

 Instituto Superior de
Engenharia do Porto


Introdução ao ANTLR 4



- ANother Tool for Language Recognition
- www.antlr.org

José Tavares – José Marinho (2021)
Baseado em:
Oliver Zeigermann
Code Generation Cambridge (2013)

1

 Instituto Superior de
Engenharia do Porto

Agenda

- Instalação
- Introdução ao *parsing* com ANTLR4
- *Use case*: Interpretador
- *Use case*: Gerador de Código

2

Instalação (<http://www.antlr.org/>)

WINDOWS

- Download
`https://www.antlr.org/download/antlr-4.9.2-complete.jar`.
- Add `antlr4-complete.jar` to CLASSPATH, either:
 - Permanently: Using System Properties dialog > Environment variables > Create or append to CLASSPATH variable
 - Temporarily, at command line:
`SET CLASSPATH=.;C:\Java\lib\antlr4-complete.jar;%CLASSPATH%`
- Create batch commands for ANTLR Tool, TestRig in dir in PATH
 - `antlr4.bat: java org.antlr.v4.Tool %*`
 - `grun.bat: java org.antlr.v4.gui.TestRig %*`

3

Instalação (<http://www.antlr.org/>)

OS X

```
$ cd /usr/local/lib
$ sudo curl -O https://www.antlr.org/download/antlr-4.9.2-complete.jar
$ export CLASSPATH=".:usr/local/lib/antlr-4.9.2-complete.jar:$CLASSPATH"
$ alias antlr4='java -jar /usr/local/lib/antlr-4.9.2-complete.jar'
$ alias grun='java org.antlr.v4.gui.TestRig'
```

4

isep Instituto Superior de Engenharia do Porto

Instalação (<http://www.antlr.org/>)

LINUX

```
$ cd /usr/local/lib
$ wget https://www.antlr.org/download/antlr-4.9.2-complete.jar
$ export CLASSPATH=".:usr/local/lib/antlr-4.9.2-complete.jar:$CLASSPATH"
$ alias antlr4='java -jar /usr/local/lib/antlr-4.9.2-complete.jar'
$ alias grun='java org.antlr.v4.gui.TestRig'
```

5

isep Instituto Superior de Engenharia do Porto

Parsing com ANTLR4?

Identificar e revelar a estrutura implícita de uma entrada numa árvore de *parse*


- A estrutura é descrita como uma gramática
- É dividido em analisador léxico e sintático
- A árvore de *parse* pode ser auto-gerada

```

graph LR
    chars["chars  
sp = 100;"] --> LEXER
    subgraph Language_recognizer [Language recognizer]
        LEXER --> tokens["tokens  
sp = 100 ;"]
        tokens --> PARSER
    end
    PARSER --> parse_tree
    subgraph parse_tree [parse tree]
        stat --> assign
        assign --> sp
        assign --> eq["="]
        assign --> expr
        assign --> semicolon[";"]
        expr --> 100
    end

```

6



Instituto Superior de Engenharia do Porto

Primeiro Exemplo

➤ **Hello.g4**

```
// Define a grammar called Hello
grammar hello;
inicio : 'hello' ID ; //match hello followed by an identifier
ID : [a-z]+ ; //match lower-case identifiers
WS : [ \t\r\n]+ -> skip ; //skip spaces, tabs, newlines
```

7


Instituto Superior de Engenharia do Porto

Primeiro Exemplo

➤ **Hello.g4**

```
// Define a grammar called Hello
grammar hello;
inicio : 'hello' ID ; //match hello followed by an identifier
ID : [a-z]+ ; //match lower-case identifiers
WS : [ \t\r\n]+ -> skip ; //skip spaces, tabs, newlines
```

- **Case-sensitive!!**
- **Correr ANTLR4 na linha de comando**


```
$ antlr4 hello.g4
$ javac hello*.java
```

8

Primeiro Exemplo

- Testar na linha de comando

```
$ grun hello inicio -gui
hello lprog
^Z
```



➤ ^Z means EOF on Windows; it's ^D in Linux

9

Exemplo Prático:
um interpretador de expressões

- Construir um interpretador a partir do zero
 - Completo e funcional
 - Analisa expressões numéricas
 - Avalia as expressões (faz cálculos)

10

Regra léxica para valores inteiros

INT : ('0'..'9')+ ;

- Regra para o *token* INT
 - Composto pelos algarismos de 0 a 9
 - Pelo menos um algarismo
 - Em alternativa pode ser:
- INT : [0-9]+ ;

11

Regra sintática para as expressões

```

expr : expr ('*' | '/') expr
      | expr ('+' | '-') expr
      | INT
      ;
  
```

- Três opções
- A precedência de operadores por ordem de opções
- Implementa automaticamente a recursividade à esquerda

12



Instituto Superior de
Engenharia do Porto

\$ antlr4

ANTLR Parser Generator Version 4.9.2

- o ____ specify output directory where all output is generated
- lib ____ specify location of grammars, tokens files
- atn generate rule augmented transition network diagrams
- encoding ____ specify grammar file encoding; e.g., euc-jp
- message-format ____ specify output style for messages in antlr, gnu, vs2005
- long-messages show exception details when available for errors and warnings
- listener generate parse tree listener (default)**
- no-listener don't generate parse tree listener
- visitor generate parse tree visitor
- no-visitor don't generate parse tree visitor (default)**
- package ____ specify a package/namespace for the generated code
- depend generate file dependencies
- D<option>=value set/override a grammar-level option
- werror treat warnings as errors
- xdbgST launch StringTemplate visualizer on generated code
- xdbgSTwait wait for STviz to close before continuing
- xforce-atn use the ATN simulator for all predictions
- xlog dump lots of logging info to antlr-timestamp.log
- xexact-output-dir all output goes into -o dir regardless of paths/package

13



Instituto Superior de
Engenharia do Porto

\$ grun

```
java org.antlr.v4.gui.TestRig GrammarName startRuleName
  [-tokens] [-tree] [-gui] [-ps file.ps] [-encoding encodingname]
  [-trace] [-diagnostics] [-SLL]
  [input-filename(s)]
Use startRuleName='tokens' if GrammarName is a lexer grammar.
Omitting input-filename makes rig read from stdin.
```

14



ISEP
Instituto Superior de
Engenharia do Porto

Testar gramática

➤ **Expressions.g4**

```


grammar Expressions;
expr : expr ('*' | '/') expr
    | expr ('+' | '-') expr
    | INT
    ;
INT : ('0'..'9')+ ;
WS : [ \t\r\n]+ -> skip ;
  
```

\$ antlr4 -o tmp -no-listener -no-visitor Expressions.g4
\$ cd tmp

Expressions.interp	INTERP File
Expressions.tokens	TOKENS File
ExpressionsLexer.interp	INTERP File
ExpressionsLexer.java	JAVA File
ExpressionsLexer.tokens	TOKENS File
ExpressionsParser.java	JAVA File

✓ Cria a pasta *tmp* com as classes java para o *Lexer* e o *Parser* →

15



ISEP
Instituto Superior de
Engenharia do Porto

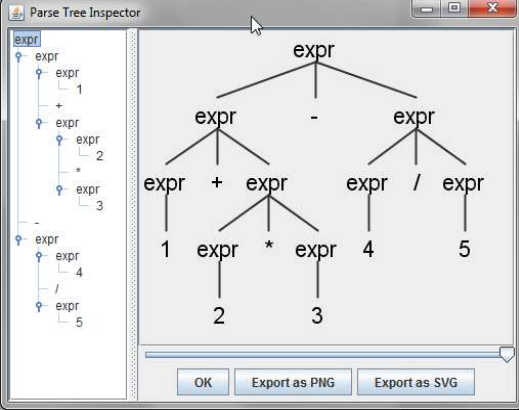
Testar gramática

➤ **Expressions.g4**

```

grammar Expressions;
expr : expr ('*' | '/') expr
    | expr ('+' | '-') expr
    | INT
    ;
INT : ('0'..'9')+ ;
WS : [ \t\r\n]+ -> skip ;
  
```

\$ javac Expressions*.java
\$ grun Expressions **expr** -gui
1+2*3-4/5
^Z



➤ *^Z means EOF on Windows; it's ^D in Linux*

16

Fazer algo com base na entrada

Opções para implementação de ações em gramáticas com ANTLR4

1. *Listeners ou Visitors (classes)*
2. Ações Semânticas na *gramática*

17

Listeners/Visitors vs Ações Semânticas

- Para além da análise sintática é normalmente necessário executar operações adicionais.
- No exemplo da gramática de análise de expressões que valida a sua sintaxe pode ser necessário avaliar estas expressões.
- Se a avaliação das expressões resultar num conjunto de instruções simples e não for necessário incorporar a avaliação das expressões numa aplicação mais complexa, então o código para a avaliação das expressões pode ser embebido na própria gramática. **(2. Ações Semânticas)**
- Caso contrário, se a análise sintática for apenas uma operação de uma aplicação mais complexa dever-se-á evitar embeber na gramática o código para avaliar as expressões, separando a gramática deste código. **(1. Listeners ou Visitors)**

18

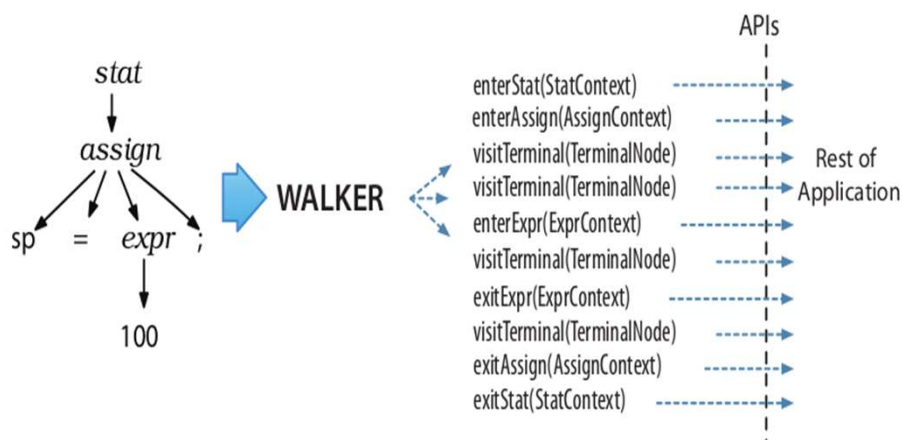
1. Listeners vs Visitors

- O ANTLR4 fornece suporte para dois mecanismos (**Listeners** e **Visitors**) que permitem percorrer a árvore de *parse*.
- Por omissão, o ANTLR4 gera uma **listener interface** da árvore de *parse*, que responde aos eventos acionados por um objeto **walker** capaz de percorrer a árvore *parse*.
- **A maior diferença entre os mecanismos Listener e Visitor consiste:**
 - os métodos de um objeto **listener** são chamados automaticamente por um objeto **walker** fornecido por ANTLR
 - os métodos de um objeto **visitor** devem chamar os nós filhos de um dado nó da árvore de *parse* para visita explícita.

19

1. Listeners vs Visitors

- **WALKER → Listener**



20

isep Instituto Superior de Engenharia do Porto

1. Listeners vs Visitors

- Os **Visitors** controlam ativamente a travessia na árvore de *parse*

- Os métodos de um objeto **visitor** devem chamar os nós filhos de um dado nó da árvore de *parse* para visitas explícitas
- Esquecer a invocação de métodos para visita aos nós filhos de um nó da árvore de *parse* significa que essas subárvores não serão visitadas

21

isep Instituto Superior de Engenharia do Porto

1. Listeners vs Visitors

- Os métodos de um **listener** não podem retornar um valor
- Os métodos de um **visitor** podem retornar um valor de um qualquer tipo personalizado.
- Os **visitors** são usados para visitar seletivamente nós numa árvore de *parse*.
- Listeners** são bons para realizar uma dada tarefa sempre que o *parser* encontra uma regra específica.
 - embora o mesmo efeito possa ser obtido com um **visitor**, o uso de um **listener** não exige que se visite manualmente os nós da árvore de *parse*.

22

isep Instituto Superior de Engenharia do Porto

Demo App

Exemplo de uso de *Visitors* e *Listeners*

23

isep Instituto Superior de Engenharia do Porto

Criar aplicação JAVA *(Netbeans)*

- Criar projeto *Netbeans* (com nome **expressions**)
- Descarregar a biblioteca ANTLR de www.antlr.org e acrescentar ao projeto (*lib\antlr-4.9.2-complete.jar*)
- Acrescentar/copiar o ficheiro ***Expressions.g4*** para a pasta *src\expressions* do projeto.
- ***Nota:*** Não tem de ser o *Netbeans*. Pode ser usado outro *IDE*.

24



Instituto Superior de
Engenharia do Porto

Criar aplicação JAVA

➤ **Gramática:**

```
// Expressions.g4
grammar Expressions;

expr : left=expr op=('*' | '/') right=expr #opExprMulDiv
    | left=expr op=('+' | '-') right=expr #opExprSumDif
    | atom=INT #atomExpr
    ;
INT : ('0'..'9')+ ;
WS : [ \t\r\n]+ -> skip ;
```

✓ Sempre que alteramos a gramática é possível testá-la fora do projeto (os ficheiros gerados podem ser enviados para uma pasta *tmp*)

```
$ antlr4 -o ./tmp -no-listener -no-visitor Expressions.g4
$ cd ./tmp
$ javac Expressions*.java
$ grun Expressions expr -gui
```

25



Instituto Superior de
Engenharia do Porto

Criar aplicação JAVA (Netbeans)

- **1. Visitors**
 - Gerar as classes para os **visitors** na pasta `src\expressions\antlr4` usando o terminal, não esquecendo a indicação do **package** do projeto (pode ser criado um *script* para correr sempre que modifica a gramática)

```
$ antlr4 -o . -package expressions.antlr4 -no-listener -visitor Expressions.g4
```

26

Criar aplicação JAVA (Netbeans)

- Acrescentar/modificar o código da classe **Expressions**

```
package expressions;

import java.io.*;
import org.antlr.v4.runtime.*;
import org.antlr.v4.runtime.tree.*;
import expressions.antlr4.*;

public class Expressions {
    public static void main(String[] args) throws IOException {
        System.out.println("Result with Visitor : ");
        parsewithVisitor();
    }
    public static void parsewithVisitor() throws IOException {
        ExpressionsLexer lexer = new ExpressionsLexer(CharStreams.fromFileName("teste.txt"));
        CommonTokenStream tokens = new CommonTokenStream(lexer);
        ExpressionsParser parser = new ExpressionsParser(tokens);
        ParseTree tree = parser.start(); // parse
        EvalVisitor eval = new EvalVisitor();
        int value = eval.visit(tree);
        System.out.println(value); // print the result
    }
}
```

27

Criar aplicação JAVA (Netbeans)

- Adicionar uma nova classe **EvalVisitor**

```
public class EvalVisitor extends ExpressionsBaseVisitor<Integer> {
    @Override
    public Integer visitstart(ExpressionsParser.StartContext ctx) {
        return visitChildren(ctx);
    }

    @Override
    public Integer visitAtomExpr(ExpressionsParser.AtomExprContext ctx) {
        return Integer.parseInt(ctx.atom.getText());
    } //g4:#atomExpr

    ...continua
}
```

- Estende **ExpressionsBaseVisitor**
uma das classes resultantes do ANTLR4 a partir da gramática

28

Criar aplicação JAVA (Netbeans)

...continuação

```
@Override
public Integer visitOpExprMulDiv(ExpressionsParser.OpExprMulDivContext ctx) {
    int left = visit(ctx.left), right = visit(ctx.right);
    switch (ctx.op.getText().charAt(0)) {
        case '*': return left * right;
        case '/': return left / right;
    }
    return 0;
} //g4:#opExprMulDiv

@Override
public Integer visitOpExprSumDif(ExpressionsParser.OpExprSumDifContext ctx) {
    int left = visit(ctx.left), right = visit(ctx.right);
    switch (ctx.op.getText().charAt(0)) {
        case '+': return left + right;
        case '-': return left - right;
    }
    return 0;
} //g4:#opExprSumDif
```

29

Criar aplicação JAVA (Netbeans)


- Falta criar um ficheiro (*teste.txt*) com uma expressão de teste ...

– Exemplo:

$3+5*2-8*2$

The screenshot shows the NetBeans IDE interface. The **Parse Tree Inspector** window displays a hierarchical tree structure for the expression $3+5*2-8*2$. The root node is **start**, which branches into **expr**, **-**, and **expr**. The first **expr** node further branches into **expr**, **+**, and **expr**. The second **expr** node branches into **expr** and *****. The leaf nodes are the numbers 3, 5, 2, 8, and 2. The **Output - expressions (run)** window shows the execution results: **run:**, **Result with Visitor :**, **-3**, and **BUILD SUCCESSFUL (total time: 0 seconds)**.

30



Instituto Superior de
Engenharia do Porto

Criar aplicação JAVA (Netbeans)

- **2.Listeners**
 - Gerar as classes para os **listeners** na pasta `src\expressions\antlr4` usando o terminal, não esquecendo a indicação do **package** do projeto (pode ser criado um *script* para correr sempre que modifica a gramática)

```
$ antlr4 -o . -package expressions antlr4 -listener -no-visitor Expressions.g4
```

31



Instituto Superior de
Engenharia do Porto

Criar aplicação JAVA (Netbeans)

- Acrescentar/modificar o código da classe **Expressions**

```
package expressions;

import java.io.*;
import org antlr.v4.runtime.*;
import org antlr.v4.runtime.tree.*;
import expressions antlr4.*;

public class Expressions {
    public static void main(String[] args) throws IOException {
        System.out.println("Result with Visitor : ");
        parseWithVisitor(); //versão anterior, para comparar

        System.out.println("Result with Listener : ");
        parseWithListener();
    }
    ...continua
}
```

32

Criar aplicação JAVA (Netbeans)

...continuação

```
public static void parseWithListener() throws IOException {
    ExpressionsLexer lexer = new ExpressionsLexer(CharStreams.fromFileName("teste.txt"));
    CommonTokenStream tokens = new CommonTokenStream(lexer);
    ExpressionsParser parser = new ExpressionsParser(tokens);
    ParseTree tree = parser.start(); // parse
    ParseTreeWalker walker = new ParseTreeWalker();
    EvalListener eListener = new EvalListener();
    walker.walk(eListener, tree);
    System.out.println(eListener.getResult()); // print the result
}
```

33

Criar aplicação JAVA (Netbeans)

- Adicionar uma nova classe **EvalListener**

```
public class EvalListener extends ExpressionsBaseListener {

    private final Stack<Integer> stack = new Stack<>();

    public int getResult() {
        return stack.peek();
    }

    @Override
    public void enterAtomExpr(ExpressionsParser.AtomExprContext ctx) {
        stack.push(Integer.valueOf(ctx.atom.getText()));
    } //g4:#atomExpr
}
```

...continuação

- Estende *ExpressionsBaseVisitor*
uma das classes resultantes do ANTLR4 a partir da gramática
- **Stack** dá suporte aos cálculos intermédios

34

Criar aplicação JAVA (Netbeans)

...continuação

```
@Override
public void exitOpExprMulDiv(ExpressionsParser.OpExprMulDivContext ctx) {
    int right = stack.pop();    int left = stack.pop();    int result;
    if (ctx.op.getText().charAt(0)=='*') {
        result = left * right;
    } else {
        result = left / right;
    }
    stack.push(result);
} //g4:#OpExprMulDiv

@Override
public void exitOpExprSumDif(ExpressionsParser.OpExprSumDifContext ctx) {
    int right = stack.pop();    int left = stack.pop();    int result;
    if (ctx.op.getText().charAt(0)=='+' || ctx.op.getText().charAt(0)=='-') {
        result = left + right;
    } else {
        result = left - right;
    }
    stack.push(result);
} //g4:#OpExprSumDif
```

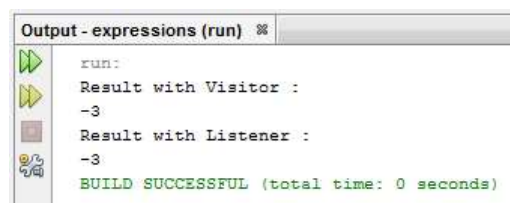
35

Criar aplicação JAVA (Netbeans)

- Falta criar um ficheiro (`teste.txt`) com uma expressão de teste ...

– Exemplo :

$3+5*2-8*2$



36

Fazer algo com base na entrada

Opções para implementação de ações em gramáticas com ANTLR4

1. Listeners ou Visitors (classes)
2. **Ações Semânticas na gramática**

37

Gramática com Ações Semânticas

- Faz as ações na gramática
- Só no final é que apresenta os resultados
(*quando completa a árvore de parse*)
- Para análise linha a linha, é necessário reescrever a gramática e usar uma classe que invoque o *parser* para cada linha

38

Gramática com Ações Semânticas

```
// ActionExpr.g4
grammar ActionExpr;      // nome da gramática

@parser::header {        // classes a incluir
    import java.util.*;
    import java.lang.*;
    import java.io.*;
}

@parser::members { // HashMap para guardar valores das variáveis
    Map<String, Integer> memory = new HashMap<String, Integer>();
}
```

39

Gramática com Ações Semânticas

```
/** Regra inicial; começa o parsing aqui. */
1prog: stat+ ;

    // só uma expressão, apresenta resultado
stat: expr NEWLINE          {System.out.println($expr.v);}

    // atribuição, guarda na tabela de hash o resultado
| ID '=' expr NEWLINE {
    String id = $ID.getText();
    memory.put(id, $expr.v);
    System.out.println(id+": "+$expr.v);
}

    // linha em branco
| NEWLINE

;
```

40

Gramática com Ações Semânticas

```

expr returns [int v]           // v serve para guardar resultado
: a=expr op=('*' | '/') b=expr {
    if ( $op.getType()==MUL )
        $v = $a.v * $b.v;
    else
        $v = $a.v / $b.v;
    }
| a=expr op=('+' | '-') b=expr {
    if ( $op.getType()==ADD )
        $v = $a.v + $b.v;
    else
        $v = $a.v - $b.v;
    }
| INT { $v = Integer.valueOf($INT.getText()); }
| ID { String id = $ID.getText();
      if ( memory.containsKey(id) )
          $v = memory.get(id);
      }
| '(' e=expr ')' { $v = $e.v; }
;

```

41

Gramática (Tokens)

```

MUL : '*' ;           // atribui nomes aos operadores para usar na
DIV : '/' ;           // gramática em cima, e identificar a operação
ADD : '+' ;
SUB : '-' ;


ID : [a-zA-Z]+ ;      // reconhece identificadores
INT : [0-9]+ ;         // reconhece inteiros

NEWLINE: '\r'? '\n' ; // devolve o fim de linha para
                     // terminar a expressão

WS : [ \t]+ -> skip ; // ignora os espaços e tabs

```

42



ISEP
Instituto Superior de
Engenharia do Porto

Testar Exemplo

```

// ActionExpr.g4
grammar ActionExpr; // nome da gramática

@parser::header {
    import java.util.*;
    import java.lang.*;
    import java.io.*;
}
...
WS : [ \t ]+ -> skip ; // toss out whitespace

```


- **Gerar Classes ANTLR4 e compilar**

```

$ antlr4 ActionExpr.g4
$ javac ActionExpr*.java

```

43



ISEP
Instituto Superior de
Engenharia do Porto

Testar Exemplo

- **teste.txt**

```

var=2+5
teste=2*3
teste+var
nova=(teste+var)*2-5*3
nova
(2*20+1)*2+15

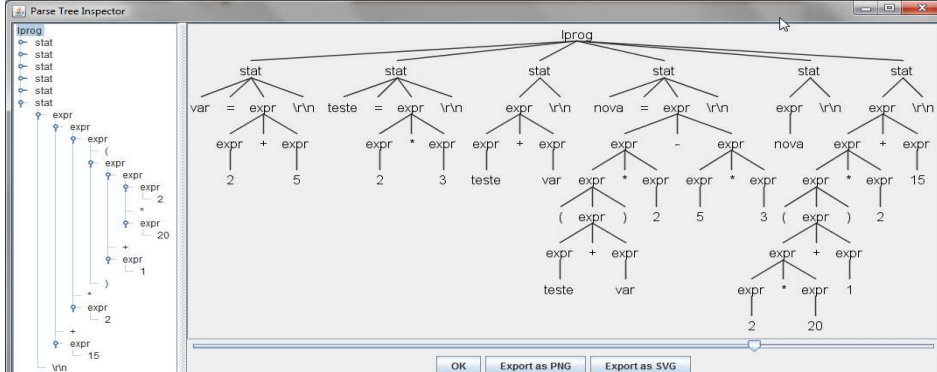
```

- **Correr exemplo**

```

$ grun ActionExpr lprog -gui teste.txt
var:7
teste:6
13
nova:11
11
97

```



44

Exercício

- Dado o exemplo anterior de gramática com ações semânticas, faça o seguinte:
 - 1) Apague o código relativo às ações semânticas;
 - 2) Usando a gramática, implemente uma aplicação que faça o mesmo (avaliar expressões aritméticas com variáveis) usando *Visitors*;
 - 3) Repita agora o ponto anterior usando *Listeners*.

45

Recursos

- Terence Parr speaking about ANTLR4
 - <http://vimeo.com/m/59285751>
- ANTLR4-Website
 - <http://www.antlr.org>
- Book
 - <http://pragprog.com/book/tpantlr2/the-definitive-antlr-4-reference>
- Based on O.Zeigermann's GitHub
 - <https://github.com/DJCordhose/antlr4-sandbox>
- Repository of grammars written for ANTLR v4
 - <https://github.com/antlr/grammars-v4>

46