Universidad San Carlos de Guatemala

Facultad de Ingeniería

Escuela de Ciencias y Sistemas

Laboratorio de Organización y Compiladores 1, Sección B

CARATULA

Proyecto de expresiones regulares, RegexIve.

Manual Técnico.

PERSONA QUE PRESENTA EL MANUAL:

Nombre:	Angel Oswaldo Arteaga García
Carné:	201901816

DATOS DEL SOLICITANTE:

Catedrático:	Ing. Manuel Castillo
Auxiliar:	Erick Lemus
Auxiliar:	René Corona

Manual Técnico

- Archivo jflex encargado del análisis sintáctico:

Este archivo es el encargado de reconocer cada token que nuestro lenguaje analiza, en este, primero se definió aquellos tokens que necesiten alguna expresión regular para poder identificarlos.

```
//EXPRESIONES REGULARES DE MI LENGUAJE
blancos = [ \t\r\n]+
entero = [0-9]
identificador = [A-Za-z]([A-Za-z]|[0-9]+|"_")*
commen = ("//".*\n)|("//".*\r)
commenM = ("<""!"[^\!]*"!"">")
cadena = ("\""([^\"]*)"\"")
caracterespecial = ("\\""n"|"\\""\'"|"\\""\"")
```

Luego de esto se definieron aquellos tokens que siguen un patrón especifico para poderse reconocer, al mismo tiempo creamos el nuevo símbolo enlazado a este para posteriormente en nuestro análisis sintáctico poder utilizarlo.

```
//MANDAMOS LOS SIMBOLOS A LA TABLA DE SIMBOLOS
//Caracteres Neutros
\n {yycolumn=1;}
{commen} {/*se ignoran*/}
{commenM} {/*se ignoran*/}
{blancos} {/*se ignoran*/}
"{" {return new Symbol(sym.TKParA,yycolumn,yyline,yytext());}
"}" {return new Symbol(sym.TKParC,yycolumn,yyline,yytext());}
"," {return new Symbol(sym.TKComa,yycolumn,yyline,yytext());}
"|" {return new Symbol(sym.TKBarra,yycolumn,yyline,yytext());}
"." {return new Symbol(sym.TKPunto,yycolumn,yyline,yytext());}
"+" {return new Symbol(sym.TKMas,yycolumn,yyline,yytext());}
"*" {return new Symbol(sym.TKAsterisco,yycolumn,yyline,yytext());}
"%" {return new Symbol(sym.TKPorcentaje,yycolumn,yyline,yytext());}
";" {return new Symbol(sym.TKPuntoComa,yycolumn,yyline,yytext());}
"?" {return new Symbol(sym.TKInterrogacion,yycolumn,yyline,yytext());}
"~" {return new Symbol(sym.TKColocho,yycolumn,yyline,yytext());}
":" {return new Symbol(sym.TKDosPuntos,yycolumn,yyline,yytext());}
"-" {return new Symbol(sym.TKGuion,yycolumn,yyline,yytext());}
">" {return new Symbol(sym.TKMayor,yycolumn,yyline,yytext());}
"CONJ" {return new Symbol(sym.TKConj,yycolumn,yyline,yytext());}
{identificador} {return new Symbol(sym.identificador,yycolumn,yyline,yytext());}
{caracterespecial} {return new Symbol(sym.caracterespecial,yycolumn,yyline,yytext());}
{cadena} {return new Symbol(sym.cadena,yycolumn,yyline,yytext());}

//USAREMOS TODOS LOS CODIGO ASCII
{entero} {return new Symbol(sym.entero,yycolumn,yyline,yytext());}
"!" {return new Symbol(sym.C33,yycolumn,yyline,yytext());}
"#" {return new Symbol(sym.C35,yycolumn,yyline,yytext());}
"$" {return new Symbol(sym.C36,yycolumn,yyline,yytext());}
"&" {return new Symbol(sym.C38,yycolumn,yyline,yytext());}
"(" {return new Symbol(sym.C40,yycolumn,yyline,yytext());}
")" {return new Symbol(sym.C41,yycolumn,yyline,yytext());}
"/" {return new Symbol(sym.C47,yycolumn,yyline,yytext());}
"<" {return new Symbol(sym.C60,yycolumn,yyline,yytext());}
"=" {return new Symbol(sym.C61,yycolumn,yyline,yytext());}
"@" {return new Symbol(sym.C64,yycolumn,yyline,yytext());}
"[" {return new Symbol(sym.C91,yycolumn,yyline,yytext());}
"]" {return new Symbol(sym.C93,yycolumn,yyline,yytext());}
"^" {return new Symbol(sym.C94,yycolumn,yyline,yytext());}
"_" {return new Symbol(sym.C95,yycolumn,yyline,yytext());}
"`" {return new Symbol(sym.C96,yycolumn,yyline,yytext());}
```

Por último, almacenamos el símbolo que no fue reconocido si así fuera el caso con un tipo de dato Errores.

```
//CUALQUIER ERROR:           SIMBOLOS NO DEFINIDOS DENTRO DEL LENGUAJE
.   {
        Application.App.TxtSalida.append("Error léxico: "+yytext()+ " Linea:"+(yyline)+" Columna:"+(yycolumn) + "\n");
        Errores err = new Errores("Léxico", "El símbolo \"" + yytext() + "\" no pertenece al lenguaje", (yyline), (yycolumn));
        Application.App.ListaErrores.add(err);
    }
```

- Archivo cup encargado del análisis sintáctico:

Determinación de los símbolos terminales y no terminales con su tipo.

```
497   // terminal [Tipo] listaterminales;
498   terminal String TKParA, TKParC, TKComa, TKBarra, TKPunto, TKMas, TKAsterisco, TKPorcentaje, TKPuntoComa, TKInterrogacion, TKColocho;
499   terminal String TKConj, TKDosPuntos, cadena, caracterespecial, entero, identificador, TKGuion, TKMayor;
500   terminal String C33, C36, C35, C38, C40, C41, C47, C60, C61, C64, C91, C93, C94, C95, C96;
501   // non terminal [Tipo] listanoterminales;
502   non terminal String INICIO, CUERPO, CONJUNTO, EXPRESION, DEFINICION, ASCII, CONJUNTOLARGO;
503   non terminal Nodo ARBOL, HOJAS;
504
```

Empezamos con la producción de la inicial encargada de leer la estructura del archivo, cuando la reconozcamos, solo mostramos el mensaje de éxito.

```
508   INICIO ::= TKParA CUERPO EXPRESION TKPorcentaje TKPorcentaje TKPorcentaje TKPorcentaje DEFINICION TKParC {:
509       App.TxtSalida.append("El archivo se ha leido correctamente.\n");
510   :};
```

Producciones del CUERPO, cuando esta se lea, almacenará los conjuntos que obtuvo en una lista dentro de la aplicación.

```
512   CUERPO ::= TKConj TKDosPuntos identificador:conjunto TKGuion TKMayor CONJUNTO TKPuntoComa{:
513       try{
514               Application.App.ListaConjuntos.add(new Conjunto(new ArrayList<>(parser.Caracteres),conjunto));
515               parser.Caracteres.clear();
516               parser.contCon++;
517           }catch(Exception e){
518               System.err.println("Error de Conjuntos: " + e);
519           }
520   :}
521           | CUERPO TKConj TKDosPuntos identificador:conjunto TKGuion TKMayor CONJUNTO TKPuntoComa{:
522               try{
523                   Application.App.ListaConjuntos.add(new Conjunto(new ArrayList<>(parser.Caracteres),conjunto));
524                   parser.Caracteres.clear();
525                   parser.contCon++;
526               }catch(Exception e){
527                   System.err.println("Error de Conjuntos: " + e);
528               }
529           :};
```

Producciones del CONJUNTO, encargada de obtener los símbolos dentro del rango del conjunto y almacenarlos en una lista

```
CONJUNTO ::= ASCII:a TKColocho ASCII:b{:
    try{
        int inicio;
        int fin;
        if (a == "\\n"){
            inicio = 10;
        } else if (a == "\\'"){
            inicio = 39;
        } else if (a == "\\\""){
            inicio = 34;
        } else {
            inicio = (int)a.charAt(0);
        }
        if (b == "\\n"){
            fin = 10;
        } else if (b == "\\'"){
            fin = 39;
        } else if (b == "\\\""){
            fin = 34;
        } else {
            fin = (int)b.charAt(0);
        }
        for (int i = inicio; i <= fin; i++){
            char temp = (char)i;
            String caracter = String.valueOf(temp);
            parser.Caracteres.add(caracter);
        }
    }catch(Exception e){
        System.err.println("Equis de: " + e);
    }
:}
            | CONJUNTOLARGO{::};
```

Producciones del CONJUNTOLARGO, tiene la misma función de almacenar los símbolos, pero esta vez específicamente los que leemos.

```
CONJUNTOLARGO ::= CONJUNTOLARGO TKComa ASCII:lexema{:
    parser.Caracteres.add(lexema);
:}
            | ASCII:lexema{:
                parser.Caracteres.add(lexema);
            :};
```

La producción ASCII solo deriva en todos los símbolos que podremos leer dentro de la definición de los conjuntos.

Las producciones de EXPRESION son las encargadas de leer la definición de la expresión regular.

```
604 > EXPRESION ::= identificador:nombre TKGuion TKMayor ARBOL:valor TKPuntoComa{: ···
701   :}
702 >             | EXPRESION identificador:nombre TKGuion TKMayor ARBOL:valor TKPuntoComa{: ···
799               :};
```

Cada vez que realiza estas producciones, procede a crear el último nodo hijo de la raíz, así mismo calculando sus primeros, últimos, anulabilidad y siguientes. Luego junta el ultimo hijo con toda la estructura generada anteriormente. Por ultimo, procede a graficar todos las tablas, arboles y afd.

```
604 EXPRESION ::= identificador:nombre TKGuion TKMayor ARBOL:valor TKPuntoComa{:
605                 //PARA EL CALCULO DE PRIMEROS
606                 int[] primeros1 = new int[1];
607                 primeros1[0] = parser.num;
608                 //PARA EL CALCULO DE ULTIMOS
609                 int[] ultimos1 = new int[1];
610                 ultimos1[0] = parser.num;
611                 //CREAMOS EL NODO
612                 Nodo fin = new Nodo(null, null, "#", parser.contId,parser.num, "N", primeros1, ultimos1);
613                 //CALCULO DE SIGUIENTES
614                 TablaSiguientes elemento = new TablaSiguientes(parser.num, null, "#");
615                 if (parser.ListaSiguientes == null){
616                     ListaSiguientes = new ArrayList<TablaSiguientes>();
617                     parser.ListaSiguientes.add(elemento);
618                 } else {
619                     parser.ListaSiguientes.add(elemento);
620                 }
621                 parser.contId++;
622 
623                 //PARA EL CALCULO DE PRIMEROS
624                 int [] primeros;
625                 if (valor.anulable == "A"){
626                     primeros = new int[valor.primeros.length + fin.primeros.length];
627                     int j = 0;
628                     for (int i = 0; i < primeros.length; i++){
629                         if (i < valor.primeros.length){
630                             primeros[i] = valor.primeros[i];
631                         } else {
632                             primeros[i] = fin.primeros[j];
633                             j++;
634                         }
635                     }
636                 } else {
637                     primeros = valor.primeros;
638                 }
639                 //PARA EL CALCULO DE ULTIMOS
640                 int [] ultimos;
641                 if (fin.anulable == "A"){
642                     ultimos = new int[valor.ultimos.length + fin.ultimos.length];
643                     int j = 0;
644                     for (int i = 0; i < ultimos.length; i++){
645                         if (i < valor.ultimos.length){
646                             ultimos[i] = valor.ultimos[i];
647                         } else {
648                             ultimos[i] = fin.ultimos[j];
649                             j++;
```

```
                    }
                }
            } else {
                ultimos = fin.ultimos;
            }
            //PARA EL CALCULO DE ANULABLES
            String anulable;
            if (fin.anulable == "A" && valor.anulable == "A"){
                anulable = "A";
            } else {
                anulable = "N";
            }
            //CREAMOS EL NODO
            Nodo raiz = new Nodo(valor, fin, ".", parser.contId,0, anulable, primeros, ultimos);
            parser.Raiz = raiz;

            //CALCULO DE SIGUIENTES
            int[] ultimosC1 = valor.ultimos;
            int[] primerosC2 = fin.primeros;
            for (int i = 0; i < ultimosC1.length; i++){
                //SABEMOS QUE ES EN LISTASIGUIENTES EN I
                int temp = ultimosC1[i];
                for (int j = 0; j < primerosC2.length; j++){
                    int nuevo = primerosC2[j];
                    if (parser.ListaSiguientes.get(temp-1).siguientes.contains(nuevo) == false){
                        ListaSiguientes.get(temp-1).siguientes.add(nuevo);
                    }
                }
            }

            graficarArbol(raiz, nombre);
            graficarSiguientes(nombre);
            graficarTransiciones(raiz.primeros, nombre);
            graficarAFD(nombre);
            //ALMACENAMOS LAS TRANSICIONES Y LOS CONJUNTOS

            try{
                int EstadoFinal = -10;
                for (int i = 0; i < parser.estados.size(); i++){
                    if (parser.estados.get(i).combinacion.contains(ListaSiguientes.size())){
                        EstadoFinal = parser.estados.get(i).S;
                    }
                }
            Transiciones tempTrans = new Transiciones(parser.Transiciones, nombre, EstadoFinal, new ArrayList<>(parser.conjuntos));
            Application.App.ListaTransiciones.add(tempTrans);
            }catch(Exception e){
                System.err.println("Error de Transiciones: " + e);
            }
```

La producción de ARBOL con respecto a la concatenación se encarga de recibir los nodos hijo, y al mismo tiempo que los lee, genera de una vez los siguientes, primeros, últimos y anulabilidad del padre a crear, y así también con las demás producciones del ARBOL.

```
ARBOL ::= TKPunto ARBOL:a ARBOL:b{:
            //PARA EL CALCULO DE PRIMEROS
            int [] primeros;
            if (a.anulable == "A"){
                primeros = new int[a.primeros.length + b.primeros.length];
                int j = 0;
                for (int i = 0; i < primeros.length; i++){
                    if (i < a.primeros.length){
                        primeros[i] = a.primeros[i];
                    } else {
                        primeros[i] = b.primeros[j];
                        j++;
                    }
                }
            } else {
                primeros = a.primeros;
            }

            //PARA EL CALCULO DE ULTIMOS
            int [] ultimos;
            if (b.anulable == "A"){
                ultimos = new int[a.ultimos.length + b.ultimos.length];
                int j = 0;
                for (int i = 0; i < ultimos.length; i++){
                    if (i < a.ultimos.length){
                        ultimos[i] = a.ultimos[i];
                    } else {
                        ultimos[i] = b.ultimos[j];
                        j++;
                    }
                }
            } else {
                ultimos = b.ultimos;
            }
```

```
836             //PARA EL CALCULO DE ANULABLES
837             String anulable;
838             if (a.anulable == "A" && b.anulable == "A"){
839                 anulable = "A";
840             } else {
841                 anulable = "N";
842             }
843             //CREAMOS EL NODO
844             Nodo NuevoPadre = new Nodo(a,b,".",parser.contId,0, anulable, primeros, ultimos);
845             //CALCULO DE SIGUIENTES
846             int[] ultimosC1 = a.ultimos;
847             int[] primerosC2 = b.primeros;
848             for (int i = 0; i < ultimosC1.length; i++){
849                 //SABEMOS QUE ES EN LISTASIGUIENTES EN I
850                 int temp = ultimosC1[i];
851                 for (int j = 0; j < primerosC2.length; j++){
852                     int nuevo = primerosC2[j];
853                     if (parser.ListaSiguientes.get(temp-1).siguientes.contains(nuevo) == false){
854                         ListaSiguientes.get(temp-1).siguientes.add(nuevo);
855                     }
856                 }
857             }
858             //ITERAMOS
859             parser.contId++;
860             RESULT = NuevoPadre;
861     :}
```

Así sucesivamente con todas las producciones del ARBOL, pero tomando en cuenta las reglas variantes para cada situación.

Para la producción de HOJAS solo cree los nodos sin hijos y también le asigné sus primeros, últimos, siguientes y anulabilidad como lo indican las reglas.

```
966 > HOJAS ::= TKParA:val0 identificador:val TKParC:val1{: ···
987   :}
988 >           | caracterespecial:val{: ···
1015            :}
1016 >          | cadena:val{: ···
1043            :};
```

```
966   HOJAS ::= TKParA:val0 identificador:val TKParC:val1{:
967               //PARA EL CALCULO DE PRIMEROS
968               int[] primeros = new int[1];
969               primeros[0] = parser.num;
970               //PARA EL CALCULO DE PRIMEROS
971               int[] ultimos = new int[1];
972               ultimos[0] = parser.num;
973               //CREAMOS EL NODO
974               Nodo NuevaHoja = new Nodo(null,null,val0+val+val1,parser.contId,parser.num, "N", primeros, ultimos);
975               //CALCULO DE SIGUIENTES
976               TablaSiguientes elemento = new TablaSiguientes(parser.num, null, val);
977               if (parser.ListaSiguientes == null){
978                   ListaSiguientes = new ArrayList<TablaSiguientes>();
979                   parser.ListaSiguientes.add(elemento);
980               } else {
981                   parser.ListaSiguientes.add(elemento);
982               }
983               //ITERAMOS
984               parser.contId++;
985               parser.num++;
986               RESULT = NuevaHoja;
987   :}
```

Para la producción de DEFINICION solo leo recursivamente los lexemas que pudieran venir en el archivo y los almaceno a en una lista creada en la aplicación.

```
1045  DEFINICION ::= identificador:nombre TKDosPuntos cadena:lexema TKPuntoComa{:
1046      try{
1047          Application.App.ListaCadenas.add(new Cadena(lexema, nombre));
1048      }catch(Exception e){
1049          System.err.println("Error de cadenas: " + e);
1050      }
1051  :}
1052              | DEFINICION identificador:nombre TKDosPuntos cadena:lexema TKPuntoComa{:
1053                  try{
1054                      Application.App.ListaCadenas.add(new Cadena(lexema, nombre));
1055                  }catch(Exception e){
1056                      System.err.println("Error de cadenas: " + e);
1057                  }
1058              :};
```

Las variables globales del archivo Cup son las siguientes:

```
parser code
{:
    public static int contId=1;
    public static Nodo Raiz;
    public static int num = 1;
    public static int contCon = 0;
    public static int contEs = 0;
    public static List<TablaSiguientes> ListaSiguientes;
    public static List<Estado> estados;
    public static List<Integer> ListaAuxiliar;
    public static int[][] Transiciones;
    public static List<String> conjuntos;
    public static List<String> Caracteres = new ArrayList<String>();
```

El método para GraficarErrores() se encarga de crear la lista HTML en caso de haber errores.

```
public static void GraficarErrores(){
    FileWriter fichero = null;
    PrintWriter pw = null;
    try {
        fichero = new FileWriter("Errores.html");
        pw = new PrintWriter(fichero);
        //EMPEZAMOS EL DOCUMENTO HTML
        pw.println("<!doctype html>");
        pw.println("<head>");
        pw.println("<meta charset=\"utf-8\">");
        pw.println("<meta name=\"author\" content=\"Angel Arteaga\">");
        pw.println("<meta name=\"descripción\" content=\"Errores de entrada\">");
        pw.println("<title>Errores De Entrada</title>");
        pw.println("</head>");
        pw.println("<style>");
        pw.println("table.GeneratedTable {width: 100%;background-color: #ffffff;border-collapse: collapse;border-width: 2px;border-color: #a2f575;border-style:
        pw.println("table.GeneratedTable td, table.GeneratedTable th {border-width: 2px;border-color: #a2f575;border-style: solid;padding: 3px;}");
        pw.println("table.GeneratedTable thead {background-color: #bcf98b;}");
        pw.println("</style>");
        pw.println("<body>");
        pw.println("<table class=\"GeneratedTable\">");
        pw.println("<thead><tr><th>Número #</th><th>Tipo de Error</th><th>Descripción</th><th>Línea</th><th>Columna</th></tr></thead>");
        pw.println("<tbody>");
        //AHORA LA PARTE DINAMICA
        for (int i = 0; i < Application.App.ListaErrores.size(); i++){
            pw.println("<tr>");
            pw.println("<td>" + i +"</td>");
            pw.println("<td>" + Application.App.ListaErrores.get(i).Tipo +"</td>");
            pw.println("<td>" + Application.App.ListaErrores.get(i).Descripcion +"</td>");
            pw.println("<td>" + Application.App.ListaErrores.get(i).Linea +"</td>");
            pw.println("<td>" + Application.App.ListaErrores.get(i).Columna +"</td>");
            pw.println("</tr>");
        }

        pw.println("</tbody>");
        pw.println("</table>");
        pw.println("</body>");
        pw.println("</html>");

    } catch (Exception e) {
        System.out.println("error, no se realizo el archivo");
    } finally {
        try {
            if (null != fichero) {
                fichero.close();
            }
        } catch (Exception e2) {
            e2.printStackTrace();
        }
```

El método graficarAFD() solo lee la matriz de transiciones para así graficarlo usando Graphviz.

```
public static void graficarAFD(String nombre){

    FileWriter fichero = null;
    PrintWriter pw = null;
    try {
        fichero = new FileWriter("afd/" + nombre + ".dot");
        pw = new PrintWriter(fichero);
        pw.println("digraph G{");

        pw.println("rankdir=LR");

        pw.println("node[shape=circle]");

        //Generamos los estados
        for (int i = 0; i < estados.size() -1; i++){
            if (estados.get(i).combinacion.contains(ListaSiguientes.size())){
                pw.println("nodo" + estados.get(i).S + " [ label =\"S" + estados.get(i).S + "\", shape=doublecircle ];");
            } else {
                pw.println("nodo" + estados.get(i).S + " [ label =\"S" + estados.get(i).S + "\"];");
            }
        }
        //realizamos las transiciones
        for (int i = 0; i < estados.size() - 1; i++){
            for (int j = 0; j < conjuntos.size() - 1; j++){
                if (Transiciones[i][j] != -1){
                    String conjunto = conjuntos.get(j);
                    if (conjunto.equals("\\n") || conjunto.equals("\\'")||conjunto.equals("\\\"")){
                        conjunto = conjunto.replace("\\", "\\\\");
                    }
                    pw.println("nodo" + i + "->nodo" + Transiciones[i][j] + " [label = \"" + conjunto + "\"];");
                }

            }
        }

        pw.println("}");
```

El método Equals() toma como parámetros dos listas de enteros y devuelve true si estas tienen los mismo elementos y false si son distintas.

```
151     public static boolean Equals(List<Integer> lista1, List<Integer> lista2){
152         if (lista1.size() != lista2.size()){
153             return false;
154         } else {
155             for (int i = 0; i < lista1.size(); i++){
156                 if (lista1.get(i) != lista2.get(i)){
157                     return false;
158                 }
159             }
160         }
161         return true;
162     }
```

El método CrearEstados() es un método que recibe a un estado y recursivamente genera los demás estados con cada nuevo estado que genere.

```
164     public static void CrearEstados(Estado S){
165         boolean encontrado = false;
166         for (int i = 0; i < S.combinacion.size(); i++){
167             int aux = S.combinacion.get(i);
168             //VALIDAR SI LA CONBINACION ES LA MISMA
169             for ( int j = 0; j < estados.size(); j++){
170                 if (Equals(estados.get(j).combinacion, ListaSiguientes.get(aux-1).siguientes) == true){
171                     encontrado = true;
172                 }
173             }
174             if (encontrado == false){
175                 Estado nuevo = new Estado(contEs, ListaSiguientes.get(aux-1).siguientes);
176                 estados.add(nuevo);
177                 contEs++;
178                 CrearEstados(nuevo);
179             }
180             encontrado = false;
181         }
182     }
```

El método de GetConjunto() recibe un String que es un conjunto y devuelve la posición dentro de la lista de conjuntos.

```
334     public static int GetConjunto (String conjunto){
335         //System.out.println("**************************************************************************");
336         int indice = -1;
337         for (int i = 0; i < conjuntos.size(); i++){
338             //System.out.println(conjuntos.get(i) + "<->" + conjunto);
339             if ((conjuntos.get(i)).contains(conjunto) || conjunto.contains(conjuntos.get(i))){
340                 //System.out.println("LO ENCONTRE PAPA");
341                 return i;
342             }
343         }
344         System.out.println("CONJUNTO: " + conjunto + " INDICE: " + indice);
345         return indice;
346     }
```

El método de graficarTransiciones() recibe un array de anteriores que siempre inicia la creación de estados, primero creamos los estados, luego creamos una lista de conjuntos, luego creamos la matriz de transiciones, y por ultimo graficamos la tabla con uso de Graphviz.

```java
public static void graficarTransiciones(int[] anteriores, String nombre){
    System.out.println("ESTOS SON LOS ESTADOS QUE LOGRÉ CREAR:");
    //PASAMOS EL ARRAY A UNA LISTA
    contEs = 0;
    List<Integer> anterioresList = new ArrayList<Integer>();
    ListaAuxiliar = new ArrayList<Integer>();
    for (int i = 0; i < anteriores.length; i++){
        anterioresList.add(anteriores[i]);
    }
    estados = new ArrayList<Estado>();
    Estado inicial = new Estado(contEs, anterioresList);
    estados.add(inicial);
    contEs++;
    //CREAMOS LOS LOS POSIBLES ESTADOS
    CrearEstados(inicial);
    for (int i = 0; i < estados.size(); i++){
        System.out.println(estados.get(i).S + ", " + estados.get(i).combinacion);
    }

    //CREAMOS LA LISTA DE CONJUNTOS
    conjuntos = new ArrayList<String>();
    for (int i = 0; i < ListaSiguientes.size(); i++){
        if (conjuntos.contains(ListaSiguientes.get(i).valor) == false){
            conjuntos.add(ListaSiguientes.get(i).valor);
        }
    }
    System.out.println(conjuntos);

    //CREAMOS LA MATRIZ DE TRANSICIONES
    Transiciones = new int[estados.size()][conjuntos.size()];
    for (int i = 0; i < estados.size(); i++){
        for  ( int j = 0; j < conjuntos.size(); j++){
            Transiciones[i][j] = -1;
        }
    }
```

```java
    //INGRESAMOS LOS DATOS A LA MATRIZ
    try{
        //recorremos la lista de estados encontrados
        for (int i = 0; i < estados.size(); i++){
            //recorremos las combinaciones del estado actual
            for (int j = 0; j < estados.get(i).combinacion.size(); j++){
                //obtenemos el indice de esa combinacion
                int indice = estados.get(i).combinacion.get(j);
                //obtenemos el string del conjunto de ese indice
                String conjunto = GetCadenaConjunto(indice);
                //recorreremos la lista de estados nuevamente
                for (int k = 0; k < estados.size(); k++){
                    //sera igual al estado que tenga la misma combinacion para obtener el estado
                    if (Equals(estados.get(k).combinacion, ListaSiguientes.get(indice - 1).siguientes)){
                        Transiciones[i][GetConjunto(conjunto)] = estados.get(k).S;
                    }
                }
            }
        }

    }catch(Exception e){
        System.err.println(e);
    }

    //IMPRIMIENDO LA MATRIZ
    for (int i = 0; i < estados.size(); i++){
        System.out.println("");
        for  ( int j = 0; j < conjuntos.size(); j++){
            System.out.print(Transiciones[i][j] + ", ");
        }
    }

    //EMPEZAMOS A GRAFICAR
    FileWriter fichero = null;
    PrintWriter pw = null;
    try {
        fichero = new FileWriter("Transiciones/" + nombre + ".dot");
        pw = new PrintWriter(fichero);
        pw.println("digraph G{");
        pw.println("tbl [");
        pw.println("shape=plaintext");
        pw.println("label=<");
```

El método de GetCadenaConjunto() se encarga de devolver el nombre del conjunto dentro de una lista respecto al índice que le entra.

```
328     public static String GetCadenaConjunto(int indice){
329         //System.out.println("**************************************************************************");
330         //System.out.println("INDICE: " + indice + " VALOR: " + ListaSiguientes.get(indice - 1).valor);
331         return ListaSiguientes.get(indice - 1).valor;
332     }
```

El método de graficarSiguientes se encarga de graficar el contenido de la lista de siguientes con ayuda de Graphviz.

```
348 public static void graficarSiguientes(String nombre){
349     FileWriter fichero = null;
350     PrintWriter pw = null;
351     try {
352         fichero = new FileWriter("siguientes/" + nombre + ".dot");
353         pw = new PrintWriter(fichero);
354         pw.println("digraph G{");
355         pw.println("tbl [");
356         pw.println("shape=plaintext");
357         pw.println("label=<");
358         pw.println("<TABLE border=\"10\" cellspacing=\"10\" cellpadding=\"10\" style=\"rounded\" bgcolor=\"/rdylgn11/1:/rdylgn11/11\" gradientangle=\"315\">");
359         pw.println("<TR>");
360         pw.println("<TD border=\"3\" colspan=\"2\"  bgcolor=\"/rdylgn11/6:/rdylgn11/9\">Simbolos</TD>");
361         pw.println("<TD border=\"3\" colspan=\"2\"  bgcolor=\"/rdylgn11/9:/rdylgn11/11\">Siguientes</TD>");
362         pw.println("</TR>");
363 
364         for (int i = 0; i < ListaSiguientes.size(); i++){
365             pw.println("<TR>");
366             pw.println("<TD border=\"3\" bgcolor=\"/rdylgn11/4:/rdylgn11/5\" gradientangle=\"270\">" + ListaSiguientes.get(i).id + "</TD>");
367             pw.println("<TD border=\"3\" bgcolor=\"/rdylgn11/3:/rdylgn11/9\" gradientangle=\"270\">" + ListaSiguientes.get(i).valor + "</TD>");
368             pw.println("<TD border=\"3\" colspan=\"2\"  bgcolor=\"/rdylgn11/1:/rdylgn11/8\">" + ListaSiguientes.get(i).siguientes + "</TD>");
369             pw.println("</TR>");
370         }
371 
372         pw.println("</TABLE>");
373         pw.println(">];");
374         pw.println("}");
375     } catch (Exception e) {
376         System.out.println("error, no se realizo el archivo");
377     } finally {
378         try {
379             if (null != fichero) {
380                 fichero.close();
381             }
382         } catch (Exception e2) {
383             e2.printStackTrace();
384         }
385     }
```

El método de graficarArbol() solo grafica el árbol de la raíz con ayuda de Graphviz.

```
413     public static void graficarArbol(Nodo act, String nombre){
414 
415         FileWriter fichero = null;
416         PrintWriter pw = null;
417         try {
418             fichero = new FileWriter("arboles/" + nombre + ".dot");
419             pw = new PrintWriter(fichero);
420             pw.println("digraph G{");
421             pw.println("rankdir=UD");
422             pw.println("node[shape=record]");
423             pw.println("concentrate=true");
424             pw.println(act.getCodigoInterno());
425             pw.println("}");
426         } catch (Exception e) {
427             System.out.println("error, no se realizo el archivo");
428         } finally {
429             try {
430                 if (null != fichero) {
431                     fichero.close();
432                 }
433             } catch (Exception e2) {
434                 e2.printStackTrace();
435             }
436         }
```

Por último se manejan los errores que puedan venir en el análisis sintáctico, si ya no se recupera del error procede a llamar el método de graficar los errores.

```
464     //-----------------------------------------para errores sintacticos-----------------------------------------------------------------------------------------------------
465     public void syntax_error(Symbol s)
466     {
467         App.TxtSalida.append("Error en la Línea " + (s.right+1) +" Columna "+(s.left+1)+ ". Identificador "+s.value + " no reconocido.\n" );
468         Errores err = new Errores("Sintáctico", "El identificador \"" + s.value + "\" no se esperaba.", s.right+1, s.left+1);
469         Application.App.ListaErrores.add(err);
470     }
471     public void unrecovered_syntax_error(Symbol s) throws java.lang.Exception
472     {
473         App.TxtSalida.append("Error en la Línea " + (s.right+1)+ " Columna "+(s.left+1)+". Identificador " +s.value + " no reconocido.\n");
474         Errores err = new Errores("Sintáctico", "El identificador \"" + s.value + "\" no se esperaba.", s.right+1, s.left+1);
475         Application.App.ListaErrores.add(err);
476         GraficarErrores();
477     }
478     //----------------------------------------------------------------------------------------------------------------------------------------------------------------
479 :}
```

- Interfaz gráfica de Java.

Las variables globales de la interfaz/aplicación son:

```
34          //VARIABLES GLOBALES
35          JFileChooser seleccionar = new JFileChooser();
36          File archivo;
37          FileInputStream entrada;
38          FileOutputStream salida;
39          public static List<Transiciones> ListaTransiciones;
40          public static List<Conjunto> ListaConjuntos;
41          public static List<Cadena> ListaCadenas;
42          public static List<Errores> ListaErrores;
```

Dentro del contructor solo inicializamos las listas donde almacenamos las transiciones, conjuntos, cadenas y errores, seteamos el icono de las imágenes.

```
47      public App() {
48          initComponents();
49          //SOLO AJUSTAMOS LA IMAGENF
50          ImageIcon imgIcon = new ImageIcon(getClass().getResource("/img/icon.jpg"));
51          Image imgEscalada = imgIcon.getImage().getScaledInstance(Imagen.getWidth(),
52                  Imagen.getHeight(), Image.SCALE_DEFAULT);
53          Icon iconoEscalado = new ImageIcon(imgEscalada);
54          Imagen.setIcon(iconoEscalado);
            ListaTransiciones = new ArrayList<Transiciones>();
            ListaConjuntos = new ArrayList<Conjunto>();
            ListaCadenas = new ArrayList<Cadena>();
            ListaErrores = new ArrayList<Errores>();
59          try {
                scanner();
61          } catch (InterruptedException ex) {
62              Logger.getLogger(App.class.getName()).log(Level.SEVERE, null, ex);
63          }
64      }
```

El método de Abrir() se encarga de recibir un archivo y regresar el contenido que hay dentro.

```
66      public String Abrir(File archivo){
67          String docu = "";
68          try{
69              entrada = new FileInputStream(archivo);
70              int ascii;
71              while((ascii = entrada.read())!= -1){
72                  char caracter = (char)ascii;
73                  docu += caracter;
74              }
75
76          } catch (Exception e){
77              System.out.println(e);
78          }
79          return docu;
80      }
```

El método de Guardar() se encarga de guardar el archivo con el string nuevo que recibe.

```
82      public String Guardar(File archivo, String doc){
83          String message = null;
84          System.out.println("");
85          try{
86              salida = new FileOutputStream(archivo);
87              byte[] txt = doc.getBytes();
88              salida.write(txt);
89              message = "Archivo Guardado";
90          } catch(Exception e) {
91              System.out.println(e);
92          }
93          return message;
94      }
```

El método scanner se encarga de setear el root al jTree.

```
96      public void scanner() throws InterruptedException {
97          String location = "Graficas";
98          File currentDir = new File(location);
99          DefaultTreeModel model = (DefaultTreeModel) jTree1.getModel();
100         DefaultMutableTreeNode root = (DefaultMutableTreeNode) model.getRoot();
101         displayDirectoryContents(currentDir, root);
102     }
```

El evento de “click” dentro del elemento de abrir dentro del menú solo se encarga de obtener el archivo a abrir y lo ubica en el TextArea.

```
375     private void jMenuItem1ActionPerformed(java.awt.event.ActionEvent evt) {
376         // TODO add your handling code here:
377         if(seleccionar.showDialog(null, "Abrir") == JFileChooser.APPROVE_OPTION){
378             archivo = seleccionar.getSelectedFile();
379             if(archivo.canRead()){
380                 if(archivo.getName().endsWith("olc")){
381                     String docu = Abrir(archivo);
382                     TxtEntrada.setText(docu);
383                 } else {
384                     JOptionPane.showMessageDialog(null, "Archivo incorrecto");
385                 }
386             }
387         }
388     }
```

El evento de “click” dentro del elemento de guardar dentro del menú se encarga de guardar el archivo si este ya fue cargado y si no, crea el nuevo archivo.

```
390     private void jMenuItem3ActionPerformed(java.awt.event.ActionEvent evt) {
391         // TODO add your handling code here:
392         if (seleccionar.showDialog(null, "Guardar") == JFileChooser.APPROVE_OPTION){
393             archivo = seleccionar.getSelectedFile();
394             if(archivo.getName().endsWith("olc")){
395                 String doc = TxtEntrada.getText();
396                 String message = Guardar(archivo, doc);
397                 if(message != null){
398                     JOptionPane.showMessageDialog(null, message);
399                 } else {
400                     JOptionPane.showMessageDialog(null, "Archivo incorrecto");
401                 }
402             } else {
403                 JOptionPane.showMessageDialog(null, "Guardar documento de texto");
404             }
405         }
406     }
```

El evento de “click dentro del elemento de guardar como dentro del menú se encarga de crear un nuevo archivo obteniendo el texto del TextArea.

```
408     private void jMenuItem2ActionPerformed(java.awt.event.ActionEvent evt) {
409         // TODO add your handling code here:
410         String doc = TxtEntrada.getText();
411         if (archivo != null) {
412             String message = null;
413             try {
414                 salida = new FileOutputStream(archivo);
415                 byte[] txt = doc.getBytes();
416                 salida.write(txt);
417                 message = "Archivo Guardado";
418             } catch (Exception e) {
419                 System.out.println(e);
420             }
421             if (message != null) {
422                 JOptionPane.showMessageDialog(null, message);
423             } else {
424                 JOptionPane.showMessageDialog(null, "Archivo incorrecto");
425             }
426         } else {
427             if (seleccionar.showDialog(null, "Guardar") == JFileChooser.APPROVE_OPTION){
428             archivo = seleccionar.getSelectedFile();
429             if(archivo.getName().endsWith("olc")){
430                 String message = Guardar(archivo, doc);
431                 if(message != null){
432                     JOptionPane.showMessageDialog(null, message);
433                 } else {
434                     JOptionPane.showMessageDialog(null, "Su archivo se encuentra vacío");
435                 }
436             } else {
437                 JOptionPane.showMessageDialog(null, "Guardar documento de texto");
438             }
439         }
```

En el botón “Generar Autómatas” solo ejecutamos nuestro analizador previamente explicado con el string que obtenemos de nuestro TextArea de entrada. Luego de un segundo procedemos a actualizar el jTree con las gráficas que logró generar.

```
443     private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
444         // TODO add your handling code here:
445         //REINICIAMOS LAS LISTAS
446         ListaTransiciones.clear();
447         ListaConjuntos.clear();
448         ListaCadenas.clear();
449         ListaErrores.clear();
450         //INTENTAR INICIALIZAR ALGO
            parser.Caracteres = new ArrayList<String>();
452         TxtSalida.setText("");
453         try {
454             String path = TxtEntrada.getText();
455             Analizadores.parser sintactico;
456             sintactico = new Analizadores.parser(new Analizadores.Lexico(new StringReader(path)));
457             sintactico.parse();
458         } catch (Exception e) {
459         }
460         //ELIMINAMOS LOS HIJOS DEL BICHO
461         DefaultTreeModel modelo = (DefaultTreeModel)jTree1.getModel();
462         DefaultMutableTreeNode root = (DefaultMutableTreeNode) modelo.getRoot();
463         root.removeAllChildren();
464         modelo.reload();
465         try {
466             TimeUnit.SECONDS.sleep(1);
467             scanner();
468         } catch (InterruptedException ex) {
469             Logger.getLogger(App.class.getName()).log(Level.SEVERE, null, ex);
470         }
471     }
```

En el botón de “Analizar Entradas” se encarga de generar el archivo json de los lexemas que estén correctos o incorrectos, esto se logra mediante las listas de transiciones, conjuntos y cadenas que anteriormente almacenamos al analizar el archivo de entrada, con ayuda de la matriz de enteros que representa a la tabla de transiciones.

```
473     private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
474         // TODO add your handling code here:
475         System.out.println("");
476         System.out.println("***************** SE VINO LO CHIDO ****************");
477         TxtSalida.setText("");
478
479         FileWriter fichero = null;
            PrintWriter pw = null;
481         try {
482             if (archivo != null){
483                 String name = archivo.getName();
484                 name = name.replaceFirst("[.][^.]+$", "");
485                 fichero = new FileWriter("Graficas/Salidas/" + name + ".json");
486                 pw = new PrintWriter(fichero);
487             } else {
488                 fichero = new FileWriter("Graficas/Salidas/Lexemas.json");
489                 pw = new PrintWriter(fichero);
490             }
491             pw.println("[");
492             //EMPEXAMOS A LEER LOS LEXEMAS:
493             for (int i = 0; i < ListaCadenas.size(); i++) {
494                 int S = 0;
495                 String cadena = ListaCadenas.get(i).Lexema;
496                 cadena = cadena.replace("\"", "");
497                 String nombre = ListaCadenas.get(i).nombre;
498                 int[][] trans = null;
499                 int EstadoFinal = -10;
500                 List<String> conjuntos = null;
501                 Conjunto ConjuntoReal = null;
502                 boolean correcto = true;
503                 //System.out.println("LEXEMA ACTUAL: " + cadena);
504                 //System.out.println("ID: " + nombre);
505                 //System.out.println("AHORA COMPARAMOS: ");
506                 //OBTENEMOS LA MATRIZ INDICE
507                 for (int j = 0; j < ListaTransiciones.size(); j++) {
508                     String nombre2 = ListaTransiciones.get(j).nombre;
509                     //System.out.println(nombre2 + "<->" + nombre);
510                     if (nombre.equals(nombre2)) {
511                         //System.out.println("ENCONTRADO!");
512                         trans = ListaTransiciones.get(j).transiciones;
513                         EstadoFinal = ListaTransiciones.get(j).EstadoFinal;
514                         conjuntos = ListaTransiciones.get(j).conjuntos;
515                         break;
516                     }
```

```
                for (int x = 0; x < trans.length; x++) {
                    for (int y = 0; y < trans[x].length; y++) {
                        //System.out.print(trans[x][y] + ", ");
                    }
                    //System.out.println("");
                }
                //RECORREMOS LA CADENA ACTUAL
                //System.out.println("AHORA RECORREMOS LA CADENA:");
                for (int j = 0; j < cadena.length(); j++) {
                    String caracter = String.valueOf(cadena.charAt(j));
                    //System.out.println("CARACTER: " + caracter);
                    if (correcto == true) {
                        //RECORREMOS LA FILA DEL ESTADO ACTUAL
                        //System.out.println("RECORREMOS LA FILA DEL ESTADO:" + S);
                        for (int x = 0; x < trans[S].length - 1; x++) {
                            //System.out.println("POSICION: " + S + ", " + x + " INDICE: " + trans[S][x]);
                            boolean encontrado = false;
                            if (trans[S][x] != -1) {
                                //ENCONTRAMOS UNA POSIBLE TRANSICION
                                //OBTENEMOS EL NOMBRE DE LOS CONJUNTO DE ESA POSICION
                                String conjuntoActual = conjuntos.get(x);
                                if (conjuntoActual.equals("\\n") || conjuntoActual.equals("\\'") || conjuntoActual.equals("\\\"")) {
                                    if (caracter.equals("\\")) {
                                        break;
                                    }
                                }
                                //OBTENEMOS LA LISTA DE CARACTERES
                                for (int z = 0; z < ListaConjuntos.size(); z++) {
                                    if (ListaConjuntos.get(z).nombre.equals(conjuntoActual)) {
                                        ConjuntoReal = ListaConjuntos.get(z);
                                    }
                                }
                                //VERIFICAMOS SI EL CARACTER ESTA EN ESE CONJUNTO
                                if (ConjuntoReal.Caracteres.contains(caracter)) {
                                    //EL NUEVO ESTADO ES:
                                    S = trans[S][x];
                                    encontrado = true;
                                    break;
                                }
                            }
                            if (x == trans[S].length - 2 && encontrado == false) {
                                correcto = false;
                            }
                        }
```

```
                //IMPRIMIMOS SI ES CORRECTO O INCORRECTO Y SEGUIMOS EL JSON
                //System.out.println("EL LEXEMA FINALIZO EN EL ESTADO: " + S);
                if (EstadoFinal == S && correcto == true) {
                    TxtSalida.append("EL LEXEMA: " + ListaCadenas.get(i).Lexema + " ES CORRECTO\n");
                    pw.println("{");
                    pw.println("\"valor\": " + ListaCadenas.get(i).Lexema + ",");
                    pw.println("\"ExpresionRegular\": \"" + ListaCadenas.get(i).nombre + "\",");
                    pw.println("\"Resultado\":" + "\"Cadena Válida\"");
                    pw.println("},");
                } else {
                    TxtSalida.append("EL LEXEMA: " + ListaCadenas.get(i).Lexema + " ES INCORRECTO\n");
                    pw.println("{");
                    pw.println("\"valor\": " + ListaCadenas.get(i).Lexema + ",");
                    pw.println("\"ExpresionRegular\": \"" + ListaCadenas.get(i).nombre + "\",");
                    pw.println("\"Resultado\":" + "\"Cadena Inválida\"");
                    pw.println("},");
                }
            }
            pw.println("]");
        } catch (Exception e) {
            System.out.println("error, no se realizo el archivo");
            TxtSalida.append("error, no se realizo el archivo\n");
        } finally {
            try {
                if (null != fichero) {
                    fichero.close();
                }
            } catch (Exception e2) {
                e2.printStackTrace();
            }
        }
```

- Clase Cadena

Objeto que es formado por un lexema y un nombre.

```java
public class Cadena {
    public String Lexema;
    public String nombre;

    public Cadena(String Lexema, String nombre) {
        this.Lexema = Lexema;
        this.nombre = nombre;
    }
```

- Clase Conjunto

Objeto formado por una lista de caracteres y por un nombre.

```java
import java.util.List;

/**
 *
 * @author Angel Arteaga
 */
public class Conjunto {
    public List<String> Caracteres;
    public String nombre;

    public Conjunto(List<String> Caracteres, String nombre) {
        this.Caracteres = Caracteres;
        this.nombre = nombre;
    }

}
```

- Clase Errores

Objeto formado por el tipo de error, su descripción, la línea donde se encontró y la fila donde se encontró.

```java
public class Errores {
    public String Tipo;
    public String Descripcion;
    public int Linea;
    public int Columna;

    public Errores(String Tipo, String Descripcion, int Linea, int Columna) {
        this.Tipo = Tipo;
        this.Descripcion = Descripcion;
        this.Linea = Linea;
        this.Columna = Columna;
    }

}
```

- Clase Estado:

Objeto formado por el entero del estado una lista de combinación asignado a ese estado.

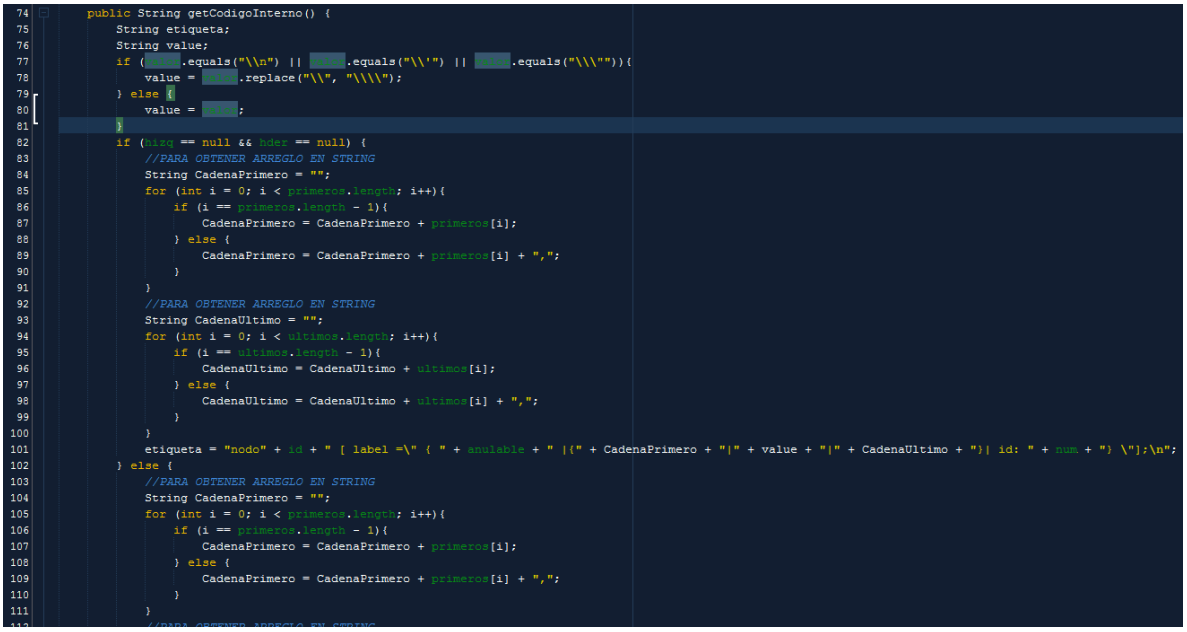
```java
public class Estado {
    public int S;
    public List<Integer> combinacion;

    public Estado(int S, List<Integer> combinacion) {
        this.S = S;
        this.combinacion = combinacion;
    }
```

- Clase Nodo

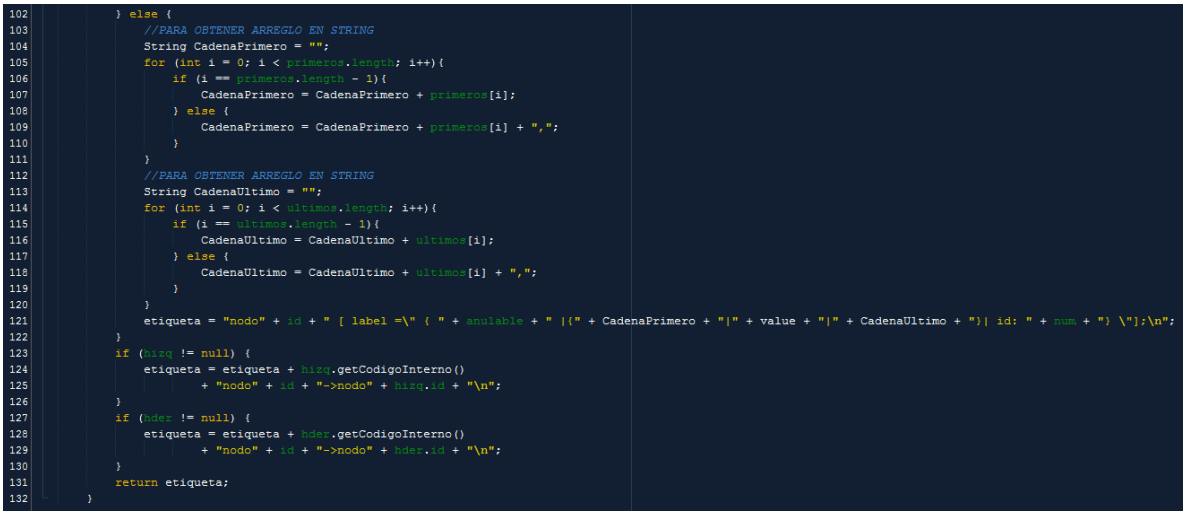
Objeto formado por su hijo izquierdo, su hijo derecho, el valor, su id, el numero de nodo, si es anulable, sus primeros y sus últimos.

```java
public class Nodo {

    public Nodo hizq;
    public Nodo hder;
    public String valor;
    public int id;
    public int num;
    public String anulable;
    public int[] primeros;
    public int [] ultimos;

    public Nodo(Nodo hizq, Nodo hder, String valor, int id, int num, String anulable, int[] primeros, int[] ultimos) {
        this.hizq = hizq;
        this.hder = hder;
        this.valor = valor;
        this.id = id;
        this.num = num;
        this.anulable = anulable;
        this.primeros = primeros;
        this.ultimos = ultimos;
    }
```

El método getCodigoInterno() se encarga de la concatenación necesaria para poder graficar todo el árbol desde la raíz en el lenguaje utilizado de Graphviz, usando la recursividad.



```
    public String getCodigoInterno() {
        String etiqueta;
        String value;
        if (value.equals("\\n") || value.equals("\\'") || value.equals("\\\"")){
            value = value.replace("\\", "\\\\");
        } else {
            value = value;
        }
        if (hizq == null && hder == null) {
            //PARA OBTENER ARREGLO EN STRING
            String CadenaPrimero = "";
            for (int i = 0; i < primeros.length; i++){
                if (i == primeros.length - 1){
                    CadenaPrimero = CadenaPrimero + primeros[i];
                } else {
                    CadenaPrimero = CadenaPrimero + primeros[i] + ",";
                }
            }
            //PARA OBTENER ARREGLO EN STRING
            String CadenaUltimo = "";
            for (int i = 0; i < ultimos.length; i++){
                if (i == ultimos.length - 1){
                    CadenaUltimo = CadenaUltimo + ultimos[i];
                } else {
                    CadenaUltimo = CadenaUltimo + ultimos[i] + ",";
                }
            }
            etiqueta = "nodo" + id + " [ label =\" { " + anulable + " |{" + CadenaPrimero + "|" + value + "|" + CadenaUltimo + "}| id: " + num + "} \"];\n";
        } else {
            //PARA OBTENER ARREGLO EN STRING
            String CadenaPrimero = "";
            for (int i = 0; i < primeros.length; i++){
                if (i == primeros.length - 1){
                    CadenaPrimero = CadenaPrimero + primeros[i];
                } else {
                    CadenaPrimero = CadenaPrimero + primeros[i] + ",";
                }
            }
```



```
        } else {
            //PARA OBTENER ARREGLO EN STRING
            String CadenaPrimero = "";
            for (int i = 0; i < primeros.length; i++){
                if (i == primeros.length - 1){
                    CadenaPrimero = CadenaPrimero + primeros[i];
                } else {
                    CadenaPrimero = CadenaPrimero + primeros[i] + ",";
                }
            }
            //PARA OBTENER ARREGLO EN STRING
            String CadenaUltimo = "";
            for (int i = 0; i < ultimos.length; i++){
                if (i == ultimos.length - 1){
                    CadenaUltimo = CadenaUltimo + ultimos[i];
                } else {
                    CadenaUltimo = CadenaUltimo + ultimos[i] + ",";
                }
            }
            etiqueta = "nodo" + id + " [ label =\" { " + anulable + " |{" + CadenaPrimero + "|" + value + "|" + CadenaUltimo + "}| id: " + num + "} \"];\n";
        }
        if (hizq != null) {
            etiqueta = etiqueta + hizq.getCodigoInterno()
                    + "nodo" + id + "->nodo" + hizq.id + "\n";
        }
        if (hder != null) {
            etiqueta = etiqueta + hder.getCodigoInterno()
                    + "nodo" + id + "->nodo" + hder.id + "\n";
        }
        return etiqueta;
    }
```

- Clase TablaSiguientes

Objeto formado por un id, un valor y una lista de enteros.

```java
public class TablaSiguientes {

    public int id;
    public String valor;
    public List<Integer> siguientes;

    public TablaSiguientes(int id, List siguientes, String valor) {
        this.id = id;
        this.siguientes = siguientes;
        this.valor = valor;
        this.siguientes = new ArrayList<Integer>();
    }
```

- Clase Transiciones

Objeto formado por una matriz de enteros, un nombre, un estado final y una lista de conjuntos.

```java
public class Transiciones {
    public int[][] transiciones;
    public String nombre;
    public int EstadoFinal;
    public List<String> conjuntos;

    public Transiciones(int[][] transiciones, String nombre, int EstadoFinal, List<String> conjuntos) {
        this.transiciones = transiciones;
        this.nombre = nombre;
        this.EstadoFinal = EstadoFinal;
        this.conjuntos = conjuntos;
    }

}
```