Universidad de San Carlos de Guatemala
Primer semestre
Facultad de ingeniería
Escuela de Ciencias y Sistemas
Organización de Lenguajes y Compiladores 1

MANUAL TECNICO

Roberto Carlos Gómez Donis

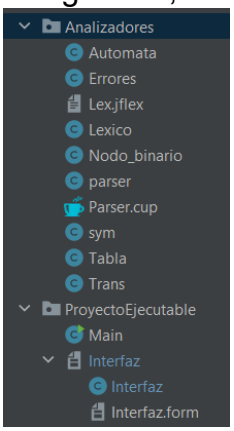
202000544

23/03/2023

Proyecto 1: ExRegan Usac

ExRegan es un programa especializado en la lectura de archivos con extensión ".olc". Su principal función es analizar las expresiones regulares contenidas en estos archivos, empleando diversos métodos como el de árbol, tabla de siguientes, tabla de transiciones, autómatas y análisis de cadena. Este software fue desarrollado íntegramente en Java, utilizando las herramientas "jflex" y "cup" para la creación del analizador léxico y sintáctico. Estas herramientas brindaron la base necesaria para desarrollar el programa desde cero. A continuación, se detallará el código fuente utilizado en su creación.

En general, esta es la estructura que se usó para la creación del software.



Empecemos por el Main. En el Main lo unico que se hace es correr la interfaz grafica

```java
package ProyectoEjecutable;


import javax.swing.*;

no usages  Roberto-Gomez1
public class Main {
    no usages  Roberto-Gomez1
    public static void main(String[] args) throws Exception{
        Roberto-Gomez1
        SwingUtilities.invokeLater(new Runnable() {
            Roberto-Gomez1
            @Override
            public void run() {
                Interfaz principal= new Interfaz();
                principal.setVisible(true);
            }
        });


    }

}
```

En la clase "Interfaz", se encuentra la creación de la interfaz gráfica del programa, así como la lógica necesaria para ingresar la información del archivo a dicha interfaz. Para ello, se definen las variables necesarias para su correcta implementación.

```
public class Interfaz extends JFrame{
    12 usages
    public JPanel panel;
    6 usages
    public  JTextArea area1,area2;
    3 usages
    public JLabel imagen;
    7 usages
    public File archivo,generar_imagen;
    3 usages
    public JMenuItem abrir, guardar, guardar_c,abrir_imagen,cerrar;
```

En el método "interfaz" se lleva a cabo únicamente la creación del frame y la definición de su estética. En el método "componentes", por su parte, se invoca a los distintos componentes creados y previamente añadidos al frame. En el método "Panel" se incluye la barra de menú, y se definen las acciones correspondientes a cada una de sus opciones. El método "AreaTexto", por su parte, se encarga de la creación y modificación de las áreas de texto. Finalmente, el método "interpretar" es aquel que se encarga de realizar el análisis del archivo, y es invocado cuando se presiona el botón de "Generar Autómata".

```
public Interfaz(){...}
1 usage   Roberto-Gomez1
public void Componentes(){...}

1 usage   Roberto-Gomez1
private void Panel(){...}
```

```
private void AreaTexto(){...}

1 usage   Roberto-Gomez1
private static void interpretar(String aux){...}
1 usage   Roberto-Gomez1
private void Botones(){...}
```

Por ultimo estas son las acciones de la barra menú.

```
ActionListener abrir_accion = new ActionListener() {...};
1 usage   Roberto-Gomez1
ActionListener guardar_accion = new ActionListener() {...};
1 usage   Roberto-Gomez1
ActionListener guardarc_accion = new ActionListener() {...};
1 usage   Roberto-Gomez1
ActionListener cerrarAccion = new ActionListener() {...};
1 usage   Roberto-Gomez1
ActionListener generar_accion= new ActionListener() {...};
1 usage   Roberto-Gomez1 *
ActionListener crear_imagen = new ActionListener() {...};
1 usage   Roberto-Gomez1
```

Las opciones "abrir_accion", "guardarc_accion" y "crear_imagen" realizan prácticamente la misma tarea, la cual es manejar la apertura de archivos y colocar su contenido en el TextArea o generar una nueva imagen. La diferencia entre cada una es que la primera se utiliza para abrir un archivo y colocar su contenido en el TextArea, la segunda para crear un nuevo archivo y la tercera para colocar la imagen en la interfaz. En cambio, "guardar_accion", "cerrar_accion" y "generar_accion" tienen funcionalidades diferentes. "guardar_accion" se encarga de guardar las modificaciones realizadas en el archivo abierto previamente, "cerrar_accion" cierra el programa y "generar_accion" se utiliza para realizar el análisis del archivo y generar el resultado correspondiente, como se mencionó anteriormente.

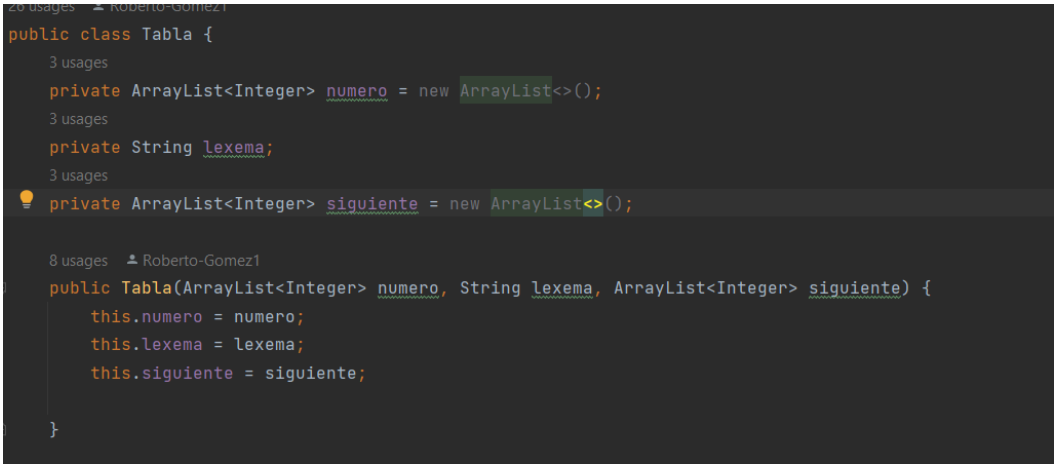
Hasta ahora hemos finalizado la parte de la interfaz gráfica y continuaremos con la lógica de desarrollo del programa. Para ello, comenzaremos explicando los objetos que se utilizaron.

Lo único que se mostrara acá es las variables utilizadas y el constructor:

El objeto "Trans" está diseñado para crear la tabla de transiciones. Para esto, se utilizaron las variables que se pueden observar en la imagen. La variable "inicial" se usó solamente para hacer una validación y encontrar el carácter o lexema y sus siguientes correspondientes. Basándose en los siguientes, se logró obtener el estado final.

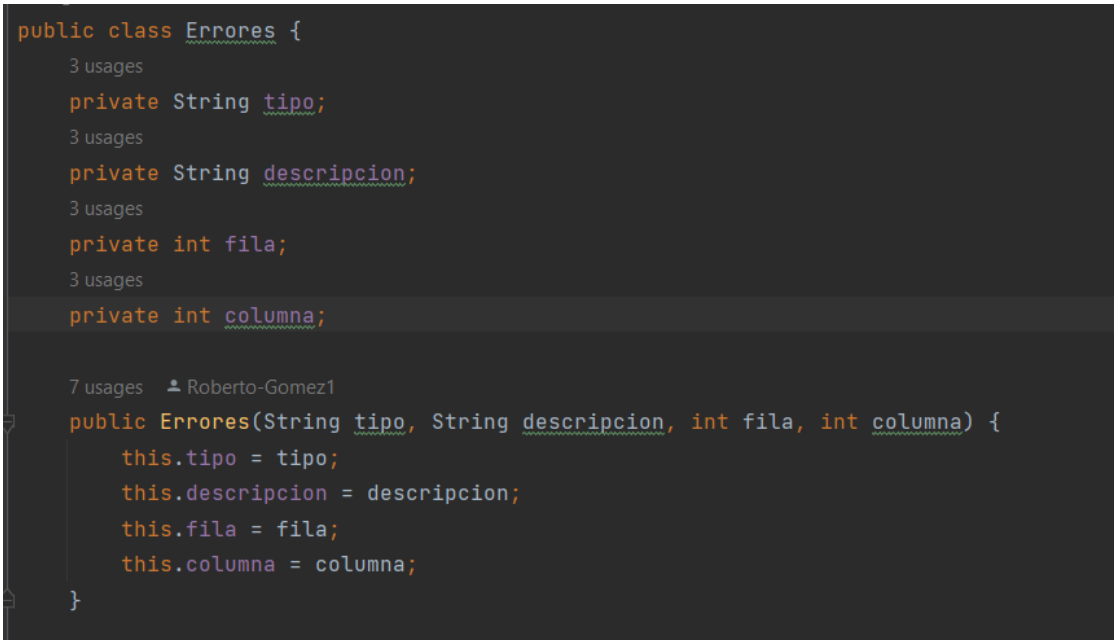
```
public class Trans {
    3 usages
    private ArrayList<Integer> inicial = new ArrayList<>();
    3 usages
    private String estado_inicial;
    3 usages
    private String lexema;
    3 usages
    private ArrayList<Integer> siguientes = new ArrayList<>();
    3 usages
    private String estado_final;

    4 usages  Roberto-Gomez1
    public Trans(ArrayList<Integer> inicial, String estado_inicial, String lexema, ArrayList<Integer> siguientes, String estado_final) {
        this.inicial = inicial;
        this.estado_inicial = estado_inicial;
        this.lexema = lexema;
        this.siguientes = siguientes;
        this.estado_final = estado_final;
    }
```

El objeto "Tabla" fue utilizado para crear las tablas de siguientes. Gracias a este objeto, también se pudo crear la tabla de transiciones. Se utilizaron variables auxiliares del tipo Tabla para guardar cada carácter con su cabecera, los estados y los siguientes de los estados. La variable "numero" se usó para guardar las cabeceras de los siguientes, mientras que la variable "lexema" se usó para guardar el carácter de las cabeceras.



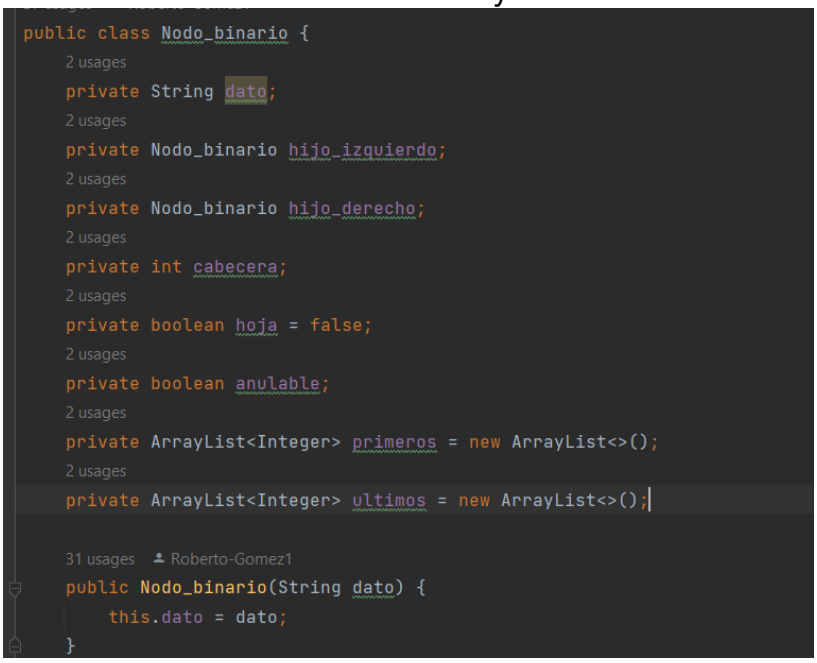
```
public class Tabla {
    3 usages
    private ArrayList<Integer> numero = new ArrayList<>();
    3 usages
    private String lexema;
    3 usages
    private ArrayList<Integer> siguiente = new ArrayList<>();

    8 usages  Roberto-Gomez1
    public Tabla(ArrayList<Integer> numero, String lexema, ArrayList<Integer> siguiente) {
        this.numero = numero;
        this.lexema = lexema;
        this.siguiente = siguiente;

    }
```

El objeto “Errores” únicamente se utilizó para la realización del reporte de errores léxicos:



```
public class Errores {
    3 usages
    private String tipo;
    3 usages
    private String descripcion;
    3 usages
    private int fila;
    3 usages
    private int columna;

    7 usages  Roberto-Gomez1
    public Errores(String tipo, String descripcion, int fila, int columna) {
        this.tipo = tipo;
        this.descripcion = descripcion;
        this.fila = fila;
        this.columna = columna;
    }
```

Aquí es donde ya empieza a entrar un poco de la logica, en estos objetos “Nodo_binario” es para la realizacion del arbol junto con sus cabeceras, siguientes, si es anulable o no y si es carácter o es operación

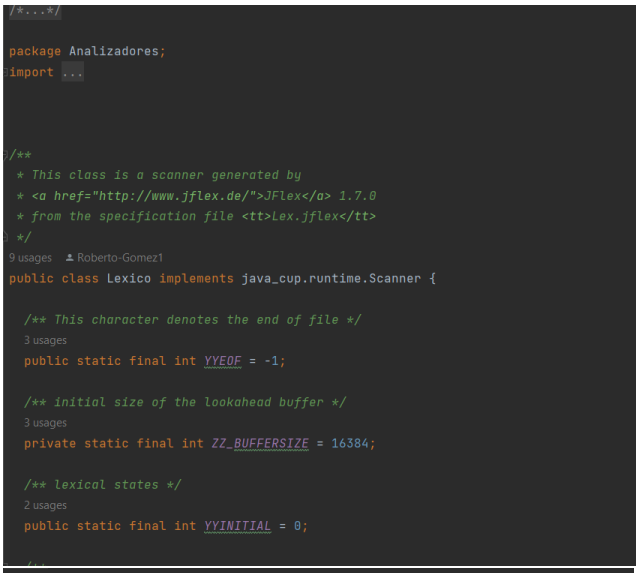


A continuación, presentamos el archivo ".jflex", el cual es esencial en la creación del Analizador Léxico. En esta clase se definen las variables que se utilizarán en el lenguaje, así como también se establecen las expresiones regulares que corresponden a dicho lenguaje.

```
package Analizadores;
import java_cup.runtime.Symbol;
import java.util.ArrayList;


%%

%{
public static ArrayList<Errores> errores= new ArrayList<>();
    //Código de usuario
%}

%class Lexico
%cup
%public
%line
%column
%char
%unicode


//SimboloLenguaje
RCONJ = "CONJ"
DOS_PUNTOS= ":"
COMA = ","
PALITO = "~"
LLAVE_ABRE= "{"
LLAVE_CIERRA= "}"
PUNTO = "."
OR = "|"
ASTERISCO= "*"
ASTERISCO= "*"
SUMA = "+"
PREGUNTA= "?"
PORCENTAJE = "%"
PUNTO_COMA = ";"
COMILLA = \"


//ExpresionRegular
ESCAPADO = "\\n" | "\\\"" | "\\\'"
NESCAPADO = [^\'\"]
SPACE = [\ \t\r\n]+
COMENTARIO_MAS= "<!"[^!]*"!>"
COMENTARIO = "//" .*
NOMBRE_M = [A-Z]
NOMBRE = [a-z]
ENTERO  = [0-9]
ASCII = [ -/:-@\[-`{-}]
CADENA = \" ([^\"] | "\\\"") + \"
CARACTER = (\" {NESCAPADO} \" ) | {ESCAPADO}
IDENTIFICADOR = [a-zA-Z_][a-zA-Z0-9_]+
FLECHA = "-" {SPACE}* ">"
```

```
%%
<YYINITIAL> {COMENTARIO_MAS}    { /*Comentario mas de una linea ignorado*/ }
<YYINITIAL> {COMENTARIO}        { /*Comentario de una linea ignorado*/ }
<YYINITIAL> {SPACE}             { /*Espacios en blanco, ignorados*/ }
<YYINITIAL> {COMILLA}           { /* ignorar las comillas*/}
<YYINITIAL> {RCONJ}             {  return new Symbol(sym.RCONJ, yyline, yycolumn,yytext());  }
<YYINITIAL> {DOS_PUNTOS}        {  return new Symbol(sym.DOS_PUNTOS, yyline, yycolumn,yytext());  }
<YYINITIAL> {FLECHA}            {  return new Symbol(sym.FLECHA, yyline, yycolumn,yytext());  }
<YYINITIAL> {COMA}              {  return new Symbol(sym.COMA, yyline, yycolumn,yytext());  }
<YYINITIAL> {PALITO}            {  return new Symbol(sym.PALITO, yyline, yycolumn,yytext());  }
<YYINITIAL> {LLAVE_ABRE}        {  return new Symbol(sym.LLAVE_ABRE, yyline, yycolumn,yytext());  }
<YYINITIAL> {LLAVE_CIERRA}      {  return new Symbol(sym.LLAVE_CIERRA, yyline, yycolumn,yytext());  }
<YYINITIAL> {PUNTO}             {  return new Symbol(sym.PUNTO, yyline, yycolumn,yytext());  }
<YYINITIAL> {OR}                {  return new Symbol(sym.OR, yyline, yycolumn,yytext());  }
<YYINITIAL> {ASTERISCO}         {  return new Symbol(sym.ASTERISCO, yyline, yycolumn,yytext());  }
<YYINITIAL> {SUMA}              {  return new Symbol(sym.SUMA, yyline, yycolumn,yytext());  }
<YYINITIAL> {PREGUNTA}          {  return new Symbol(sym.PREGUNTA, yyline, yycolumn,yytext());  }
<YYINITIAL> {PORCENTAJE}        {  return new Symbol(sym.PORCENTAJE, yyline, yycolumn,yytext());  }
<YYINITIAL> {PUNTO_COMA}        {  return new Symbol(sym.PUNTO_COMA, yyline, yycolumn,yytext());  }
<YYINITIAL> {IDENTIFICADOR}     {  return new Symbol(sym.IDENTIFICADOR, yyline, yycolumn,yytext());   }
<YYINITIAL> {NOMBRE}            {  return new Symbol(sym.NOMBRE, yyline, yycolumn,yytext());   }
<YYINITIAL> {CARACTER}          {  return new Symbol(sym.CARACTER, yyline, yycolumn,yytext());   }
<YYINITIAL> {CADENA}            {  return new Symbol(sym.CADENA, yyline, yycolumn,yytext());   }
<YYINITIAL> {NOMBRE_M}          {  return new Symbol(sym.NOMBRE_M, yyline, yycolumn,yytext());  }
<YYINITIAL> {ENTERO}            {  return new Symbol(sym.ENTERO, yyline, yycolumn,yytext());   }
<YYINITIAL> {ASCII}             {  return new Symbol(sym.ASCII, yyline, yycolumn,yytext());   }


<YYINITIAL> . {
        String errLex = "Error léxico : '"+yytext()+"' en la línea: "+(yyline+1)+" y columna: "+(yycolumn+1);
        errores.add(new Errores("Lexico", "El caracter: "+yytext()+" no pertenece al lenguaje",(yyline+1) , (yycolumn+1)));
```

A partir de esta estructura creada se crea la clase Lexico.java y Sys.java, donde una es el analizador en si y la otra es de tokens.

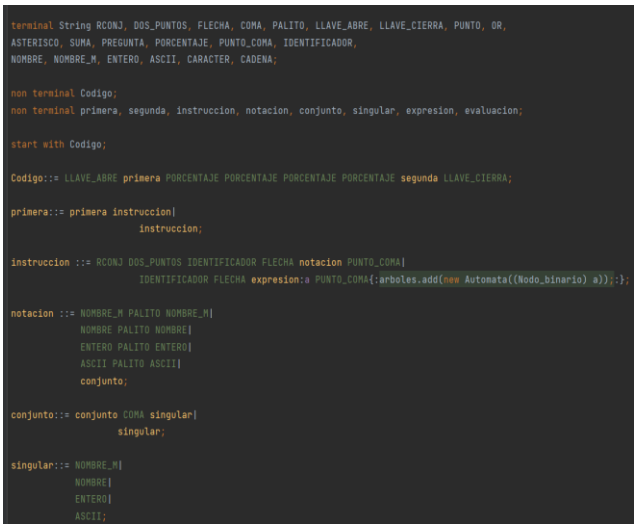


```
/*...*/

package Analizadores;
import ...



/**
 * This class is a scanner generated by
 * <a href="http://www.jflex.de/">JFlex</a> 1.7.0
 * from the specification file <tt>Lex.jflex</tt>
 */
9 usages  Roberto-Gomez1
public class Lexico implements java_cup.runtime.Scanner {

  /** This character denotes the end of file */
  3 usages
  public static final int YYEOF = -1;

  /** initial size of the lookahead buffer */
  3 usages
  private static final int ZZ_BUFFERSIZE = 16384;

  /** lexical states */
  2 usages
  public static final int YYINITIAL = 0;
```

```
package Analizadores;

/** CUP generated class containing symbol constants. */
2 usages  Roberto-Gomez1
public class sym {
  /* terminals */
  1 usage
  public static final int LLAVE_ABRE = 7;
  1 usage
  public static final int PUNTO = 9;
  1 usage
  public static final int CADENA = 22;
  1 usage
  public static final int LLAVE_CIERRA = 8;
  1 usage
  public static final int OR = 10;
  1 usage
  public static final int SUMA = 12;
  1 usage
  public static final int RCONJ = 2;
  1 usage
  public static final int FLECHA = 4;
  1 usage
  public static final int PALITO = 6;
  1 usage
```

Posteriormente se presenta lo que es el “cup”, este archivo se utilizo para la creación de nuestro Analizador sintáctico, aquí definimos todo lo que es nuestra gramática y sobreotdo se empieza a guardar las expresiones para empezar el análisis. Luego

de terminar esta clase se crea la clase “parser.java” que es el analizador.



```
terminal String RCONJ, DOS_PUNTOS, FLECHA, COMA, PALITO, LLAVE_ABRE, LLAVE_CIERRA, PUNTO, OR,
ASTERISCO, SUMA, PREGUNTA, PORCENTAJE, PUNTO_COMA, IDENTIFICADOR,
NOMBRE, NOMBRE_M, ENTERO, ASCII, CARACTER, CADENA;

non terminal Codigo;
non terminal primera, segunda, instruccion, notacion, conjunto, singular, expresion, evaluacion;

start with Codigo;

Codigo::= LLAVE_ABRE primera PORCENTAJE PORCENTAJE PORCENTAJE PORCENTAJE segunda LLAVE_CIERRA;

primera::= primera instruccion|
           instruccion;

instruccion ::= RCONJ DOS_PUNTOS IDENTIFICADOR FLECHA notacion PUNTO_COMA|
                IDENTIFICADOR FLECHA expresion:a PUNTO_COMA{:arboles.add(new Automata((Nodo_binario) a));:};

notacion ::= NOMBRE_M PALITO NOMBRE_M|
             NOMBRE PALITO NOMBRE|
             ENTERO PALITO ENTERO|
             ASCII PALITO ASCII|
             conjunto;

conjunto::= conjunto COMA singular|
            singular;

singular::= NOMBRE_M|
            NOMBRE|
            ENTERO|
            ASCII;
```

```
                Nodo_binario padre = new Nodo_binario(a);
                padre.setHijo_izquierdo((Nodo_binario) b);
                RESULT = padre;
                :}|
                SUMA:a expresion:b {:
                Nodo_binario padre = new Nodo_binario(a);
                padre.setHijo_izquierdo((Nodo_binario) b);
                RESULT = padre;
                :}|
                PREGUNTA:a expresion:b {:
                Nodo_binario padre = new Nodo_binario(a);
                padre.setHijo_izquierdo((Nodo_binario) b);
                RESULT = padre;
                :}|
                CARACTER:a {:
                     Nodo_binario hoja = new Nodo_binario(a);
                     hoja.setHoja(true);
                     hoja.setAnulable(false);
                     RESULT = hoja;
                :}|
                LLAVE_ABRE IDENTIFICADOR:a LLAVE_CIERRA{:
                  Nodo_binario hoja = new Nodo_binario(a);
                  hoja.setHoja(true);
                  hoja.setAnulable(false);
                  RESULT = hoja;
                  :};

egunda::= segunda evaluacion|
                  evaluacion;

valuacion::= IDENTIFICADOR DOS_PUNTOS CADENA PUNTO_COMA;
```

```
package Analizadores;

import ...

/** CUP v0.11b 20160615 (GIT 4ac7450) generated parser.
 */
5 usages  Roberto-Gomez1
@SuppressWarnings({"rawtypes"})
public class parser extends java_cup.runtime.lr_parser {

 no usages  Roberto-Gomez1
public final Class getSymbolContainer() { return sym.class; }

  /** Default constructor. */
  1 usage  Roberto-Gomez1
  @Deprecated
  public parser() {super();}

  /** Constructor which sets the default scanner. */
  2 usages  Roberto-Gomez1
  @Deprecated
  public parser(java_cup.runtime.Scanner s) {super(s);}

  /** Constructor which sets the default scanner. */
  1 usage  Roberto-Gomez1
  public parser(java_cup.runtime.Scanner s, java_cup.runtime.SymbolFactory sf) {super(s,sf);}
```

Finalmente esta la clase “Automata” que es la que se encarga de la logica para la creacion del metodo de arbol, tablas y automatas. A continuacion las variables utilizadas:

```
private Nodo_binario arbol_expresion;
21 usages
private ArrayList<Trans> transiciones = new ArrayList<>();
18 usages
private ArrayList<Tabla> estados = new ArrayList<>();
35 usages
private ArrayList<Tabla> aux_tabla = new ArrayList<>();
14 usages
private ArrayList<Tabla> aux_nombre = new ArrayList<>();

7 usages
private int conteo = 1;
6 usages
private int contador_arbol,contador_siguientes,contador_transiciones,contador_afd,contador_afnd,contador_html =1;
7 usages
private int contador = 1;
23 usages
private String aa = "";
```

Al principio se puede apreciar como hacemos la asignacion del estado de aceptacion al arbol creado por crear, tambien se aprecia que se asigna si es anulable.

```
public Automata(Nodo_binario arbol_expresion) {
    Nodo_binario primero = new Nodo_binario( dato: ".");
    Nodo_binario hash = new Nodo_binario( dato: "#");
    hash.setHoja(true);
    hash.setAnulable(false);
    primero.setHijo_derecho(hash);
    primero.setHijo_izquierdo(arbol_expresion);
    this.arbol_expresion = primero;
    asignacion(this.arbol_expresion);
    conteo=0;
    metodo(this.arbol_expresion);
    aa= "";
    String cadena= "digraph G {\n"+crear_arbol(this.arbol_expresion,conteo)+"}";
    generar(cadena);
    this.estados.add(new Tabla(this.arbol_expresion.getPrimeros(), lexema: "S0",this.arbol_expresion.getUltimos()));
    tabla_siguientes();
    contador=0;
    aa="";
    tabla_trans();
    contador=0;
    aa="";
    afd();
    ingresar_error();
}
```

Luego se empieza hacer la asignacion de la cabecera y siguientes, para esto se utiliza el metodo de asignacion y metodo (metodos recursivos)

```java
public void asignacion(Nodo_binario aux){
    if (aux == null){
        return;
    }
    if (aux.isHoja()){
        aux.setCabecera(conteo);
        conteo++;
        return;
    }
    asignacion(aux.getHijo_izquierdo());
    asignacion(aux.getHijo_derecho());
}
```

```java
public void metodo (Nodo_binario aux){
    if (aux ==null){
        return;
    }

    if (aux.isHoja()){
        aux.getPrimeros().add(aux.getCabecera());
        aux.getUltimos().add(aux.getCabecera());
        return;
    }

    metodo(aux.getHijo_izquierdo());
    metodo(aux.getHijo_derecho());
    if (aux.getDato().equals("*")){
        aux.setAnulable(true);
        aux.getPrimeros().addAll(aux.getHijo_izquierdo().getPrimeros());
        aux.getUltimos().addAll(aux.getHijo_izquierdo().getUltimos());

    }else if (aux.getDato().equals("+")){
        aux.setAnulable(aux.getHijo_izquierdo().isAnulable());
        aux.getPrimeros().addAll(aux.getHijo_izquierdo().getPrimeros());
        aux.getUltimos().addAll(aux.getHijo_izquierdo().getUltimos());
    }else if (aux.getDato().equals("?")){
        aux.setAnulable(true);
        aux.getPrimeros().addAll(aux.getHijo_izquierdo().getPrimeros());
        aux.getUltimos().addAll(aux.getHijo_izquierdo().getUltimos());
    }else if (aux.getDato().equals("|")){
        aux.setAnulable(aux.getHijo_izquierdo().isAnulable() || aux.getHijo_derecho().isAnulable());
        aux.getPrimeros().addAll(aux.getHijo_izquierdo().getPrimeros());
        aux.getPrimeros().addAll(aux.getHijo_derecho().getPrimeros());
        aux.getUltimos().addAll(aux.getHijo_izquierdo().getUltimos());
```

```java
    }else if (aux.getDato().equals("|")){
        aux.setAnulable(aux.getHijo_izquierdo().isAnulable() || aux.getHijo_derecho().isAnulable());
        aux.getPrimeros().addAll(aux.getHijo_izquierdo().getPrimeros());
        aux.getPrimeros().addAll(aux.getHijo_derecho().getPrimeros());
        aux.getUltimos().addAll(aux.getHijo_izquierdo().getUltimos());
        aux.getUltimos().addAll(aux.getHijo_derecho().getUltimos());
    }else if (aux.getDato().equals(".")){
        aux.setAnulable(aux.getHijo_izquierdo().isAnulable() && aux.getHijo_derecho().isAnulable());
        if (aux.getHijo_izquierdo().isAnulable()){
            aux.getPrimeros().addAll(aux.getHijo_izquierdo().getPrimeros());
            aux.getPrimeros().addAll(aux.getHijo_derecho().getPrimeros());
        }else{
            aux.getPrimeros().addAll(aux.getHijo_izquierdo().getPrimeros());
        }
        if (aux.getHijo_derecho().isAnulable()){
            aux.getUltimos().addAll(aux.getHijo_izquierdo().getUltimos());
            aux.getUltimos().addAll(aux.getHijo_derecho().getUltimos());
        }else{
            aux.getUltimos().addAll(aux.getHijo_derecho().getUltimos());
        }
    }
```

Posterior se hace la creacion del arbol en graphviz, para esto se utiliza el metodo crear_arbol y generar() (metodos recursivos)

```
public String crear_arbol(Nodo_binario actual, int padre){
    String graficar = "";
    conteo += 1;
    int aux = conteo;
    if (actual == null){
        conteo -=1;
        return graficar;
    }
    if(actual.isHoja()){
        graficar+= "N_"+aux+ "[shape = none label =<\n"
                + "<TABLE border=\"1\" cellspacing=\"2\" cellpadding=\"10\" > \n
                + "<TR>\n"
                + "<TD colspan=\"3\">"+actual.isAnulable()+"</TD>\n"
                + "</TR>\n"
                + "<TR>\n"
                + "<TD>"+actual.getPrimeros()+"</TD>\n"
                + "<TD >"+actual.getDato()+"</TD>\n"
                + "<TD>"+actual.getUltimos()+"</TD>\n"
                + "</TR>\n"
                + "<TR>\n"
                + "<TD colspan=\"3\">"+actual.getCabecera()+"</TD>\n"
                + "</TR>\n"
                + "</TABLE>>];";
        int num = actual.getCabecera();
        ArrayList<Integer> lista = new ArrayList<>();
        lista.add(num);
        Tabla nuevo = new Tabla(lista,actual.getDato(),actual.getUltimos());
        this.aux_nombre.add(nuevo);
    }else{
        graficar+= "N_"+aux+"[shape = none label=<\n"
```

```
        graficar+= "N_"+aux+"[shape = none label=<\n"                                    ⚠ 52
                + "<TABLE border=\"1\" cellspacing=\"2\" cellpadding=\"10\" > \n"
                + "<TR>\n"
                + "<TD colspan=\"3\">"+actual.isAnulable()+"</TD>\n"
                + "</TR>\n"
                + "<TR>\n"
                + "<TD >"+actual.getPrimeros()+"</TD>\n"
                + "<TD >" + actual.getDato() +"</TD>\n"
                + "<TD >"+actual.getUltimos()+"</TD>\n"
                + "</TR>\n"
                + "</TABLE>>];";
        if(actual.getDato().equals(".")){
            Tabla nuevo = new Tabla(actual.getHijo_izquierdo().getUltimos(),actual.getDato(),actual.getHijo_derecho().getPrimeros());
            this.aux_tabla.add(nuevo);

        }else if (actual.getDato().equals("+") || actual.getDato().equals("*")){
            Tabla nuevo = new Tabla(actual.getHijo_izquierdo().getUltimos(),actual.getDato(),actual.getHijo_izquierdo().getPrimeros());
            this.aux_tabla.add(nuevo);
        }
    }

    if (padre != 0 ){
        graficar+="N_"+padre+ " -> N_" + aux +";\n";
    }
    graficar+= crear_arbol(actual.getHijo_izquierdo(), aux);
    graficar+= crear_arbol(actual.getHijo_derecho(),aux);

    return graficar;
}
```

Meotod de generar()

Para finalizar los ultimos metodos son tabla_siguientes(), tabla_trans(), afd(), ingresar_error(). Estos metodos como su nombre lo indica es para la creacion de cada uno, en caso de “ingresar_error” es para el reporte de errores lexicos

```
public void tabla_siguientes (){
    //this.estados.add(new Tabla(this.aux_tabla.get(0).getNumero(),"S1",this.aux_tabla.get(0).getSiguiente()));
    //disyuncion de primeros
    for (int i=0;i<this.aux_tabla.size();i++){
        Tabla hola = this.aux_tabla.get(i);
        ArrayList<Integer> numero = hola.getNumero();

        if (numero.size() >1){
            this.aux_tabla.remove(i);
            for(Integer num: numero){
                ArrayList<Integer> lista = new ArrayList<~>();
                lista.add(num);
                this.aux_tabla.add(i, new Tabla(lista,hola.getLexema(),hola.getSiguiente()));
            }
        }
    }
    //concatenacion de siugientes
    for (int i=0; i<this.aux_tabla.size();i++){
        ArrayList<Integer> numero = this.aux_tabla.get(i).getNumero();
        for(int j=i+1; j<this.aux_tabla.size();j++){
            Tabla hola = this.aux_tabla.get(j);
            if (numero.equals(hola.getNumero())){
                this.aux_tabla.remove(j);
                this.aux_tabla.get(i).getSiguiente().addAll(hola.getNumero());

            }
        }
    }
```

```
    //ordenamiento de siguientes
    for (int i=0;i<this.aux_nombre.size();i++){
        ArrayList<Integer> numero = this.aux_nombre.get(i).getNumero();
        for (int j=0;j<this.aux_tabla.size();j++){
            if (numero.equals(this.aux_tabla.get(j).getNumero())){
                Tabla hola = this.aux_tabla.get(j);
                this.aux_tabla.remove(j);
                Integer [] arreglo = hola.getSiguiente().toArray(new Integer[0]);
                Arrays.sort(arreglo);
                ArrayList<Integer> auxilio = new ArrayList<~>(Arrays.asList(arreglo));
                this.aux_tabla.add(j, new Tabla(hola.getNumero(),this.aux_nombre.get(i).getLexema(),auxilio));
            }
        }
    }

    //ordenamiento
    for (int i =0;i<this.aux_tabla.size()-1;i++){
        for (int j =0;j<this.aux_tabla.size()-i-1;j++){
            Tabla hola = this.aux_tabla.get(j);
            ArrayList<Integer> numero = hola.getNumero();
            int mmm = numero.get(0);
            Tabla hola1 = this.aux_tabla.get(j+1);
            ArrayList<Integer> numero1 = hola1.getNumero();
            int mmm1 = numero1.get(0);
            if(mmm>mmm1){
                this.aux_tabla.remove(j);
                this.aux_tabla.add(j, hola1);
                this.aux_tabla.remove( index: j+1);
                this.aux_tabla.add( index: j+1,hola);
            }
        }
```

```
    for (int i = 0; i < this.aux_tabla.size(); i++) {
        Tabla numero = this.aux_tabla.get(i);
        boolean duplicado = false;
        for (int j = i + 1; j < this.aux_tabla.size(); j++) {
            Tabla hola = this.aux_tabla.get(j);
            if (numero.getSiguiente().equals(hola.getSiguiente())) {
                duplicado = true;
                break;
            }
        }
        if (!duplicado) {
            String mm = "S" + contador;
            this.estados.add(new Tabla(numero.getNumero(), mm, numero.getSiguiente()));
            contador++;
        }
    }
    contador=1;



    ArrayList<Integer> le = new ArrayList<>();
    this.aux_tabla.add(new Tabla(this.aux_nombre.get(this.aux_nombre.size()-1).getNumero(),tH

    aa +=  "digraph G{\n"
            + "graph [ratio=fill];\n"
            + "node [label=\"\\N\", fontsize=15, shape=plaintext];\n"
            + "graph [bb=\"0,0,352,154\"];\n"
            + "arset [label=<\n"
            + "<TABLE ALIGN=\"LEFT\">\n"
```

```
aa +=  "digraph G{\n"
        + "graph [ratio=fill];\n"
        + "node [label=\"\\N\", fontsize=15, shape=plaintext];\n"
        + "graph [bb=\"0,0,352,154\"];\n"
        + "arset [label=<\n"
        + "<TABLE ALIGN=\"LEFT\">\n"
        + "<TR><TD colspan=\"2\"  bgcolor=\"lemonchiffon4\">Hoja</TD>\n"
        + "<TD bgcolor=\"lemonchiffon4\">Siguientes</TD></TR>\n"
        + "<TR><TD> Numero </TD>\n"
        + "<TD> Lexema</TD>\n"
        + "<TD> Siguiente </TD></TR>\n";
for (Tabla este : this.aux_tabla){
    aa += "<TR><TD>"+contador+"</TD>\n"
            + "<TD>"+este.getLexema()+"</TD>\n"
            + "<TD>"+este.getSiguiente()+"</TD></TR>\n";
    contador++;
}
aa+="</TABLE>>];\n}";

FileWriter fichero = null;
try {
    File directory = new File( pathname: "Imagenes\\SIGUIENTES_202000544");
    if (!directory.exists()) {
        directory.mkdirs();
    }
    File file = new File( pathname: "Imagenes\\SIGUIENTES_202000544\\TablaSiguientes" + contador_siguientes + ".dot");
    while (file.exists()) {
        contador_siguientes++;
        file = new File( pathname: "Imagenes\\SIGUIENTES_202000544\\TablaSiguientes" + contador_siguientes + ".dot");
    }
```

Omitire la parte de la creacion de imágenes para los proximos metodos ya que es lo mismo para todos.

```java
public void tabla_trans() {

    for (int i = 0; i < this.estados.size(); i++) {
        Tabla hola = this.estados.get(i);
        ArrayList<Integer> numero = (i == 0) ? hola.getNumero() : hola.getSiguiente();
        if (numero.size() > 1) {
            for (Integer num : numero) {
                ArrayList<Integer> lista = new ArrayList<>();
                lista.add(num);
                this.transiciones.add(new Trans(lista, hola.getLexema(),  lexema: "", lista,  estado_final: ""));
            }
        } else {
            ArrayList<Integer> lista = new ArrayList<>();
            lista.add(numero.get(0));
            Trans nose = new Trans(lista, hola.getLexema(),  lexema: "", lista,  estado_final: "");
            boolean addTransition = true;
            for (Trans t : this.transiciones) {
                if (t.getInicial().equals(nose.getInicial()) && t.getSiguientes().equals(nose.getSiguientes())) {
                    addTransition = false;
                    break;
                }
            }
            if (addTransition) {
                this.transiciones.add(nose);
            }
        }
```

```java
    for (int i = 0; i < this.transiciones.size(); i++) {
        Trans numero = this.transiciones.get(i);
        for (int j = 0; j < this.aux_nombre.size(); j++) {
            Tabla hola = this.aux_nombre.get(j);
            for (int k = 0; k < this.aux_tabla.size(); k++) {
                Tabla queso = this.aux_tabla.get(k);
                if (numero.getInicial().equals(hola.getNumero()) && numero.getInicial().equals(queso.getNumero())) {
                    this.transiciones.remove(i);
                    this.transiciones.add(i, new Trans(numero.getInicial(), numero.getEstado_inicial(), hola.getLexema(), queso.getSiguiente(), nu
                }
            }
        }
    }
    for (int i = 0; i < this.transiciones.size(); i++) {
        Trans numero = this.transiciones.get(i);
        for (int j = 0; j < this.estados.size(); j++) {
            Tabla hola = this.estados.get(j);
            if (numero.getSiguientes().equals(hola.getSiguiente())) {
                this.transiciones.remove(i);
                this.transiciones.add(i, new Trans(numero.getInicial(), numero.getEstado_inicial(), numero.getLexema(), numero.getSiguientes(), hu
            }
        }
    }
```

```java
aa += "digraph G{\n" +
        "graph [ratio=fill];\n" +
        "node [label=\"\\N\", fontsize=15, shape=plaintext];\n" +
        "graph [bb=\"0,0,352,154\"];\n" +
        "arset [label=<\n" +
        "<TABLE ALIGN=\"LEFT\">\n" +
        "<TR><TD bgcolor=\"lemonchiffon4\">Estado</TD>\n";
for (int i = 0; i < this.aux_tabla.size(); i++) {
    aa += "<TD bgcolor =\"lemonchiffon4\"> " + this.aux_tabla.get(i).getLexema() + "</TD>\n";
}
aa += "</TR>\n";
int total = this.estados.size()+this.aux_nombre.size();
int temporal=0;
for (int i = 0; i < this.estados.size(); i++) {
int temp= temporal;
if(i==0) {
    aa += "<TR><TD bgcolor=\"lemonchiffon4\">" + this.estados.get(i).getLexema() + " " + this.estados.get(i).getNumero() + "</TD>\n";
}else{
    aa += "<TR><TD bgcolor=\"lemonchiffon4\">" + this.estados.get(i).getLexema() + " " + this.estados.get(i).getSiguiente() + "</TD>\n";
}
    for (int j = 0; j < this.aux_nombre.size(); j++) {
        if(this.estados.get(i).getLexema().equals(this.transiciones.get(temp).getEstado_inicial()) && this.transiciones.get(temp).getLexema()
            aa +="<TD>"+this.transiciones.get(temp).getEstado_final()+"</TD>\n";
            if(temp <=total-2) {
                temp++;
            }
            temporal=temp;
        } else if (!this.estados.get(i).getLexema().equals(this.transiciones.get(temp).getEstado_inicial()) || !this.transiciones.get(temp).g
            aa +="<TD>--</TD>\n";

            }
```

```java
public void afd() {
    aa += "digraph G {\nrankdir=LR;\n";
    for (int i = 0; i < this.estados.size(); i++) {
        if(i==this.estados.size()-1) {
            aa+=this.estados.get(i).getLexema()+"[shape=doublecircle];\n";
        }else{
            aa+=this.estados.get(i).getLexema()+"[shape=circle];\n";
        }
    }
    for(int i=0;i<this.transiciones.size();i++){
        if(!this.transiciones.get(i).getLexema().equals("#")){
            String label = this.transiciones.get(i).getLexema().replaceAll( regex: "\"",  replacement: "\\\\\"");
            aa+=this.transiciones.get(i).getEstado_inicial()+" -> "+this.transiciones.get(i).getEstado_final()+" [label=\""+label+"\"];\n";
        }
    }
    aa+="}";
```