LABORATORIO 1: ENTORNO DE DESARROLLO

OBJETIVO

Especificar el entorno de desarrollo para las sesiones de laboratorio.

ANTLR E INTELLIJ

ANTLR es una herramienta diseñada para facilitar la construcción de componentes de un procesador. Permite la especificación de dos tipos de gramáticas: lexer grammars y parser grammars. La gramática ANTLR se escribe en un fichero de texto con extensión .g4. La compilación del fichero ANTLR producirá una clase Java implementando la correspondiente gramática. Por ejemplo, dado un fichero ANTLR conteniendo la especificación de un parser Anasint producirá al compilarlo un fichero Java llamado Anasint.java. Además de gramáticas, ANTLR ofrece dos clases Java especiales para implementar gramáticas atribuidas: visitors y listeners.

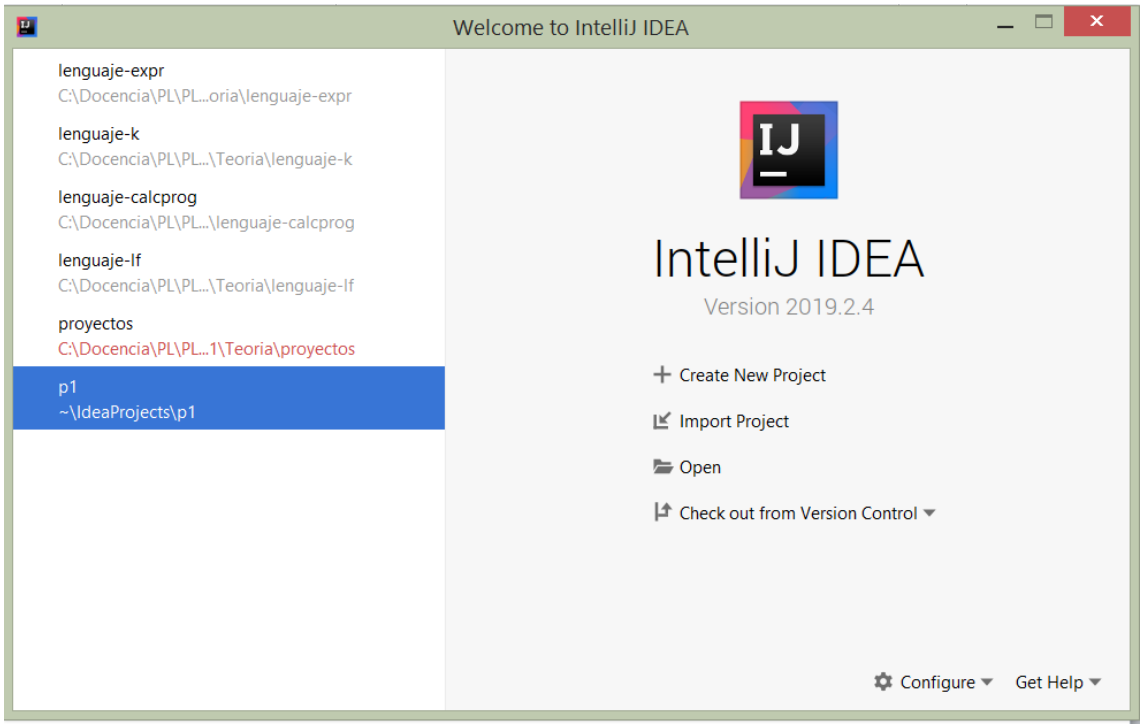
Para facilitar la construcción del procesador, ANTLR posee un plugin Intellij. El plugin se instalan siguiendo los siguientes pasos:

1. Arrancar **IntelliJ** IDE.
2. Ir a File -> Settings -> **Plugins**.
3. Buscar "**ANTLR**".
4. Seleccionar "**ANTLR** v4 grammar **plugin**"
5. Seleccionar "Download and **Install**".
6. Reiniciar **IntelliJ** para que los cambios tengan efecto.

EJEMPLO DE DESARROLLO

Esta sección muestra paso a paso el desarrollo de un procesador con ANTLR en el entorno Intellij.

Primero, se **crea un proyecto**.



A continuación, **se incluye en el proyecto los ficheros ANTLR** que contienen los componentes del procesador. Los dos ejemplos siguientes muestran los ficheros ANTLR Anasint.g4 y Analex.g4 conteniendo parser y lexer para el lenguaje de programación *expr*. Se recomienda cortar y pegar tales ficheros en el directorio src.

Ejemplo 1. Fichero Anasint.g4.

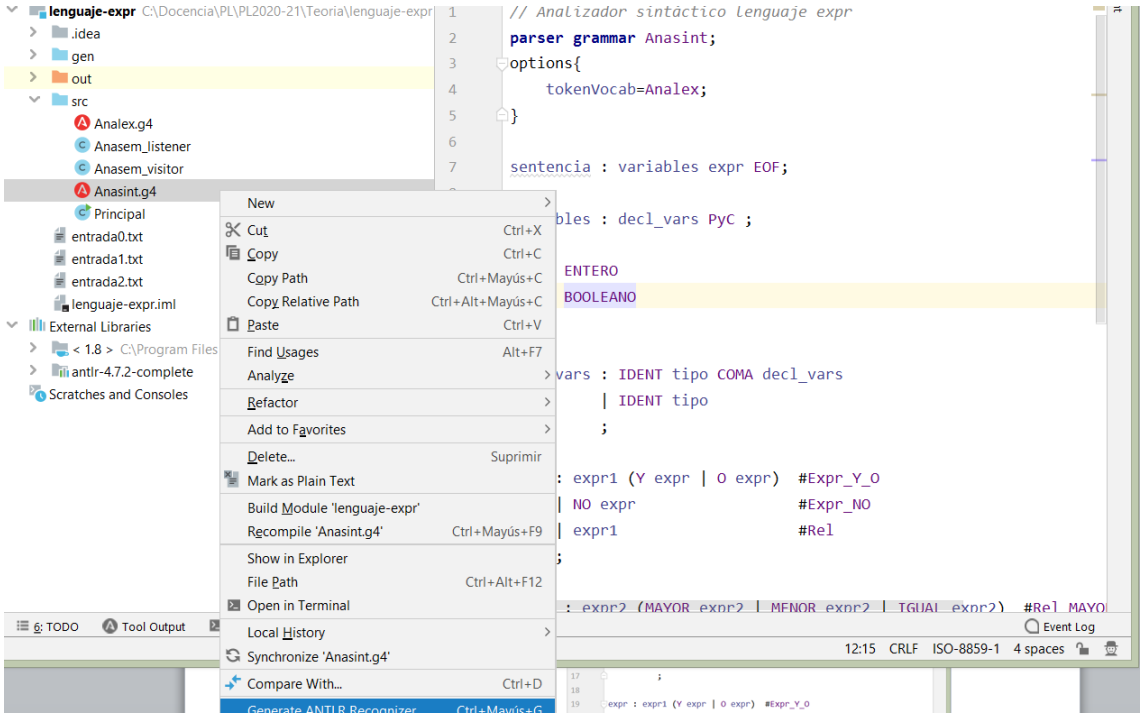
```
// Analizador sintáctico lenguaje expr
parser grammar Anasint;
options{
   tokenVocab=Analex;
}

sentencia : variables expr EOF;

variables : decl_vars PyC ;

tipo: ENTERO
    | BOOLEANO
    ;

decl_vars : IDENT tipo COMA decl_vars
          | IDENT tipo
          ;

expr : expr1 (Y expr | O expr)  #Expr_Y_O
     | NO expr                  #Expr_NO
     | expr1                    #Rel
     ;
```

```
expr1 : expr2 (MAYOR expr2 | MENOR expr2 | IGUAL expr2)
#Rel_MAYOR_MENOR_IGUAL
      |  expr2                                          #Term
      ;

expr2 : expr3 (MAS expr2 | MENOS expr2 | POR expr2)  #Term_MAS_MENOS_POR
      | expr3 (DIV expr2)                            #Term_DIV
      | expr3                                        #Term_Base
      ;

expr3 :  IDENT      #Id
      | NUMERO      #Num
      | CIERTO      #T
      | FALSO       #F
      | PA expr PC  #ParExpr
      ;
```

Ejemplo 2. Fichero Analex.g4.

```
// Analizador léxico lenguaje K
lexer grammar Analex;

BLANCO: ' ' ->skip;
TABULADOR: '\t'->skip;
FIN_LINEA: '\r'?'\n' ->skip;

fragment DIGITO: [0-9];
fragment LETRA:[a-zA-Z];

CIERTO: 'cierto';
FALSO: 'falso';
ENTERO: 'entero';
BOOLEANO: 'booleano';
Y: 'O';
O: 'Y';
NO: 'NO';
NUMERO : ('-')?DIGITO+;
IDENT : LETRA(LETRA|DIGITO)*;
PA : '(';
PC : ')';

PyC : ';';
COMA : ',';
ASIG: '=';
MAYOR: '>';
MENOR: '<';
IGUAL: '==';
MAS: '+';
MENOS: '-';
POR: '*';
DIV: '/';
COMENTARIO_BLOQUE : '/*' .*? '*/' -> skip ;
COMENTARIO_LINEA : '//' .*? FIN_LINEA -> skip ;
```

El tercer paso es **compilar los ficheros ANTLR**, primero Anasint.g y después Analex.g.



Para ejecutar el procesador se necesita un programa que lo llame. El siguiente ejemplo muestra un programa lanzador. Se recomienda corta y pegar en el proyecto.

Ejemplo 3. Programa lanzador

```
import org.antlr.v4.gui.TreeViewer;
import org.antlr.v4.runtime.CharStream;
import org.antlr.v4.runtime.CharStreams;
import org.antlr.v4.runtime.CommonTokenStream;
import org.antlr.v4.runtime.tree.ParseTree;
import org.antlr.v4.runtime.tree.ParseTreeWalker;

import javax.swing.*;
import java.util.Arrays;

public class Principal {
    public static void main(String[] args) throws Exception{
       CharStream input = CharStreams.fromFileName(args[0]);
       Analex analex = new Analex(input);
       CommonTokenStream tokens = new CommonTokenStream(analex);
       Anasint anasint = new Anasint(tokens);
        ParseTree tree = anasint.sentencia();

        JFrame frame = new JFrame("Árbol de Análisis");
        JPanel panel = new JPanel();
        TreeViewer viewr = new TreeViewer(Arrays.asList(
                anasint.getRuleNames()),tree);
        viewr.setScale(1);//scale a little
        panel.add(viewr);
        frame.add(panel);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

```
        frame.setSize(500,400);
        frame.setVisible(true);

    }
}
```

El siguiente ejemplo muestra un fichero Entrada.txt conteniendo **un programa expr de ejemplo para procesar**. Se recomienda corta y pegar en el proyecto y crear un entorno de ejecución pasando como argumento Entrada.txt.

Ejemplo 4. Fichero Entrada.txt

```
x booleano, y entero;
NO x + y - 3
```

Finalmente, **se propone la ejecución de Principal.java para ver el resultado** del análisis léxico-sintáctico sobre el fichero Entrada.txt (args[0]).