Nueva Guatemala de la Asunción

# Manual de Técnico

## Proyecto 1: OLC-1

Elaborado por: Estuardo Gabriel Son Mux
Carné: 202003894
Fecha: 23/02/2022

# Índice

# Alcances y Objetivos del Programa

El programa realizado tiene como objetivo el análisis léxico y sintáctico de archivos EXP (extensión ".exp"), los cuales tienen la información para la creación de árboles y autómatas tanto deterministas con el método del árbol, como no determinista con el método de Thompson y graficarlos por medio de la librería Graphviz o tablas de HTML. Posteriormente al análisis se procede a realizar un análisis de las frases incluidas en el archivo EXP con una de las expresiones para verificar su validez.

# Especificaciones Técnicas

## Requisitos de Hardware

- Computadora con todos sus componentes para su correcto funcionamiento
- Laptop

## Requisitos de Software

### Sistema Operativo

El programa fue desarrollado en una laptop con sistema operativo Windows 10, Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz   1.80 GHz.

Especificaciones del dispositivo

| | |
|---|---|
| Nombre del dispositivo | DESKTOP-B6R45EM |
| Procesador | Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz   1.80 GHz |
| RAM instalada | 8.00 GB |
| Id. del dispositivo | 077D5E5A-52E8-4649-AED3-5A498B8678A7 |
| Id. del producto | 00325-80915-48720-AAOEM |
| Tipo de sistema | Sistema operativo de 64 bits, procesador x64 |
| Lápiz y entrada táctil | Compatibilidad con entrada táctil con 10 puntos táctiles |

### Lenguaje de Programación

JAVA

#### *Librerías*

- Html2image
- Java-cup-11a
- Java-cup-11b
- JFlex
- Json-simple

### IDE o Editor de Código

Apache Netbeans IDE 12.6

# Lógica del Programa

## Estructura de Archivos EXP

Los archivos EXP poseen la siguiente estructura.

```
{
////// CONJUNTOS

CONJ: letra -> a~z;
CONJ: digito -> 0~9;

/////// EXPRESIONES REGULARES

ExpReg1 -> . {letra} * | "_" | {letra} {digito};
ExpresionReg2 -> . {digito} . "." + {digito};
RegEx3 -> . {digito} * | "_" | {letra} {digito};

%%
%%

ExpReg1 : “primerLexemaCokoa”;
ExpresionReg2 : “34.44”;
}
```

Para ellos se hizo un reconocimiento de tokens con la siguiente tabla de tokens.

| Token | Lexema | Patrón |
|---|---|---|
| **LLAVE_APERTURA** | { | { |
| **LLAVE_CIERRE** | } | } |
| **DOS_PTS** | : | : |
| **PT_COMA** | ; | ; |
| **ASIGNACION** | -> | -> |
| **SEPARADOR** | %% | (\%)+ |
| **CONJ** | CONJ | CONJ |
| **IDENTIFICADOR** | Expresion1 | (_)*[a-zA-ZnÑ]+[_a-zA-Z0-9ñÑ]* |
| **CONJUNTO** | a,e,i,o,u | ([a-z]\~[a-z])\|([^a-zA-Z0-9\~]\~[^a-zA-Z0-9\~])\|([A-Z]\~[A-Z])\|([0-9]\~[0-9])\|([^\~](,[^\~])+) |
| **EXPRESION** | ..+{digito}"."+{digito} | [\.\|*\+\?]([a-zA-Z0-9ñÑ\|\?\+*\.]\|(\"((\\[^\n])\|([^\\\"\n]))+\")\|(\{(_)*[a-zA-ZnÑ]+[_a-zA-Z0-9ñÑ]*\}))+ |
| **ESPACIO** | | [\ \r\t\f\t] |
| **SALTO** | \n | [\ \n] |
| **FRASE** | "34.51" | \”(\\\”\|\\n\|\\\’\|[^\"])\” |

Para el análisis sintáctico se utilizó la siguiente gramática:

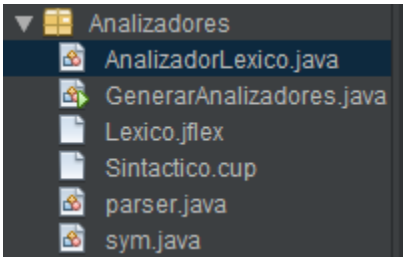
```
Inicio -> LLAVE_APERTURA CONJUNTOS EXPRESIONES SEPARADOR VERIFICACIONES LLAVE_CIERRE

CONJUNTOS-> CONJUNTOS CONJ DOS_PTS IDENTIFICADOR ASIGNACION CONJUNTO PT_COMA |
                        CONJ DOS_PTS IDENTIFICADOR ASIGNACION CONJUNTO PT_COMA


EXPRESIONES-> EXPRESIONES IDENTIFICADOR ASIGNACION EXPRESION PT_COMA |
                          IDENTIFICADOR ASIGNACION EXPRESION PT_COMA


VERIFICACIONES-> VERIFICACIONES IDENTIFICADOR ASIGNACION FRASE PT_COMA |
                                IDENTIFICADOR ASIGNACION FRASE PT_COMA
```

## Archivos Utilizados

Para la elaboración del proyecto en Java se separaron los archivos en 3 paquetes los cuales son:

- **Analizadores:** En este paquete se almacenan los archivos utilizados para la creación de los analizadores, tanto léxico como sintáctico.
- **Arboles:** En este paquete se almacenan todos los archivos utilizados para la creación de las estructuras para la realización del método del árbol y los autómatas finitos deterministas y no deterministas.
- **Ocl1_proyecto1:** En este paquete se encuentran los archivos para la creación de la ventana en donde se visualizan todos los componentes y la ejecución de las funciones.

## Package Analizadores




### *Analizadores.java*

En este archivo se encuentra la generación de los archivos AnalizadorLexico.java, parser.java y sym.java, esto mediante la utilización de las librerías JFlex y Cup que mediante la utilización de archivos de extensión .jflex y .cup genera dichos archivos para el análisis Léxico y Sintáctico respectivamente.

```java
package Analizadores;

public class GenerarAnalizadores {
    public static void main(String []args){
        try{
            String  opcFlex[] = {"src/Analizadores/Lexico.jflex","-d","src/Analizadores"};
            JFlex.Main.generate(opcFlex);
            String opcCup [] ={"-destdir","src/Analizadores","-parser","parser","src/Analizadores/Sintactico.cup"};
            java_cup.Main.main(opcCup);
        } catch(Exception e){
            e.printStackTrace();
        }
    }
}
```

### *AnalizadorLexico.java*

Archivo generado por medio de la librería JFlex para el análisis sintáctico de los archivos de entrada del programa.

```java
/* The following code was generated by JFlex 1.4.3 on 21/02/22 15:47 */

package Analizadores;

import ocll_proyectol.*;
import static ocll_proyectol.OCLl_Proyectol.ListaErrores;
import java_cup.runtime.Symbol;
import java.util.*;


/**
 * This class is a scanner generated by
 * <a href="http://www.jflex.de/">JFlex</a> 1.4.3
 * on 21/02/22 15:47 from the specification file
 * <tt>src/Analizadores/Lexico.jflex</tt>
 */
public class AnalizadorLexico implements java_cup.runtime.Scanner {

  /** This character denotes the end of file */
```

*Parte del código de AnalizadorLexico.java.*

```
//Expresiones
IDENTIFICADOR = (_)*[a-zA-ZnÑ]+[_a-zA-Z0-9ñÑ]*
CONJUNTO = ([a-z]\~[a-z])|([^a-zA-Z0-9\~]\~[^a-zA-Z0-9\~])|([A-Z]\~[A-Z])|([0-9]\~[0-9])|([^\~](,[^\~])+)
EXPRESION = [\.\|\*\+\?]([a-zA-Z0-9ñÑ\|\?\+\*\.]|(\"((\\[^\n])|([^\\\"\n]))+\")|(\{(_)*[a-zA-ZnÑ]+[_a-zA-Z0-9ñÑ]*\}))+
ESPACIO = [\ \r\t\f\t]
SALTO = [\ \n]

%%

<YYINITIAL> {CONJ} { return new Symbol(sym.CONJ, yyline, yycolumn, yytext());}

<YYINITIAL> {LLAVE_APERTURA} { return new Symbol(sym.LLAVE_APERTURA, yyline, yycolumn, yytext());}
<YYINITIAL> {LLAVE_CIERRE} { return new Symbol(sym.LLAVE_CIERRE, yyline, yycolumn, yytext());}
<YYINITIAL> {DOS_PTS} { return new Symbol(sym.DOS_PTS, yyline, yycolumn, yytext());}
<YYINITIAL> {PT_COMA} { return new Symbol(sym.PT_COMA, yyline, yycolumn, yytext());}
<YYINITIAL> {ASIGNACION} { return new Symbol(sym.ASIGNACION, yyline, yycolumn, yytext());}
<YYINITIAL> {SEPARADOR} { return new Symbol(sym.SEPARADOR, yyline, yycolumn, yytext());}
<YYINITIAL> {IDENTIFICADOR} { return new Symbol(sym.IDENTIFICADOR, yyline, yycolumn, yytext());}
<YYINITIAL> {CONJUNTO} { return new Symbol(sym.CONJUNTO, yyline, yycolumn, yytext());}
<YYINITIAL> {EXPRESION} { return new Symbol(sym.EXPRESION, yyline, yycolumn, yytext()); }
```

*Parte de la sintaxis del archivo Lexico.jflex.*

### *Parser.java y sym.java*

Son archivos generados por medio de la librería java-cup mediante el uso del archivo Sintactico.cup los cuales son de utilidad para el análisis Sintáctico del archivo de entrada durante la ejecución del programa. Esto permite la recolección de errores, así como los conjuntos, expresiones y frases dentro del archivo EXP.

```
//----------------------------------------------------
// The following code was generated by CUP v0.11a beta 20060608
// Mon Feb 21 15:47:55 CST 2022
//----------------------------------------------------

package Analizadores;
import static ocll_proyecto1.OCLl_Proyecto1.ListaErrores;
import static ocll_proyecto1.OCLl_Proyecto1.ListaConjuntos;
import static ocll_proyecto1.OCLl_Proyecto1.ListaVerificaciones;
import static ocll_proyecto1.OCLl_Proyecto1.ListaExpresiones;
import java_cup.runtime.*;
import java.util.ArrayList;

/** CUP v0.11a beta 20060608 generated parser.
  * @version Mon Feb 21 15:47:55 CST 2022
  */
public class parser extends java_cup.runtime.lr_parser {
```

*Parte del código de parser.java.*

```
package Analizadores;

/** CUP generated class containing symbol constants. */
public class sym {
  /* terminals */
  public static final int PT_COMA = 11;
  public static final int CONJ = 6;
  public static final int EOF = 0;
  public static final int CONJUNTO = 3;
  public static final int SEPARADOR = 7;
  public static final int ASIGNACION = 12;
  public static final int error = 1;
  public static final int DOS_PTS = 10;
  public static final int FRASE = 5;
  public static final int LLAVE_CIERRE = 9;
  public static final int EXPRESION = 4;
  public static final int LLAVE_APERTURA = 8;
  public static final int IDENTIFICADOR = 2;
}
```

*Parte del código de sym.java.*

```
start with INICIO;

INICIO::= LLAVE_APERTURA CONJUNTOS EXPRESIONES SEPARADORES:sep VERIFICACIONES LLAVE_CIERRE
    {:
        if(!sep.equals("%%")){
            System.out.println("Exceso de separadores");
        }
        System.out.println("{\n"+instruccion+"%%\n"+segVerificar+"}");
    :}
;

SEPARADORES::= SEPARADOR:sep SEPARADORES:seps
    {:
        RESULT = sep+seps;
    :}
    | SEPARADOR:sep
    {:
        RESULT = sep;
    :}
```

*Parte del contenido del archivo Sintactico.cup.*

## Package Arboles


### *CambioNotacion.java*

En este se utiliza para como centro para el llamado a las funciones y procedimiento para la realización del método del árbol, la tabla de siguientes y tabla de transiciones para la formación del autómata finido determinista y el método de Thompson para la formación del autómata finito no determinista. Esto lo hace recibiendo como dos parámetros los cuales son la expresión regular la cual se desea realizar el árbol y el nombre que recibe dicha expresión.

```
package Arboles;


public class CambioNotacion {

    public TablaTransicion tablaTransicion;

    public CambioNotacion(String texto, String nombre){
        int largo = texto.length();
        FormacionArbol pila = new FormacionArbol(nombre);

        for(int i = largo-1; i>=0; i--){
            char caracter = texto.charAt(i);
            NodoArbol nuevo;

            if (((int) caracter) == 34){
                String cadena = "";

                for(int j = i-1; j>=0; j--){
```

Posteriormente realiza una lectura de la cadena carácter por carácter de derecha a izquierda realizando validaciones para ingresarlos como NodoArbol dentro de una pila y extrayendo la cantidad necesaria para la concatenación o cualquier otra operación que indique la expresión.

```
else if ((((int) caracter) >= 97 && ((int) caracter) <= 122) ||
    nuevo = new NodoArbol(String.valueOf(caracter));
    pila.insertar(nuevo);
}


else if (((int) caracter) == 46){
    nuevo = new NodoArbol(String.valueOf(caracter), false);
    nuevo.izquierda = pila.extraer();
    nuevo.derecha = pila.extraer();
    pila.insertar(nuevo);
}

else if (((int) caracter) == 42){
    nuevo = new NodoArbol(String.valueOf(caracter), true);
    nuevo.izquierda = pila.extraer();
    pila.insertar(nuevo);
}
```

*Ejemplos de validaciones para la formación del árbol.*

Por último, se ejecutan los métodos de la pila con la raíz del árbol para la creación de las tablas de siguientes, transiciones y la gráfica de los autómatas.

```
pila.crearArchivo();
pila.tablaSiguientes.generarHTML();

this.tablaTransicion = new TablaTransicion(pila.tablaSiguientes, nombre);
this.tablaTransicion.ingresarColumnas(pila.tablaSiguientes.primero);

this.tablaTransicion.ingresarFila(pila.raiz.primeros.imprimirLista());
this.tablaTransicion.llenarTabla();

this.tablaTransicion.htmlTabla();
this.tablaTransicion.crearArchivo();

pila.generarAFN();
```

*Implementación de las funciones de pila para la creación de los reportes.*

### *FormacionArbol.java*

En este archivo se encuentran todos los procedimiento y funciones para la creación de una pila de NodoArbol, mediante la implementación de la clase nodo que implementara la relación entre la pila.

```
package Arboles;

import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.util.logging.Level;
import java.util.logging.Logger;


public class FormacionArbol {

    public Nodo primero, ultimo;
    public NodoArbol raiz = null;
    public String dot ="", dotAFN ="";
    public String nombre = "";
    public int contadorHojas = 1, contadorAFN = 0;
    public int contadorNodos = 1;

    public ListaSiguientes tablaSiguientes;
    public FormacionArbol(String nombre){
        this.primero = null;
        this.ultimo = null;
        this.nombre = nombre;
        tablaSiguientes = new ListaSiguientes(nombre);
    }
```

## Métodos en FormacionArbol.java

```
public void insertar(NodoArbol nodo){...12 lines }

public NodoArbol extraer(){...12 lines }

public void recorrer(){...8 lines }

public String generarDot(){...8 lines }

public void recorrerArbol(NodoArbol nuevo){...42 lines }

public void crearArchivo(){...26 lines }

public void numerarNodos(NodoArbol nodo){...19 lines }

public void PrimUltSig(NodoArbol nodo){...78 lines }

public void imprimirArbol(NodoArbol nodo){...9 lines }

public void generarAFN(){...34 lines }

public int [] recorrerAFN(NodoArbol nodo){...238 lines }
```

## Método insertar(NodoArbol nodo)

Método que se utiliza para ingresar nodos del tipo NodoArbol dentro de la pila.

```
public void insertar(NodoArbol nodo){
    Nodo nuevo = new Nodo(nodo);

    if (this.primero != null){
        this.ultimo.siguiente = nuevo;
        nuevo.anterior = this.ultimo;
        this.ultimo = nuevo;

    }else {
        this.primero = this.ultimo = nuevo;
    }
}
```

*Código del método insertar.*

## Función extraer()

Función que devuelve el contenido del primer nodo de la pila.

```
public NodoArbol extraer(){
    Nodo aux = this.ultimo;
    if (this.primero == this.ultimo){
        this.primero = null;
    }
    else{
        this.ultimo.anterior.siguiente = null;
    }
    this.ultimo = this.ultimo.anterior;
    aux.anterior = null;
    return aux.contenido;
}
```

*Código de la función extraer.*

## Método recorrer()

Método para imprimir en consola el contenido de la pila.

```java
public void recorrer(){
    Nodo aux = this.primero;

    while(aux!=null){
        System.out.println(aux.contenido.texto);
        aux = aux.siguiente;
    }
}
```

*Código del método recorrer.*

### Función generarDot()

Función que se utiliza para generar el archivo dot que contiene la información para generar el árbol del método del árbol, retornado la información como una cadena String.

```java
public String generarDot(){
    this.dot = "digraph G{\n";

    this.recorrerArbol(this.raiz);

    this.dot += "}\n";
    return this.dot;
}
```

*Código de la función generarDot.*

### Método recorrerArbol(NodoArbol nuevo)

Método utilizado para recorrer el árbol para concatenar a la variable String dot la información de cada nodo perteneciente al árbol.

```java
public void recorrerArbol(NodoArbol nuevo){
    NodoArbol aux = nuevo;

    if(aux.izquierda != null){
        dot += "N"+aux.noNodo+"->";
        if(!aux.izquierda.hoja){
            dot+= "N"+aux.izquierda.noNodo+"\n";
        }else{
            dot+= aux.izquierda.noNodo+"\n";
        }
        this.recorrerArbol(nuevo.izquierda);
    }

    if(aux.hoja){
        if(aux.texto.equals("\\n")){
            dot += aux.noNodo+"[label=\"N\n\\\\n\nP: "+aux.primeros.imprimirLista()+
        }
        else if(aux.texto.equals("\\\"") || aux.texto.equals("\\\'")){
            dot += aux.noNodo+"[label=\"N\n\\\\"+aux.texto+"\nP: "+aux.primeros.impr
        }
        else{
            dot += aux.noNodo+"[label=\"N\n"+aux.texto+"\nP: "+aux.primeros.imprimi
        }
    }
    if(!aux.hoja){
        if (aux.anulable){
            dot += "N"+aux.noNodo+"[label=\"A\n"+aux.texto+"\nP: "+aux.primeros.impr
        }else{
            dot += "N"+aux.noNodo+"[label=\"N\n"+aux.texto+"\nP: "+aux.primeros.impr
        }
    }

    if(aux.derecha != null){
        dot += "N"+aux.noNodo+"->";
        if(!aux.derecha.hoja){
```

*Parte del código del método recorrerArbol.*

### Método crearArchivo()

Método para la creación del archivo .dot y la creación de la imagen .png dentro de la capeta Arboles_202003894.

```
public void crearArchivo(){
    String dirActual = System.getProperty("user.dir");

    File filedot = new File(dirActual+"/Reportes_202003894/Arboles_202003894/"+ this.nombre +".dot");
    try {
        filedot.createNewFile();
    } catch (IOException ex) {
        Logger.getLogger(FormacionArbol.class.getName()).log(Level.SEVERE, null, ex);
    }
    finally{
        try {
            try (FileWriter escritor = new FileWriter(filedot)) {
                escritor.write(this.generarDot());
            }

            ProcessBuilder proceso = new ProcessBuilder("dot", "-Tpng","-o",dirActual+"/Reportes_202003
            proceso.redirectErrorStream(true);
            proceso.start();
            }  catch (IOException ex) {
                Logger.getLogger(FormacionArbol.class.getName()).log(Level.SEVERE, null, ex);
            }
        catch (Exception e){
            e.printStackTrace();
        }
    }
}
```

*Método crearArchivo.*

### Método numerarNodos(NodoArbol nodo)

Método para recorrer el árbol e ir numerando los nodos hijos del árbol en orden ascendente de forma recursiva.

```
public void numerarNodos(NodoArbol nodo){

    if (nodo.izquierda != null){
        this.numerarNodos(nodo.izquierda);
    }

    if(nodo.hoja){
        nodo.noNodo = this.contadorHojas;
        this.contadorHojas ++;
    }
    if(!nodo.hoja){
        nodo.noNodo = this.contadorNodos;
        this.contadorNodos ++;
    }

    if (nodo.derecha != null){
        this.numerarNodos(nodo.derecha);
    }
}
```

*Método numerarNodos.*

### Método PrimUltSig(NodoArbol nodo)

Método para modificar los atributos del cada uno de los NodoArbol pertenecientes al método del árbol para en ellos el listado de nodos primeros, últimos y siguientes.

```
public void PrimUltSig(NodoArbol nodo){
    if (nodo.izquierda != null){
        this.PrimUltSig(nodo.izquierda);
    }

    if (nodo.derecha != null){
        this.PrimUltSig(nodo.derecha);
    }


    if (".".equals(nodo.texto) && !nodo.hoja){
        nodo.primeros.insertarLista(nodo.izquierda.primeros);

        if (nodo.izquierda.anulable == true && nodo.derecha.anulable == true){
            nodo.anulable = true;
        }
        if (nodo.izquierda.anulable){
            nodo.primeros.insertarLista(nodo.derecha.primeros);
        }
        if (nodo.derecha.anulable){
            nodo.ultimos.insertarLista(nodo.izquierda.ultimos);
        }
        nodo.ultimos.insertarLista(nodo.derecha.ultimos);

        NodoPU aux = nodo.izquierda.ultimos.primero;
        while(aux != null){
            this.tablaSiguientes.insertar(aux.Lexema, aux.noNodo, nodo.derecha.p
            aux = aux.siguiente;
        }
    }
```

*Parte del código del método PrimUltSig.*

## Método imprimirArbol

Método para imprimir en consola el árbol generado en in-order.

```
public void imprimirArbol(NodoArbol nodo){
    if (nodo.izquierda != null){
        this.imprimirArbol(nodo.izquierda);
    }
    System.out.println(nodo.texto);
    if (nodo.derecha != null){
        this.imprimirArbol(nodo.derecha);
    }
}
```

*Código del método*

## Método generarAFN()

Método para la creación del archivo .dot y .png del método de Thompson.

```
public void generarAFN(){

    this.dotAFN ="digraph G{\nrankdir = LR;\n" ;
    this.dotAFN +="Inicio[style=invis];\n";
    int arreglo [] = this.recorrerAFN(this.raiz.izquierda);
    this.dotAFN += "Inicio->"+(arreglo[0])+" [label=\"Inicio\"];\n";
    this.dotAFN += arreglo[1]+" [shape=doublecircle];\n";
    this.dotAFN += "}";

    String dirActual = System.getProperty("user.dir");

    File filedot = new File(dirActual+"/Reportes_202003894/AFND_202003894/"+ this.nombre +".dot");
    try {
        filedot.createNewFile();
    } catch (IOException ex) {
        Logger.getLogger(FormacionArbol.class.getName()).log(Level.SEVERE, null, ex);
    }
    finally{
        try {
            try (FileWriter escritor = new FileWriter(filedot)) {
                escritor.write(this.dotAFN);
            }

            ProcessBuilder proceso = new ProcessBuilder("dot", "-Tpng","-o",dirActual+"/Reportes_2020
            proceso.redirectErrorStream(true);
            proceso.start();
            } catch (IOException ex) {
                Logger.getLogger(FormacionArbol.class.getName()).log(Level.SEVERE, null, ex);
            }
        catch (Exception e){
            e.printStackTrace();
        }
    }
}
```

*Código del método generarAFN.*

### Función recorrerAFN(NodoArbol nodo)

Función recursiva que recorre todos los nodos del árbol del método del árbol para generar el método de Thompson y generar así un String con formato DOT con el cuál generar la gráfica del mismo.

```
public int [] recorrerAFN(NodoArbol nodo){

    int [] datosIzquierda = new int [2];
    int [] datosDerecha = new int [2];
    if (nodo.izquierda != null && !nodo.izquierda.hoja){
       datosIzquierda = recorrerAFN(nodo.izquierda);
    }

    if (nodo.derecha != null && !nodo.derecha.hoja){
       datosDerecha = recorrerAFN(nodo.derecha);
    }

    int [] regreso = new int [2];
    regreso[0]= this.contadorAFN;


    if (nodo.texto.equals(".") && nodo.izquierda.hoja && nodo.derecha.hoja){

        this.dotAFN += contadorAFN+"[shape=circle];\n";
        this.dotAFN += (contadorAFN+1)+"[shape=circle];\n";
        this.dotAFN += (contadorAFN+2)+"[shape=circle];\n";

        String izq = "", der = "";
        if (nodo.izquierda.texto.equals("\\n")){ izq = "\\\\n"; }
        else if(nodo.izquierda.texto.equals("\\\"")){ izq = "\\\\\""; }
        else if(nodo.izquierda.texto.equals("\\'")){ izq = "\\\\'"; }
        else{izq = nodo.izquierda.texto;}
        if (nodo.derecha.texto.equals("\\n")){ der = "\\\\n"; }
        else if(nodo.derecha.texto.equals("\\\"")){ der = "\\\\\""; }
        else if(nodo.derecha.texto.equals("\\'")){ der = "\\\\'"; }
        else{der = nodo.derecha.texto;}

        this.dotAFN += contadorAFN+"->"+(contadorAFN+1)+" [label=\""+izq+"\"];\n";
```

*Parte del código de la función recorrerAFN.*

### *Nodo.java*

Archivo utilizado para la creación de los objetos nodos que formaran la pila.

```
package Arboles;


public class Nodo {

    public NodoArbol contenido;
    public Nodo siguiente, anterior;

    public Nodo(NodoArbol nodo){
        this.contenido = nodo;
        this.siguiente = null;
        this.anterior = null;
    }

}
```

*Código de Nodo.java.*

### *NodoArbol.java*

Archivo utilizado para la creación de los nodos que formaran parte del árbol correspondiente a cada uno de los arboles generados por el método del árbol.

```java
package Arboles;

public class NodoArbol {

    public NodoArbol izquierda, derecha;
    public boolean anulable, hoja, estadoFinal = false;
    public int noNodo;
    public String texto;
    public ListaPU ultimos;
    public ListaPU primeros;

    public NodoArbol(String contenido){
        this.anulable = false;
        this.izquierda = null;
        this.hoja = true;
        this.noNodo = 0;
        this.derecha = null;
        this.texto = contenido;
        this.primeros= new ListaPU();
        this.ultimos= new ListaPU();
    }
 public NodoArbol(String contenido, boolean anulable){
     this.anulable = anulable;
     this.izquierda = null;
     this.hoja = false;
     this.noNodo = 0;
     this.derecha = null;
     this.texto = contenido;
     this.primeros = new ListaPU();
     this.ultimos = new ListaPU();
 }
```

*Código de NodoArbol.java*

### *NodoCabecera.java*

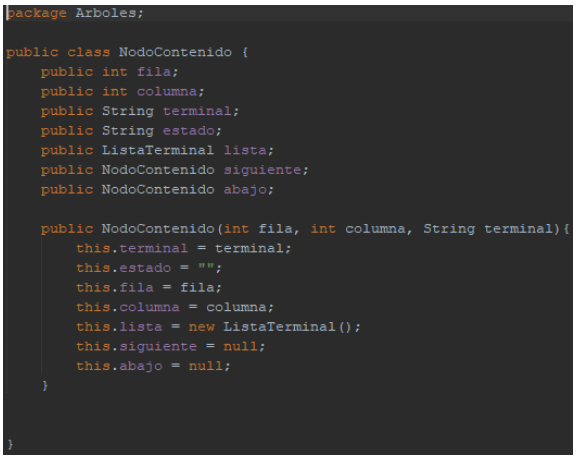
Con esta clase se crean los objetos que servirán como cabecera de las filas y columnas de la tabla de transición. Posee dos constructores dependiendo del caso en que se desee crear un nodo cabecero para fila o columna.

```java
package Arboles;

public class NodoCabecera {

    public String nombre = "";
    public String lexema;
    public int numero;
    public ListaEstado listaTransicion;
    public boolean estadoFinal = false;
    public NodoCabecera siguiente;
    public NodoCabecera abajo;

    public NodoCabecera(String lexema, int numero){
        this.numero = numero;
        this.listaTransicion = new ListaEstado();
        this.lexema = lexema;
        this.siguiente = null;
        this.abajo = null;
    }
 public NodoCabecera(String lexema, String estado, int numero){
     this.listaTransicion = new ListaEstado();
     this.numero = numero;
     this.nombre = estado;
     this.lexema = lexema;
     this.siguiente = null;
     this.abajo = null;
 }
```

*Código de Nodocabecera.java*

### *NodoContenido.java*

Con esta clase se crean los objetos Nodo que irán dentro y formaran la tabla de transiciones.

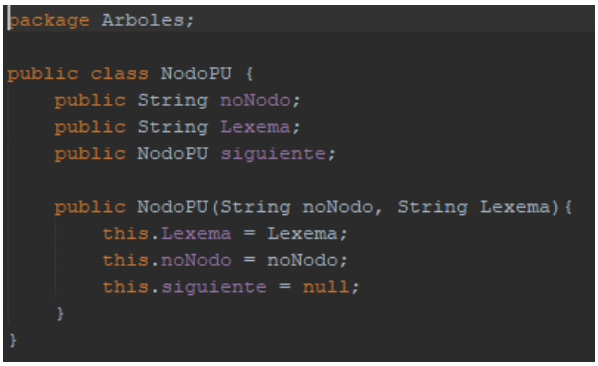


```java
package Arboles;

public class NodoContenido {
    public int fila;
    public int columna;
    public String terminal;
    public String estado;
    public ListaTerminal lista;
    public NodoContenido siguiente;
    public NodoContenido abajo;

    public NodoContenido(int fila, int columna, String terminal){
        this.terminal = terminal;
        this.estado = "";
        this.fila = fila;
        this.columna = columna;
        this.lista = new ListaTerminal();
        this.siguiente = null;
        this.abajo = null;
    }


}
```

*Código de NodoContenido.java*

### *NodoPU.java*

Archivo que contiene la clase para la creación de los nodos que formarán parte de la lista de Nodos que serán los primero o últimos de un nodo NodoArbol.

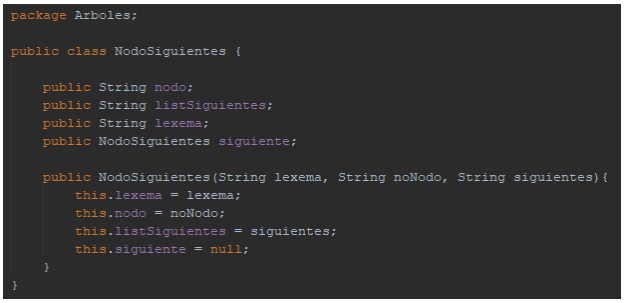


```java
package Arboles;

public class NodoPU {
    public String noNodo;
    public String Lexema;
    public NodoPU siguiente;

    public NodoPU(String noNodo, String Lexema){
        this.Lexema = Lexema;
        this.noNodo = noNodo;
        this.siguiente = null;
    }
}
```

*Código de NodoPU.java*

### *NodoSiguientes.java*

Archivo que contiene la clase para la creación de los nodos que formarán parte de la lista de Nodos siguientes de un terminal.



```java
package Arboles;

public class NodoSiguientes {

    public String nodo;
    public String listSiguientes;
    public String lexema;
    public NodoSiguientes siguiente;

    public NodoSiguientes(String lexema, String noNodo, String siguientes){
        this.lexema = lexema;
        this.nodo = noNodo;
        this.listSiguientes = siguientes;
        this.siguiente = null;
    }
}
```

*Código de NodoSiguientes.java*

### *NodoTerminal.java*

Archivo que contiene la clase para la creación de los nodos que formaran parte de la ListaTerminal la cual contiene el numero de terminal que pertenece a un estado.

```java
package Arboles;

public class NodoTerminal {
    public int numero;
    public NodoTerminal siguiente;

    public NodoTerminal(int numero){
        this.numero = numero;
        this.siguiente = null;
    }
}
```

*Código de NodoTerminal.java*

### *TablaTransiciones.java*

Archivo que contiene la clase para la creación de la tabla de transiciones la cual funciona como una matriz ya que contiene dos listas de nodos cabeceras correspondientes a los estados (filas) y terminales (columnas).

```java
package Arboles;

import static ocl1_proyecto1.OCL1_Proyecto1.ListaConjuntos;
import gui.ava.html.image.generator.HtmlImageGenerator;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;


public class TablaTransicion {

    public ListaCabecera columTransicion;
    public ListaCabecera filaTransicion;
    public ListaSiguientes tablaSiguiente;
    public String nombre;
    public int contadorEstado = 0;

    public TablaTransicion(ListaSiguientes listaSiguientes, String nombre){...6 lines }

    public void ingresarColumnas(NodoSiguientes primero){...8 lines }

    public String ingresarFila(String lista){...3 lines }

    public void llenarTabla(){...28 lines }

    public void htmlTabla(){...49 lines }

    public String generarDot(){...29 lines }

    public void crearArchivo(){...26 lines }
```

### *ListaCabecera.java*

Clase utilizada para la creación de la lista de nodoCabecera que forman la tabla de transiciones.

```java
package Arboles;

public class ListaCabecera {

    public NodoCabecera primero;
    public NodoCabecera ultimo;
    public int contadorTerminal = 1;
    public int contadorEstado = 0;

    public ListaCabecera(){
        this.primero = null;
        this.ultimo = null;
    }

    public void insertar(String lexema){...17 lines }

    public NodoCabecera buscar(String nodo){...12 lines }

    public String insertarFila(String lexema){...18 lines }

    public NodoCabecera buscarFila(String nodo){...12 lines }

    public void imprimir(){...9 lines }
}
```

## Métodos y funciones de ListaCabecera.java

### Método Insertar(String Lexema)

Método para insertar nodos dentro de la Lista de cabeceras para formar las columnas.

```java
public void insertar(String lexema){
    NodoCabecera aux = this.buscar(lexema);
    if (aux!=null){
        return;
    }
    else{
        NodoCabecera nuevo = new NodoCabecera(lexema, this.contadorTerminal);
        if (primero != null){
            this.ultimo.siguiente = nuevo;
            this.ultimo = nuevo;
        }
        else{
            this.primero = this.ultimo = nuevo;
        }
        this.contadorTerminal++;
    }
}
```

*Código del método Insertar.*

### Función buscar(String nodo)

Función que retorna un objeto del tipo NodoCabecera si encuentra una coincidencia con la cadena de texto que recibe como parámetro en una lista que sea de columnas.

```java
public NodoCabecera buscar(String nodo){
    NodoCabecera aux = this.primero;

    while(aux!=null){
        if(aux.lexema.equals(nodo)){
            return aux;
        }

        aux = aux.siguiente;
    }
    return null;
}
```

*Código de la función buscar.*

### Función InsertarFila(String Lexema)

Función para insertar nodos dentro de la Lista de cabeceras para formar las Filas y retorna una cadena String con el nombre del Estado creado.


*Código del método Insertar.*

### Función buscarFila(String nodo)

Función que retorna un objeto del tipo NodoCabecera si encuentra una coincidencia con la cadena de texto que recibe como parámetro en una lista que sea de Filas.

```
public NodoCabecera buscarFila(String nodo){
    NodoCabecera aux = this.primero;

    while(aux!=null){
        if(aux.lexema.equals(nodo)){
            return aux;
        }

        aux = aux.abajo;
    }
    return null;
}
```

*Código de la función buscarFila.*

### Método imprimir()

Método para imprimir en consola la lista de columnas, correspondiente a los terminales.

```
public void imprimir(){
    NodoCabecera aux = this.primero;

    while(aux != null){
        System.out.println(aux.numero+" "+aux.lexema);

        aux = aux.siguiente;
    }
}
```

*Código del método imprimir.*

### *ListaEstado.java*

Archivo que sirve para la creación de la lista de Estados que puede tener un NodoCabecera del tipo Fila.

```java
package Arboles;

public class ListaEstado {

    public NodoContenido primero;
    public NodoContenido ultimo;

    public ListaEstado(){
        this.primero = null;
        this.ultimo = null;
    }

    public void insertarEnFila(int fila, int columna, String Terminal, String [] terminales){...43 lines }

    public void insertarEnColumna(int fila, int columna, String Terminal, String [] terminales){...43 lines }

    public NodoContenido buscarEnFila(int columna){...12 lines }

    public NodoContenido buscarEnColumna(int fila){...12 lines }
}
```

*Código de Listado.java*

## Métodos y funciones de ListaEstado.java

### Método insertarEnFila(int fila, int columna, String Terminal, String [] terminales)

Método que inserta en la lista de forma ascendente cada nuevo nodo a ingresar.

```java
public void insertarEnFila(int fila, int columna, String Terminal, String [] terminales){
    NodoContenido aux = this.buscarEnFila(columna);
    if(aux != null){
        for(int i = 0; i < terminales.length ; i++){
            aux.lista.insertar(Integer.parseInt(terminales[i]));
        }
        return;
    }
    else{
        NodoContenido nuevo = new NodoContenido(fila, columna, Terminal);
        for(int i = 0; i < terminales.length ; i++){
            nuevo.lista.insertar(Integer.parseInt(terminales[i]));
        }
        if (this.primero == null){
            this.primero = this.ultimo = nuevo;
        }
        else if (this.primero.columna < nuevo.columna && this.primero.siguiente == null){
            this.primero.siguiente = nuevo;
            this.ultimo = nuevo;
        }
        else if (this.primero.columna > nuevo.columna){
            nuevo.siguiente = this.primero;
            this.primero = nuevo;
        }
        else{
            NodoContenido aux2 = this.primero;
            while(aux2 != null){
                if (aux2 == this.ultimo && aux2.columna < nuevo.columna){
                    aux2.siguiente = nuevo;
                    this.ultimo = nuevo;
                    break;
                }
```

*Parte del código de la función insertarEnFila.*

### Método insertarEnColumna(int fila, int columna, String Terminal, String [] terminales)

Método que inserta en la lista de forma ascendente cada nuevo nodo a ingresar.

```java
public void insertarEnColumna(int fila, int columna, String Terminal, String [] terminales){
    NodoContenido aux = this.buscarEnColumna(fila);
    if(aux != null){
        for(int i = 0; i < terminales.length ; i++){
            aux.lista.insertar(Integer.parseInt(terminales[i]));
        }
        return;
    }
    else{
        NodoContenido nuevo = new NodoContenido(fila, columna, Terminal);
        for(int i = 0; i < terminales.length ; i++){
            nuevo.lista.insertar(Integer.parseInt(terminales[i]));
        }
        if (this.primero == null){
            this.primero = this.ultimo = nuevo;
        }
        else if (this.primero.fila < nuevo.fila && this.primero.abajo == null){
            this.primero.abajo = nuevo;
            this.ultimo = nuevo;
        }
        else if (this.primero.fila > nuevo.fila){
            nuevo.abajo = this.primero;
            this.primero = nuevo;
        }
        else{
            NodoContenido aux2 = this.primero;
            while(aux2 != null){
                if (aux2 == this.ultimo && aux2.fila < nuevo.fila){
                    aux2.abajo = nuevo;
                    this.ultimo = nuevo;
                    break;
                }
                else if (aux2 != this.ultimo && aux2.abajo.fila > nuevo.fila && aux2.fila < nuevo.fila){
```

*Parte del código de la función insertarEnColumna.*

### Función buscarEnFila(int columna)

Función que retorna un objeto del tipo NodoContenido si encuentra una coincidencia con el número de columna que recibe como parámetro en la lista actual.

```java
public NodoContenido buscarEnFila(int columna){
    NodoContenido aux = this.primero;

    while(aux!=null){
        if(aux.columna == columna){
            return aux;
        }

        aux = aux.siguiente;
    }
    return null;
}
```

*Código de buscarEnFila.*

### Función buscarEnColumna(int fila)

Función que retorna un objeto del tipo NodoContenido si encuentra una coincidencia con el número de fila que recibe como parámetro en la lista actual.

```java
public NodoContenido buscarEnColumna(int fila){
    NodoContenido aux = this.primero;

    while(aux!=null){
        if(aux.fila == fila){
            return aux;
        }

        aux = aux.abajo;
    }
    return null;
}
```

*Código de buscarEnColumna.*

### *ListaPU.java*

En este archivo se encuentra la clase utilizada para la creación d ellos listados de nodos Primeros y Últimos de un nodoArbol.

```java
package Arboles;

public class ListaPU {

    public NodoPU primero;
    public NodoPU ultimo;

    public ListaPU(){
        this.primero = null;
        this.ultimo = null;
    }

    public void insertar(String noNodo, String lexema){...11 lines }

    public void insertarLista(ListaPU lista){...8 lines }

    public String imprimirLista(){...16 lines }
}
```

### Métodos y funciones de ListaPU.java

### Método insertar(String noNodo, String Lexema)

Método que inserta dentro de la Lista de primeros y últimos un nuevo nodo.

```java
public void insertar(String noNodo, String lexema){
    NodoPU nuevo = new NodoPU(noNodo, lexema);

    if (this.primero != null){
        this.ultimo.siguiente = nuevo;
        this.ultimo = nuevo;
    }
    else{
        this.primero = this.ultimo = nuevo;
    }
}
```

*Código método insertar.*

### Método insertarLista(ListaPU lista)

Método para insertar varios nodos dentro de la lista Primeros o últimos sucesivamente.

```java
public void insertarLista(ListaPU lista){
    NodoPU aux = lista.primero;

    while(aux != null){
        this.insertar(aux.noNodo, aux.Lexema);
        aux = aux.siguiente;
    }
}
```

*Código de insertarLista.*

### Función imprimirLista()

Función que retorna una cadena de texto con el contenido de toda la lista actual.

```java
public String imprimirLista(){
    NodoPU aux = this.primero;
    String cadena = "";

    while (aux != null){
        if (aux == this.primero){
            cadena += aux.noNodo;
        }else{
            cadena += ","+aux.noNodo;
        }

        aux = aux.siguiente;
    }

    return cadena;
}
```

*Código de imprimirLista.*

### *ListaSiguientes.java*

En este archivo se encuentra la clase con la cual se crean las listas de nodos siguientes a un nodo terminal.

```
package Arboles;

import gui.ava.html.image.generator.HtmlImageGenerator;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.util.logging.Level;
import java.util.logging.Logger;

public class ListaSiguientes {

    public NodoSiguientes primero;
    public NodoSiguientes ultimo;
    public String nombre;

    public ListaSiguientes(String nombre){
        this.primero = null;
        this.ultimo = null;
        this.nombre = nombre;
    }

    public void insertar(String lexema, String nodo, String siguientes){...37 lines }

    public NodoSiguientes buscar(String nodo){...12 lines }

    public void generarHTML(){...26 lines }
}
```

## Métodos y funciones de ListaSiguientes.java

### Método insertar(String lexema, String nodo, String siguientes)

Método que ingresa en orden ascendente cada nodo que se desee ingresar dentro de la lista.

```
public void insertar(String lexema, String nodo, String siguientes){
    NodoSiguientes aux = this.buscar(nodo);
    if (aux!=null){
        aux.listSiguientes += ","+siguientes;
    }
    else{
        NodoSiguientes nuevo = new NodoSiguientes(lexema, nodo, siguientes);
        if (this.primero == null){
            this.primero = this.ultimo = nuevo;
        }
        else if (Integer.parseInt(this.primero.nodo) < Integer.parseInt(nuevo.nodo) && this.primero.siguiente
            this.primero.siguiente = nuevo;
            this.ultimo = nuevo;
        }
        else if (Integer.parseInt(this.primero.nodo) > Integer.parseInt(nuevo.nodo)){
            nuevo.siguiente = this.primero;
            this.primero = nuevo;
        }
        else{
            NodoSiguientes aux2 = this.primero;
            while(aux2 != null){
                if (aux2 == this.ultimo && Integer.parseInt(aux2.nodo) < Integer.parseInt(nuevo.nodo)){
                    aux2.siguiente = nuevo;
                    this.ultimo = nuevo;
                    break;
                }
                else if (aux2 != this.ultimo && Integer.parseInt(aux2.siguiente.nodo) > Integer.parseInt(nue
                    nuevo.siguiente = aux2.siguiente;
                    aux2.siguiente = nuevo;
                    break;
                }

                aux2 = aux2.siguiente;
            }
        }
```

### Función buscar(String nodo)

Función que retorna un objeto del tipo NodoSiguientes el cual su atributo nodo sea igual al parámetro nodo que recibe la función.

```
public NodoSiguientes buscar(String nodo){
    NodoSiguientes aux = this.primero;

    while(aux!=null){
        if(aux.nodo.equals(nodo)){
            return aux;
        }

        aux = aux.siguiente;
    }
    return null;
}
```

*Código de buscar.*

### Método generarHTML()

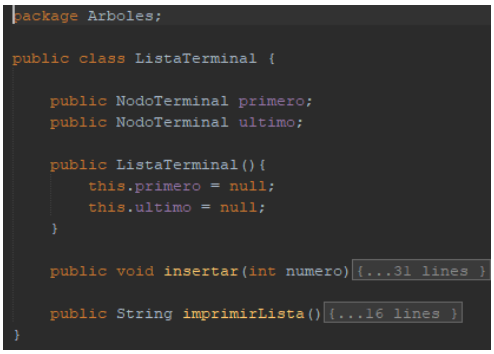
Método que genera un archivo html dentro de la carpeta Siguientes_202003894 y una imagen png con el contenido de la tabla html.

```
public void generarHTML(){
    NodoSiguientes aux = this.primero;
    String rutaActual = System.getProperty("user.dir");

    String html = "<table border=\"2\">\n<tr>\n<th colspan=\"2\">Hoja</th>\n<th>Siguientes</th>\n<tr>\n";

    while(aux != null){
        html += "</tr>\n<td>"+aux.lexema+"</td>\n<td>"+aux.nodo+"</td>\n<td>"+aux.listSiguientes+"</td>\n<tr>\n";
        aux = aux.siguiente;
    }
    html += "</table>";

    File archivo = new File(rutaActual+"/Reportes_202003894/Siguientes_202003894/"+this.nombre+".html");
    try {
        archivo.createNewFile();
        FileWriter escritor = new FileWriter(archivo);
        escritor.write(html);
        escritor.close();
    } catch (IOException ex) {
        Logger.getLogger(ListaSiguientes.class.getName()).log(Level.SEVERE, null, ex);
    }

    HtmlImageGenerator hig = new HtmlImageGenerator();
    hig.loadHtml(html);
    hig.saveAsImage(new File(rutaActual+"/Reportes_202003894/Siguientes_202003894/"+this.nombre+".png"));
}
```

*Código de generarHTML.*

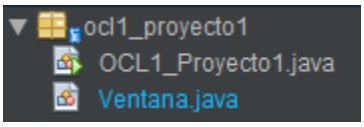
### *ListaTerminal.java*

Archivo que contiene la clase para la creación de Listas de Terminales que posee cada estado de la tabla de transiciones.



```
package Arboles;

public class ListaTerminal {

    public NodoTerminal primero;
    public NodoTerminal ultimo;

    public ListaTerminal(){
        this.primero = null;
        this.ultimo = null;
    }

    public void insertar(int numero){...31 lines }

    public String imprimirLista(){...16 lines }
}
```

## Package ocl1_proyecto1




### *OCL1_Proyecto1.java*

Archivo principal del proyecto que contiene la inicialización de las listas: ListaConjuntos, ListaExpresiones y ListaErrores. Así mismo tiene la inicialización de la ventana en que se mostrará la ejecución.

```java
package ocl1_proyecto1;

import java.util.ArrayList;
import java.util.List;

public class OCL1_Proyecto1 {

    public static List ListaConjuntos = new ArrayList<List>(), ListaExpresiones = new ArrayList<List>(), ListaVerificaciones = new ArrayList<List>(), ListaErrores = new ArrayList<List>();

    public static void main(String[] args) {
        Ventana ventana = new Ventana();
        ventana.setVisible(true);
    }

}
```

### *Ventana.java*

Archivo que contiene la configuración de la ventana que se visualizara durante la ejecución de la ventana, esto incluye: apariencia y funcionamiento.

```java
package ocl1_proyecto1;
import Analizadores.*;
import Arboles.*;
import static ocl1_proyecto1.OCL1_Proyecto1.ListaConjuntos;
import static ocl1_proyecto1.OCL1_Proyecto1.ListaExpresiones;
import static ocl1_proyecto1.OCL1_Proyecto1.ListaVerificaciones;
import static ocl1_proyecto1.OCL1_Proyecto1.ListaErrores;
import java.awt.Color;
import java.awt.Font;
import java.util.List;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileWriter;
import java.io.IOException;
import java.io.StringReader;
import java.util.ArrayList;
import java.util.Scanner;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.*;
import javax.swing.filechooser.FileFilter;
import javax.swing.filechooser.FileNameExtensionFilter;
import org.json.simple.JSONArray;
import org.json.simple.JSONObject;
```

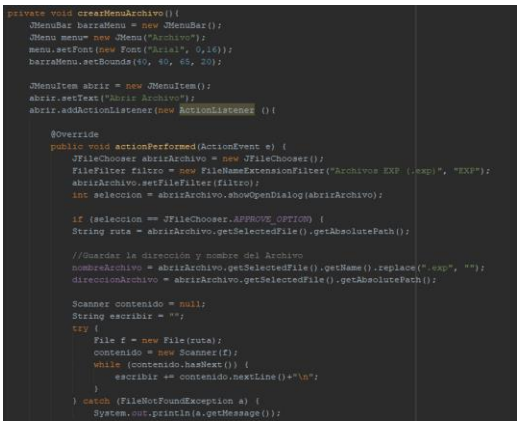
```java
public class Ventana extends JFrame{

    private JPanel panel = new JPanel();
    private JTextArea entrada = new JTextArea();
    private JTextArea salida = new JTextArea();
    private JLabel imagen = new JLabel();
    private List arboles;
    private String nombreArchivo = "", direccionArchivo = "";

    public Ventana(){
        this.generarDirectorios();

        this.setSize(800, 600);
        this.setTitle("OLC1 Proyecto 1");
        this.panel.setLayout(null);
        this.panel.setBackground(Color.GRAY);

        this.crearMenuArchivo();
        this.crearLabels();
        this.configurarTextArea();
        this.crearBotones();
        this.visualizadorImagenes();

        this.setDefaultCloseOperation(EXIT_ON_CLOSE);
        this.getContentPane().add(this.panel);
    }
```

```java
private void crearMenuArchivo(){...138 lines }

private void crearLabels(){...12 lines }

private void configurarTextArea(){...16 lines }

private void visualizadorImagenes(){...8 lines }

private void crearBotones(){...141 lines }

private void generarDirectorios(){...73 lines }
```

## Métodos y funciones de Ventana.js

### Método crearMenuArchivo()

Este método se utiliza para crear el menú desplegable que contiene todas las opciones para la creación o apertura de archivos.




*Parte del código de crearMenuArchvo.*

## Método crearLabels()

Método para crear los labels que contendrán texto para indicar que sección del programa es cada una.

```
private void crearLabels(){
    JLabel textol = new JLabel("Archivo de Entrada");
    textol.setBounds(40, 80, 200, 20);
    textol.setFont(new Font("Arial", 0,16));

    JLabel texto2 = new JLabel("Salida");
    texto2.setBounds(40,375, 200, 20);
    texto2.setFont(new Font("Arial", 0,16));

    this.panel.add(textol);
    this.panel.add(texto2);
}
```

*Código de crearLabels.*

## Método configurarTextArea()

Método para crear las áreas de texto de entrada y salida de información.

```
private void configurarTextArea(){
    this.entrada.setBounds(0, 0, 340, 220);
    JScrollPane scroll = new JScrollPane(this.entrada);
    scroll.setBounds(40, 110, 340, 220);
    scroll.setVerticalScrollBarPolicy(ScrollPaneConstants.VERTICAL_SCROLLBAR_ALWAYS);

    this.panel.add(scroll);


    this.salida.setBounds(0, 0, 710, 110);
    JScrollPane scroll2 = new JScrollPane(this.salida);
    scroll2.setBounds(40, 395, 710, 110);;
    scroll2.setVerticalScrollBarPolicy(ScrollPaneConstants.VERTICAL_SCROLLBAR_ALWAYS);

    this.panel.add(scroll2);
}
```

*Código de configurarTextArea.*

## Método visualizadorImagenes()

Método para crear el JScrollPane que se utiliza para mostrar las imágenes de reportes generados durante la ejecución.

```
private void visualizadorImagenes(){
    this.imagen.setBounds(0, 0, 340, 220);
    JScrollPane scroll = new JScrollPane(this.imagen);
    scroll.setBounds(405, 110, 340, 220);
    scroll.setVerticalScrollBarPolicy(ScrollPaneConstants.VERTICAL_SCROLLBAR_ALWAYS);

    this.panel.add(scroll);
}
```

*Código de visualizadorImagenes.*

## Método crearBotones()

Método para configurar los botones con los que se ejecutaran las acciones del programa las cuales son: Analizar el archivo de entrada, Crear Autómatas, Analizar las Frases y Visualizar Imágenes.

```
private void crearBotones(){
    JButton automata = new JButton("Crear Automatas");
    automata.setBounds(40, 350, 150, 20);
    this.panel.add(automata);

    automata.addActionListener(new ActionListener(){
        @Override
        public void actionPerformed(ActionEvent e) {
            arboles = new ArrayList<>();
            String textoSalida = "";
            for(int i=0; i< ListaExpresiones.size(); i++){
                List lista = (ArrayList) ListaExpresiones.get(i);
                try{
                    CambioNotacion cambio = new CambioNotacion(((String) lista.get(1)),((String) lista.get(0)));
                    arboles.add(new ArrayList<>(){{add(String.valueOf(lista.get(0))); add(cambio);}});
                    textoSalida += "Automatas creados correctamente con la expresion "+String.valueOf(lista.get(0)
                }catch(Exception a){
                    System.out.println(a.toString());
                    textoSalida += "La expresion regular "+String.valueOf(lista.get(0))+" posee un error\n";
                }
            }
            salida.setText(textoSalida);
        }
    });

    JButton analizarE = new JButton("Analizar Entradas");
    analizarE.setBounds(230, 350, 150, 20);
    this.panel.add(analizarE);

    analizarE.addActionListener(new ActionListener(){
        @Override
        public void actionPerformed(ActionEvent e) {
            salida.setText("");
            JSONArray listaAceptados = new JSONArray();
```

*Parte del código de crearBotones.*

## Método generarDirectorios()

Método para generar las carpetas donde se almacenarán los reportes que se generen durante la ejecución del programa.

```
private void generarDirectorios(){
    String directorioActual = System.getProperty("user.dir");
    File reportes = new File(directorioActual+"/Reportes_202003894");
    if (!reportes.exists()) {
        if (reportes.mkdirs()) {
            System.out.println("Directorio creado");
            File arboles = new File(directorioActual+"/Reportes_202003894/Arboles_202003894");
            if (!arboles.exists()) {
                if (arboles.mkdirs()) {
                    System.out.println("Directorio creado");
                } else {
                    System.out.println("Error al crear directorio");
                }
            }

            File afnd = new File(directorioActual+"/Reportes_202003894/AFND_202003894");
            if (!afnd.exists()) {
                if (afnd.mkdirs()) {
                    System.out.println("Directorio creado");
                } else {
                    System.out.println("Error al crear directorio");
                }
            }

            File afd = new File(directorioActual+"/Reportes_202003894/AFD_202003894");
            if (!afd.exists()) {
                if (afd.mkdirs()) {
                    System.out.println("Directorio creado");
                } else {
                    System.out.println("Error al crear directorio");
                }
            }

            File siguientes = new File(directorioActual+"/Reportes_202003894/Siguientes_202003894");
```

*Parte del código de generarDirectorios.*