MANUAL TÉCNICO

Bienvenido a la aplicación creada para comparación de proyectos codificados, el objetivo de esta aplicación es comparar 2 archivos de entrada que resultan ser dos proyectos que contienen clases, métodos, declaración de variables, llamada a métodos, llamada de variables, sección de comentarios, declaración y desarrollo de sentencias y condicionales, entre otras. Las mencionadas anteriormente se encuentran dentro de una estructura de clase inicial, incluyendo los id de cada una de las cosas a comparar anteriormente se toman en cuenta en la comparación, se crea un desarrollo de puntaje para calcular el porcentaje de parecido o copia de uno y el otro que pueda haber, a continuación, se muestra un manual de instrucciones ilustrativas del manejo de la aplicación y su funcionamiento.

1. Para el desarrollo de esta aplicación se utilizaron las herramientas de analizador léxico JFLEX, y también analizador sintáctico CUP. Los cuales se desarrollan por medio de una plantilla de código alterable donde se añaden los símbolos, expresiones regulares, palabras reservadas e instrucciones específicas para satisfacer las necesidades del programa y sus requerimientos.

ANALIZADOR LÉXICO JFLEX

La estructura de dicho analizador es la siguiente:

```
/*-------------------------------------------------
 ------------  1ra Area: Codigo de Usuario ---------
 -------------------------------------------------*/

//------> Paquetes,importaciones
package analizadores;
import java_cup.runtime.*;
import javax.swing.JOptionPane;

/*-----------------------------------------------------------
  ------------  2da Area: Opciones y Declaraciones ---------
  -----------------------------------------------------------*/
%%
%{
    //----> Codigo de usuario en sintaxis java
%}

//-------> Directivas
%public
%class Analizador_Lexico_FCA
%cupsym Simbolos_FCA
%cup
%char
%column
%full
%ignorecase
%line
%unicode

//------> Expresiones Regulares

cadena              = [\"][^\"\n]+[\"]
id                  = [A-z0-9]+("_"|[A-z]|[0-9])*
decimal             = [0-9]+("."[0-9]*)

LineTerminator = \r|\n|\r\n
InputCharacter = [^\r\n]

comentariosimple    = "##" {InputCharacter}* {LineTerminator}?

//------> Estados

%%

/*-------------------------------------------------
  ------------  3ra Area: Reglas Lexicas ---------
  -------------------------------------------------*/
```

Esta parte es donde se realizan las especificaciones de los símbolos a utilizar, llamada a expresiones regulares para aceptar en el análisis.

```
/*-------------------------------------------------
  ------------  3ra Area: Reglas Lexicas ---------
  -------------------------------------------------*/

//-----> Simbolos

","         { System.out.println("Reconocio "+yytext()+" coma"); return new Symbol(Simbolos_FCA.coma, yycolumn, yyline, yytext()); }
"{"         { System.out.println("Reconocio "+yytext()+" llava"); return new Symbol(Simbolos_FCA.llava, yycolumn, yyline, yytext()); }
"}"         { System.out.println("Reconocio "+yytext()+" llavc"); return new Symbol(Simbolos_FCA.llavc, yycolumn, yyline, yytext()); }
"("         { System.out.println("Reconocio "+yytext()+" para"); return new Symbol(Simbolos_FCA.para, yycolumn, yyline, yytext()); }
")"         { System.out.println("Reconocio "+yytext()+" parc"); return new Symbol(Simbolos_FCA.parc, yycolumn, yyline, yytext()); }
";"         { System.out.println("Reconocio "+yytext()+" pyc"); return new Symbol(Simbolos_FCA.pyc, yycolumn, yyline, yytext()); }
":"         { System.out.println("Reconocio "+yytext()+" dosp"); return new Symbol(Simbolos.dosp, yycolumn, yyline, yytext()); }
"="         { System.out.println("Reconocio "+yytext()+" igual"); return new Symbol(Simbolos.igual, yycolumn, yyline,yytext());}
"+"         { System.out.println("Reconocio "+yytext()+" mas"); return new Symbol(Simbolos.mas, yycolumn, yyline, yytext()); }
"-"         { System.out.println("Reconocio "+yytext()+" menos"); return new Symbol(Simbolos.menos, yycolumn, yyline, yytext()); }
"*"         { System.out.println("Reconocio "+yytext()+" por"); return new Symbol(Simbolos.por, yycolumn, yyline, yytext()); }
"/"         { System.out.println("Reconocio "+yytext()+" div"); return new Symbol(Simbolos.div, yycolumn, yyline, yytext()); }
"["         { System.out.println("Reconocio "+yytext()+" cora"); return new Symbol(Simbolos.cora, yycolumn, yyline, yytext()); }
"]"         { System.out.println("Reconocio "+yytext()+" corc"); return new Symbol(Simbolos.corc, yycolumn, yyline, yytext()); }

//-----> Palabras reservadas
"GenerarReporteEstadistico"         { System.out.println("Reconocio "+yytext()+" gre"); return new Symbol(Simbolos_FCA.gre, yycolumn, yyline, yytext()); }
"Compare"         { System.out.println("Reconocio "+yytext()+" comp"); return new Symbol(Simbolos_FCA.comp, yycolumn, yyline, yytext()); }


//-------> Simbolos ER
{cadena}    { System.out.println("Reconocio "+yytext()+" cadena"); return new Symbol(Simbolos_FCA.cadena, yycolumn, yyline, yytext()); }
{id}    { System.out.println("Reconocio "+yytext()+" id"); return new Symbol(Simbolos.id, yycolumn, yyline, yytext()); }
{decimal}    { System.out.println("Reconocio "+yytext()+" deci"); return new Symbol(Simbolos.deci, yycolumn, yyline, yytext()); }


//------> Espacios
{comentariosimple}      {System.out.println("Comentario: "+yytext()); }
[ \t\r\n\f]             {/* Espacios en blanco, se ignoran */}

//------> Errores Lexicos
.                       { System.out.println("Error Lexico"+yytext()+" Linea "+yyline+" Columna "+yycolumn);}
```

ANALIZADOR SINTÁCTICO CUP

En donde se especifican las palabras reservadas y se desarrollan las especificaciones para la lectura del archivo. El cual consta de la siguiente estructura:

```
/*---------------- 1ra Area: Codigo de Usuario ----------------------*/
//-------> importaciones, paquetes
package analizadores;
import java_cup.runtime.Symbol;
import ventana_compi.Nodo;



//------> Codigo para el parser,variables, metodos
parser code{:

        //Podemos crear variables, listas, etc;
        // deben de ser staticas si se quieren usar fuera

        public static Nodo raiz;

        public Nodo getRaiz(){
            return raiz;
        }


        public void syntax_error(Symbol s)
        {
            System.err.println("Error en la Linea " + (s.right+1) +" Columna "+(s.left+1)+ ". Identificador "+s.value + " no reconocido. Se ha recuperado de
        }

        public void unrecovered_syntax_error(Symbol s) throws java.lang.Exception
        {
            System.err.println("Error en la Linea " + (s.right+1)+ " Columna "+(s.left+1)+". Identificador " +s.value + " no reconocido.");
        }

:}

//------> Codigo para las acciones gramaticales
action code
{:  :}
```

En esta área de declaraciones, se inicializan la declaración de terminales que son los símbolos en este caso y los no terminales que serán los encabezados de las instrucciones que se construyeron a continuación.

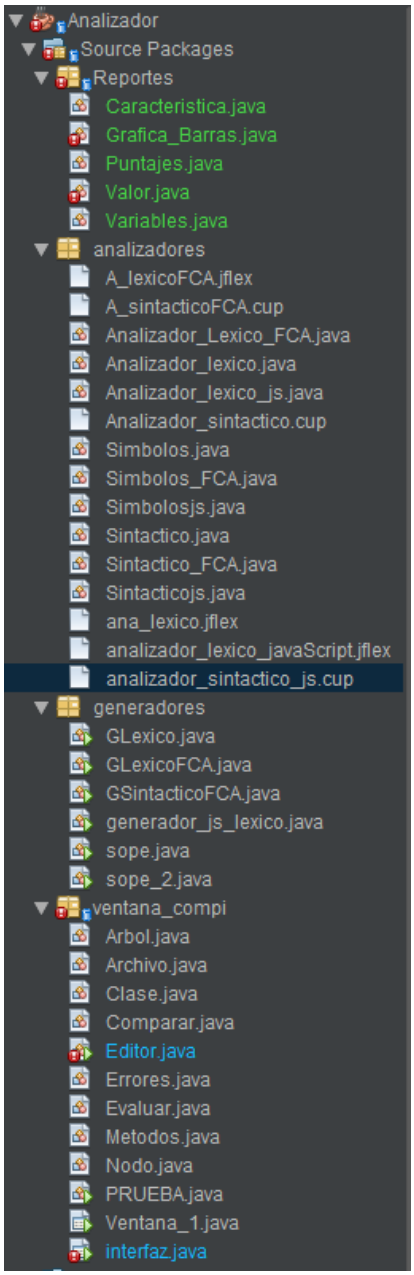
```
/*---------------- 2da Area: Declaraciones -----------------------*/

//------> declaracion de terminales

terminal String mas, deci,coma, por,id, div, str, menos,dosp,pyc, keis, bre, defi, digito, cadena,  para, parc, igual, llavea, llavec,clase,vare,lete,const


//------> declaracion de no terminales
non terminal Nodo  INICIO,  INSTRUCCIONES, INSTRUCCION;
non terminal Nodo DECLARACION, DECLARACIONES;
non terminal Nodo OPERACIONES, OPERACION;
non terminal Nodo COMPARACIONES;
non terminal Nodo ASIGNACIONES;
non terminal Nodo O;
non terminal Nodo FUNCION;
non terminal Nodo METODOS;
non terminal Nodo FORMETOD;
non terminal Nodo IFMETOD;
non terminal Nodo IFELSEMETOD;
non terminal Nodo WHILEMETOD;
non terminal Nodo MIXELSEMETOD;
non terminal Nodo DOWHILEMETOD;
non terminal Nodo SWITCHMETOD;
non terminal Nodo METODOVACIO;
non terminal Nodo CONTENTFOR;
non terminal Nodo CONTENIDOS;
non terminal Nodo SENTENCIAS;
non terminal Nodo ELSEMETODE, CLASE;




//------> precedencia
precedence left mas, menos;
precedence left por, div;
```

```
//------> precedencia
precedence left mas, menos;
precedence left por, div;



start with INICIO;

/*---------------- 3ra Area: Reglas Semanticas -----------------------*/

INICIO::= CLASE:a   {: parser.raiz = a; :};

CLASE::= clase id:a llavea:b INSTRUCCIONES:c llavec:g  {:
                                                Nodo superpadre = new Nodo("CLASE","",0,0);
                                                Nodo padre = new Nodo("CLASE","",0,0);
                                                    padre.AddHijo(new Nodo("id", a, aright, aleft));
                                                    padre.AddHijo(new Nodo("llavea", b, bright, bleft));
                                                    padre.AddHijo(c);
                                                    padre.AddHijo(new Nodo("llavec", g, gright, gleft));
                                                    superpadre.AddHijo(padre);
                                                    RESULT = superpadre;
                                            :}
                                             ;

INSTRUCCIONES::= INSTRUCCIONES:a FUNCION:b            {:Nodo padre = new Nodo("INSTRUCCIONES","",0,0);
                                                      padre.AddHijo(a);
                                                      padre.AddHijo(b);
                                                      RESULT = padre;  :}


       | FUNCION:a                            {: Nodo padre = new Nodo("INSTRUCCIONES","",0,0);
                                                    padre.AddHijo(a);
                                                    RESULT = padre; :}

       | INSTRUCCIONES:a SENTENCIAS:b           {: Nodo padre = new Nodo("INSTRUCCIONES","",0,0);
                                                    padre.AddHijo(a);
                                                    padre.AddHijo(b);
```

En esta parte se construyen toda la estructura especifica de cómo debe venir el formato archivo de entrada, y las posibles variantes que pueda tener de contenido dentro de él para que pueda reconocer diferentes archivos de entrada.

```
INICIO::= CLASE:a   {: parser.raiz = a; :};

CLASE::= clase id:a llavea:b INSTRUCCIONES:c llavec:g  {:
                                                    Nodo superpadre = new Nodo("CLASE","",0,0);
                                                    Nodo padre = new Nodo("CLASE","",0,0);
                                                        padre.AddHijo(new Nodo("id", a, aright, aleft));
                                                        padre.AddHijo(new Nodo("llavea", b, bright, bleft));
                                                        padre.AddHijo(c);
                                                        padre.AddHijo(new Nodo("llavec", g, gright, gleft));
                                                        superpadre.AddHijo(padre);
                                                        RESULT = superpadre;
                                                :}
                                                 ;

INSTRUCCIONES::= INSTRUCCIONES:a FUNCION:b             {:Nodo padre = new Nodo("INSTRUCCIONES","",0,0);
                                                        padre.AddHijo(a);
                                                        padre.AddHijo(b);
                                                        RESULT = padre;  :}


       | FUNCION:a                              {: Nodo padre = new Nodo("INSTRUCCIONES","",0,0);
                                                    padre.AddHijo(a);
                                                    RESULT = padre; :}

       | INSTRUCCIONES:a SENTENCIAS:b           {: Nodo padre = new Nodo("INSTRUCCIONES","",0,0);
                                                    padre.AddHijo(a);
                                                    padre.AddHijo(b);
                                                    RESULT = padre;:}

       | SENTENCIAS:a                           {: Nodo padre = new Nodo("INSTRUCCIONES","",0,0);
                                                    padre.AddHijo(a);
                                                    RESULT = padre; :}

       | INSTRUCCIONES:a DECLARACION:b           {: Nodo padre = new Nodo("INSTRUCCIONES","",0,0);
                                                    padre.AddHijo(a);
                                                    padre.AddHijo(b);
                                                    RESULT = padre; :}
```

El proyecto se manejó con 4 package que básicamente son apartados ordenados de su contenido mostrado a continuación.

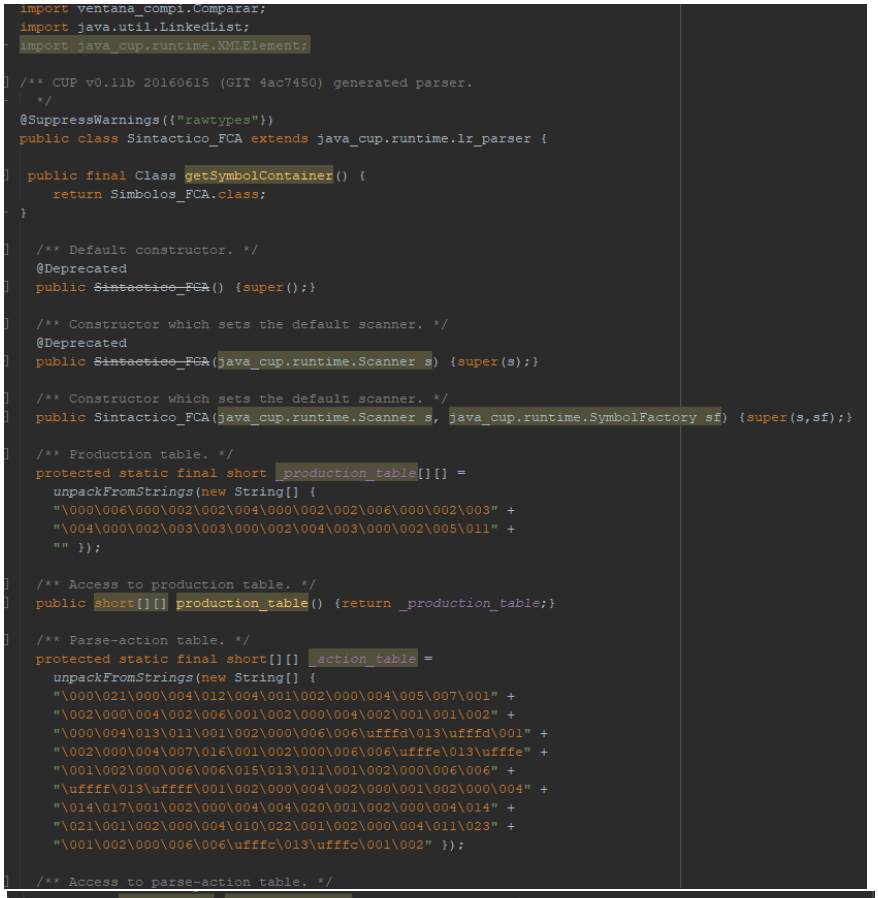


En la package de Analizadores se encuentran los analizadores FCA y los JS, con sus respectivos analizadores JFLEX y CUP para su completo uso.

Package Generadores se encuentran los generadores léxicos y sintácticos correspondientes.

Package Ventana_compi, se encuentran los archivos Java donde se desarrolla el árbol, archivo, la clase de la función compare, editor, manejo de errores, evaluar, métodos, Nodos, y la interfaz gráfica del proyecto con sus funciones programadas.

Este es un vistazo de el archivo autogenerado del sintáctico de FCA



```
import ventana_compi.Comparar;
import java.util.LinkedList;
import java_cup.runtime.XMLElement;

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

 public final Class getSymbolContainer() {
    return Simbolos_FCA.class;
}

  /** Default constructor. */
  @Deprecated
  public Sintactico_FCA() {super();}

  /** Constructor which sets the default scanner. */
  @Deprecated
  public Sintactico_FCA(java_cup.runtime.Scanner s) {super(s);}

  /** Constructor which sets the default scanner. */
  public Sintactico_FCA(java_cup.runtime.Scanner s, java_cup.runtime.SymbolFactory sf) {super(s,sf);}

  /** Production table. */
  protected static final short _production_table[][] =
    unpackFromStrings(new String[] {
    "\000\006\000\002\002\004\000\002\002\006\000\002\003" +
    "\004\000\002\003\003\000\002\004\003\000\002\005\011" +
    "" });

  /** Access to production table. */
  public short[][] production_table() {return _production_table;}

  /** Parse-action table. */
  protected static final short[][] _action_table =
    unpackFromStrings(new String[] {
    "\000\021\000\004\012\004\001\002\000\004\005\007\001" +
    "\002\000\004\002\006\001\002\000\004\002\001\001\002" +
    "\000\004\013\011\001\002\000\006\006\ufffd\013\ufffd\001" +
    "\002\000\004\007\016\001\002\000\006\006\ufffe\013\ufffe" +
    "\001\002\000\006\006\015\013\011\001\002\000\006\006" +
    "\uffff\013\uffff\001\002\000\004\002\000\001\002\000\004" +
    "\014\017\001\002\000\004\004\020\001\002\000\004\014" +
    "\021\001\002\000\004\010\022\001\002\000\004\011\023" +
    "\001\002\000\006\006\ufffc\013\ufffc\001\002" });

  /** Access to parse-action table. */
  public short[][] action_table() {return _action_table;}

  /** <code>reduce_goto</code> table. */
  protected static final short[][] _reduce_table =
    unpackFromStrings(new String[] {
    "\000\021\000\004\002\004\001\001\000\002\001\001\000" +
    "\002\001\001\000\002\001\001\000\010\003\012\004\011" +
    "\005\007\001\001\000\002\001\001\000\002\001\001\000" +
    "\002\001\001\000\006\004\013\005\007\001\001\000\002" +
    "\001\001\000\002\001\001\000\002\001\001\000\002\001" +
    "\001\000\002\001\001\000\002\001\001\000\002\001\001" +
    "\000\002\001\001" });

  /** Access to <code>reduce_goto</code> table. */
  public short[][] reduce_table() {return _reduce_table;}

  /** Instance of action encapsulation class. */
  protected CUP$Sintactico_FCA$actions action_obj;

  /** Action encapsulation object initializer. */
  protected void init_actions()
    {
      action_obj = new CUP$Sintactico_FCA$actions(this);
    }

  /** Invoke a user supplied parse action. */
  public java_cup.runtime.Symbol do_action(
    int                        act_num,
    java_cup.runtime.lr_parser parser,
    java.util.Stack            stack,
    int                        top)
    throws java.lang.Exception
  {
    /* call code in generated class */
    return action_obj.CUP$Sintactico_FCA$do_action(act_num, parser, stack, top);
  }

  /** Indicates start state. */
  public int start_state() {return 0;}
  /** Indicates start production. */
  public int start_production() {return 0;}

  /** <code>EOF</code> Symbol index. */
  public int EOF_sym() {return 0;}
```

UNA VISTA DEL AUTOGENERADO DE SIMBOLOS

```java
public class sope_2 {
    public static void main(String[] args)
    {
        String opciones[] = new String[7];

        //Seleccionamos la opción de dirección de destino
        opciones[0] = "-destdir";

        //Le damos la dirección, carpeta donde se va a generar el parser.java & el simbolosxxx.java
        opciones[1] = "src/analizadores";

        //Seleccionamos la opción de nombre de archivo simbolos
        opciones[2] = "-symbols";

        //Le damos el nombre que queremos que tenga
        opciones[3] = "Simbolosjs";

        //Seleccionamos la opcion de clase parser
        opciones[4] = "-parser";

        //Le damos nombre a esa clase del paso anterior
        opciones[5] = "Sintacticojs";

        //Le decimos donde se encuentra el archivo .cup
        opciones[6] = "src/analizadores/Analizador_sintactico_js.cup";
        try
        {
            java_cup.Main.main(opciones);
        }
        catch (Exception ex)
        {
            System.out.print(ex);
        }
    }
}
```

INTERFAZ PROGRAMADA Y PARTE DE LA FUNCIÓN COMPARE

```java
public void comparar_archivos_proyectos(String ruta_proy1, String ruta_proy2){

    try{
        DirectoryStream<Path> stream_p1 = Files.newDirectoryStream(Paths.get(ruta_proy1), "*.js");

       //Recorremos los archivos del proyecto 1
        for (Path file_p1: stream_p1) {
            DirectoryStream<Path> stream_p2 = Files.newDirectoryStream(Paths.get(ruta_proy2), "*.js");

            String nombre_archivo1 = file_p1.getFileName().toString();
            File archivo = new File (file_p1.toString());
            FileReader fr = new FileReader (archivo);
            Archivo nuevo_archivo1 = null;
            Archivo nuevo_archivo2 = null;
            //Vamos a comparar el archivo del proyecto1 con los archivos de la carpeta2 para saber si se llaman igual
            for(Path file_p2 : stream_p2){
                String nombre_archivo2 = file_p2.getFileName().toString();
                File archivo2 = new File (file_p2.toString());
                FileReader fr2 = new FileReader (archivo2);
                //Si se llaman igual se comienza el proceso para analizar copias
                if(nombre_archivo1.equals(nombre_archivo2)){
                    System.out.println("Los nombres son iguales, vamos a comparar --> " + file_p1.getFileName());
                    //--> 1ero vamos a analizar el archivo1 del proyecto 1
                    try{

                        System.out.println("---------- " + nombre_archivo1 + " en PROYECTO 1 ---------- ");
                        Nodo raiz = null;
                        //Mandamos a analizar el archivo del proyecto 1
                        Sintacticojs parse = new Sintacticojs(new Analizador_lexico_js(new BufferedReader(fr)));
                        parse.parse();

                        raiz = parse.getRaiz();
                        if(raiz == null){
                            System.out.println("No se genero bien el arbol");
                        }else{

                            nuevo_archivo1 = new Archivo(nombre_archivo1, new LinkedList<>(), new LinkedList<>(), new L
```