

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
FACULTAD DE INGENIERÍA
LENGUAJES FORMALES DE PROGRAMACION
CATEDRÁTICO: ING. DAMARIS CAMPOS
TUTOR ACADÉMICO: LUISA MARIA ORTIZ



MANUAL TÉCNICO **PROYECTO 1**

Álvaro Gabriel Ceballos

Gil CARNÉ:

202300673

SECCIÓN: A-

GUATEMALA, 26 DE MARZO DEL 2,025

ÍNDICE

ÍNDICE	1
INTRODUCCIÓN	1
OBJETIVOS	3
1. GENERAL	3
2. ESPECÍFICOS	3
ALCANCES DEL SISTEMA	4
ESPECIFICACIÓN TÉCNICA	5
• REQUISITOS DE HARDWARE	6
• REQUISITOS DE SOFTWARE	6
Clase Tokens	7
7	
Clase ErrorLexico	8
Analizador Lexico	9
Automata Individual	14
Procedimiento GraficarAutomata	14
15	
Menu.java	17
Procedimiento procesarAutomata	20

INTRODUCCIÓN

Este manual se encontrará de forma más desarrollada la explicación del código del proyecto, con el fin de que un programador pueda entender el código del programa, y así poder implementarlo si así lo desea.

Este manual le permite al programador ver la lógica del programa de una forma mucho más comprensible, lo que permitirá un mejor desarrollo de la aplicación

en el caso de que el programador desee implementar alguna función de este programa en el suyo.

OBJETIVOS

1. GENERAL

- 1.1. Ayudar al programador a poder entender el programa de una mejor manera

2. ESPECÍFICOS

- 2.1. Explicar la lógica de las funcionalidades del programa de una forma más sencilla
- 2.2. Brindar información necesaria para la comprensión del proyecto de una forma más técnica

ALCANCES DEL SISTEMA

Este manual tiene el objetivo de explicar de una forma mucho más explícita las funcionalidades del código del proyecto 1, con el fin de que el programador sea capaz de entender correctamente las líneas de código utilizadas para el desarrollo del programa.

También se tiene como objetivo que el programador logre entender mejor la lógica del programa, y que de esta forma el programador sea capaz de implementar varias funcionalidades de este programa en su código.

En base a este manual, el programador no sólo será capaz de replicar el programa, sino de mejorarlo, implementando múltiples funcionalidades nuevas.

ESPECIFICACIÓN TÉCNICA

- **REQUISITOS DE HARDWARE**

Para poder correr este programa, es necesario que el usuario tenga instalado java con NetBeans para el correcto desarrollo de la practica

Se debe de contar con un sistema operativo Windows de 64 bits, con una memoria RAM recomendada de 8GB, ya que con esto se garantiza de que el programa pueda correr sin ningún tipo de dificultad.

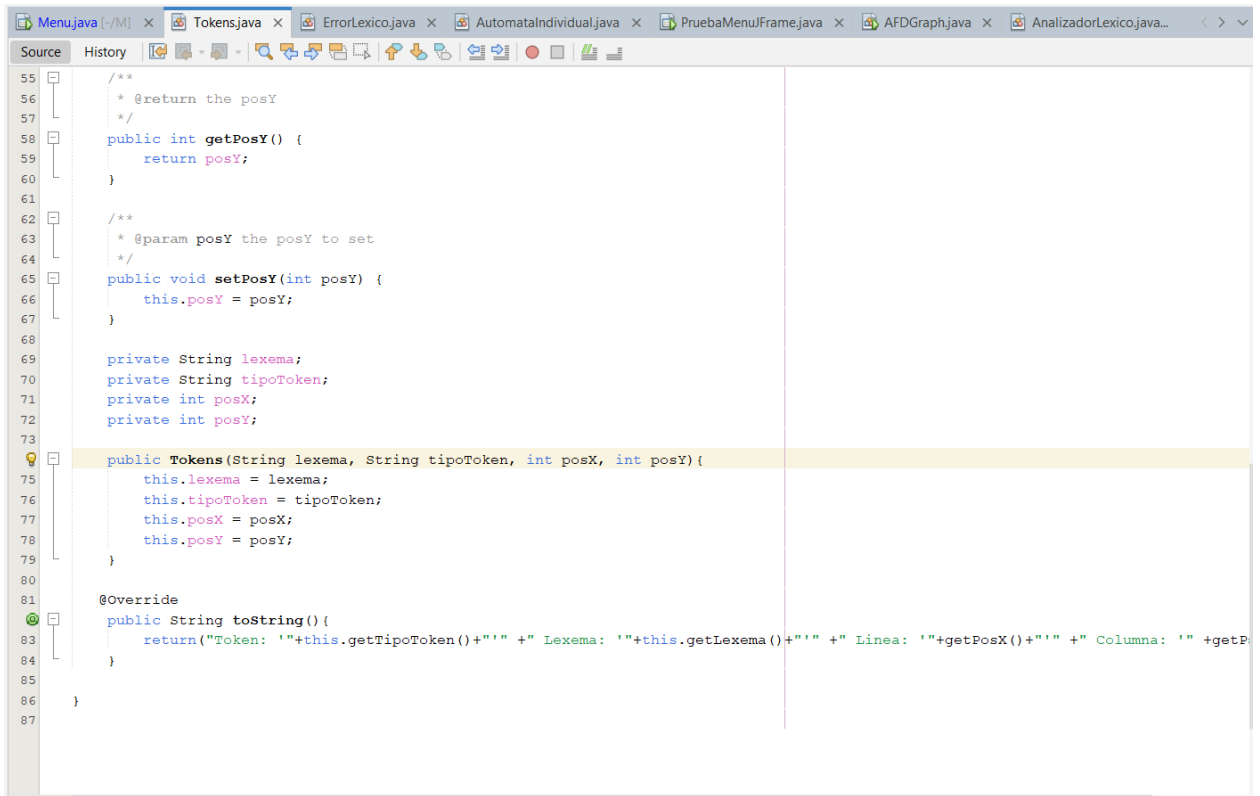
Se debe de contar con GraphViz instalado para la correcta generación de los gráficos, para intalarlo debe de instalar el ejecutable correspondiente a su sistema operativo que se encuentra en: <https://graphviz.org/download/>

- **REQUISITOS DE SOFTWARE**

- Debe contar con un sistema operativo medianamente moderno, como Windows 10, esto con el fin de que a la hora de desarrollar el proyecto no tenga algún tipo de error.

EXPLICACION DEL CODIGO

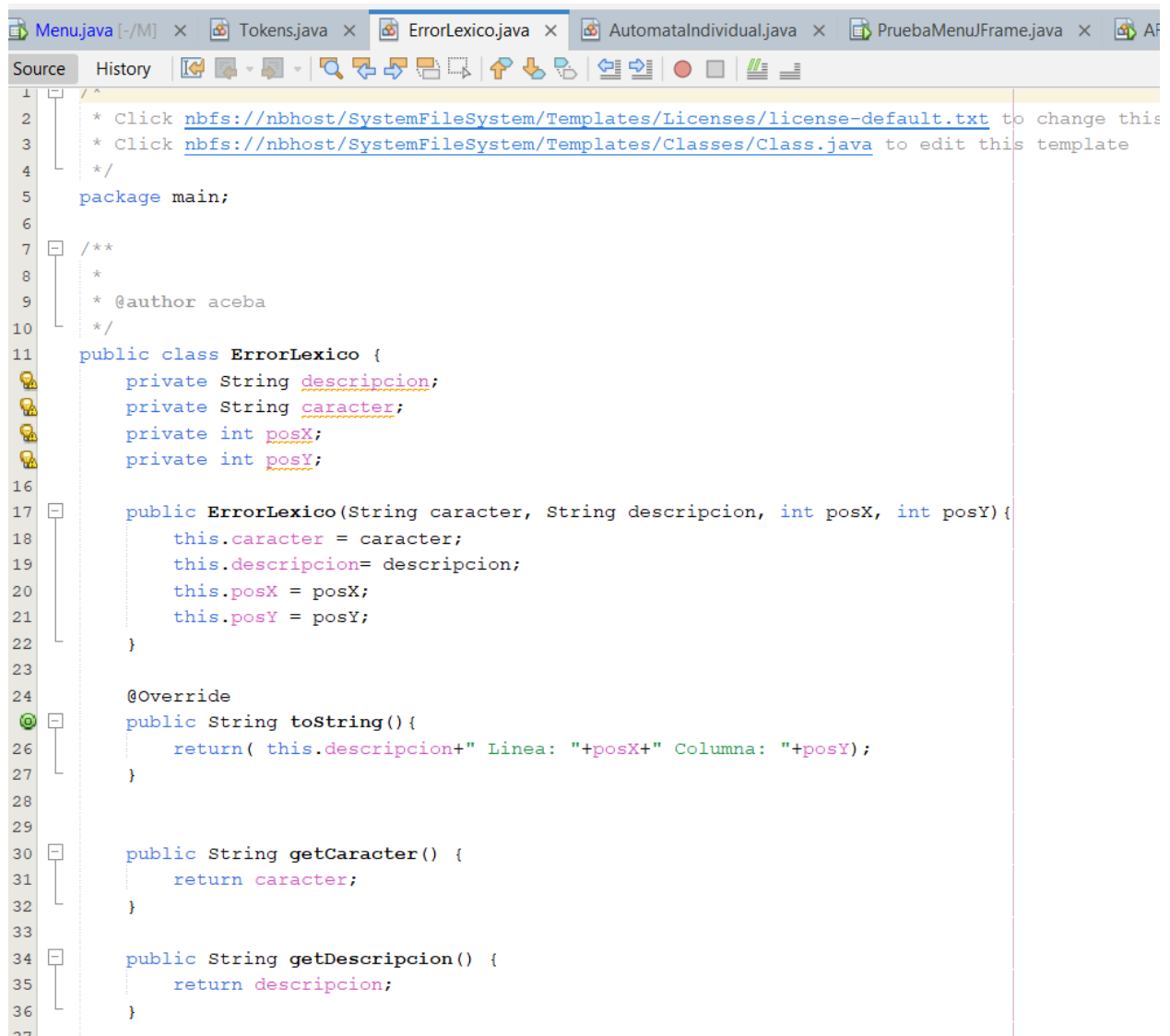
Clase Tokens



```
55  /**
56   * @return the posY
57   */
58  public int getPosY() {
59      return posY;
60  }
61
62  /**
63   * @param posY the posY to set
64   */
65  public void setPosY(int posY) {
66      this.posY = posY;
67  }
68
69  private String lexema;
70  private String tipoToken;
71  private int posX;
72  private int posY;
73
74  public Tokens(String lexema, String tipoToken, int posX, int posY) {
75      this.lexema = lexema;
76      this.tipoToken = tipoToken;
77      this.posX = posX;
78      this.posY = posY;
79  }
80
81  @Override
82  public String toString() {
83      return "Token: '" + this.getTipoToken() + "' Lexema: '" + this.getLexema() + "' Linea: '" + getPosY() + "' Columna: '" + getPosX() + "'";
84  }
85
86  }
87
```

Clase de tóquense definen todos los atributos que tendrá nuestro token como la posición en x posición en que se llama el tipo de token junto con su constructor y sus getters y setters

Clase ErrorLexico



```
1  * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this
2
3  * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
4  */
5  package main;
6
7  /**
8   *
9   * @author aceba
10  */
11  public class ErrorLexico {
12      private String descripcion;
13      private String caracter;
14      private int posX;
15      private int posY;
16
17      public ErrorLexico(String caracter, String descripcion, int posX, int posY){
18          this.caracter = caracter;
19          this.descripcion= descripcion;
20          this.posX = posX;
21          this.posY = posY;
22      }
23
24      @Override
25      public String toString(){
26          return( this.descripcion+" Linea: "+posX+" Columna: "+posY);
27      }
28
29
30      public String getCaracter() {
31          return caracter;
32      }
33
34      public String getDescripcion() {
35          return descripcion;
36      }
37  }
```

De error léxico se define en todos los atributos de la clase como la descripción el carácter la posición en XY la posición e junto con su constructor y getters y settlers

Analizador Lexico

```
*
* @author aceba
*/
public class AnalizadorLexico {
    private List<Tokens> ListaTokens;
    private List<ErrorLexico> ListaErrores;
    private int posX;
    private int posY;
    private String buffer;
    private int estado;
    private int iArchivo;

    public AnalizadorLexico(){
        this.ListaTokens = new ArrayList<>();
        this.ListaErrores = new ArrayList<>();
    }

    public void nuevoToken(String caracter, String token, int posX, int posY){
        this.ListaTokens.add(new Tokens(caracter, token, posX, posY));
        this.buffer = "";
    }

    public void nuevoError(String caracter, int posX, int posY){
        this.ListaErrores.add(new ErrorLexico(caracter, "Caracter "+caracter+" no reconocido en el lenguaje", posX, posY));
        this.buffer = "";
    }

    public void analizarArchivo(StringBuilder cadena){
        this.ListaTokens.clear();
        this.ListaErrores.clear();
        this.estado = 0;
        this.iArchivo = 0;
        this.buffer = "";
    }
}
```

Como se puede observar en nuestro analizador léxico primeramente hemos definido distintos arrays sobre los cuales se irán almacenando tanto los tokens como los errores, que luego servirán para la tabla de reportes de errores y de tokens.

```

while(this.iArchivo < cadena.length()){
    if(this.estado == 0){
        q0(cadena.charAt(this.iArchivo));
    }
    else if(this.estado == 4){
        q4(cadena.charAt(this.iArchivo));
    }
    else if(this.estado == 1){
        q1(cadena.charAt(this.iArchivo));
    }
    else if(this.estado == 3){
        q3(cadena.charAt(this.iArchivo));
    }
    iArchivo++;
}

//Este estado es el iniciar, que a su vez es capaz de reconocer simbolos
public void q0(char caracter){
    if (caracter == '{') {
        this.nuevoToken(String.valueOf(caracter), "LlaveAbrir", this.posX, this.posY);
        this.posY++;
    } else if (caracter == '}') {
        this.nuevoToken(String.valueOf(caracter), "LlaveCerrar", this.posX, this.posY);
        this.posY++;
    } else if (caracter == '[') {
        this.nuevoToken(String.valueOf(caracter), "CorcheteAbrir", this.posX, this.posY);
        this.posY++;
    } else if (caracter == ']') {
        this.nuevoToken(String.valueOf(caracter), "CorcheteCerrar", this.posX, this.posY);
        this.posY++;
    } else if (caracter == ':') {
        this.nuevoToken(String.valueOf(caracter), "DosPuntos", this.posX, this.posY);
        this.posY++;
    } else if (caracter == ',') {
        this.nuevoToken(String.valueOf(caracter), "Coma", this.posX, this.posY);
    }
}

```

Dicha imagen se encuentra en los estados que tendrá el analizador léxico, indicándonos a qué estado se debe de ir movilizand dependiendo de cómo se trabaje.

```

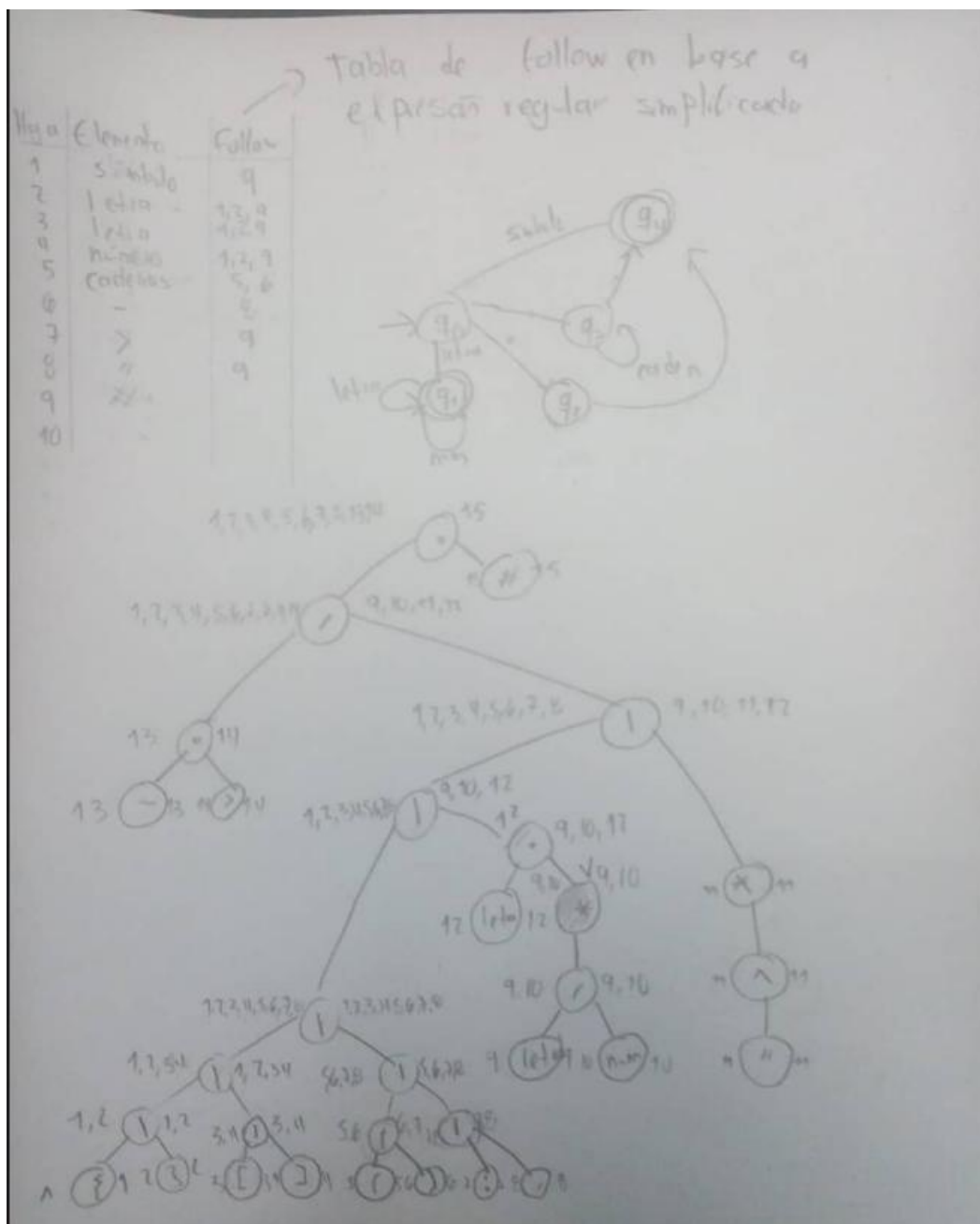
65 }
71 this.nuevoToken(String.valueOf(caracter), "LlaveAbrir", this.posX, this.posY);
72 this.posY++;
73 } else if (caracter == '}') {
74     this.nuevoToken(String.valueOf(caracter), "LlaveCerrar", this.posX, this.posY);
75     this.posY++;
76 } else if (caracter == '[') {
77     this.nuevoToken(String.valueOf(caracter), "CorcheteAbrir", this.posX, this.posY);
78     this.posY++;
79 } else if (caracter == ']') {
80     this.nuevoToken(String.valueOf(caracter), "CorcheteCerrar", this.posX, this.posY);
81     this.posY++;
82 } else if (caracter == ':') {
83     this.nuevoToken(String.valueOf(caracter), "DosPuntos", this.posX, this.posY);
84     this.posY++;
85 } else if (caracter == ',') {
86     this.nuevoToken(String.valueOf(caracter), "Coma", this.posX, this.posY);
87     this.posY++;
88 } else if (caracter == '(') {
89     this.nuevoToken(String.valueOf(caracter), "ParentesisAbrir", this.posX, this.posY);
90     this.posY++;
91 } else if (caracter == ')') {
92     this.nuevoToken(String.valueOf(caracter), "ParentesisCerrar", this.posX, this.posY);
93     this.posY++;
94 } else if (caracter == '=') {
95     this.nuevoToken(String.valueOf(caracter), "SignoIgual", this.posX, this.posY);
96     this.posY++;
97 } else if (caracter == '\n') {
98     this.posX++;
99     this.posY = 0;
100 } else if (caracter == ' ') {
101     this.posY++;
102 } else if (caracter == '\t') {
103     this.posY += 4;
104 } else if (caracter == '-') {
105     this.buffer += caracter;
106     this.posY++;
107     this.estado = 4;

```

Como se puede observar este es el analizador léxico, el cual se le ha hecho un previo análisis, ya que antes de se hizo una expresión regular y sobre esa expresión regular se trabajó un método del árbol junto con su tabla de transiciones y sus respectivos nodos, hoy y sobre esto que trabajamos ya se definieron todos los estados necesarios para poder hacer funcionar correctamente el programa en base al archivo de entrada que obtuvimos.

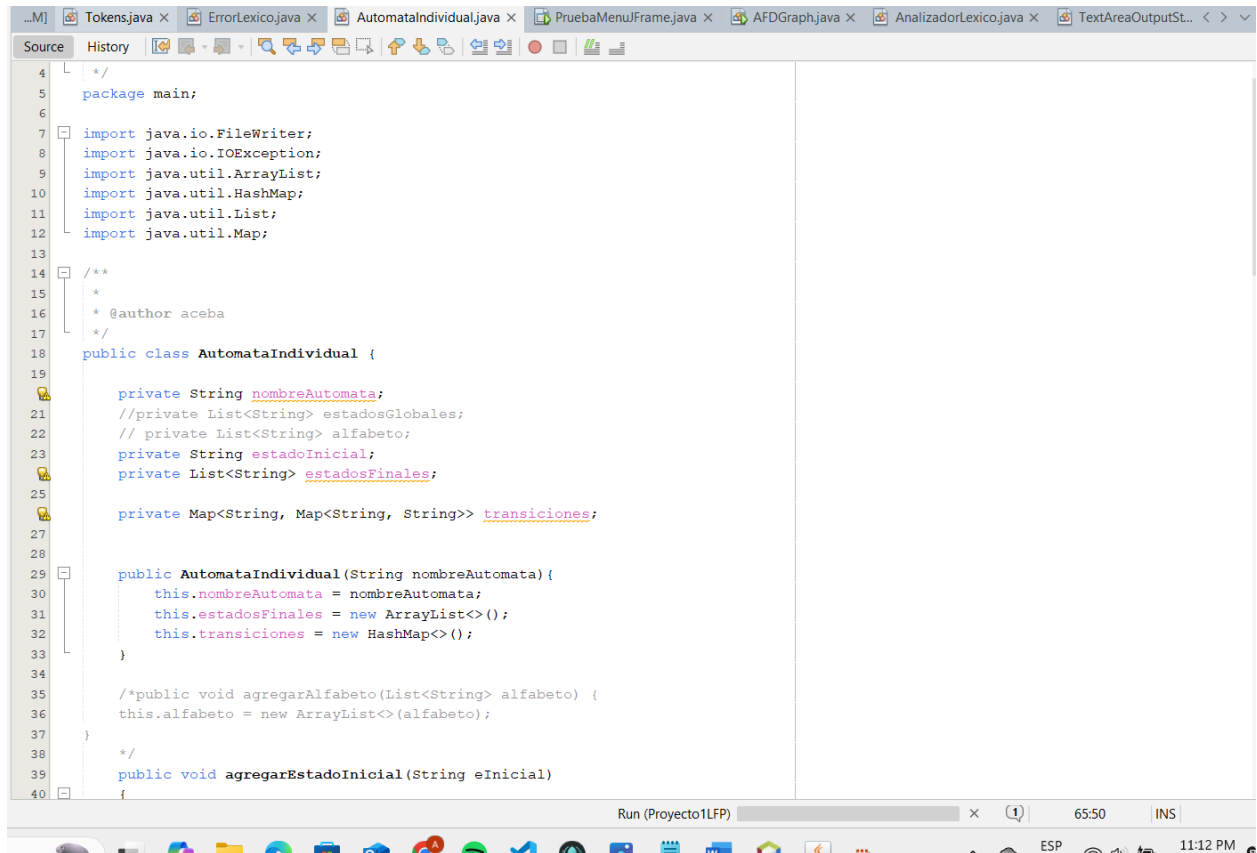
Símbolos = $\{ \{ \} [] : | () \} =$
 Identificador = $(\text{letra}(\text{letra}|\text{número})^*) \#$
 flecha = $(\rightarrow) \#$
 letra = $[a-z]$
 número = $[0-9]$
 cadenas = $"[^\wedge"]^*" \#$
 Expresión regular del proyecto expresión
regular
simplificada
 $((\{ \} [] : | ()) | (\text{letra}(\text{letra}|\text{número})^*) | "[^\wedge"]^* | (\rightarrow)) \#$
 Nombre: Alvaro Gabriel Oballos G. I
 Correo: 727300673

Como se puede observar se ha hecho la gramática de nuestra aplicación, el cual contiene símbolos identificadores flechas cadenas de texto letras números entre otros.



Luego sobre esa expresión regular se realizó un método del árbol identificando todos los nodos iniciales y finales, para luego hacer la tabla de transición y un autómata finito determinista para poder realizar el analizador léxico.

Automata Individual



```
4  /*
5  package main;
6
7  import java.io.FileWriter;
8  import java.io.IOException;
9  import java.util.ArrayList;
10 import java.util.HashMap;
11 import java.util.List;
12 import java.util.Map;
13
14 /**
15  *
16  * @author aceba
17  */
18 public class AutomataIndividual {
19
20     private String nombreAutomata;
21     //private List<String> estadosGlobales;
22     // private List<String> alfabeto;
23     private String estadoInicial;
24     private List<String> estadosFinales;
25
26     private Map<String, Map<String, String>> transiciones;
27
28
29     public AutomataIndividual(String nombreAutomata) {
30         this.nombreAutomata = nombreAutomata;
31         this.estadosFinales = new ArrayList<>();
32         this.transiciones = new HashMap<>();
33     }
34
35     /*public void agregarAlfabeto(List<String> alfabeto) {
36         this.alfabeto = new ArrayList<>(alfabeto);
37     }
38
39     */
40     public void agregarEstadoInicial(String eInicial)
41     {
```

Como se puede observar esta es nuestra clase de autómata individual que es donde se irán almacenando los distintos autómatas finitos deterministas que se vayan encontrando en el sistema al momento de graficarlos, como se puede observar hoy tiene un string del nombre del autómata junto con el del estado inicial los estados finales los cuales serían una lista ya que pueden haber varios y por último en las transiciones tiene un hash map y ese hash map en su primer valor tendrá el nombre y en su segundo valor tendrá otro #el cual contendrá el valor del alfabeto sobre el cual se trabajará y el estado del destino al que se quiere llegar, es por eso que se ha trabajado con un hash map entro de otro HashMap

Procedimiento GraficarAutomata

```

6   }
7
8   public void GraficandoAutomata(){
9       String ArchivoDOTGenerado = "C:\\Users\\aceba\\OneDrive\\Desktop\\Practical\\-LFP-202300673-\\Proyecto1\\AFD.dot";
10      String ArchivoPNGGenerado = "C:\\Users\\aceba\\OneDrive\\Desktop\\Practical\\-LFP-202300673-\\Proyecto1\\AFD.png";
11
12      try{
13          FileWriter escritorArchivoDOT = new FileWriter(ArchivoDOTGenerado);
14          escritorArchivoDOT.write("digraph G {\n");
15          escritorArchivoDOT.write("rankdir=LR;\n");
16          escritorArchivoDOT.write("node [shape=circle];\n");
17          String transitionString = "";
18          String infoAuto = "";
19
20          for(Map.Entry<String, Map<String, String>> entry : this.transiciones.entrySet()){
21              String sInicial = entry.getKey();
22              if (sInicial.equals(estadoInicial)) {
23                  infoAuto += "invisible_start [shape=point, width=0.1, height=0.1]\n";
24                  infoAuto += "invisible_start -> " + sInicial + " [style=bold]\n";
25              }
26              if (this.estadosFinales.contains(sInicial)) {
27                  infoAuto += sInicial + " [shape=doublecircle]\n";
28              }
29              Map<String, String> transicion = entry.getValue();
30
31              for(Map.Entry<String, String> entry2: transicion.entrySet()){
32                  String etiqueta = entry2.getKey();
33
34                  String estadoFinal = entry2.getValue();
35
36                  transitionString += sInicial + "->" + estadoFinal + "[label="+etiqueta+"]\n";
37
38                  if(this.estadosFinales.contains(estadoFinal)){
39                      infoAuto+=estadoFinal+"[shape=doublecircle]\n";
40                  }else{
41                      infoAuto+=estadoFinal+"[shape=circle]\n";
42                  }
43              }
44          }
45      }
46  }

```

Este será nuestro procedimiento que nos servirá para poder graficar nuestro autómata, primero se inician 2 variables tipo string que será donde se dirá almacenar tanto el archivo. Dot como el archivo PNG. Primeramente se generará el archivo. 2 y una vez generado dicho archivo este se pasará un PNG este PNG se pasará a nuestra interfaz gráfica

Luego hoy seguirá iterando sobre nuestro HashMap de transiciones, tanto los estados iniciales y finales, indicando que al encontrarse un estado inicial este se hoy hará un círculo junto con una flecha, pero si es un estado final este se creará con un círculo doble.

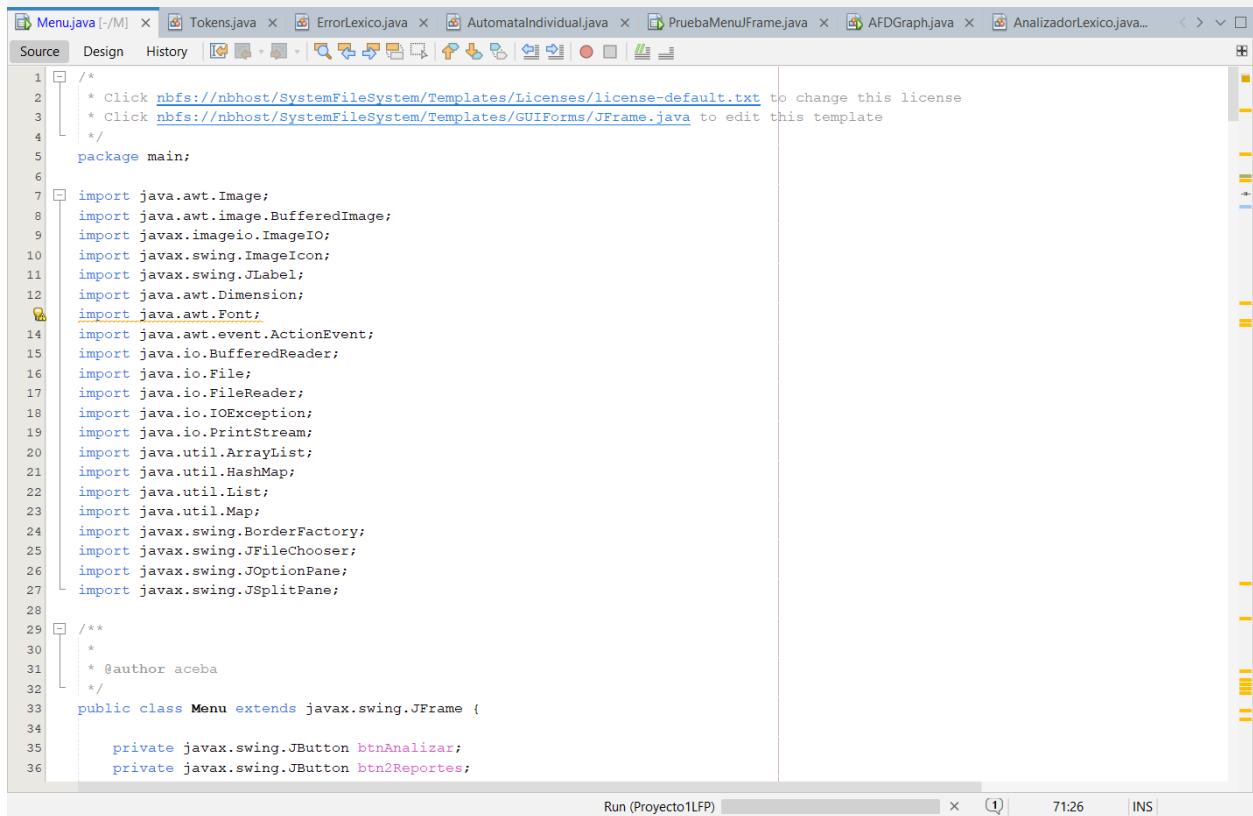
```

-         }
-         escritorArchivoDOT.write(transitionString);
-         escritorArchivoDOT.write(infoAuto);
-         escritorArchivoDOT.write("{}");
-         escritorArchivoDOT.close();
-
-     } catch (IOException e) {
-         System.out.println("Error al escribir el archivo.");
-         e.printStackTrace();
-     }
-
-     String[] comando = {"dot", "-Tpng", ArchivoDOTGenerado, "-o", ArchivoPNGGenerado};
-
-     try{
-         ProcessBuilder builder = new ProcessBuilder(comando);
-         builder.inheritIO();
-         Process proceso = builder.start();
-         int exitCode = proceso.waitFor();
-
-         if(exitCode == 0){
-             System.out.println("Conversion completada");
-         }else{
-             System.err.println("Error en la conversion de codigo");
-         }
-     } catch (IOException | InterruptedException e) {
-         e.printStackTrace();
-     }
-
- }
-
- }

```

Hora como puedes ver si termina escribiendo el archivo. Dot y fuera del try catch se crea una variable llamada comando la cual nos servirá para poder transformar nuestro punto dot a un PNG, hp donde se hace por medio de un proceso builder, y se indica que si exist code es igual a cero la convención estará completada y de lo contrario hubo algún error a la hora de convertir el código.

Menu.java



```
1  /*
2   * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
3   * Click nbfs://nbhost/SystemFileSystem/Templates/GUIForms/JFrame.java to edit this template
4   */
5   package main;
6
7   import java.awt.Image;
8   import java.awt.image.BufferedImage;
9   import javax.imageio.ImageIO;
10  import javax.swing.ImageIcon;
11  import javax.swing.JLabel;
12  import java.awt.Dimension;
13  import java.awt.Font;
14  import java.awt.event.ActionEvent;
15  import java.io.BufferedReader;
16  import java.io.File;
17  import java.io.FileReader;
18  import java.io.IOException;
19  import java.io.PrintWriter;
20  import java.util.ArrayList;
21  import java.util.HashMap;
22  import java.util.List;
23  import java.util.Map;
24  import javax.swing.BorderFactory;
25  import javax.swing.JFileChooser;
26  import javax.swing.JOptionPane;
27  import javax.swing.JSplitPane;
28
29  /**
30   *
31   * @author aceba
32   */
33  public class Menu extends javax.swing.JFrame {
34
35      private javax.swing.JButton btnAnalizar;
36      private javax.swing.JButton btn2Reportes;
```

Este archivo es donde ocurrirá la magia, hola ya que será nuestro archivo j frame form hola dónde se cargará todo nuestro programa, inicialmente se pudieron observar todas estas librerías que generalmente son librerías de la interfaz gráfica como por ejemplo las librerías para poder visualizar imágenes el buffer para los iconos los label los botones las listas entre otros.

```

    * @author aceba
    */
public class Menu extends javax.swing.JFrame {

    private javax.swing.JButton btnAnalizar;
    private javax.swing.JButton btn2Reportes;
    private javax.swing.JButton btnGenerarGrafico;
    private javax.swing.JComboBox<String> JCBNombreAutomata;
    private javax.swing.JFileChooser fileChooser;
    private javax.swing.JTextArea txtResultados;
    private javax.swing.JTabbedPane tabbedPane;
    private javax.swing.JScrollPane scrollPane;
    private javax.swing.JPanel panelGrafico;
    private AnalizadorLexico analizadorLexico;
    private Map<String, AutomataIndividual> automata;

    /**
     * Creates new form Menu
     */
    public Menu() {

        this.setSize(1000, 1000);
        this.setPreferredSize(new Dimension(1000, 1000));
        this.setLocationRelativeTo(null);
        initComponentsManuak();
        redirectSystemOutput();
        analizadorLexico = new AnalizadorLexico();
        automata = new HashMap<>();

        this.addComponentListener(new java.awt.event.ComponentAdapter() {
            public void componentResized(java.awt.event.ComponentEvent evt) {

```

En este caso todas las propiedades han sido creadas manualmente pero si lo desean también lo pueden hacer contra la droga como se puede ver hoy en nuestro public class menú se puede observar todos los componentes que hemos creado para el programa. Y dentro de nuestro public menú sí colocaron todas las condiciones como por ejemplo de que el archivo tendría medida de 1000 * 1000 pixeles y que siempre estaría colocada el centro, también indicando que aquí se miseria harán todos los componentes manuales que yo he creado junto con su redirect system output, que es básicamente un procedimiento que pasa lo que tenemos en la consola a un textbox. También se crea una nueva instancia del analizador léxico y del autómata.

```

private void initComponentsManuak() {
    setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
    setTitle("Graficador de automata");
    setSize(1800, 700);
    getContentPane().setBackground(new java.awt.Color(196, 255, 51));
    btnAnalizar = new javax.swing.JButton("Analizar Archivo");
    btn2Reportes = new javax.swing.JButton("Generar Reporte");
    btnGenerarGrafico = new javax.swing.JButton("Graficar");
    JCBNombreAutomata = new javax.swing.JComboBox<>();
    fileChooser = new javax.swing.JFileChooser();
    txtResultados = new javax.swing.JTextArea();
    scrollPane = new javax.swing.JScrollPane(txtResultados);
    panelGrafico = new javax.swing.JPanel();

    btnAnalizar.setBackground(new java.awt.Color(51, 66, 255));
    btn2Reportes.setBackground(new java.awt.Color(51, 66, 255));
    btnGenerarGrafico.setBackground(new java.awt.Color(51, 66, 255));

    btnAnalizar.addActionListener(this::btnAnalizarActionPerformed);
    btn2Reportes.addActionListener(this::btnGenerarReporteActionPerformed);
    btnGenerarGrafico.addActionListener(this::btnGenerarGraficoActionPerformed);

    txtResultados.setEditable(false);
    txtResultados.setFont(new java.awt.Font("Monospaced", java.awt.Font.CENTER_BASELINE, 15));

    panelGrafico.setBorder(BorderFactory.createTitledBorder("Visualización del Autómata"));
    panelGrafico.setLayout(new java.awt.BorderLayout());

    JSplitPane splitPane = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT, scrollPane, panelGrafico);
    splitPane.setDividerLocation(900);
    splitPane.setResizeWeight(0.5);

    javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());

```

Aquí en los siguientes componentes de manual únicamente se van creando todo lo visual por ejemplo los botones el TXT el JFileChooser entre otros.

```

115         .addComponent(splitPane)
116         .addGroup(layout.createSequentialGroup()
117             .addComponent(btnAnalizar)
118             .addPreferredGap(LayoutStyle.ComponentPlacement.RELATED)
119             .addComponent(btn2Reportes)
120             .addPreferredGap(LayoutStyle.ComponentPlacement.RELATED)
121             .addComponent(JCBNombreAutomata, javax.swing.GroupLayout.PREFERRED_SIZE, 150, javax.swing.GroupLayout.PREFERRED_SIZE)
122             .addPreferredGap(LayoutStyle.ComponentPlacement.RELATED)
123             .addComponent(btnGenerarGrafico)
124             .addGap(0, 0, Short.MAX_VALUE))
125         .addContainerGap());
126
127     layout.setVerticalGroup(
128         layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
129             .addGroup(layout.createSequentialGroup()
130                 .addContainerGap()
131                 .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
132                     .addComponent(btnAnalizar)
133                     .addComponent(btn2Reportes)
134                     .addComponent(JCBNombreAutomata, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE)
135                     .addComponent(btnGenerarGrafico))
136                 .addPreferredGap(LayoutStyle.ComponentPlacement.RELATED)
137                 .addComponent(splitPane)
138                 .addContainerGap())
139             );
140
141     pack();
142 }
143
144 private void btnAnalizarActionPerformed(java.awt.event.ActionEvent evt) {
145     if (fileChooser.showOpenDialog(this) == JFileChooser.APPROVE_OPTION) {
146         File archivo = fileChooser.getSelectedFile();
147         analizarArchivo(archivo);
148     }
149 }
150

```

Procedimiento procesarAutomata

```

9
10
11 private void procesarAutomata(List<Tokens> tokensAnalizados) {
12     automata.clear();
13     txtResultados.append("\n Comenzando inicializacion para verificacion de construccion de automatas\n");
14
15     List<String> nombresAutomatas = new ArrayList<>();
16     List<String> descripcionesLista = new ArrayList<>();
17     List<String> estadosGlobales = new ArrayList<>();
18     List<String> alfabeto = new ArrayList<>();
19     List<String> estadosIniciales = new ArrayList<>();
20     List<String> estadosFinales = new ArrayList<>();
21     List<String> TransicionesLista = new ArrayList<>();
22     // I VA EN 0
23     int i = 0;
24     while (i < tokensAnalizados.size()) {
25
26         if (tokensAnalizados.get(i).getTipoToken().equals("Identificador") && i + 2 < tokensAnalizados.size() && tokensAnalizados.get(i + 1).getTipoToken().equals("Palabra Reservada")) {
27             String nombre = tokensAnalizados.get(i).getLexema();
28             nombresAutomatas.add(nombre);
29             AutomataIndividual afd = new AutomataIndividual(nombre);
30             estadosGlobales.clear();
31             estadosFinales.clear();
32             txtResultados.append("Nombre del automata verificado: " + nombre);
33             //I VA EN 1
34
35             i += 3;
36             //I VA EN 4
37
38             while (i < tokensAnalizados.size() && !tokensAnalizados.get(i).getTipoToken().equals("LlaveCerrar")) {
39                 if (tokensAnalizados.get(i).getTipoToken().equals("Palabra Reservada") && tokensAnalizados.get(i + 1).getTipoToken().equals("Identificador")) {
40                     String descripcionActual = tokensAnalizados.get(i + 1).getLexema();
41                     descripcionesLista.add(descripcionActual);
42                     System.out.println("Descripcion: " + descripcionActual + "\n");
43                     i += 4;
44                 }
45             }
46         }
47     }
48 }

```

Luego del analizador léxico se podría decir que este es el corazón del programa, ya que en base a todo este código se podrán generar las gráficas. Ahora iniciaremos lo que se hace es que se crea una lista para los nombres de los autómatas descripciones estados globales el alfabeto estados iniciales estados finales y las transiciones. Hoy y luego para ir identificando todo lo que está en nuestro archivo que está inicializaremos una variable llamada i, ahora y colocaremos de que mientras la variable i que hemos creado para iteración sea menor al tamaño de los tokens analizados este seguirá corriendo. Ahora inicialmente se han creado una condición donde se indica que si el token obtenido es un identificador y que si este en tus siguientes 2 iteraciones es una llave de apertura, se da por entendido que aquí vendrá el nombre del autómata, por lo que se almacena el nombre del autómata en la lista para el nombre de los autómatas y se crea una nueva instancia del autómata individual y este autómata individual nos servirá para poder graficar. En resumen todas estas listas únicamente servirán para ir mostrando en nuestro j text box como una información más detallada de todo el archivo de entrada, y el autómata individual es que el que se utilizará para poder graficar.

```

169 AutomataIndividual afd = new AutomataIndividual(nombre);
170 estadosGlobales.clear();
171 estadosFinales.clear();
172 txtResultados.append("Nombre del automata verificado: " + nombre);
173 //I VA EN 1
174
175 i += 3;
176 //I VA EN 4
177
178 while (i < tokensAnalizados.size() && !tokensAnalizados.get(i).getTipoToken().equals("LlaveCerrar")) {
179     if (tokensAnalizados.get(i).getTipoToken().equals("Palabra Reservada") && tokensAnalizados.get(i).getLexema().equals("CorcheteAbrir")) {
180         String descripcionActual = tokensAnalizados.get(i + 2).getLexema();
181         descripcionesLista.add(descripcionActual);
182         System.out.println("Descripcion: " + descripcionActual + "\n");
183         i += 4;
184         //I VA EN 8
185     }
186
187     if (tokensAnalizados.get(i).getTipoToken().equals("Palabra Reservada") && tokensAnalizados.get(i).getLexema().equals("CorcheteCerrar")) {
188         i += 2;
189
190         if (i < tokensAnalizados.size() && tokensAnalizados.get(i).getTipoToken().equals("CorcheteAbrir")) {
191             i++;
192             java.util.Set<String> estadosUnicos = new java.util.HashSet<>();
193
194             while (i < tokensAnalizados.size() && !tokensAnalizados.get(i).getTipoToken().equals("CorcheteCerrar")) {
195                 if (tokensAnalizados.get(i).getTipoToken().equals("Identificador")) {
196                     String estadoActual = tokensAnalizados.get(i).getLexema();
197                     if (!estadosUnicos.contains(estadoActual)) {
198                         estadosUnicos.add(estadoActual);
199                         estadosGlobales.add(estadoActual);
200                         System.out.println("Estado encontrado : " + estadoActual);
201                     } else {
202                         System.out.println("Se encontró un estado duplicado : " + estadoActual);
203                     }
204                 }
205             }
206         }
207     }
208 }

```

Ahora colocamos otra condición de que siempre cuando el token obtenido sea diferente a una llave de cerrar este se va a ir repitiendo hasta que lo encuentre, y otra condición donde si el token obtenido es una palabra reservada, y si su lexema es descripción sabemos que este será una descripción por lo que al identificar esto sumaremos nuestra variable i

en 2, y luego si nuestro token obtenido es una corchete para abrir este sumará uno para poder avanzar de posición y luego mientras sea diferente a un corchete de cerrar sabremos que esta es una descripción. Hola ahora bien identificaremos nuestros estados que contiene el autómata. Sabemos que estos vienen en un archivo de entrada en el que inicializan con los corchetes por lo que le indicaremos el programa que mientras el lexema obtenido sea diferente un corchete de salida este seguirá siendo la misma instrucción. Una vez definido eso identificamos que si el token equivale a un identificador este será el estado inicial por lo que se Añade a nuestra lista de estados globales junto con una lista de Estados Unidos y está lista de estados únicos verifica que no hayan estado repetidos en dicha lista. Hola ahora identificamos de que si se encuentra una coma este avanzará uno, y luego de esto siempre avanzará uno y luego avanzarán 2 más. Esto es suma importancia tener un control muy detallado de la iteración en la que vamos ya que el momento de programarlo debemos de ser muy conscientes en exactamente qué token nos encontramos analizando ya que de no hacerlo puede que aunque las condiciones estén bien definidas nuestro programa se queda ciclado infinitamente debido a que no logró avanzar correctamente la siguiente posición como por ejemplo que se quede en una coma y al no avanzar de posición esta se quede comparando y buscando los estados pero al estar en una coma nunca los encontrará es por eso que hay que tener un control muy minucioso de cuándo debemos de avanzar nuestra integración. Y así nos iremos con todo lo necesario de nuestro archivo de entrada hasta llegar a las transiciones.

```

80         if (i < tokensAnalizados.size() && tokensAnalizados.get(i).getTipoToken().equals("Coma")) {
81             i++;
82         }
83     }
84 }
85
86 if (tokensAnalizados.get(i).getTipoToken().equals("Palabra Reservada") && tokensAnalizados.get(i).getLexema().equals("tra
87     i += 1;
88
89 while (!tokensAnalizados.get(i + 1).getTipoToken().equals("LlaveCerrar")) {
90     i += 2;
91     String estadoActual = tokensAnalizados.get(i).getLexema();
92     i += 3;
93     while (!tokensAnalizados.get(i).getTipoToken().equals("ParentesisCerrar")) {
94         String entrada = tokensAnalizados.get(i).getLexema();
95         i += 2;
96         String estadoDestino = tokensAnalizados.get(i).getLexema();
97         System.out.println("Estado actual : " + estadoActual + " Entrada: " + entrada + " EstadoDestino: " + estadoDes
98         afd.agregarTransicion(estadoActual, entrada, estadoDestino);
99
100         if (tokensAnalizados.get(i + 1).getTipoToken().equals("Coma")) {
101             i += 2;
102         } else {
103             i++;
104         }
105     }
106 }
107
108 automata.put(nombre, afd);
109 actualizarComboBoxAutomatas(nombresAutomatas);
110
111 }
112
113 } else {
114     i++;
115 }

```

Para las transiciones hay que tener un mayor cuidado ya que en estas haremos uso de nuestro automata. Hoy luego de hacer todas las verificaciones necesarias para que pueda leer tanto el estado inicial como el alfabeto y el estado objetivo debemos de asegurarnos que al final coloquemos una instancia de nuestro autómata el cual tendrá en su primer posición en nombre y el segundo en la f de que la FD contendrá tanto el alfabeto como el estado objetivo y por último se debe actualizar el combobox del nombre de los autómatas para que siempre se pueda visualizar actualizado.

```

public void actualizarComboBoxAutomatas(List<String> nombresAutomatas) {
    JNombreAutomata.removeAllItems();
    for (String nombre : nombresAutomatas) {
        JCBNombreAutomata.addItem(nombre);
    }

    if (JCBNombreAutomata.getItemCount() > 0) {
        JCBNombreAutomata.setSelectedIndex(0);
    }

    System.out.println("Automatas disponibles para escoger: " + nombresAutomatas);
}

public AutomataIndividual getAutomataSeleccionado() {
    String nombre = (String) JCBNombreAutomata.getSelectedItem();
    return automata.get(nombre);
}

public void analizarArchivo(File archivo) {
    try {
        txtResultados.setText("");
        automata.clear();
        StringBuilder content = new StringBuilder();
        BufferedReader br = new BufferedReader(new FileReader(archivo));
        String linea;
        while ((linea = br.readLine()) != null) {
            content.append(linea).append("\n");
        }
        analizadorLexico.analizarArchivo(content);
        mostrarTokensEnGUI();
        procesarAutomata(analizadorLexico.getTokens());
    } catch (IOException e) {
        JOptionPane.showMessageDialog(this, "Error al leer el archivo: " + e.getMessage(),

```

Run (Proyecto11 FP)

luego se crea métodos para poder actualizar el combobox de los autómatas y para analizar el archivo para actualizar combo sobre la tomada sólo se va integrando la lista del nombre de los autómatas y para analizar el archivo se hace por medio de un try catch y por medio de un string builder y un buffer reader para poder leer correctamente el archivo que se seleccione en nuestra interfaz gráfica.

```

public void mostrarImagenAutomata(String nombreAutomata) {
    try {
        panelGrafico.removeAll();
        String rutaImagen = "C:\\Users\\aceba\\OneDrive\\Desktop\\Practical\\-LFP-202300673-\\Proyecto1\\AFD.png";
        BufferedImage imagen = ImageIO.read(new File(rutaImagen));
        Image imagenEscalada = imagen.getScaledInstance(
            panelGrafico.getWidth() - 20,
            panelGrafico.getHeight() - 20,
            Image.SCALE_SMOOTH);
        JLabel labelImagen = new JLabel(new ImageIcon(imagenEscalada));
        panelGrafico.add(labelImagen);
        panelGrafico.revalidate();
        panelGrafico.repaint();
        tabbedPane.setSelectedIndex(1);
    } catch (IOException e) {
        txtResultados.append("\nError al cargar la imagen: " + e.getMessage());
    }
}

public void mostrarTokensEnGUI() {

```


y por último se muestra la imagen de nuestro autómeta dentro de nuestra interfaz grafica.