ANTLR

ANTLR Intro

- ANTLR is a parser generator, a tool that helps you to create parsers. **A parser takes a piece of text and transforms it in an organized structure**, a *parse tree*, also known as a *Abstract Syntax Tree (AST)*.

What you need to get parse tree

- define a lexer and parser grammar
- invoke ANTLR: it will generate a lexer and a parser in your target language (e.g., Java, Python, C#, JavaScript)
- use the generated lexer and parser: you invoke them passing the code to recognize and they return to you a parse tree

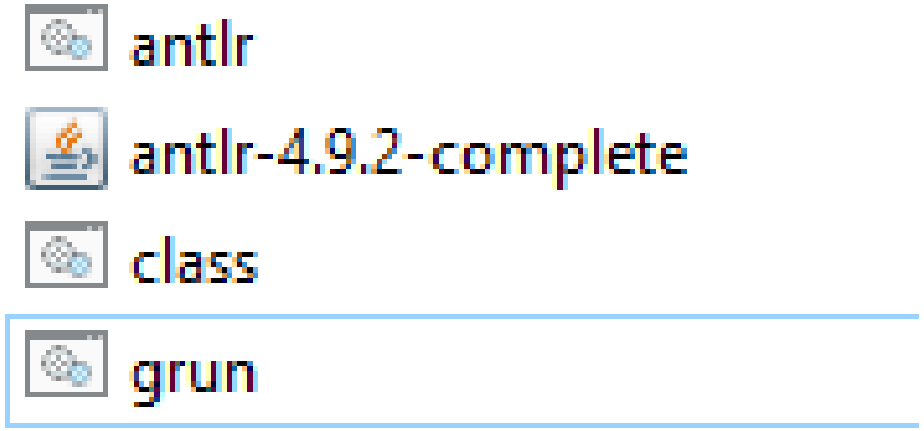
Setup ANTLR

- Have atleast Java 1.7
- Install ANTLR Jar File
  - https://www.antlr.org/download/antlr-4.9.2-complete.jar

- copy the downloaded tool where you usually put third-party java libraries (ex. /usr/local/lib or C:\Program Files\Java\libs)

Setup Antlr

- Create antlr.bat file insert “java org.antlr.v4.Tool %*”
- Create class.bat insert “SET CLASSPATH=.;%CLASSPATH%”
- Create grun.bat insert “java org.antlr.v4.gui.TestRig %*”

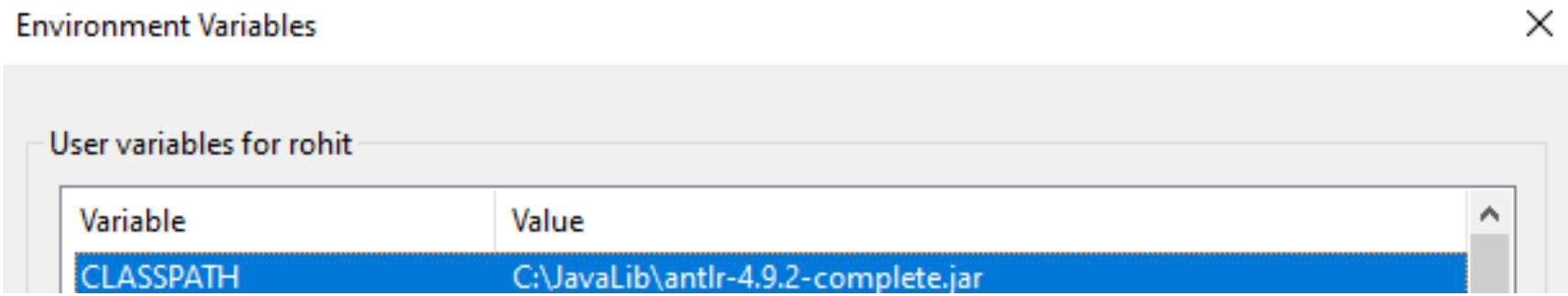


Folder should look something like this

Setup Antlr

- add the tool to your CLASSPATH. Add it to your startup script (ex. .bash_profile)

- (optional) add also aliases to your startup script to simplify the usage of ANTLR



Executing the instructions on Linux/Mac OS

Raw Copy

```
# 1.
sudo cp antlr-4.9.2-complete.jar /usr/local/lib/
# 2. and 3.
# add this to your .bash_profile
export CLASSPATH=".:/usr/local/lib/antlr-4.9.2-complete.jar:$CLASSPATH"
# simplify the use of the tool to generate lexer and parser
alias antlr4='java -Xmx500M -cp "/usr/local/lib/antlr-4.9.2-complete.jar:$CLASSPATH" org.antlr.v4.Tool'
# simplify the use of the tool to test the generated code
alias grun='java -Xmx500M -cp "/usr/local/lib/antlr-4.9.2-complete.jar:$CLASSPATH" org.antlr.v4.gui.TestRig'
```

Executing the instructions on Windows

```
1.  // 1. Copy antlr-4.9.2-complete.jar in C:\Program Files\Java\libs (or
    wherever you prefer)
2.  // 2. Append the location of ANTLR to the CLASSPATH variable on your
    system, or create a CLASSPATH variable if you have not done so before
3.  // you can do to that by pressing WIN + R and typing sysdm.cpl, then
    selecting Advanced (tab) > Environment variables > System Variables
4.  // CLASSPATH -> .;C:\Program Files\Java\libs\antlr-4.9.2-
    complete.jar;%CLASSPATH%
5.  // 3. Add aliases
6.  // create antlr4.bat
7.  java org.antlr.v4.Tool %*
8.  // create grun.bat
9.  java org.antlr.v4.gui.TestRig %*
10. // put them in the system PATH or any of the directories included in
    your PATH
```

Typical Workflow

- When you use ANTLR you start writing a *grammar*, a file with extension .g4, which contains rule of the language you are analzing.
- You then use the antlr4 program to generate the files that your program will actually use, such as the lexer and the parser.

antlr4 <options> <grammar-file-g4>

Sample main program for invoking your grammar

- You simple swap out “YourLexer” and “YourParser” for the appropriate names of your lexer and parser.

```
import sys
from antlr4 import *
from YourLexer import YourLexer
from YourParser import YourParser


def main(argv):
    if len(sys.argv) > 1:
        in = FileStream(sys.argv[1])
    else:
        in = InputStream(sys.stdin.readline())

    lexer = YourLexer(in)
    tokens = CommonTokenStream(lexer)
    parser = YourParser(tokens)
    tree = parser.prog()
    print(tree.toStringTree(recog=parser))

if __name__ == '__main__':
    main(sys.argv)
```

ANTLR Grammar

- ANTLR generates code from a grammar, which describes what is valid in the language and how the language is structured.
- Found in arithmetic.g4

Compiling

- ANTLR turns this grammar file into two parts: a lexer, which reads the input stream and turns it into tokens; and a parser, which associates the tokens with the elements of the grammar we named above.

```
antlr4 –Dlanguage=Python3 arithmetic.g4
```

- This command generates arithmeticLexer.py, arithmeticParser.py, and arithmeticListener.py.
- The listener is a new design element of ANTLR 4 and is designed to make it easier to write code that handles events from the parser, without being impacted if the grammar is modified and re-compiled.

Tree Walking

- We start by reading the input text stream and passing it through the lexer and parser builds a tree.
- Now we need to handle the expressions. This is done by passing tree through a handleExpression function

Output

- Run command

  echo “2 * 8 - 7 + 2” | python arithmetic.py
- Create a txt file inp.txt containing “2 * 8 - 7 + 2”

  Python arithmetic.py < inp.txt

- Output would

```
Parsed expression 2*8-7+2 has value 11
```

References

- https://tomassetti.me/antlr-mega-tutorial/
- https://jason.whitehorn.us/blog/2021/02/08/getting-started-with-antlr-for-python/
- https://dzone.com/articles/antlr-4-with-python-2-detailed-example
- https://faun.pub/introduction-to-antlr-python-af8a3c603d23
- https://gist.github.com/jeroendeswaef/563cd2ab68ab895aedff
- https://github.com/antlr/grammars-v4