalsIgnoreCase("optional")
                )
236         {
237            SequenceNode optionalSequence =
                  createSequenceFromXml(xml, highlights,
                  errorNode, markNextTokens,
                  highLightNext);
238            lastHighlightConsumed = highLightNext; //
                  the next token should also be
                  highlighted
239            return new OptionNode(optionalSequence);
240         }
241         throw new XMLnodeDoesNotExist("Unknown xml
               node.");
242      }
243   }
```

E.7.3 GrammarInterface.java

```
1  package Kernel;
2
3  import GuiAPI.GuiApi;
4
5  public class GrammarInterface extends
       GrammarCompiler
6  {
7     private static GrammarInterface instance = null;
8
9     private GrammarInterface()
10    {
11       super();
12    }
13
14    public void compile()
15    {
16       compileSourceCode();
17       createBnfComponents();
18       GuiApi.updateDiagrams();
19    }
20
21    public static synchronized GrammarInterface
          getInstance()
```

```
22    {
23       if(instance == null)
24       {
25          instance = new GrammarInterface();
26       }
27       return instance;
28    }
29 }
```

E.7.4 SourceCodeCompiler.java

```
1  package Kernel;
2
3  import java.io.ByteArrayInputStream;
4  import java.io.IOException;
5  import java.io.InputStream;
6  import java.io.UnsupportedEncodingException;
7  import java.lang.reflect.Constructor;
8  import java.lang.reflect.InvocationTargetException;
9  import java.util.Stack;
10
11 import org.antlr.runtime.ANTLRInputStream;
12 import org.antlr.runtime.CharStream;
13 import org.antlr.runtime.CommonTokenStream;
14 import org.antlr.runtime.Lexer;
15 import org.antlr.runtime.RecognitionException;
16 import org.antlr.runtime.TokenStream;
17
18 import Grammar.BnfParser;
19
20 public class SourceCodeCompiler extends GrammarBase
21 {
22    /**
23     * Stack of rules which are in the error (top
          down).
24     * Each rule is a stack trace indicating which
          steps in these rules have been taken (
          starting with the rule name as the first
          element)
25     *
26     * Example:
27     *       compilationUnit
```

```
28      *       typeDeclaration
29      *       classDeclaration 'class' IDENTIFIER '{'
            fieldDeclaration
30      *
31      * This means the last '}' is missing (if
            fieldDeclaration weren't there, the error
            would be in that.
32      */
33     protected Stack<Stack<String>> errorTrace;
34     protected int errorLine, errorCharPositionInLine;
35     protected boolean popLast;
36     private Constructor<Lexer> LexerConstructor;
37     private Constructor<BnfParser> ParserConstructor;
38
39     protected SourceCodeCompiler()
40     {
41        super();
42        boolean success = false;
43        errorTrace = null;
44        errorLine = -1;
45        errorCharPositionInLine = -1;
46        popLast = false;
47
48        /*
49         * Load classes
50         */
51        try
52        {
53           @SuppressWarnings("unchecked")
54           Class<Lexer> CLexer = (Class<Lexer>) Class.
                forName("Grammar." + Variables.
                grammarName + "Lexer", false,
                classLoader);
55           @SuppressWarnings("unchecked")
56           Class<BnfParser> CParser = (Class<BnfParser
                >) Class.forName("Grammar." + Variables
                .grammarName + "Parser", false,
                classLoader);
57
58           LexerConstructor = CLexer.getConstructor(
                new Class[]{CharStream.class});
59           ParserConstructor = CParser.getConstructor(
                new Class[]{TokenStream.class});
60
```

```
61          //A little check to make sure files can be
               loaded
62          InputStream is = new ByteArrayInputStream((
               new String()).getBytes("UTF-8"));
63          ANTLRInputStream input = new
               ANTLRInputStream(is);
64
65          Lexer lexer = LexerConstructor.newInstance(
               new Object[]{input});
66
67          CommonTokenStream tokens = new
               CommonTokenStream(lexer);
68
69          @SuppressWarnings("unused")
70          BnfParser parser = ParserConstructor.
               newInstance(new Object[]{tokens});
71
72          success = true;
73       }
74       catch (SecurityException e)
75       {
76          e.printStackTrace();
77       }
78       catch (ClassNotFoundException e)
79       {
80          e.printStackTrace();
81       }
82       catch (NoSuchMethodException e)
83       {
84          e.printStackTrace();
85       }
86       catch (UnsupportedEncodingException e)
87       {
88          e.printStackTrace();
89       }
90       catch (IllegalArgumentException e)
91       {
92          e.printStackTrace();
93       }
94       catch (IOException e)
95       {
96          e.printStackTrace();
97       }
98       catch (InstantiationException e)
```

```
99          {
100            e.printStackTrace();
101         }
102         catch (IllegalAccessException e)
103         {
104            e.printStackTrace();
105         }
106         catch (InvocationTargetException e)
107         {
108            e.printStackTrace();
109         }
110         if( !success )
111         {
112            System.out.println("Invalid Jar file!");
113            System.exit(1);
114         }
115      }
116
117      public Stack<Stack<String>> getErrorTrace()
118      {
119         return errorTrace;
120      }
121
122      public int getErrorLine()
123      {
124         return errorLine;
125      }
126
127      public int getErrorCharPositionInLine()
128      {
129         return errorCharPositionInLine;
130      }
131
132      protected void compileSourceCode()
133      {
134         try
135         {
136            if( getSourceCode() == null )
137               return;
138            InputStream is = new ByteArrayInputStream(
                  getSourceCode().getBytes("UTF-8"));
139            ANTLRInputStream input = new
                  ANTLRInputStream(is);
140
```

```
141             Lexer lexer = LexerConstructor.newInstance(
                    new Object[]{input});
142
143             CommonTokenStream tokens = new
                    CommonTokenStream(lexer);
144
145             BnfParser parser = ParserConstructor.
                    newInstance(new Object[]{tokens});
146
147             try
148             {
149                 parser.bnf();
150                 //System.out.println("No errors :)");
151             }
152             catch(RuntimeException e)
153             {
154                 //System.out.println("Failed, errors in
                        code :(");
155             }
156             errorTrace = parser.trace;
157             errorLine = parser.errorLine;
158             errorCharPositionInLine = parser.
                    errorCharPositionInLine;
159             popLast = parser.popLast;
160
161             return;
162         }
163         catch (UnsupportedEncodingException e)
164         {
165         }
166         catch (IOException e)
167         {
168         }
169         catch (RecognitionException e)
170         {
171         }
172         catch (InstantiationException e1)
173         {
174             System.out.println("Invalid Jar file!");
175             System.exit(1);
176         }
177         catch (IllegalAccessException e1) {
178             System.out.println("Invalid Jar file!");
179             System.exit(1);
```

```
180         }
181         catch (InvocationTargetException e1)
182         {
183            System.out.println("Invalid Jar file!");
184            System.exit(1);
185         }
186         errorTrace = null;
187      }
188 }
```

E.7.5 Variables.java

```
1  package Kernel;
2
3  import java.awt.Color;
4  import java.util.Hashtable;
5
6  public class Variables
7  {
8     public static final String xmlVersion = "1.0";
9
10    public static String grammarName = "invalid";
11    public static Hashtable<String, String>
         idToVariable;
12
13    public static final Color highlightColor = new
         Color(255,150,50);
14 }
```

E.8 KernelAPI

E.8.1 KernelApi.java

```
1  package KernelAPI;
2
3  import java.io.File;
4  import java.io.IOException;
5  import java.util.ArrayList;
```

```
6 import java.util.List;
7 import java.util.Stack;
8
9 import net.hydromatic.clapham.parser.ProductionNode;
10
11 import Kernel.GrammarInterface;
12
13 public class KernelApi
14 {
15    //GETTERS
16    public static String getSourceCode()
17    {
18       return GrammarInterface.getInstance().
             getSourceCode();
19    }
20
21    public static Stack<Stack<String>> getErrorTrace
          ()
22    {
23       return GrammarInterface.getInstance().
             getErrorTrace();
24    }
25
26    public static ArrayList<String> getGrammars()
27    {
28       return GrammarInterface.getInstance().
             getGrammars();
29    }
30
31    public static List<ProductionNode>
          getGrammarProductionNodes()
32    {
33       return GrammarInterface.getInstance().
             getProductionNodes();
34    }
35
36    public static int getErrorLine()
37    {
38       return GrammarInterface.getInstance().
             getErrorLine();
39    }
40    public static int getErrorCharPositionInLine()
41    {
```

```
42        return GrammarInterface.getInstance().
              getErrorCharPositionInLine();
43    }
44
45    public static void saveSourceCode() throws
          IOException
46    {
47        GrammarInterface.getInstance().saveSourceCode
              ();
48    }
49    public static void setSourceCode(String code)
50    {
51        GrammarInterface.getInstance().setSourceCode(
              code);
52        GrammarInterface.getInstance().compile();
53    }
54    public static void readSourceFile( File file )
55    {
56        GrammarInterface.getInstance().readSourceCode(
               file );
57        GrammarInterface.getInstance().compile();
58    }
59    public static void reloadSourceCode()
60    {
61        GrammarInterface.getInstance().
              reloadSourceCode();
62        GrammarInterface.getInstance().compile();
63    }
64 }
```

E.9 Library

E.9.1 Lock.java

```
1 package Library;
2
3 public class Lock
4 {
5    private boolean isTaken;
6
```

```
7     public Lock()
8     {
9        isTaken = false;
10    }
11
12    public synchronized void P()
13    {
14       while( isTaken )
15       {
16          try {
17             wait();
18          } catch (InterruptedException e)
19          {
20          }
21       }
22       isTaken = true;
23    }
24
25    public synchronized void V()
26    {
27       isTaken = false;
28       notify();
29    }
30 }
```

E.9.2 StdLibrary.java

```
1 package Library;
2
3 import java.io.File;
4 import java.io.FileInputStream;
5 import java.io.FileNotFoundException;
6 import java.io.FileReader;
7 import java.io.IOException;
8 import java.io.InputStream;
9 import java.io.InputStreamReader;
10 import java.io.Reader;
11
12 public class StdLibrary
13 {
14    public static String xmlEscapeString( String str
         )
```

```
15    {
16       return str.replace("\"", "&quot;").replace
             ("&", "&amp;").replace("'", "&apos;").
             replace("<", "&lt;").replace(">", "&gt;");
17    }
18
19    public static String xmlUnEscapeString( String
         str )
20    {
21       return str.replace("&quot;", "\"").replace("&
             amp;", "&").replace("&apos;", "'").replace
             ("&lt;", "<").replace("&gt;", ">");
22    }
23
24    public static String readFileAsString(InputStream
          inputstream)
25    {
26       if( inputstream == null )
27       {
28          return null;
29       }
30         return readFileAsString(new
              InputStreamReader(inputstream));
31     }
32    public static String readFileAsString(File file)
33    {
34         try
35         {
36          return readFileAsString(new FileInputStream
               (file));
37       }
38         catch (FileNotFoundException e)
39       {
40          return null;
41       }
42     }
43    /**
44     * Reads a file and returns it's content
45     * @param filePath path of the file to read
46     * @return Content of the file specified, returns
          null if file does not exist or there's an
         IOException.
47     */
```

```
48    public static String readFileAsString(String
          filePath)
49    {
50          try
51          {
52           return readFileAsString(new FileReader(
                 filePath));
53          }
54          catch (FileNotFoundException e)
55          {
56           return null;
57          }
58     }
59
60    public static String readFileAsString(Reader
          reader)
61    {
62          try {
63           StringBuffer fileData = new StringBuffer
                 (1000);
64           char[] buf = new char[1024];
65           int numRead=0;
66           while((numRead=reader.read(buf)) != -1){
67               String readData = String.valueOf(buf,
                     0, numRead);
68               fileData.append(readData);
69               buf = new char[1024];
70           }
71           reader.close();
72           return fileData.toString();
73          }
74        catch (IOException e) {}
75        return null;
76     }
77 }
```

E.10 Xml

E.10.1 XmlNode.java

```
1 package Xml;
2
3 import java.io.StringReader;
4 import java.util.ArrayList;
5
6 import javax.xml.parsers.DocumentBuilder;
7 import javax.xml.parsers.DocumentBuilderFactory;
8
9 import org.w3c.dom.Document;
10 import org.w3c.dom.Element;
11 import org.w3c.dom.NamedNodeMap;
12 import org.w3c.dom.Node;
13 import org.w3c.dom.NodeList;
14 import org.xml.sax.InputSource;
15
16 import Exceptions.XMLLoadException;
17 import Exceptions.XMLTextDoesNotExist;
18 import Exceptions.XMLattributeDoesNotExist;
19 import Exceptions.XMLnodeDoesNotExist;
20
21 public class XmlNode
22 {
23    private Node node;
24    private Node currentNodeGiven;
25
26    public XmlNode(String xml, String version) throws
          XMLLoadException
27    {
28       String xmlVersion;
29       currentNodeGiven = null;
30       try
31       {
32          DocumentBuilderFactory dbf =
              DocumentBuilderFactory.newInstance();
33          DocumentBuilder db = dbf.newDocumentBuilder
              ();
34          Document doc = db.parse(new InputSource(new
               StringReader(xml)));
35          node = doc.getDocumentElement();
36          xmlVersion = getAttribute("version");
37       } catch (Exception e)
38       {
39          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;
```

```
74        Node sibling = temp.getNextSibling();
75        while(sibling != null)
76        {
77           if(sibling.getNodeType() == Node.
                 ELEMENT_NODE)
78              return true;
79           sibling = sibling.getNextSibling();
80        }
81        return false;
82     }
83     public String getName()
84     {
85        return node.getNodeName();
86     }
87     public ArrayList<XmlNode> getAllChildNodes()
          throws XMLnodeDoesNotExist
88     {
89        ArrayList<XmlNode> childNodes = new ArrayList<
             XmlNode>();
90        NodeList nodes = node.getChildNodes();
91        for(int i=0; i<nodes.getLength(); i++)
92        {
93           Node node = nodes.item(i);
94           if (node.getNodeType() == Node.ELEMENT_NODE
                 )
95              childNodes.add(new XmlNode(node));
96        }
97        return childNodes;
98     }
99     public ArrayList<XmlNode> getChildNodes(String
          nodeName) throws XMLnodeDoesNotExist
100    {
101       ArrayList<XmlNode> childNodes = new ArrayList<
             XmlNode>();
102       NodeList nodes = node.getChildNodes();
103       for(int i=0; i<nodes.getLength(); i++)
104       {
105          Node node = nodes.item(i);
106          if (node.getNodeType() == Node.ELEMENT_NODE
                 )
107             if(((Element) node).getTagName().
                   equalsIgnoreCase(nodeName))
108                childNodes.add(new XmlNode(node));
109       }
```

```
110         return childNodes;
111     }
112     public XmlNode getChildNode(String nodeName, int
            skipAmount) throws XMLnodeDoesNotExist
113     {
114         NodeList nodes = node.getChildNodes();
115         for(int i=0; i<nodes.getLength(); i++)
116         {
117             Node node = nodes.item(i);
118             if (node.getNodeType() == Node.ELEMENT_NODE
                    )
119                 if(((Element) node).getTagName().
                        equalsIgnoreCase(nodeName))
120                     if(skipAmount-- > 0)
121                         continue;
122                     else
123                         return new XmlNode(node);
124         }
125         throw new XMLnodeDoesNotExist( nodeName );
126     }
127     public XmlNode getChildNode(String nodeName)
            throws XMLnodeDoesNotExist
128     {
129         return getChildNode(nodeName, 0);
130     }
131     public int getIntAttribute(String attName) throws
             XMLattributeDoesNotExist
132     {
133         return Integer.parseInt(getAttribute(attName))
                ;
134     }
135     public boolean getBooleanAttribute( String
            attName ) throws XMLattributeDoesNotExist
136     {
137         return Boolean.parseBoolean( getAttribute(
                attName) );
138     }
139     public String getAttribute(String attName) throws
             XMLattributeDoesNotExist
140     {
141         NamedNodeMap nodeMap = node.getAttributes();
142         for(int i=0;i<nodeMap.getLength();i++)
143         {
144             Node node = nodeMap.item(i);
```

```
145             if(node.getNodeType() != Node.
                    ATTRIBUTE_NODE)
146             {
147                 System.out.println("XmlNode -
                        getAttribute: Node: " + node + " is
                        not an attribute! :S");
148             }
149             if(node.getNodeName().equalsIgnoreCase(
                    attName))
150             {
151                 return node.getNodeValue();
152             }
153         }
154         throw new XMLattributeDoesNotExist(attName);
155     }
156     public String getText() throws
            XMLTextDoesNotExist
157     {
158         NodeList nodes = node.getChildNodes();
159         for(int i=0; i<nodes.getLength(); i++)
160         {
161             Node node = nodes.item(i);
162             if (node.getNodeType() == Node.TEXT_NODE)
163                 return node.getNodeValue();
164         }
165         throw new XMLTextDoesNotExist();
166     }
167 }
```

Bibliography

[1] Gregory R. Andrews. *Foundations of Multithreaded, Parallel, and Distributed Programming*. Addison-Wesley, 2007.

[2] Ed. D. Crocker. Augmented bnf for syntax specifications: Abnf. http://tools.ietf.org/html/rfc5234, January 2008.

[3] Matthew C. Jadud. Methods and tools for exploring novice compilation behaviour. 2006.

[4] Helen Sharp Yvonne Rogers and Jenny Preece. *Interaction Design: Beyond Human-Computer Interaction*. John Wiley & Sons, second edition, 2007.