Implementación de Gramáticas con ANTLR v4

Castañeda Hernández Lina Maria

Joaquín Fernando Sánchez Cifuentes

Lenguajes de programación y Transducción

2026

1. Introducción

En la presente práctica se desarrollaron dos implementaciones utilizando ANTLR v4:

1. Una gramática básica denominada *Hello.g4*, cuyo objetivo fue comprender el funcionamiento fundamental del análisis léxico y sintáctico.
2. La implementación de una calculadora aritmética presentada en el Capítulo 4 del libro *The Definitive ANTLR 4 Reference*, en la cual se incorporó evaluación semántica mediante el patrón Visitor.

   Link Github: https://github.com/linacastaneda/Calculadora_ANTLRv4

2. Implementación Inicial: Hello.g4

2.1 Descripción

El archivo *Hello.g4* constituye una gramática mínima cuyo propósito esreconocer expresiones de la forma:

hello Nombre

Este ejemplo permitió comprender la estructura básica de una gramática ANTLR y el proceso de generación del analizador sin incorporar aún lógica semántica.

2.2 Gramática Hello.g4

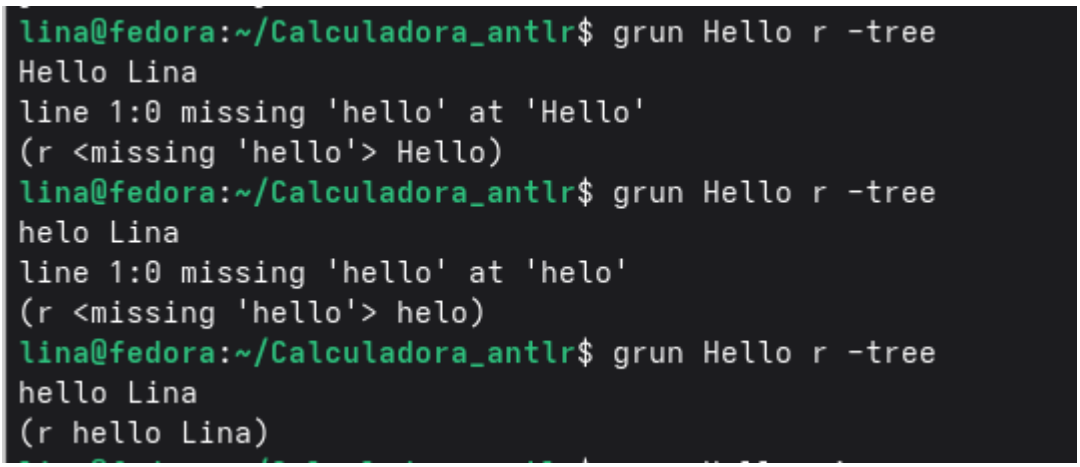
Explicación

- grammar Hello; define el nombre de la gramática.
- r es la regla principal.
- 'hello' es un literal reconocido directamente.
- ID define un identificador compuesto únicamente por letras.

- WS ignora espacios en blanco mediante la acción -> skip.

2.3 Generación y prueba

Comandos ejecutados:



Este resultado confirma que el parser reconoce correctamente la estructura definida por la gramática y construye el árbol sintáctico correspondiente.

2.4 Análisis

El ejemplo Hello.g4 permitió comprender:

- La diferencia entre reglas de parser y reglas léxicas.
- La generación automática del lexer y parser.
- La construcción de árboles sintácticos.
- El funcionamiento del TestRig (grun).

3. Implementación Calculadora con ANRLR

3.1 Descripción General

La segunda fase consistió en implementar una calculadora aritmética con soporte para:

- Operaciones básicas (+, -, *, /)
- Variables
- Paréntesis
- Precedencia de operadores

A diferencia del ejemplo anterior, esta implementación no solo reconoce estructura, sino que también interpreta y evalúa expresiones.

Archivos escritos manualmente

LabeledExpr.g4 → Gramática del lenguaje
Calc.java → Programa principal
EvalVisitor.java → Implementación semántica
t.expr → Archivo de entrada con expresiones

3.2 Archivos generados automáticamente por ANTLR

- LabeledExprLexer.java
- LabeledExprParser.java
- LabeledExprVisitor.java
- LabeledExprBaseVisitor.java
- .tokens
- .interp

Estos archivos son generados automáticamente por ANTLR a partir de la gramática. En ellos se implementa la infraestructura necesaria para el análisis léxico y sintáctico del lenguaje definido.

3.3 Archivos compilados

- .class

Generados tras la compilación mediante el compilador de Java.

Estos corresponden a la versión ejecutable del programa.

4. Gramática (LabeledExpr.g4)

```
grammar LabeledExpr; // rename to distinguish from Expr.g4

prog:   stat+ ;

stat:   expr NEWLINE                # printExpr
    |   ID '=' expr NEWLINE         # assign
    |   NEWLINE                     # blank
    ;

expr:   expr op=('*'|'/') expr      # MulDiv
    |   expr op=('+'|'-') expr      # AddSub
    |   INT                         # int
    |   ID                          # id
    |   '(' expr ')'                # parens
    ;

MUL :   '*' ; // assigns token name to '*' used above in grammar
DIV :   '/' ;
ADD :   '+' ;
SUB :   '-' ;
ID  :   [a-zA-Z]+ ;      // match identifiers
INT :   [0-9]+ ;         // match integers
NEWLINE:'\r'? '\n' ;     // return newlines to parser (is end-statement signal)
WS  :   [ \t]+ -> skip ; // toss out whitespace
```

4.1 Regla principal

prog: stat+ ;

La regla prog indica que un programa está compuesto por una o más sentencias. Esto permite evaluar múltiples expresiones dentro de un mismo archivo de entrada.

4.2 Sentencias

stat:
    expr NEWLINE   # printExpr
  | ID '=' expr NEWLINE # assign
  | NEWLINE   # blank
;

Tipos de sentencias:

- Imprimir expresión
- Asignación de variable
- Línea en blanco

Las etiquetas (#printExpr, #assign, #blank) permiten que ANTLR genere métodos diferenciados en la interfaz Visitor para cada alternativa, facilitando así la implementación semántica posterior.

4.3 Expresiones

```
expr:
    expr op=('*'|'/') expr # MulDiv
  | expr op=('+'|'-') expr # AddSub
  | INT                  # int
  | ID                   # id
  | '(' expr ')'         # parens
;
```

Se implementa:

- Multiplicación y división
- Suma y resta
- Números enteros
- Identificadores
- Paréntesis

ANTLR v4 maneja automáticamente la precedencia y asociatividad mediante la transformación interna.

5. Generación del Parser

Se utilizó el siguiente comando:

antlr4 -no-listener -visitor LabeledExpr.g4

```
lina@fedora:~/Calculadora_antlr$ antlr4 -no-listener -visitor LabeledExpr.g
lina@fedora:~/Calculadora_antlr$ ls LabeledExpr*.java
LabeledExprBaseVisitor.java  LabeledExprParser.java
LabeledExprLexer.java        LabeledExprVisitor.java
```

- -visitor genera la interfaz Visitor.
- -no-listener evita generar listener.

Esto produce las clases necesarias para el lexer y el parser.

6. Programa Principal (Calc.java)

```java
/* Visit http://www.pragmaticprogrammer.com/titles/tpantlr2 for more book information.
***/
import org.antlr.v4.runtime.*;
import org.antlr.v4.runtime.tree.ParseTree;

import java.io.FileInputStream;
import java.io.InputStream;

public class Calc {
    public static void main(String[] args) throws Exception {
        String inputFile = null;
        if ( args.length>0 ) inputFile = args[0];
        InputStream is = System.in;
        if ( inputFile!=null ) is = new FileInputStream(inputFile);
        ANTLRInputStream input = new ANTLRInputStream(is);
        LabeledExprLexer lexer = new LabeledExprLexer(input);
        CommonTokenStream tokens = new CommonTokenStream(lexer);
        LabeledExprParser parser = new LabeledExprParser(tokens);
        ParseTree tree = parser.prog(); // parse

        EvalVisitor eval = new EvalVisitor();
        eval.visit(tree);
    }
}
```

El flujo principal es:

ANTLRInputStream input = new ANTLRInputStream(is);
LabeledExprLexer lexer = new LabeledExprLexer(input);
CommonTokenStream tokens = new CommonTokenStream(lexer);
LabeledExprParser parser = new LabeledExprParser(tokens);
ParseTree tree = parser.prog();

EvalVisitor eval = new EvalVisitor();
eval.visit(tree);

Flujo de ejecución:

Archivo → Lexer → TokenStream → Parser → ParseTree → Visitor → Resultado

El uso de generics (<Integer>) indica que cada método de visita retorna un valor entero.

La implementación incluye una estructura de almacenamiento:

Map<String, Integer> memory = new HashMap<>();

Esta estructura permite asociar identificadores con valores, habilitando el manejo de variables dentro del lenguaje.

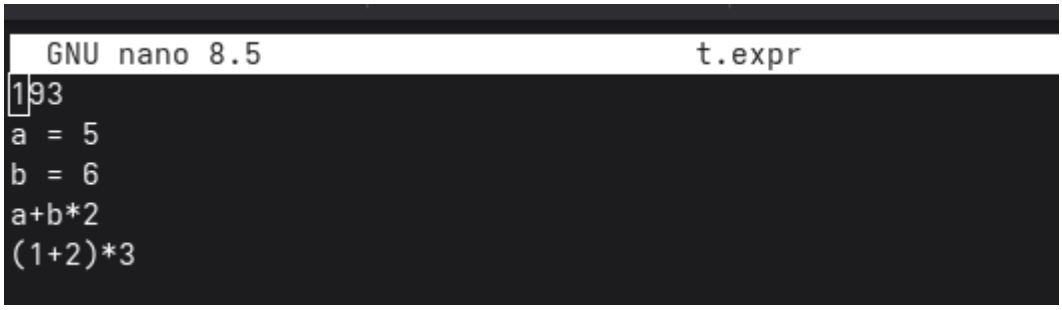
7. Implementación Semántica (EvalVisitor.java)

La clase:

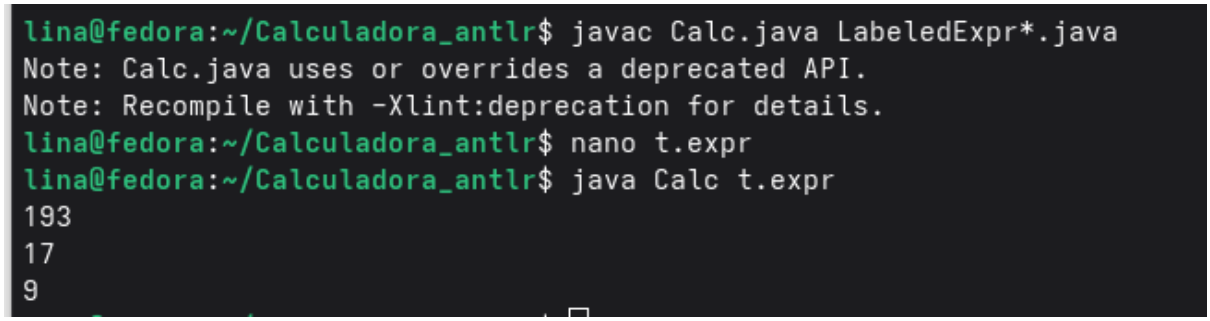
public class EvalVisitor extends LabeledExprBaseVisitor<Integer>
Utiliza generics (<Integer>) para indicar que cada visita devuelve un entero.

8. Prueba de Funcionamiento

Archivo t.expr:



193
a = 5
b = 6
a+b*2
(1+2)*3



Comandos ejecutados:

javac Calc.java EvalVisitor.java LabeledExpr*.java
java Calc t.expr

Salida obtenida:

193
17
9

9. Análisis del Resultado

*9.1 Expresión: a+b*2*

1. b*2 = 6*2 = 12
2. a + 12 = 5 + 12 = 17

Se respeta la precedencia de operadores.

*9.2 Expresión: (1+2)*3*

1. (1+2) = 3
2. 3*3 = 9

Los paréntesis alteran la precedencia correctamente.

10. Conclusiones

- Se logró implementar un lenguaje aritmético básico con variables.
- La gramática permanece independiente del código Java.
- La semántica fue implementada usando el patrón Visitor.
- ANTLR genera automáticamente la infraestructura del parser.
- La precedencia se maneja sin acciones embebidas en la gramática.