Universidad de San Carlos de Guatemala

Facultad de Ingeniería

Escuela de ciencias y sistemas

Laboratorio de Organización de Lenguajes y Compiladores 1

Sección A

Manual de Técnico

Jonathan Alexander Alvarado Fernández

Carné: 201903004

Fecha de entrega: 06 de marzo de 2022

Paquete App:

Dentro de este paquete se encuentran 2 clases.

AppFrame: Esta clase contiene la interfaz que es descrita en el manual de usuario.

FileSystemModel: Esta clase contiene un modelo de sistema de archivos que se encarga de manejar el uso del jtree que se encuentra en el AppFrame.

Paquete Arbol:

Dentro de este paquete se encuentra únicamente la clase “Nodo”:

Nodo:

```
    private String Valor;
    private int Identificador;
    private boolean Anulabilidad;
    private ArrayList<Integer> Primeros;
    private ArrayList<Integer> Ultimos;
    private Nodo Izquierda;
    private Nodo Derecha;
    private ArrayList<String> terms;

    public Nodo(String Valor, int Identificador, boolean Anulabilidad, ArrayList<Integer> Primeros, ArrayList<Integer> Ultimos,
        this.Valor = Valor;
        this.Identificador = Identificador;
        this.Anulabilidad = Anulabilidad;
        this.Primeros = Primeros;
        this.Ultimos = Ultimos;
        this.Izquierda = Izquierda;
        this.Derecha = Derecha;
    }

    public ArrayList<String> getTerms(){
        this.terms = new ArrayList<String>();
        getTerms(this);
        return this.terms;
    }

    private boolean YaIngresado(String value){
        for(int i = 0; i < this.terms.size(); i++){
            if(terms.get(i).equals(value)){
                return true;
            }
```

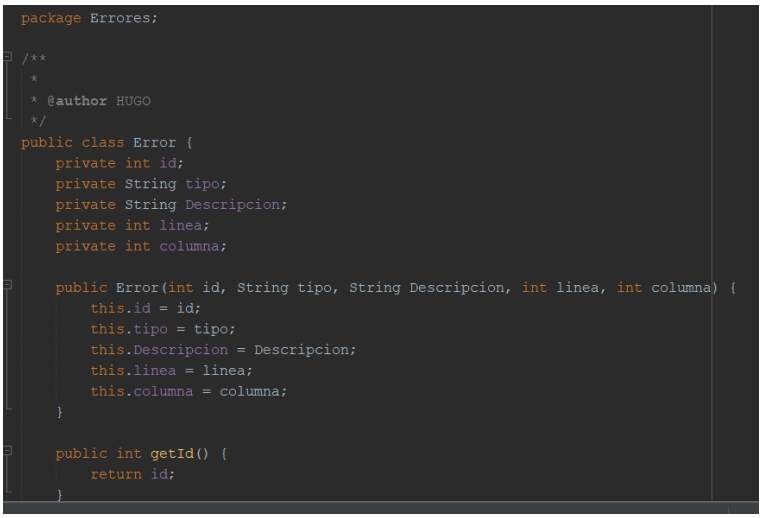
```
    public void GenerateSiguientes() {
        App.AppFrame.siguientes = new ArrayList<Tabla.Siguiente>();
        for (int i = 1; i <= this.Derecha.Identificador; i++) {
            App.AppFrame.auxiliarSigpor = new ArrayList<Integer>();
            GenerateSiguientes(this, i);
            Nom(this, i);
            //System.out.println(App.AppFrame.aux);
            App.AppFrame.siguientes.add(new Siguiente(App.AppFrame.aux, i, App.AppFrame.auxiliarSigpor));
        }
    }

    private boolean YaExiste(int index) {
        for (int i = 0; i < App.AppFrame.auxiliarSigpor.size(); i++) {
            if (AppFrame.auxiliarSigpor.get(i).equals(index)) {
                return true;
            }
        }
        return false;
    }

    private void Nom(Nodo nodo, int index) {
        if (nodo != null) {
            Nom(nodo.getIzquierda(), index);

            if (nodo.Identificador > 0) {
                if (nodo.getIdentificador() == index) {
                    App.AppFrame.aux = nodo.getValor();
                    return;
                }
```

esta clase se encarga del manejo del árbol de expresiones que es generado a partir de una expresión regular ingresada en la entrada del programa.
Contiene Métodos para el manejo de las primeras posiciones, ultimas posiciones, numero de nodo, anulabilidad.
También contiene métodos que realizan la generación de su tabla se siguientes.

Paquete Errores:

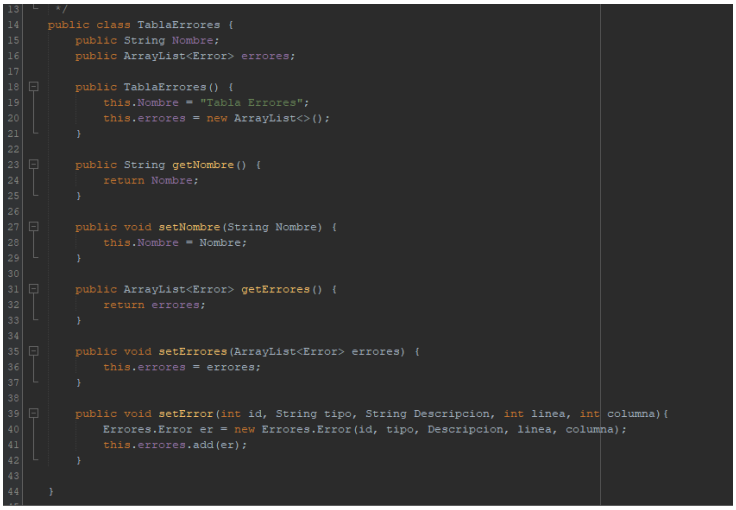
Este paquete contiene dos clases auxiliares para el manejo de errores dentro de la gramática.

Error: Representa cada línea de error.



```java
package Errores;

/**
 *
 * @author HUGO
 */
public class Error {
    private int id;
    private String tipo;
    private String Descripcion;
    private int linea;
    private int columna;

    public Error(int id, String tipo, String Descripcion, int linea, int columna) {
        this.id = id;
        this.tipo = tipo;
        this.Descripcion = Descripcion;
        this.linea = linea;
        this.columna = columna;
    }

    public int getId() {
        return id;
    }
```

TablaErrores: Contiene el nombre del archivo que se analizó y una lista de objetos error.



```java
 */
public class TablaErrores {
    public String Nombre;
    public ArrayList<Error> errores;

    public TablaErrores() {
        this.Nombre = "Tabla Errores";
        this.errores = new ArrayList<>();
    }

    public String getNombre() {
        return Nombre;
    }

    public void setNombre(String Nombre) {
        this.Nombre = Nombre;
    }

    public ArrayList<Error> getErrores() {
        return errores;
    }

    public void setErrores(ArrayList<Error> errores) {
        this.errores = errores;
    }

    public void setError(int id, String tipo, String Descripcion, int linea, int columna){
        Errores.Error er = new Errores.Error(id, tipo, Descripcion, linea, columna);
        this.errores.add(er);
    }

}
```

Paquete Grafica:

Este paquete contiene las clases que serán utilizadas para la graficación de reportes con graphviz, la generación de html de errores y el archivo json de salida.

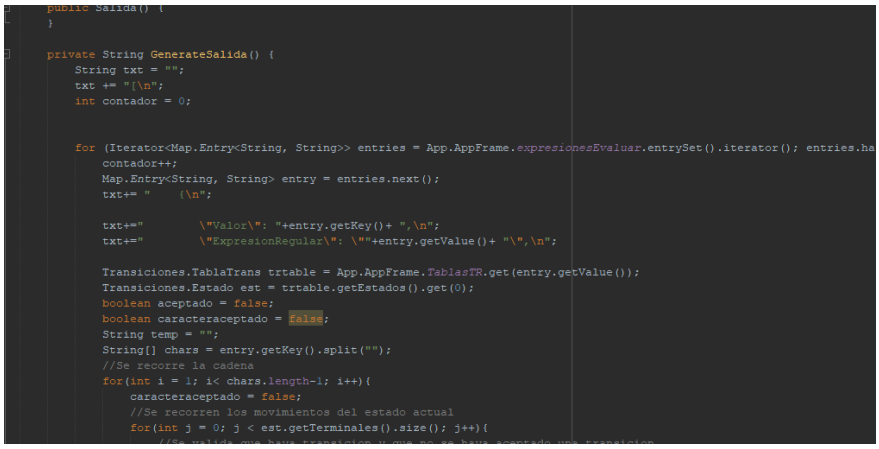
AFD:

```
*/
public class AFD {

    public AFD() {
    }

    private String Nodos(Transiciones.TablaTrans tbl, int fin) {
        boolean encontrado = false;
        String txt = "{\n";
        for (int i = 0; i < tbl.getEstados().size(); i++) {
            txt += "\n";
            txt += tbl.getEstados().get(i).getNombre();
            if (esAceptacion(tbl.getEstados().get(i).getLista(), fin)) {
                txt += "[shape=doublecircle fixedsize=true]";
            } else {
                txt += "[shape=circle fixedsize=true]";
            }
        }
        txt += "}";
        return txt;
    }

    private String Arrows(Transiciones.TablaTrans tbl) {
        String txt = "";
        for (int i = 0; i < tbl.getEstados().size(); i++) {
            for (int j = 0; j < tbl.getEstados().get(i).getTerminales().size(); j++) {
                if (!tbl.getEstados().get(i).getTerminales().get(j).equals("--")) {
                    txt += "\n";
                    txt += tbl.getEstados().get(i).getNombre() + " -> " + tbl.getEstados().get(i).getTerminales().get(j)+"[la
                }
            }
        }
```

Árbol:

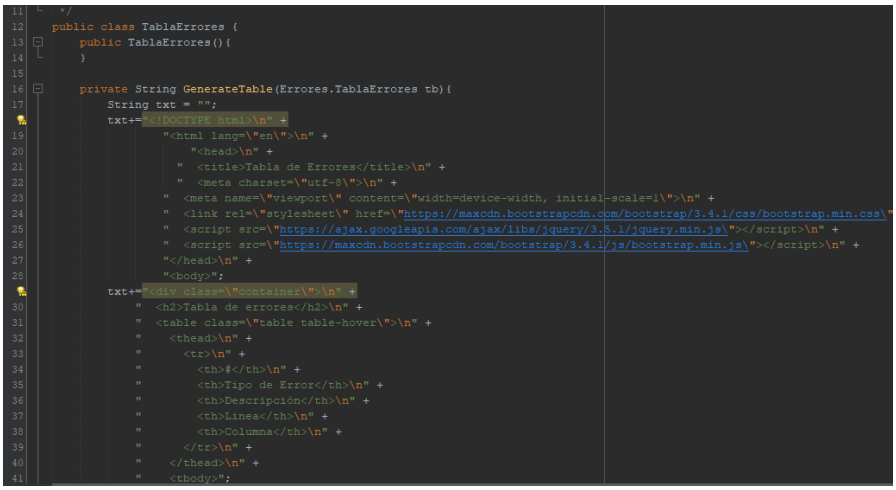
```
*/
public class Arbol {

    public Arbol() {
    }

    public String texto(Nodo nodo){
        String txt = "";
        if(nodo != null){
            if(nodo.getIzquierda() == null && nodo.getDerecha() == null){
                if(nodo.getValor().length()== 3){
                    txt += String.valueOf(nodo.getIdentificador())+"[label=\""+nodo.getPrimeros()+"|{"+nodo.isAnulabilidad()+"
                }else{
                    txt += String.valueOf(nodo.getIdentificador())+"[label=\""+nodo.getPrimeros()+"|{"+nodo.isAnulabilidad()+"
                }

            }else{
                if(nodo.getIzquierda() != null){
                    txt += String.valueOf(nodo.getIdentificador())+"[label=\""+nodo.getPrimeros()+"|{"+nodo.isAnulabilidad()+"
                    txt += texto(nodo.getIzquierda());
                }
                if(nodo.getDerecha() != null){
                    txt += String.valueOf(nodo.getIdentificador())+"[label=\""+nodo.getPrimeros()+"|{"+nodo.isAnulabilidad()+"
                    txt += texto(nodo.getDerecha());
                }
            }
        }
        return txt;
    }
```

Salida:



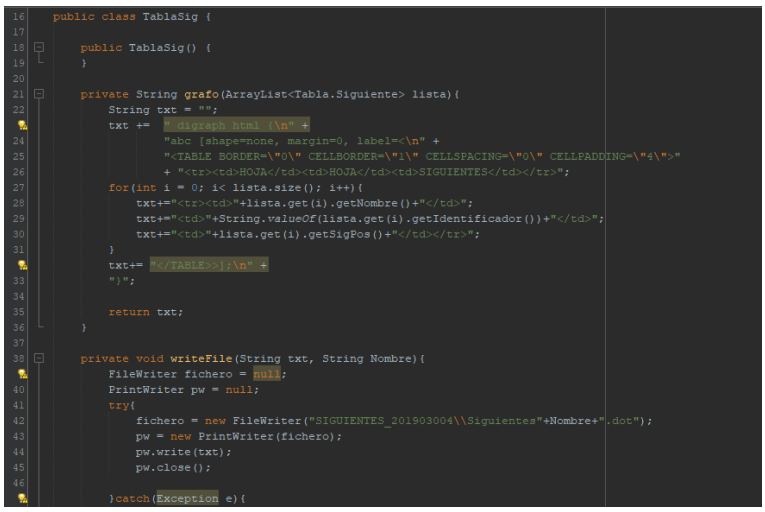
```
public Salida() {
}

private String GenerateSalida() {
    String txt = "";
    txt += "[\n";
    int contador = 0;


    for (Iterator<Map.Entry<String, String>> entries = App.AppFrame.expresionesEvaluar.entrySet().iterator(); entries.ha
        contador++;
        Map.Entry<String, String> entry = entries.next();
        txt+= "    {\n";

        txt+="        \"Valor\": "+entry.getKey()+ ",\n";
        txt+="        \"ExpresionRegular\": \""+entry.getValue()+ "\",\n";

        Transiciones.TablaTrans trttable = App.AppFrame.TablasTR.get(entry.getValue());
        Transiciones.Estado est = trttable.getEstados().get(0);
        boolean aceptado = false;
        boolean caracteraceptado = false;
        String temp = "";
        String[] chars = entry.getKey().split("");
        //Se recorre la cadena
        for(int i = 1; i< chars.length-1; i++){
            caracteraceptado = false;
            //Se recorren los movimientos del estado actual
            for(int j = 0; j < est.getTerminales().size(); j++){
```

TablaErrores:



```
*/
public class TablaErrores {
    public TablaErrores(){
    }

    private String GenerateTable(Errores.TablaErrores tb){
        String txt = "";
        txt+="<!DOCTYPE html>\n" +
                "<html lang=\"en\">\n" +
                   "<head>\n" +
                "  <title>Tabla de Errores</title>\n" +
                "  <meta charset=\"utf-8\">\n" +
                "  <meta name=\"viewport\" content=\"width=device-width, initial-scale=1\">\n" +
                "  <link rel=\"stylesheet\" href=\"https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/css/bootstrap.min.css\"
                "  <script src=\"https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js\"></script>\n" +
                "  <script src=\"https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/js/bootstrap.min.js\"></script>\n" +
                "</head>\n" +
                "<body>";
        txt+="<div class=\"container\">\n" +
            "  <h2>Tabla de errores</h2>\n" +
            "  <table class=\"table table-hover\">\n" +
            "    <thead>\n" +
            "      <tr>\n" +
            "        <th>#</th>\n" +
            "        <th>Tipo de Error</th>\n" +
            "        <th>Descripción</th>\n" +
            "        <th>Linea</th>\n" +
            "        <th>Columna</th>\n" +
            "      </tr>\n" +
            "    </thead>\n" +
            "    <tbody>";
```

TablaSig:



```
public class TablaSig {

    public TablaSig() {
    }

    private String grafo(ArrayList<Tabla.Siguiente> lista){
        String txt = "";
        txt +=  " digraph html {\n" +
                "abc [shape=none, margin=0, label=<\n" +
                "<TABLE BORDER=\"0\" CELLBORDER=\"1\" CELLSPACING=\"0\" CELLPADDING=\"4\">"
                + "<tr><td>HOJA</td><td>HOJA</td><td>SIGUIENTES</td></tr>";
        for(int i = 0; i< lista.size(); i++){
            txt+="<tr><td>"+lista.get(i).getNombre()+"</td>";
            txt+="<td>"+String.valueOf(lista.get(i).getIdentificador())+"</td>";
            txt+="<td>"+lista.get(i).getSigPos()+"</td></tr>";
        }
        txt+= "</TABLE>>];\n" +
        "}";

        return txt;
    }

    private void writeFile(String txt, String Nombre){
        FileWriter fichero = null;
        PrintWriter pw = null;
        try{
            fichero = new FileWriter("SIGUIENTES_201903004\\Siguientes"+Nombre+".dot");
            pw = new PrintWriter(fichero);
            pw.write(txt);
            pw.close();

        }catch(Exception e){
```

TablaTrans:

```
public class TablaTrans {

    public TablaTrans() {
    }

    private String grafo(Transiciones.TablaTrans tbl){
        String txt = "";
        txt +=  " digraph html {\n" +
                "abc [shape=none, margin=0, label=<\n" +
                "<TABLE BORDER=\"0\" CELLBORDER=\"1\" CELLSPACING=\"0\" CELLPADDING=\"4\">";
                //+ "<tr><td>HOJA</td><td>HOJA</td><td>SIGUIENTES</td></tr>";/
                txt+= "<tr><td>Estado</td   >";
                for(int i = 0; i < tbl.getTerms().size()-1; i++){
                    txt+= "<td>"+tbl.getTerms().get(i)+"</td>";
                }
                txt+= "</tr>";
        for(int i = 0; i< tbl.getEstados().size(); i++){
            txt+="<tr><td>"+tbl.getEstados().get(i).getNombre()+tbl.getEstados().get(i).getLista()+"</td>";
            for(int j = 0; j < tbl.getEstados().get(i).getTerminales().size(); j++){
                txt+= "<td>"+tbl.getEstados().get(i).getTerminales().get(j)+"</td>";
            }
            txt+="</tr>";
        }
        txt+= "</TABLE>>];\n" +
        "}";

        return txt;
    }

    private void writeFile(String txt, String Nombre){
        FileWriter fichero = null;
        PrintWriter pw = null;
```

Paquete Parser:

Este paquete contiene el analizador léxico hecho con jFlex y el analizador sintáctico hecho con JCUP.

Lexico.jflex: Contiene las reglas léxicas del lenguaje.

```
BLANCOS=[ \r\t]+
D=[0-9]+
DD=[0-9]+("."[  |0-9]+)?
LETR = [A-Za-z0-9]
ID =    [A-Za-z0-9_-]+
RCONCOMA = [\!-\)](","[\!-\)])*
T = \"([^\"])*\"
CMUL = \<\!([^\"])*\!\>
CUL = (\/)(\/)(.)*
SEP = (\%)(\%)
SL = (\\)(n)
CS = (\\)(\')
CD = (\\)(\")
CONCAT = (\.)


%%

"CONJ" {return new Symbol(sym.CONJ,yyline,yychar,yytext());}
":" {return new Symbol(sym.DOSPT,yyline,yychar,yytext());}
";" {return new Symbol(sym.PTCOMA,yyline,yychar,yytext());}
"->" {return new Symbol(sym.ASIGN,yyline,yychar,yytext());}
"(" {return new Symbol(sym.LLAVEIZQ,yyline,yychar,yytext());}
")" {return new Symbol(sym.LLAVEDER,yyline,yychar,yytext());}
"|" {return new Symbol(sym.DISYUNCION,yyline,yychar,yytext());}
"*" {return new Symbol(sym.KLEEN,yyline,yychar,yytext());}
"+" {return new Symbol(sym.CERRADURAMAS,yyline,yychar,yytext());}
"?" {return new Symbol(sym.CERRADURAINTER,yyline,yychar,yytext());}
"~" {return new Symbol(sym.HASTA,yyline,yychar,yytext());}
\n {yychar=1;}
```

Sintactico.cup: Contiene las reglas sintácticas del lenguaje.

```
terminal String REVALUAR;

non terminal inicio, instrucciones, instruccion, instr;
non terminal operacion, operaciones;
non terminal Nodo regex;
start with inicio;

inicio ::= LLAVEIZQ instr LLAVEDER
    | error LLAVEDER
;

instr::= instrucciones instr
    | instrucciones operaciones
;

instrucciones::= CONJ DOSPT IDENTIFICADOR:a ASIGN SIMBOLO:b HASTA:c SIMBOLO:d PTCOMA{:AppFrame.conjuntos.put(a, b+c+d);:}
    | CONJ DOSPT IDENTIFICADOR:a ASIGN RRCONCOMA:b PTCOMA{:AppFrame.conjuntos.put(a, b);:}
    | IDENTIFICADOR:a  ASIGN regex:b PTCOMA {:
        AppFrame.expresiones.add(a);
        AppFrame.primeros = new ArrayList<Integer>();
        AppFrame.ultimos = new ArrayList<Integer>();
        AppFrame.identificador++;
        AppFrame.primeros.add(AppFrame.identificador);
        AppFrame.ultimos.add(AppFrame.identificador);
        Nodo c = new Nodo("#", AppFrame.identificador, false,AppFrame.primeros, AppFrame.ultimos, null, null);
        AppFrame.identificador = 0;



        AppFrame.primeros = b.getPrimeros();
        AppFrame.ultimos = new ArrayList<Integer>();
```

Paquete Tabla:

Este paquete contiene clases auxiliares para el manejo de las tablas siguientes.

Siguiente:

```java
public class Siguiente {
    private int identificador;
    private ArrayList<Integer> SigPos;
    private String Nombre;

    public Siguiente(String Nombre,int identificador, ArrayList<Integer> sig) {
        this.Nombre = Nombre;
        this.identificador = identificador;
        this.SigPos = sig;
    }

    public String getNombre() {
        return Nombre;
    }

    public void setSigPos(int id){
        this.SigPos.add(id);
    }

    public int getIdentificador() {
        return identificador;
    }

    public ArrayList<Integer> getSigPos() {
        return SigPos;
    }


}
```

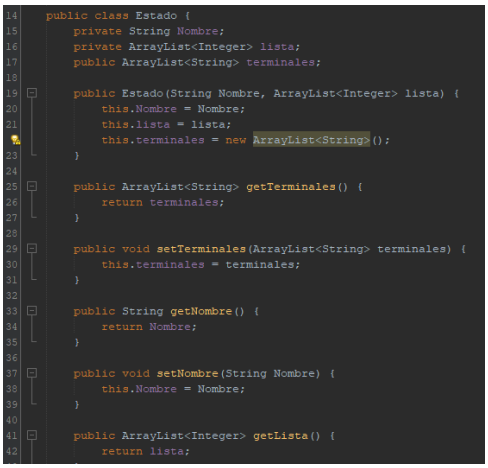
Table:

```java
package Tabla;

import java.util.ArrayList;

/**
 *
 * @author HUGO
 */
public class Table {
    private ArrayList<Siguiente> siguientes;

    public Table() {
        siguientes = new ArrayList<Siguiente>();
    }

    public void setSiguientes(ArrayList<Siguiente> siguientes) {
        this.siguientes = siguientes;
    }

    public ArrayList<Siguiente> getSiguientes() {
        return siguientes;
    }


}
```

Paquete Transiciones:

Este paquete contiene clases auxiliares para el manejo de las tablas de transiciones.

Estado:



```java
public class Estado {
    private String Nombre;
    private ArrayList<Integer> lista;
    public ArrayList<String> terminales;

    public Estado(String Nombre, ArrayList<Integer> lista) {
        this.Nombre = Nombre;
        this.lista = lista;
        this.terminales = new ArrayList<String>();
    }

    public ArrayList<String> getTerminales() {
        return terminales;
    }

    public void setTerminales(ArrayList<String> terminales) {
        this.terminales = terminales;
    }

    public String getNombre() {
        return Nombre;
    }

    public void setNombre(String Nombre) {
        this.Nombre = Nombre;
    }

    public ArrayList<Integer> getLista() {
        return lista;
```

TablaTrans:

```java
public class TablaTrans {

    private String nombre;
    private ArrayList<Estado> estados;
    private ArrayList<String> terms;

    public TablaTrans() {
        this.nombre = "";
        this.estados = new ArrayList<Estado>();
        this.terms = new ArrayList<String>();
    }

    private boolean YaIngresado(ArrayList<Integer> lista, int Valor) {
        for (int i = 0; i < lista.size(); i++) {
            if (lista.get(i).equals(Valor)) {
                return true;
            }
        }
        return false;
    }

    private boolean YaExisten(ArrayList<Integer> lista, ArrayList<ArrayList<Integer>> listas) {
        for (int i = 0; i < listas.size(); i++) {
            if (listas.get(i).equals(lista)) {
                return true;
            }
        }
        return false;
    }
```

Paquete expanalizer:

ExpAnalyzer: Contiene el método main y los métodos para la compilación de los analizadores a la hora de la ejecución del programa.

```java
public class ExpAnalyzer {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        generarCompilador("src/Parser/");
        App.AppFrame app = new App.AppFrame();
        app.setVisible(true);
    }

    public static void generarCompilador(String path) {
        try {
            String opcFlex[] = {path + "Lexico.jflex", "-d", path};
            jflex.Main.generate(opcFlex);

            String opcCUP[] = {"-destdir", path, "-parser", "Sintactico", path + "Sintactico.cup"};
            java_cup.Main.main(opcCUP);
        } catch (Exception e) {
        }
    }
}
```