UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

ESCUELA DE CIENCIAS Y SISTEMAS

ORGANIZACIÓN DE LENGUAJES Y COMPILADORES 1

PRIMER SEMESTRE 2,022

PROYECTO 1

Manual Técnico

Paulo Vlademir Argueta Ortega

202010751

09/03/2021

Programa utilizado

IntelliJ IDEA es un entorno de desarrollo integrado para el desarrollo de programas informáticos. Es desarrollado por JetBrains, y está disponible en dos ediciones: edición para la comunidad y edición comercial.

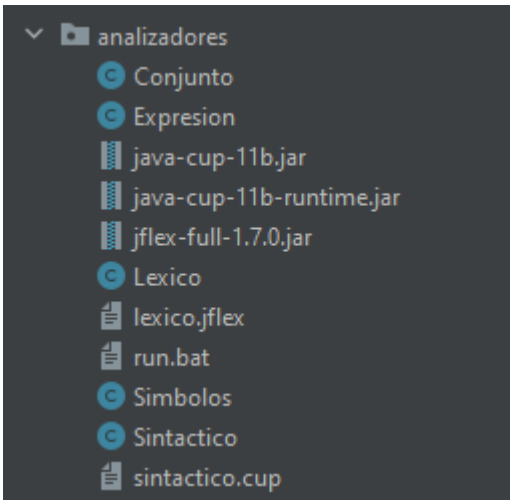
Lenguaje utilizado

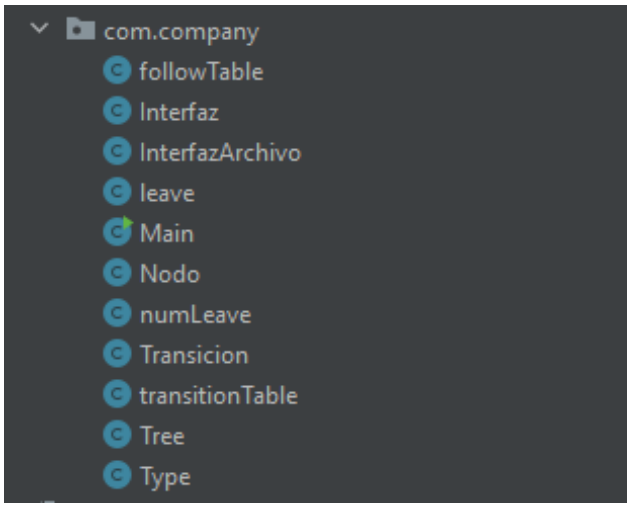
Java es un lenguaje de programación y una plataforma informática que fue comercializada por primera vez en 1995 por Sun Microsystems. Hay muchas aplicaciones y sitios web que no funcionarán, probablemente, a menos que tengan Java instalado, y cada día se crean más. Java es rápido, seguro y fiable

Sobre el código

Clases

El proyecto está conformado por las siguientes clases siendo la principal llamada main. Estas clases ayudan a ordenar y ejecutar el código de una mejor manera, en las de la carpeta de analizadores se encuentran las clases de flex y cup que sirven para analizar el texto de entrada mientras que afuera se encuentran las que realizan los árboles, tablas y autómatas.





Interfaz grafica

Para comenzar el código lo primero escrito fue la interfaz gráfica con sus botones y respectivas funciones logrando darle funcionalidad a dicha interfaz.

```
public Interfaz (){
    this.setTitle("");
    this.setBounds( x: 600,  y: 250,  width: 1180,  height: 610);
    this.setLayout(null);
    this.getContentPane().setBackground(Color.darkGray);

    archivo = new JButton( text: "Archivo");
    archivo.setBounds( x: 15, y: 15, width: 85, height: 20);
    archivo.addActionListener( l: this);

    area1 = new JTextArea();
    JScrollPane scroll = new JScrollPane(area1);
    scroll.setVerticalScrollBarPolicy(ScrollPaneConstants.VERTICAL_SCROLLBAR_ALWAYS);
    scroll.setBounds( x: 20,  y: 45,  width: 700,  height: 300);

    area2 = new JTextArea();
    area2.setBounds( x: 20,  y: 370,  width: 700,  height: 125);

    area3 = new JTextPane();
    JScrollPane scroll3 = new JScrollPane(area3);
    scroll3.setVerticalScrollBarPolicy(ScrollPaneConstants.VERTICAL_SCROLLBAR_ALWAYS);
    scroll3.setBounds( x: 740,  y: 45,  width: 400,  height: 400);


    generarAutomatas = new JButton( text: "Generar Automatas");
    generarAutomatas.setBounds( x: 20, y: 525, width: 200, height: 25);
    generarAutomatas.addActionListener( l: this);

    generarEntradas = new JButton( text: "Analizar Entradas");
    generarEntradas.setBounds( x: 250, y: 525, width: 200, height: 25);
    generarEntradas.addActionListener( l: this);

    verImagen = new JButton( text: "Ver Imagen");
    verImagen.setBounds( x: 880, y: 470, width: 100, height: 20);
    verImagen.addActionListener( l: this);
```

Principales Funciones del Código:

Método del Árbol:

```
//==========================================================Metodo del Arbol==========================================================
        ArrayList<Nodo> leaves = new ArrayList();
        ArrayList<ArrayList> table = new ArrayList();

        Tree arbol = new Tree(expresiones.get(i).getExpresionArray(), leaves, table); // CREA EL ARBOL
        Nodo raiz = arbol.getRoot();

        raiz.getNode(); // DETERMINA SI LOS NODOS SON ANULABLES, SUS PRIMEROS Y ULTIMOS
        raiz.follow();

        String graphvizArbol = Tree.contenido+"}";
        EscribirArchivo(graphvizArbol, ruta: "./reportes/arboles_202010751/Arbol"+(i+1)+".dot");
        ProcessBuilder proceso0;
        proceso0 = new ProcessBuilder( ...command: "dot", "-Tpng", "-o","./reportes/arboles_202010751/Arbol"+(i+1)+".png","./reportes/arboles_202010751/Arbol"+(i+1)+".dot");
        proceso0.redirectErrorStream(true);
        try {
            proceso0.start();
        } catch (IOException e) {
            e.printStackTrace();
        }
```

Este método es el primer método en ejecutarse al analizar el texto ya que crea el árbol, determina sus nodos anulables además de sus primeros y últimos.

Al crear el árbol se aprovecha a ir concatenando un string para generar el graphviz y poder visualizarlo.

Tabla de siguientes:

```
    //System.out.println("============================TABLA SIGUIENTES============================");
    followTable ft = new followTable();
    String graphviz = ft.graphviz(table);
    EscribirArchivo(graphviz, ruta: "./reportes/siguientes_202010751/Tabla"+(i+1)+".dot");
    ProcessBuilder proceso;
    proceso = new ProcessBuilder( ...command: "dot", "-Tpng", "-o","./reportes/siguientes_202010751/Tabla"+(i+1)+".png","./reportes/siguientes_202010751/Tabla"+(i+1)+".dot");
    proceso.redirectErrorStream(true);
    try {
        proceso.start();
    } catch (IOException e) {
        e.printStackTrace();
    }
```

Este método le sigue al método del árbol, una vez creado el árbol accedemos a cada uno de sus nodos y dentro de ellos a cada uno de sus siguientes parra poder graficarla por medio de un string concatenado.

Tabla de Transiciones:

```
//System.out.println("============================TABLA TRANSICIONES============================");
graphviz = tran.graphviz(table);
EscribirArchivo(graphviz, ruta: "./reportes/transiciones_202010751/Tabla"+(i+1)+".dot");
ProcessBuilder proceso2;
proceso2 = new ProcessBuilder( ...command: "dot", "-Tpng", "-o","./reportes/transiciones_202010751/Tabla"+(i+1)+".png","./reportes/transiciones_202010751/Tabla"+(i+1)+".dot");
proceso2.redirectErrorStream(true);
try {
    proceso2.start();
} catch (IOException e) {
    e.printStackTrace();
}
```

Este método realiza algo parecido a la tabla de siguientes. Accedemos a los nodos, vemos las transiciones y las graficamos en una tabla.

AFD:

```
//System.out.println("============================AFD============================");
String automataGraphviz = tran.Automata();
EscribirArchivo(automataGraphviz, ruta: "./reportes/afd_202010751/Automata"+(i+1)+".dot");
ProcessBuilder proceso3;
proceso3 = new ProcessBuilder( ...command: "dot", "-Tpng", "-o","./reportes/afd_202010751/Automata"+(i+1)+".png","./reportes/afd_202010751/Automata"+(i+1)+".dot");
proceso3.redirectErrorStream(true);
try {
    proceso3.start();
} catch (IOException e) {
    e.printStackTrace();
}
```

Gramática Libre de Contexto:

No terminales:

INICIO

INICIO2

CONJUNTOS

COMBINACIONES

COMASLETRA

COMASNUMERO

EXPRESIONES

COMBINACIONESEXPRESIONES

PORCENTAJES

ENTRADAS

Terminales:

T = { llavea, llavec, dospuntos, puntocoma, flecha, punto, asterisco, coma, or, mas, guiononduladó, interrogación, conj, porcentajes, comentariolineas, cadena, comentario, numero, letra, id, símbolos, caracterEspecial }

Producciones:

INICIO -> llavea INICIO2 llavec

INICIO2 -> CONJUNTOS

| INICIO2 CONJUNTOS

| INICIO2 EXPRESIONES

| EXPRESIONES

| INICIO2 PORCENTAJES

| PORCENTAJES

| INICIO2 ENTRADAS

| ENTRADAS

| error

CONJUNTOS -> conj dospuntos id flechaCOMBINACIONES puntocoma

COMBINACIONES -> letra guiononduladó letra

| numero guiononduladó numero

| simbolos guiononduladó simbolos

| llavea guiononduladó llavec

| letra COMASLETRA

| numero COMASLETRA

| cadena COMASLETRA

| caracterEspecial COMASLETRA

COMASLETRA -> coma letra COMASLETRA

| coma letra

| coma caracterEspecial COMASLETRA

| coma caracterEspecial

| coma cadena COMASLETRA

| coma cadena

| coma numero COMASLETRA

| coma numero

EXPRESIONES -> id flecha COMBINACIONESEXPRESIONES puntocoma

COMBINACIONESEXPRESIONES -> punto COMBINACIONESEXPRESIONES

| punto

| llavea COMBINACIONESEXPRESIONES

| llavea

| llavec COMBINACIONESEXPRESIONES

| llavec

| id COMBINACIONESEXPRESIONES

| id

| asterisco COMBINACIONESEXPRESIONES

| asterisco

| interrogacion COMBINACIONESEXPRESIONES

| interrogacion

| or COMBINACIONESEXPRESIONES

| or

| cadena COMBINACIONESEXPRESIONES

| cadena

| mas COMBINACIONESEXPRESIONES

| mas

| caracterEspecial COMBINACIONESEXPRESIONES

| caracterEspecial

PORCENTAJES -> porcentajes

ENTRADAS -> id dospuntos cadena puntocoma

Símbolo inicial:

S = INICIO