Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Curso: Laboratorio Organización de Lenguajes & Compiladores 1
Auxiliar: José Diego Pérez Toralla
Auxiliar: Maynor Octavio Piló Tuy

Manual Técnico

Juan F. Urbina S. **2019060651**
Sección: **C**

Guatemala, Marzo de 2023

Índice

Introducción

Dentro del primer proyecto del laboratorio de Organización de Lenguajes & Compiladores 1 la aplicación realizada fue en base al uso de JFlex y Cup en el lenguaje de programación Java, abordando los conceptos de gramáticas y expresiones regulares para la creación de lo que es el análisis léxico y sintáctico para analizar por consiguiente las cadenas o el archivo de prueba respectivo, la utilización de listas enlazadas para el almacenamiento de los errores, & tokens que son almacenados para su utilización en los que son los árboles y tablas graficadas posteriormente; utlizando asimismo el modo gráfico para obtener aparte de los archivos, generar, graficas, nuevos documentos y observar las imagenes de las expresiones regulares aceptadas.

Se empleo el lenguaje de programación Java como herramienta en la función de las estructuras y analisis, para almacenar la información que la interfaz gráfica le enviaba, y la interfaz gráfica se utilizaba para la obtención y envío de información a las estructuras/actualización de tokens y realización gráficas; se utilizó Graphviz para graficar las estructuras de las tablas0 y árboles.

Se pudo determinar que la implementación de las estructuras y el almacenamiento y/o arreglo de la información obtenida en cada una de ellas no mostro ningún problema durante su ejecución.

Requisitos del Sistema

- ***Sistema Operativo:*** Windows 7 o superior
- ***CPU:*** Intel Pentium D o AMD Athlon 64 (K8) 2.6GHz. (Requisitos Mínimo)
- ***RAM:*** 600MB
- ***Lenguaje Utilizado:*** Java
- ***IDE:*** NetBeans 16
- ***JDK:*** 19.0 (Open JDK)
- ***Versión Java:*** 19.0

Explicación del Código

Análisis Léxico

Utiliza las siguientes expresiones regulares para obtener los parámetros a la hora de leer el archivo, obteniendo los errores léxicos que tenga el archivo.

```
//esppacios
espacios_muchos = [ \t\r\n\f]+

//letras & digitos
letra = [a-zA-Z]
digito = [0-9]
numero = {digito}+
s_flecha = -(\s)*>

//comentarios
comentario_simple = "//" [^"\n"]*
comentario_multiple = "<!" [^"!>"]* "!>"

//Pal Proyecto
llave_abierta = "{"
llave_cerrada = "}"
dos_puntos = ":"
punto_coma = ";"

s_porcentaje = "%%"
virgilla         = "~"
coma     = ","
punto = "."
or = "|"
asterisco = "*"
s_mas = "+"
s_interrogacion = "?"
fin_linea = "\\n"
s_comilla = "\\\'"
doble_comilla = "\\\""
range = [!-/] | [:-@] | [\[-`] | [\{-\}]
espacio = "\" \""
conj_sym = ["c"|"C"]["o"|"O"]["n"|"N"]["j"|"J"]
ident = {letra}({letra}|{digito}|"_")*
string_dat = [^"\\\""]* "\\\""
s_frase = "\"" {string_dat}* [^"\""]*  {string_dat}* "\""
```

Análisis Sintáctico

Se encarga de obtener los errores sintácticos y mediante una gramática regular cada en cada producción obtener los datos requeridos.

Se analizarán las cadenas y separarán los tokens para verificarlos posteriormente.

```
/*------------------------------3. Terminales------------------------------*/

terminal numero;
terminal llave_abierta,llave_cerrada;
terminal dos_puntos,punto_coma,s_flecha;
terminal s_porcentaje,virgilla,coma,punto;
terminal or,asterisco,s_mas,s_interrogacion;
terminal fin_linea,s_comilla,doble_comilla;
terminal range,espacio,conj_sym,ident,s_frase,letra;

/*------------------------------4. No terminales------------------------------*/

non terminal INICIO;
non terminal STARTS;
non terminal CONTENIDO;
non terminal CONJUNTO;
non terminal CONTENIDOR;
non terminal ER;
non terminal DEFCONJ;
non terminal SEPCOMAS;
non terminal RANGO;
non terminal DATOSEP;
non terminal SEPCOMASR;
non terminal DATORANGO;
non terminal DEFER;
non terminal OP;
non terminal REFCONJ;
non terminal CADENAS;
non terminal CADENASR;
non terminal NAMECOBJ;


start with INICIO;
```

```
/*----------------------------6. Producciones----------------------------*/

INICIO::= STARTS                {: /**/ :}
;

/*{ contenido*/
STARTS::= llave_abierta CONTENIDO           {: /**/ :}
;

/**/
CONTENIDO::= CONJUNTO CONTENIDOR            {: /*CONJ contenido*/ :}
    | ER CONTENIDOR                         {: /*EXPREG contenido*/ :}
;

/*conj: nombrecontenido -> definicion*/
CONJUNTO ::= conj_sym dos_puntos NAMECOBJ:a s_flecha DEFCONJ     {:
                                                                    TempConjName =(String) a;
                                                                    ConjTemp = new Conjunto(TempConjType,(String) a,TempConjVar1,TempConjVar2,TempConjText);
                                                                    Lista_Conjunto.add(ConjTemp);
                                                                    TempConjText = "";
                                                                    TempConjVar1="";
                                                                    TempConjVar2="";
                                                                :}
;
```

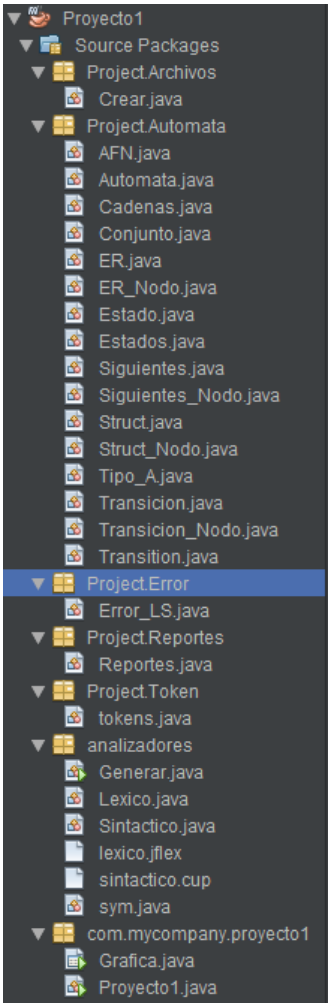
```
/*a,b,c,d a,c,d o 1~9*/
DEFCONJ ::= SEPCOMAS    {: TempConjType="comas"; :}
    | RANGO             {: TempConjType="rango"; :}
;

/*a,c,d*/
SEPCOMAS ::= DATOSEP:a SEPCOMASR    {: TempConjText += a + ","; :}
;

DATOSEP ::= numero:a        {: RESULT = a; :}
    | letra:a               {: RESULT = a; :}
    | range:a               {: RESULT = a; :}
    | asterisco:a           {: RESULT = a; :}
    | s_mas:a               {: RESULT = a; :}
    | coma:a                {: RESULT = a; :}
    | punto:a               {: RESULT = a; :}
    | dos_puntos:a          {: RESULT = a; :}
    | punto_coma:a          {: RESULT = a; :}
    | s_interrogacion:a     {: RESULT = a; :}
    | llave_abierta:a       {: RESULT = a; :}
    | or:a                  {: RESULT = a; :}
    | llave_cerrada:a       {: RESULT = a; :}
    | doble_comilla:a       {: RESULT = a; :}
    | s_comilla:a           {: RESULT = a; :}
    | fin_linea:a           {: RESULT = a; :}
    | s_frase:a             {: RESULT = a; :}
;

SEPCOMASR ::= coma SEPCOMAS         {: /**/ :}
    | punto_coma                    {: /**/ :}
;

RANGO ::= DATORANGO:a virgilla DATORANGO:b punto_coma   {:
                                                            TempConjVar1 =(String) a;
                                                            TempConjVar2 =(String) b;
                                                        :}
;
```

```
DATORANGO ::= numero:a          {: RESULT = a; :}
    | letra:a                   {: RESULT = a; :}
    | range:a                   {: RESULT = a; :}
    | espacio:a                 {: RESULT = a; :}
    | asterisco:a               {: RESULT = a; :}
    | s_mas:a                   {: RESULT = a; :}
    | coma:a                    {: RESULT = a; :}
    | punto:a                   {: RESULT = a; :}
    | dos_puntos:a              {: RESULT = a; :}
    | punto_coma:a              {: RESULT = a; :}
    | s_interrogacion:a         {: RESULT = a; :}
    | llave_abierta:a           {: RESULT = a; :}
    | or:a                      {: RESULT = a; :}
    | llave_cerrada:a           {: RESULT = a; :}
    | doble_comilla:a           {: RESULT = a; :}
    | s_comilla:a               {: RESULT = a; :}
    | fin_linea:a               {: RESULT = a; :}
;

/*NombreExpresionRegular -> contenidoER*/
ER ::= ident:a s_flecha DEFER       {:
                                        temp.name = (String) a;
                                        Lista_ER.add(temp);
                                    :}
;

DEFER ::= OP:a                          {: temp.insert((String)a,"OP",false); :}
    | REFCONJ:a                         {: temp.insert((String)a,"REFCONJ",true); :}
    | s_frase:a DEFER                   {: temp.insert((String)a,"s_frase",true); :}
    | espacio:a DEFER                   {: temp.insert((String)a,"espacio",true); :}
    | fin_linea:a DEFER                 {: temp.insert((String)a,"fin_linea",true); :}
    | s_comilla:a DEFER                 {: temp.insert((String)a,"s_comilla",true); :}
    | doble_comilla:a DEFER             {: temp.insert((String)a,"doble_comilla",true); :}
    | punto_coma:a                      {: temp = new ER(); :}
;

OP ::= or:a DEFER                   {: RESULT = a; :}
    | asterisco:a DEFER             {: RESULT = a; :}
    | s_mas:a DEFER                 {: RESULT = a; :}
    | s_interrogacion:a DEFER       {: RESULT = a; :}
    | punto:a DEFER                 {: RESULT = a; :}
;
```

```
REFCONJ ::= llave_abierta NAMECOBJ:a llave_cerrada DEFER     {: RESULT = a; :}
;

NAMECOBJ::= ident:a         {: RESULT = a; :}
          | letra:a         {: RESULT = a; :}
;

CONTENIDOR ::= s_porcentaje CADENASR    {: /**/ :}
    | CONTENIDO                         {: /**/ :}
;

CADENAS ::= s_porcentaje CADENASR
    |ident:a dos_puntos s_frase:b punto_coma CADENASR       {:
                                                               String temptext = (String) b;
                                                               temptext = temptext.substring(0,0) + temptext.substring(0+1);
                                                               temptext = temptext.substring(0,temptext.length()-1);
                                                               temptext = temptext.replace("\\\"","\"");
                                                               CadenasTemp = new Cadenas((String) a , temptext);
                                                               Lista_Cadena.add(CadenasTemp);
                                                           :}
;

CADENASR ::= llave_cerrada      {: /**/ :}
     | CADENAS                  {: /**/ :}
;
```

Código Java

Obtendrá los tokens ingresados, formará los árboles y tablas y analizará o mostrará si la cadena es válida o no es válida.

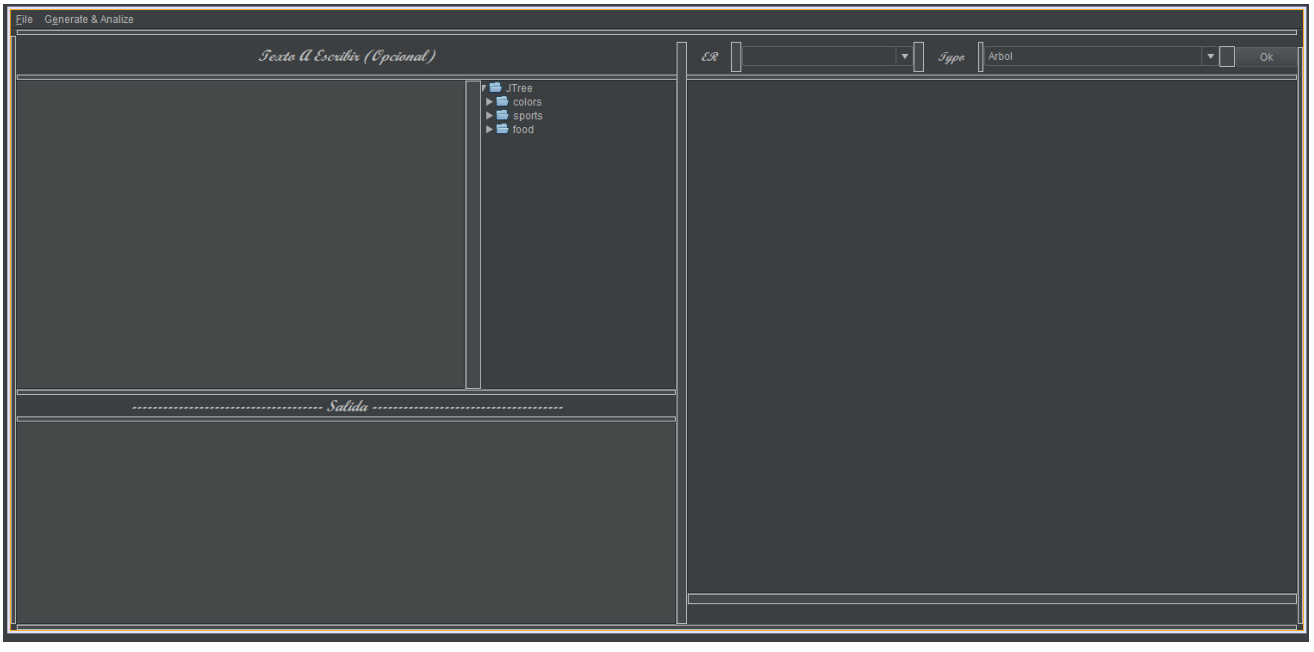


Por tener varias clases en java se dará un recorrido breve por cada uno:

- Las clases que terminen con "nodo" son los nodos que utilizarán las principales para la formación de las tablas y los árboles.
- Las clases AFN, Conjunto, Siguientes, Struct & ER se encargan de recopilar la información y hacer los grafos que obtendrá de las otras clases respectivas.
- La clase Reportes se encarga de hacer el archivo HTML de los errores.
- La clase Proyecto1 solo corre la clase Grafica, que es la que tiene el modo gráfico.

```java
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 */
package com.mycompany.proyecto1;
/**
 *
 * @author Pacos
 */
public class Proyecto1 {

    public static void main(String[] args) {
        //System.out.println("Hello World!");
        //ProyectoGrafica graph = new ProyectoGrafica();
        Grafica graf = new Grafica();
        graf.setVisible( b:true);
        graf.setLocationRelativeTo( c:null);
    }
}
```

Gráfica del Proyecto



Función al Abrir el programa

- Analiza las carpetas de reportes para ver si tiene o no tiene archivos creados, en caso de tener más de algún archivo en la sección de reportes, agrega los archivos de imagen de las expresiones regulares creadas.

```
private void formWindowOpened(java.awt.event.WindowEvent evt) {
    // TODO add your handling code here:
    boolean tiene_archivos = false;

    String path = "src\\Reportes";
    //String per = "hola.png";
    //String[]fs = per.split("[\".\"]");
    //System.out.println(fs[0]);
    //String[] as = path.split("\\\\");
    //System.out.println(as[0]);
    String pathAFD = path + "\\AFD_201906051\\";
    File carpeta = new File( pathname:pathAFD);
    File[] lista = carpeta.listFiles();
    int cuenta = 0;
    //System.out.println(lista.length);
    String [] its;
    List<String> ist = new ArrayList<>();
    for(int i = 0; i < lista.length; i++){
        if(lista[i].getName().endsWith( suffix:".png")) {
            String [] temp = lista[i].getName().split( regex:"[\".\"]");
            //System.out.println(lista[i].getName().split(".png"));
            ist.add(temp[0]);
            this.tiene_archivos = true;
            cuenta++;
        }
    }
    for(String sd : ist){ /*System.out.println(sd);*/ this.ER_Main_List.add( e:sd); }
    this.ER_Menu.setModel(new DefaultComboBoxModel( items:this.ER_Main_List.toArray()));
    if(this.tiene_archivos){
        this.AutomataCreado = true;
    }
```

Funciones Dentro de File

- En el ítem "nuevo", limpia la ruta, la consola, y el texto para que uno ingrese un nuevo archivo para analizar, no se puede analizar el archivo sin antes usar la opción de "Guardar Como".

```
private void newMenuItemActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    this.Consola.setText( t:"");
    this.analizado = false;
    this.AutomataCreado = false;
    this.generado = false;
    this.ruta.setText( text:"");
    this.Texto_Ing.setText( t:"");
}
```

- En el Item "Open" abre la carpeta de "Archivos" para buscar archivos con extensión ".olc" mostrando en la pantalla de texto todo lo que tiene el mismo archivo.

```
private void openMenuItemActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    this.fc.setFileFilter(filter);
    this.fc.setCurrentDirectory(new File( pathname:"src\\Archivos\\"));
    this.fc.addChoosableFileFilter( filter:this.filter);
    if(this.fc.showOpenDialog( parent:this) == JFileChooser.APPROVE_OPTION){
        try{
            Texto_Ing.setText( t:"");
            List<String> ErTemp = new ArrayList<>();
            String text = readUnicodeClassic( fileName:fc.getSelectedFile().toString());
            Texto_Ing.setText( t:text);
            ruta.setText( text:fc.getSelectedFile().toString());
            this.analizado = false;
            this.generado = false;
            this.AutomataCreado = false;
        }catch(Exception e){}
    }

}
```

- En el ítem "Save" En el caso de abrir un archivo o haberlo guardado como, busca el archivo y le guarda los nuevos cambios que este tendrá.

```
private void saveMenuItemActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    if(this.ruta.getText() == ""){
        System.out.println( x:"Error: Debe guardar el archivo");
        this.Consola.setText( t:"Error: Debe guardar el archivo");
    }else{
        this.Consola.setText( t:" ");
        try(FileWriter fw = new FileWriter( fileName:this.ruta.getText())){
            this.analizado = false;
            this.AutomataCreado = false;
            this.generado = false;
            fw.write( str:this.Texto_Ing.getText());
            this.Consola.setText( t:"Guardado");
        }catch(Exception e) { }
    }
}
```

- En el ítem "Save As" Guarda el archivo del área de texto como nuevo archivo con extensión ".olc".

```
private void saveAsMenuItemActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    this.fc.setFileFilter(filter);
    this.fc.setCurrentDirectory(new File( pathname:"src\\Archivos\\"));
    this.fc.addChoosableFileFilter( filter:this.filter);
    if(fc.showSaveDialog( parent:this) == JFileChooser.APPROVE_OPTION){
        try {
            Consola.setText( t:"");
            List<String> ErTemp = new ArrayList<>();
            System.out.println("Archivo: " + fc.getSelectedFile());
            File fichero = fc.getSelectedFile();
            try(FileWriter fw = new FileWriter( file:fichero)){
                fw.write( str:this.Texto_Ing.getText());
                Consola.setText("Archivo guardado: " + fc.getSelectedFile());
                this.analizado = false;
                this.AutomataCreado = false;
                this.generado = false;
                this.ruta.setText( text:fc.getSelectedFile().toString());
            }catch(Exception e) { }
        }catch(Exception e){ }
    }
}
```

- En el Item "Exit" Sale del programa.

```
private void exitMenuItemActionPerformed(java.awt.event.ActionEvent evt) {
    System.exit( status:0);
}
```

Funciones Dentro Generate & Analize

- En el ítem "Analizar" con el archivo abierto, analiza línea a línea, guardando los tokens y obteniendo posteriormente las expresiones regulares, también analiza que no haya ningún error en el archivo.

```
private void analizeMenuItemActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    if(this.ruta.getText() != ""){
        try{
            Reportes reportes = new Reportes();
            Boolean Errores = false;
            lexico = new Lexico(new BufferedReader(new FileReader( fileName:this.ruta.getText())));
            sintactico = new Sintactico( s:lexico);
            sintactico.parse();
            try{
                System.out.println( x:"<>---<>---<> ERRORES <>---<>---<>");
                if(lexico.errores.size() > 0 || sintactico.errores.size() > 0){
                    if(lexico.errores.size() > 0){
                        System.out.println( x:"------> ERRORES LEXICOS <------");
                        this.Consola.setText( t:"Error Lexico encontrado... agregando a la tabla de errores");
                        for(Error_LS ertab : lexico.errores){ System.out.println( x:ertab.show()); }
                    }
                    if(sintactico.errores.size() > 0){
                        System.out.println( x:"------> ERRORES SINTACTICOS <------");
                        this.Consola.setText( t:"Error Sintactico encontrado... agregando a la tabla de errores");
                        for(Error_LS ertab : sintactico.errores) { System.out.println( x:ertab.show()); }
                    }

                    Errores = true;
                    this.analizado = false;
                    this.AutomataCreado = false;
                }else{ Errores = false; }
                //generar reportes
                System.out.println( x:"<>---<>---<>--- | ---<>---<>---<>");
                reportes.GenerarReporte( errores:lexico.errores, errores2:sintactico.errores);
                if(Errores == true){ reportes.GenerarReporte( errores:lexico.errores, errores2:sintactico.errores); }
                if(Errores == false){
                    System.out.println( x:"Expresiones Regulares: {");
                    for(ER er : sintactico.Lista_ER){ System.out.println("\t" + er.name); er.GestionArbol(); er.GenerarHermano(); }
                    System.out.println( x:"}");

                    System.out.println( x:"------> Analidis completado <------");
                    this.analizado = true;
                    this.AgregarA_ER_Main( er:sintactico.Lista_ER);
                }
            }catch(Exception e){ this.Consola.setText( t:"Error con el Archivo de entrada: Agregando errores en tabla"); }
```

- En el ítem "Generar" una vez analizado el archivo genera todos los grafos de los árboles y tablas, también generando el archivo de salida JSON.

```
private void genreMenuItemActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    if(analizado){
        if(generado == false){
            System.out.println( x: "Generando Automatas {");
            for(ER er : sintactico.Lista_ER){ System.out.println("\t::- " + er.name + " -::"); er.GenraraAFND(); }
            System.out.println( x: "}");
            this.AutomataCreado = true;
            this.generado = true;
            this.Consola.setText( t: "Automatas Creados Correctamente");
        }else{ this.Consola.setText( t: "Automatas Existentes"); }
    }
    try{
        if(this.AutomataCreado){
            System.out.println( x: "------> Validando Cadenas <------");
            if(sintactico.Lista_Cadena.size() > 0){
                boolean encontrado = false;
                for(Cadenas valu : sintactico.Lista_Cadena){
                    encontrado = false;
                    for(ER vela : sintactico.Lista_ER){
                        if(valu.name.equals( anObject: vela.name)){
                            System.out.println("Cadena: " + valu.string + ", En la Expresion Regular: " + vela.name);
                            valu.Encontrado = true;
                            vela.ValidarCadena( cadena: valu.string,  conjList: sintactico.Lista_Conjunto,  i: valu);
                            encontrado = true;
                            break;
                        }
                    }
                    if(encontrado == false){ System.out.println("Expresion Regular no Encontrada: [" + valu.name + "] para la cadena [" + valu.string + "]"); }
                }
                String texto = "";
                JSONArray Main = new JSONArray();
                JSONObject sec;
                for(Cadenas i : sintactico.Lista_Cadena) {
                    sec = new JSONObject();
                    if(i.validacion){
                        texto += "La cadena: [" + i.string + "] es VALIDA para la Expresion Regular: [" + i.name + "]\n";
                        System.out.println("La cadena: [" + i.string + "] es VALIDA para la Expresion Regular: [" + i.name + "]");
                        //this.Consola.setText("La cadena: [" + i.string + "] es VALIDA para la Expresion Regular: [" + i.name + "]");
```

- En el ítem "Borrar archivos Directorios" Borra todos los archivos de imagen de los directorios de reportes.

```
private void deleteArchivesMenuItemActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    String direcA = "src\\Reportes\\ARBOLES_201906051\\";
    String direcS = "src\\Reportes\\SIGUIENTES_201906051\\";
    String direcT = "src\\Reportes\\TRANSICIONES_201906051\\";
    String direcAFD = "src\\Reportes\\AFD_201906051\\";
    String direcAFND = "src\\Reportes\\AFND_201906051\\";
    File path = new File( pathname: direcA);
    if(path.listFiles().length > 0){
        File a1 = new File( pathname: direcA);
        File a2 = new File( pathname: direcS);
        File a3 = new File( pathname: direcT);
        File a4 = new File( pathname: direcAFD);
        File a5 = new File( pathname: direcAFND);
        for(File fe : Objects.requireNonNull( obj: a1.listFiles())){ if(!fe.isDirectory()) { fe.delete(); } }
        for(File fe : Objects.requireNonNull( obj: a2.listFiles())){ if(!fe.isDirectory()) { fe.delete(); } }
        for(File fe : Objects.requireNonNull( obj: a3.listFiles())){ if(!fe.isDirectory()) { fe.delete(); } }
        for(File fe : Objects.requireNonNull( obj: a4.listFiles())){ if(!fe.isDirectory()) { fe.delete(); } }
        for(File fe : Objects.requireNonNull( obj: a5.listFiles())){ if(!fe.isDirectory()) { fe.delete(); } }
        this.actualizar();

        //List<String> lista = new ArrayList<>();
        this.ER_Main_List.clear();
        this.ER_Menu.setModel(new DefaultComboBoxModel( items: this.ER_Main_List.toArray()));
    }
}
```

- En el ítem "Abrir Directorio Archivos Prueba" Abre la carpeta donde se encuentran los archivos de prueba, para agregar nuevos.

```
private void openDirectoryMenuItemActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    String path = "src\\";
    String nombre = "Archivos";
    Path rute = Paths.get( first:path, more:nombre);
    if(Files.exists( path:rute)){
        try{ Desktop.getDesktop().open( file:rute.toFile()); }
        catch(Exception e){ }
    }

}
```

Funciones del Programa

Consta de 2 pestañas, 'File' y 'Generate & Analize', tambien tiene 2 combobox para la búsqueda de las imagenes y mostrarlas.

File

- Nuevo: Crea un nuevo archivo con extensión olc.
- Open: Abre un menú desplegable en la carpeta de archivos para abrir archivos de extensión olc.
- Save: Guarda el archivo con extensión olc.
- Save As: Guarda un nuevo archivo con extensión olc.
- Exit: Sale del programa.

Generate & Analize

- Analizar: Analiza el archivo abierto, en caso de haber errores los agrega a la tabla de errores.
- Generar: Genera los grafos de las expresiones regulares, el archivo de salida y muestra cuales cadenas son aceptadas y cuales cadenas no son aceptadas.
- Borrar archivos Directorios: Borra todos los archivos del directorio para poner nuevos.
- Abrir Directorio Archivos Prueba: Abre la carpeta en donde están los archivos de prueba, ya sea para agregar o para borrar archivos.

Conclusiones

- La utilización de expresiones regulares ayuda en la simplificación y obtención de mejor forma de expresiones que uno quiere obtener.
- El análisis sintáctico sirve para obtener reglas del funcionamiento del léxico que uno creó, para el análisis de un archivo.