# Manual Técnico

## Introducción:

La aplicación REGEXIVE, el cual se construyó con un analizador Léxico, y un analizador sintáctico. Nos permite la lectura de gramáticas regulares y el mismo genera varios métodos, con el cual se evalúa la gramática. Entre ellos el método del árbol y el Método por Thompson.

Datos del sistema donde se realizó:

- Windows 10 Home
- Intel(R) Core(TM) i7-8565U CPU @ 180GHz 1.99 GHz
- 12 GB de memoria RAM.
- Sistema operativo de 64 bits.
- java version "1.8.0_281"
- Netbeans IDE 8.1
- JFlex 1.6.1
- JCup 11b
- Gson 2.8.2

## Diccionario de Clases;

- A_Lexico.jflex: Dentro de esta clase que se utiliza jflex, se realiza toda la construcción léxica con la cual nuestro programa va a poder aceptar los caracteres.

```
//-----> Simbolos

"."         { System.out.println("Reconocio "+yytext()+" conca"); return new Symbol(Simbolos.conca, yycolumn, yyline, yytext()); }
"|"         { System.out.println("Reconocio "+yytext()+" or"); return new Symbol(Simbolos.or, yycolumn, yyline, yytext()); }
"*"         { System.out.println("Reconocio "+yytext()+" cierre"); return new Symbol(Simbolos.cierre, yycolumn, yyline, yytext()); }
"+"         { System.out.println("Reconocio "+yytext()+" mas"); return new Symbol(Simbolos.mas, yycolumn, yyline, yytext()); }
"{"         { System.out.println("Reconocio "+yytext()+" corchea"); return new Symbol(Simbolos.corchea, yycolumn, yyline, yytext()); }
"}"         { System.out.println("Reconocio "+yytext()+" corchec"); return new Symbol(Simbolos.corchec, yycolumn, yyline, yytext()); }
"-"         { System.out.println("Reconocio "+yytext()+" guion"); return new Symbol(Simbolos.guion, yycolumn, yyline, yytext()); }
">"         { System.out.println("Reconocio "+yytext()+" mayork"); return new Symbol(Simbolos.mayork, yycolumn, yyline, yytext()); }
"<"         { System.out.println("Reconocio "+yytext()+" menork"); return new Symbol(Simbolos.menork, yycolumn, yyline, yytext()); }
"%"         { System.out.println("Reconocio "+yytext()+" prct"); return new Symbol(Simbolos.prct, yycolumn, yyline, yytext()); }
":"         { System.out.println("Reconocio "+yytext()+" dspts"); return new Symbol(Simbolos.dspts, yycolumn, yyline, yytext()); }
";"         { System.out.println("Reconocio "+yytext()+" ptcoma"); return new Symbol(Simbolos.ptcoma, yycolumn, yyline, yytext()); }
"~"         { System.out.println("Reconocio "+yytext()+" curv"); return new Symbol(Simbolos.curv, yycolumn, yyline, yytext()); }
"?"         { System.out.println("Reconocio "+yytext()+" inte"); return new Symbol(Simbolos.inte, yycolumn, yyline, yytext()); }
"\\n"       { System.out.println("Reconocio "+yytext()+" sltln"); return new Symbol(Simbolos.sltln, yycolumn, yyline, yytext()); }
"\\'"       { System.out.println("Reconocio "+yytext()+" cmllsmpl"); return new Symbol(Simbolos.cmllsmpl, yycolumn, yyline, yytext()); }
"\\\""      { System.out.println("Reconocio "+yytext()+" cmlldbl"); return new Symbol(Simbolos.cmlldbl, yycolumn, yyline, yytext()); }
","         { System.out.println("Reconocio "+yytext()+" coma"); return new Symbol(Simbolos.coma, yycolumn, yyline, yytext()); }

//-----> Palabras reservadas

"CONJ"          { System.out.println("Reconocio "+yytext()+" conj"); return new Symbol(Simbolos.conj, yycolumn, yyline, yytext()); }

//-------> Simbolos ER
{digito}    { System.out.println("Reconocio "+yytext()+" digito"); return new Symbol(Simbolos.digito, yycolumn, yyline, yytext()); }
{letra}     { System.out.println("Reconocio "+yytext()+" letra"); return new Symbol(Simbolos.letra, yycolumn, yyline, yytext()); }
{esp}     { System.out.println("Reconocio "+yytext()+" esp"); return new Symbol(Simbolos.esp, yycolumn, yyline, yytext()); }
{cadena}    { System.out.println("Reconocio "+yytext()+" cadena"); return new Symbol(Simbolos.cadena, yycolumn, yyline, yytext()); }
{id}     { System.out.println("Reconocio "+yytext()+" id"); return new Symbol(Simbolos.id, yycolumn, yyline, yytext()); }
```

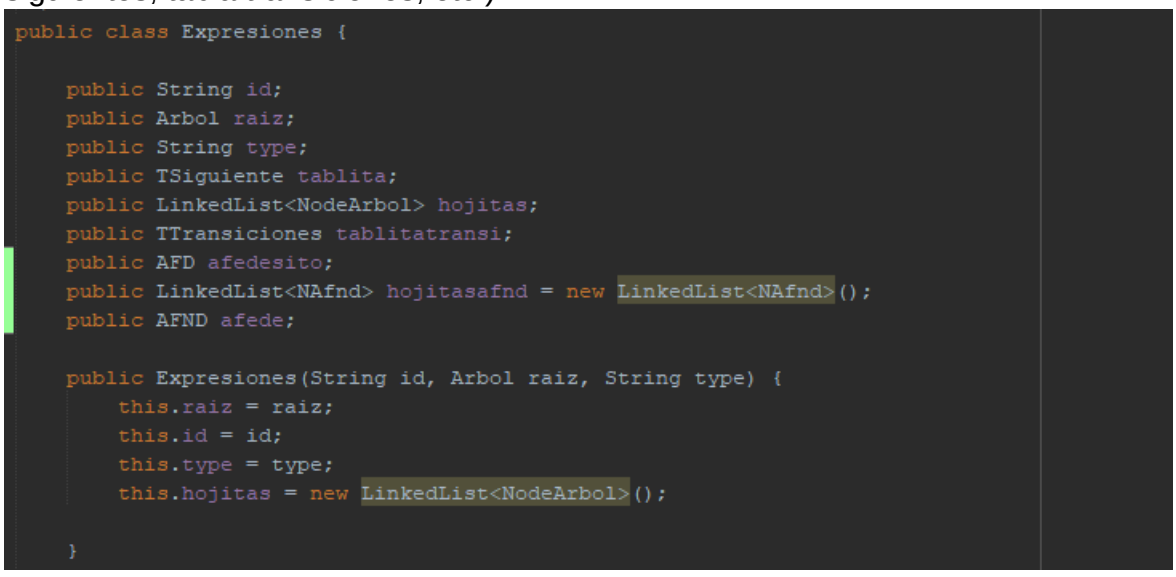
- A_Sintactico.jcup : Dentro de esta clase que se utiliza jcup, se escribran todas las reglas semánticas las cuales nuestro programa podrá aceptar, del mismo también leemos los caracteres necesarios para que luego se puedan construir con lenguaje java los métodos.

```
/*--------------- 3ra Area: Reglas Semanticas ----------------------*/


INICIO::= corchea ENCABEZADO:a prct prct
          prct prct VALIDACIONES:b corchec       {:parser.enca = a; parser.lista_valida = b;:}
          | error
        ;
ENCABEZADO ::= ENCABEZADO:a EXPRESION:b         {: RESULT = a; RESULT.add(b); :}
          | ENCABEZADO:a CONJUNTOS:b            {: RESULT = a; RESULT.add(b);:}
          | EXPRESION:a                     {: RESULT = new LinkedList<>(); RESULT.add(a);:}
          | CONJUNTOS:a                   {: RESULT = new LinkedList<>(); RESULT.add(a);:}
        ;
CONJUNTOS ::= conj dspts id:a guion mayork VALORES:b ptcoma    {: RESULT = new Conjuntos(a.toString(), b,"Conjunto"); :}
        | error
        ;
VALORES ::= VAL:a curv:b VAL:c        {:RESULT = a.toString()+b+c.toString();:}
        | LIST_VAL:a                  {:RESULT = a.toString();:}
        ;
VAL ::= letra:a                 {: RESULT = a.toString();:}
        | digito:a              {: RESULT = a.toString();:}
        | sltln:a               {: RESULT = a.toString();:}
        | cmllsmpl:a            {: RESULT = a.toString();:}
        | cmlldbl:a             {: RESULT = a.toString();:}
        | esp:a                 {: RESULT = a.toString();:}
        | id:a                  {: RESULT = a.toString();:}
        | cadena:a              {: RESULT = a.toString();:}
    ;
LIST_VAL ::= LIST_VAL:a coma:b VAL:c        {:RESULT = a.toString()+b+c.toString();:}
        | VAL:a                     {: RESULT = a.toString();:}
        ;
EXPRESION ::= id:a guion mayork E:b ptcoma     {:RESULT = new Expresiones(a.toString(), new Arbol(b, a.toString(), parser.contHojas),"Expresion"); parser.contHojas = 1; :}
        ;

E ::= conca E:a E:b             {: RESULT = new NodeArbol(".","", -1,a,b,true,null,null,false);   :}
    | or E:a E:b                {: RESULT = new NodeArbol("|","", -1,a,b,true,null,null,false); :}
    | cierre E:a                {: RESULT = new NodeArbol("*","", -1,a,null,true,null,null,false); :}
    | mas E:a                   {: RESULT = new NodeArbol("+","", -1,a,null,false,null,null,false); :}
    | inte E:a                  {: RESULT = new NodeArbol("?","", -1,a,null,true,null,null,false); :}
    | digito:a                  {: RESULT = new NodeArbol(a.toString(), "", parser.contHojas,null,null,false,String.valueOf(parser.contHojas),String.valueOf(parser.contHojas),true); parser.con
    | corchea id:a corchec      {: RESULT = new NodeArbol(a.toString(), "", parser.contHojas,null,null,false,String.valueOf(parser.contHojas),String.valueOf(parser.contHojas),true); parser.con
    | sltln:a                    {: RESULT = new NodeArbol(a.toString(), "", parser.contHojas,null,null,false,String.valueOf(parser.contHojas),String.valueOf(parser.contHojas),true); parser.con
    | cmllsmpl:a                {: RESULT = new NodeArbol(a.toString(), "", parser.contHojas,null,null,false,String.valueOf(parser.contHojas),String.valueOf(parser.contHojas),true); parser.con
    | cmlldbl:a                 {: RESULT = new NodeArbol(a.toString(), "", parser.contHojas,null,null,false,String.valueOf(parser.contHojas),String.valueOf(parser.contHojas),true); parser.con
    | cadena:a                  {: RESULT = new NodeArbol(a.toString(), "", parser.contHojas,null,null,false,String.valueOf(parser.contHojas),String.valueOf(parser.contHojas),true); parser.con
    ;
```

- Expresiones: Esta clase es donde se podrán almacenar las gramáticas regulares que se describan en nuestra área de texto. Dentro de este mismo es donde crearemos todos los métodos que requiera (árbol, tabla de siguientes, tabla transiciones, etc.).



```
public class Expresiones {

    public String id;
    public Arbol raiz;
    public String type;
    public TSiguiente tablita;
    public LinkedList<NodeArbol> hojitas;
    public TTransiciones tablitatransi;
    public AFD afedesito;
    public LinkedList<NAfnd> hojitasafnd = new LinkedList<NAfnd>();
    public AFND afede;

    public Expresiones(String id, Arbol raiz, String type) {
        this.raiz = raiz;
        this.id = id;
        this.type = type;
        this.hojitas = new LinkedList<NodeArbol>();

    }
```

- ErroresS: Dentro de este se almacenan los errores que puedan surgir por medio de nuestro A_Sintactico, un orden no aceptado por ejemplos.

```java
public class ErroresS {
    public String tipo;
    public String descripcion;
    public int fila;
    public int columna;

    public ErroresS(String tipo, String descripcion, int fila, int columna){
        this.tipo = tipo;
        this.descripcion = descripcion;
        this.fila = fila;
        this.columna = columna;
    }
```

- ErroresL: Dentro de este se almacenan los errores que se puedan surgir por medio de A_Lexico, algún símbolo no previamente señalizado.

```java
public class ErroresL {

    String tipo;
    String caracter;
    int fila;
    int columna;

    public ErroresL(String tipo, String caracter,int fila,int columna) {
        this.tipo = tipo;
        this.caracter = caracter;
        this.fila = fila;
        this.columna = columna;
    }
```