

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 2**

Álvaro Gabriel Ceballos

Gil CARNÉ:

202300673

SECCIÓN: A-

GUATEMALA, 2 DE MAYO 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	¡Error! Marcador no definido.
Procedimiento GraficarAutomata	¡Error! Marcador no definido.
¡Error! Marcador no definido.	
Menu.java	¡Error! Marcador no definido.
Procedimiento procesarAutomata	36

# 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 2, con el fin de que el programador sea capaz de entender correctamente las líneas de código utilizadas para el desarrollo del programa.

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

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 caso así lo desee el programador.

# 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.

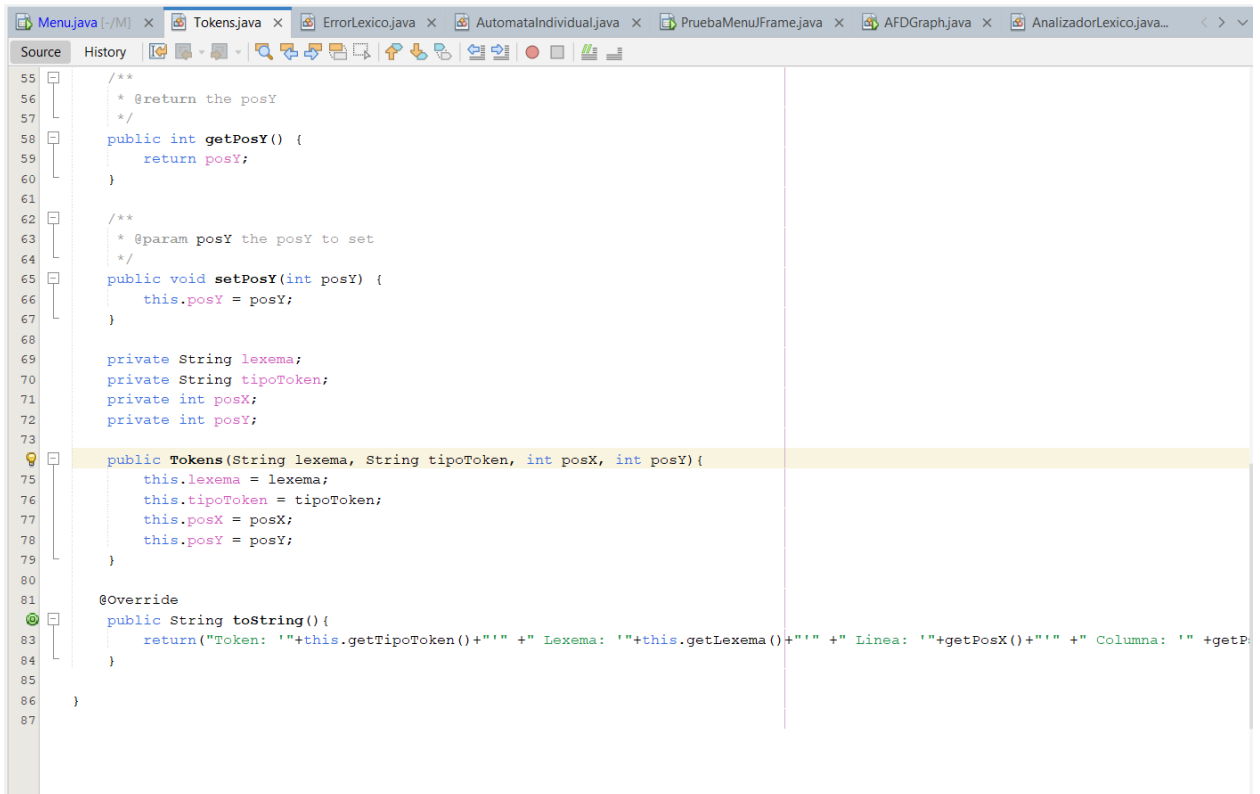
Se debe de contar con GraphViz instalado para la correcta generación de los gráficos, para intalarlo debe de instalar el ejecutable correspondiente a su sistema operativo que se encuentra en: <https://graphviz.org/download/>

- **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

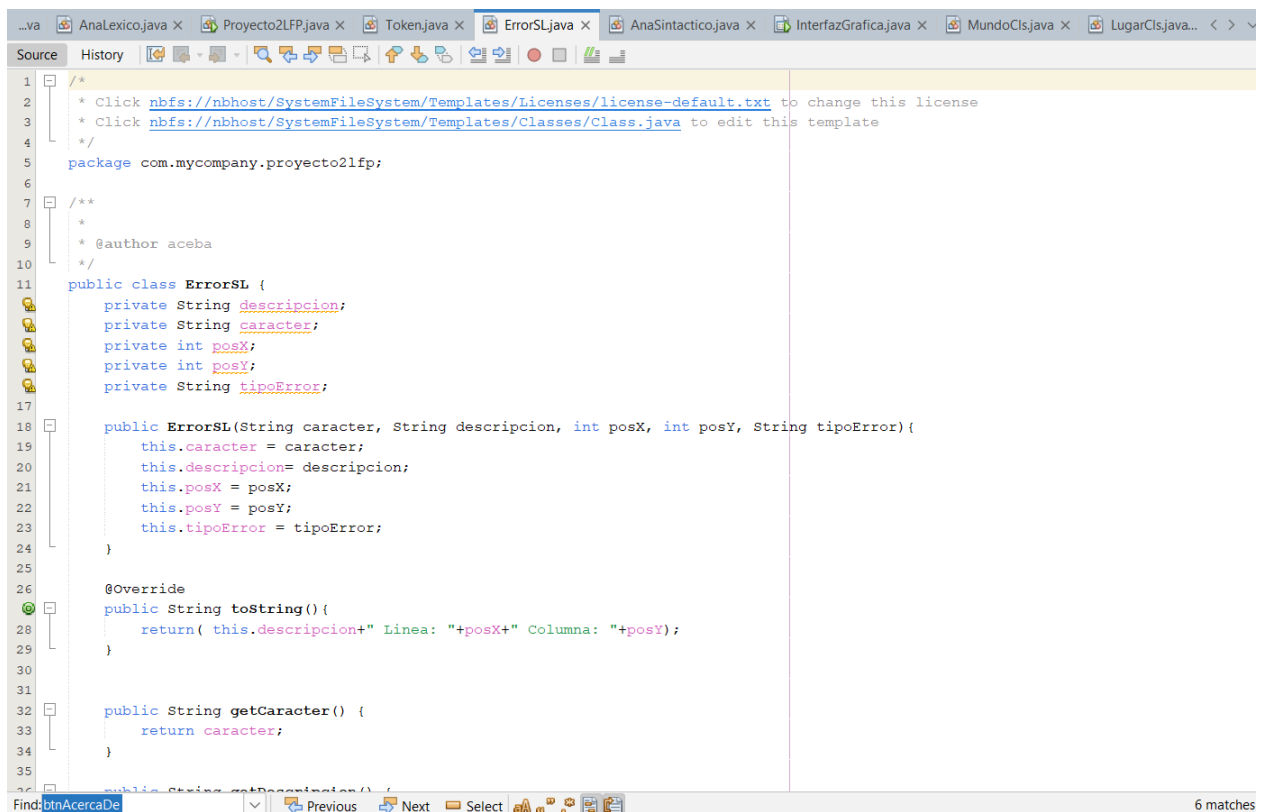


```
55  /**
56   * @return the posY
57   */
58  public int getPosY() {
59      return posY;
60  }
61
62  /**
63   * @param posY the posY to set
64   */
65  public void setPosY(int posY) {
66      this.posY = posY;
67  }
68
69  private String lexema;
70  private String tipoToken;
71  private int posX;
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: '" + getPosX() + "'" + " Columna: '" + getPosY() + "'";
84  }
85
86  }
87  }
```

Clase de tokens se 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 ErrorSL

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



```
1  /*
2  * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
3  * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
4  */
5  package com.mycompany.proyecto2lfp;
6
7  /**
8   *
9   * @author aceba
10  */
11  public class ErrorSL {
12      private String descripcion;
13      private String caracter;
14      private int posX;
15      private int posY;
16      private String tipoError;
17
18      public ErrorSL(String caracter, String descripcion, int posX, int posY, String tipoError){
19          this.caracter = caracter;
20          this.descripcion= descripcion;
21          this.posX = posX;
22          this.posY = posY;
23          this.tipoError = tipoError;
24      }
25
26      @Override
27      public String toString(){
28          return( this.descripcion+" Linea: "+posX+" Columna: "+posY);
29      }
30
31
32      public String getCaracter() {
33          return caracter;
34      }
35
36      public String getDescripcion() {
```



## Clase ErrorSintactico

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package com.mycompany.proyecto21fp;

/**
 *
 * @author aceba
 */
public class ErrorSintactico {
    private String descripcion;
    private String caracter;
    private int posX;
    private int posY;
    private String tipoError;

    public ErrorSintactico(String caracter, String descripcion, int posX, int posY, String tipoError){
        this.caracter = caracter;
        this.descripcion = descripcion;
        this.posX = posX;
        this.posY = posY;
        this.tipoError = tipoError;
    }

    @Override
    public String toString(){
        return ( this.descripcion+" Línea: "+posX+" Columna: "+posY);
    }

    public String getCaracter() {
        return caracter;
    }

    public String getDescripcion() {

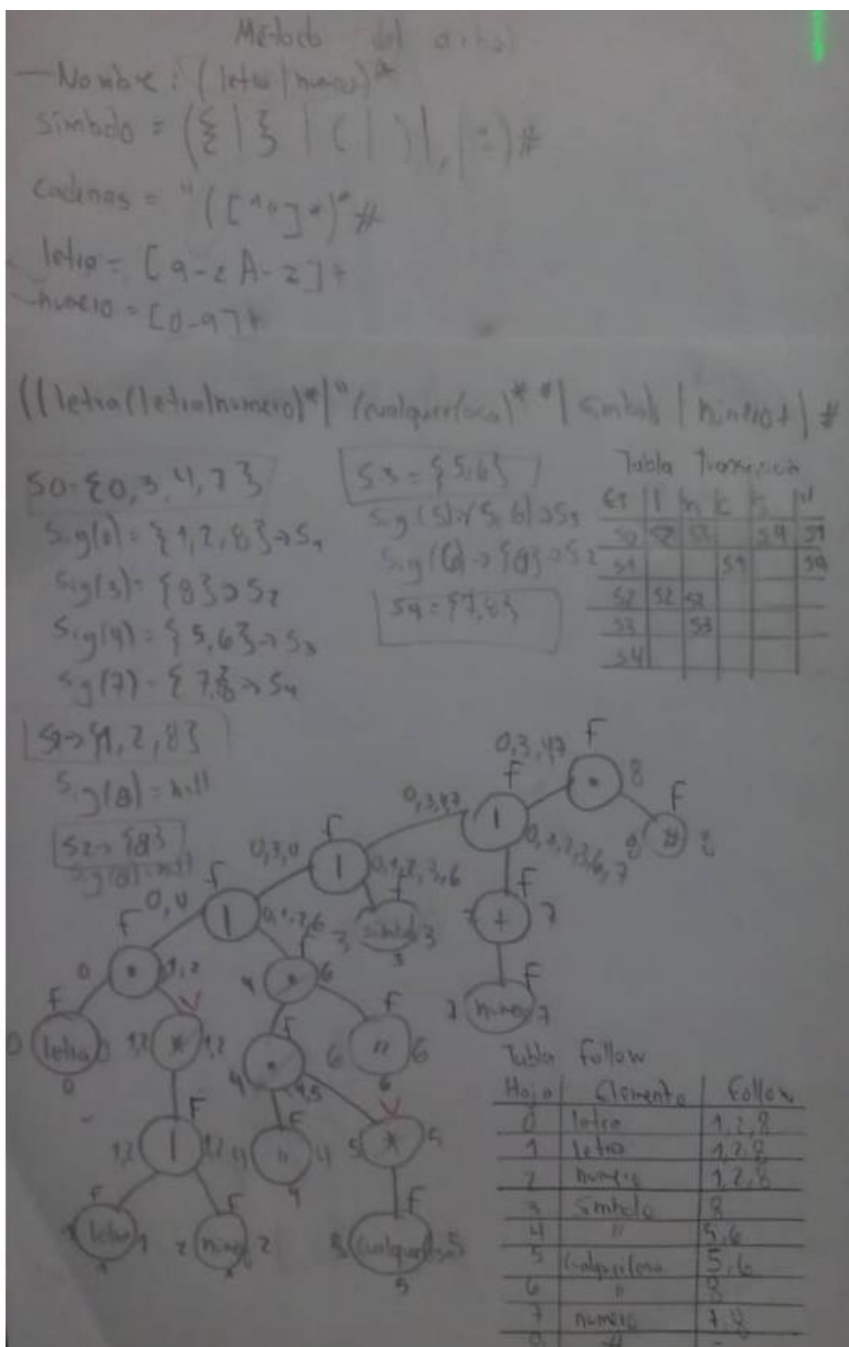
```

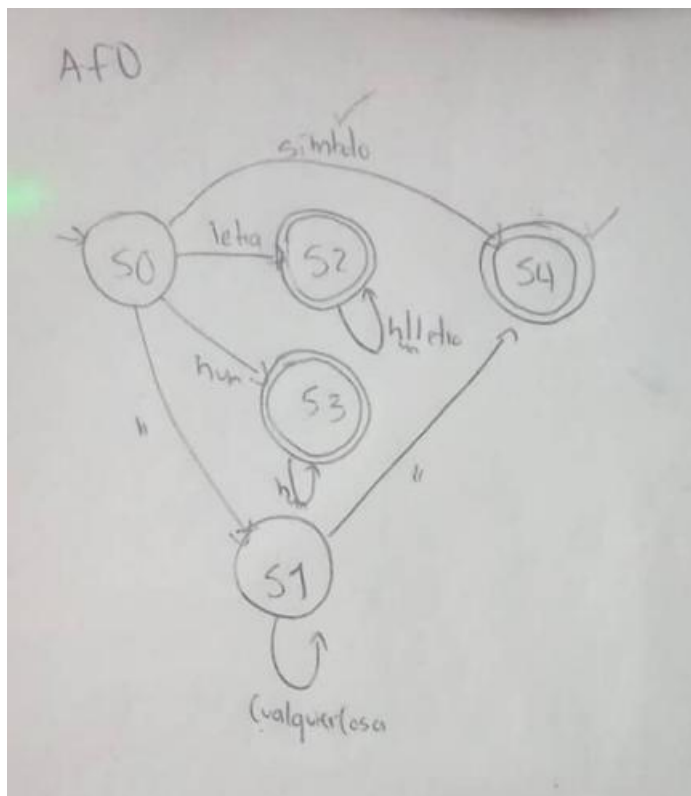
Contiene exactamente los mismos parámetros de la clase de errorSL, hp pero se decidió realizar otra clase con el fin de poder crear 2 archivos HTML con mucha mayor facilidad.

## Analizador Lexico

Contiene exactamente los mismos parámetros de la clase de errorSL, hp pero se decidió realizar otra clase con el fin de poder crear 2 archivos HTML con mucha mayor facilidad.

Hola lo primero que se hizo antes de empezar a programar cualquier cosa fue definir nuestra expresión regular, seguido de realizar el método del árbol junto con la tabla de follow y transiciones, esto con el fin de poder hallar un autómata finito determinista que sea capaz de reconocer todos los lexemas y tokens del archivo de entrada, para luego proseguir a simplificar el autómata finito determinista. Una vez conociendo cuáles serían nuestras transiciones y estados ya podemos comenzar a programar.





Una vez creada y sabiendo cuáles serían nuestros estados procedimos a programar nuestro analizador léxico, inicialmente se crea una lista de tokens y de error léxico para poder ir almacenando lo justo con variables de post x para ir sabiendo cuál es la fila y la columna del token con un buffer para cuando venga más de una cadena poder almacenarlos dentro del buffer y poderlo mostrar como una cadena de caracteres y una posición de archivo para saber en qué posición estamos pero en esta ocasión tendrá pocos usos.

```

    * @author aceba
    */
    public class AnaLexico {
        private List<Token> ListaTokens;
        private List<ErrorSL> ListaErrores;
        private int posX;
        private int posY;
        private String buffer;
        private int estado;
        private int iArchivo;

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

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

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

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

```

Se pudo observar se crearon muchos procedimientos unas para poder agregar los tokens para poder agregar los errores pero el más importante es el de analizar archivo ya que en este vamos a definir cómo se irá haciendo la transición entre nuestros estados que fue lo que le definimos en nuestro afd.

```

public void analizarArchivo(StringBuilder cadena){
    this.ListaTokens.clear();
    this.ListaErrores.clear();
    this.estado = 0;
    this.buffer = "";
    this.iArchivo = 0;
    this.posX = 0;
    this.posY = 0;
    while(this.iArchivo < cadena.length()){
        if(this.estado == 0){
            q0q4(cadena.charAt(this.iArchivo));
        }
        else if(this.estado == 3){
            q3(cadena.charAt(this.iArchivo));
        }

        else if(this.estado == 2){
            q2(cadena.charAt(this.iArchivo));
        }

        else if(this.estado == 1){
            q1(cadena.charAt(this.iArchivo));
        }

        iArchivo++;

    }
    if(!buffer.isEmpty()) {
        if(estados == 3) {
            nuevoToken(buffer, "Numero", posX, posY);
        }
    }
}

```

```

    }
    public void q0q4(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), "DosPuntos", this.posX, this.posY);
            this.posY++;
        } else if (caracter == ',') {
            this.nuevoToken(String.valueOf(caracter), "Coma", this.posX, this.posY);
            this.posY++;
        } else if (caracter == '(') {
            this.nuevoToken(String.valueOf(caracter), "ParentesisAbrir", this.posX, this.posY);
            this.posY++;
        } else if (caracter == ')') {
            this.nuevoToken(String.valueOf(caracter), "ParentesisCerrar", this.posX, this.posY);
            this.posY++;
        } else if (caracter == '\n') {
            this.posX++;
            this.posY = 0;
        } else if (caracter == ' ') {
            this.posY++;
        } else if (caracter == '\t') {
            this.posY += 4;
        }
        else if (Character.isDigit(caracter)) {
            this.buffer += caracter;
            this.posY++;
            this.estado = 3;
        }
        else if (Character.isLetter(caracter)) {
            this.buffer += caracter;

```

Luego se prosiguió con la programación de cada uno de los estados como el estado inicial no tenía nada lo reconfiguramos para que el estado inicial también fuera equivalente al estado cuatro y que así pudiera aceptar símbolos como se puede observar se definen las llaves de apertura de cierre:, (y cierre.

```

    public void q3(char caracter){
    if(Character.isDigit(caracter)) {
        this.buffer += caracter;
        this.posY++;
    } else {
        int tokenPosY = this.posY - buffer.length();
        this.nuevoToken(buffer, "Numero", this.posX, tokenPosY);
        this.estado = 0;
        this.iArchivo--;
    }
}

public void q2(char caracter) {
    if (Character.isDigit(caracter) || Character.isLetter(caracter)) {
        this.buffer += caracter;
        this.posY += 1;
    } else {
        if (this.buffer.equals("world")) {
            this.nuevoToken(this.buffer, "PRWorld", posX, posY);
            this.estado = 0;
            this.posY += 1;
            this.iArchivo -= 1;
        } else if (this.buffer.equals("place")) {
            this.nuevoToken(this.buffer, "PRPlace", posX, posY);
            this.estado = 0;
            this.posY += 1;
            this.iArchivo -= 1;
        } else if (this.buffer.equals("at")) {
            this.nuevoToken(this.buffer, "PRAt", posX, posY);
            this.estado = 0;
            this.posY += 1;
            this.iArchivo -= 1;
        } else if (this.buffer.equals("playa") || this.buffer.equals("cueva") ||
            this.buffer.equals("templo") || this.buffer.equals("jungla") ||

```

En los estados q3 y q2 definimos que se podrán aceptar números en el caso de cutres y en q 2 se podrán aceptar todos los lexemas que repitan una letra o un carácter una o más veces es por eso que utilizamos este estado para poder reconocer todas las palabras reservadas.

```

        this.iArchivo -= 1;
    } else if (this.buffer.equals("at")) {
        this.nuevoToken(this.buffer, "PRAt", posX, posY);
        this.estado = 0;
        this.posY += 1;
        this.iArchivo -= 1;
    } else if (this.buffer.equals("playa") || this.buffer.equals("cueva") ||
        this.buffer.equals("templo") || this.buffer.equals("jungla") ||
        this.buffer.equals("montana") || this.buffer.equals("pueblo") ||
        this.buffer.equals("isla") || this.buffer.equals("rio") ||
        this.buffer.equals("volcan") || this.buffer.equals("pantano")) {

        this.nuevoToken(this.buffer, "PRLugar", posX, posY);
        this.estado = 0;
        this.posY += 1;
        this.iArchivo -= 1;

    } else if (this.buffer.equals("tesoro") || this.buffer.equals("llave") ||
        this.buffer.equals("arma") || this.buffer.equals("objetomagico") ||
        this.buffer.equals("pocion") || this.buffer.equals("trampa") ||
        this.buffer.equals("libro") || this.buffer.equals("herramienta") ||
        this.buffer.equals("bandera") || this.buffer.equals("gema")) {

        this.nuevoToken(this.buffer, "PRObjeto", posX, posY);
        this.estado = 0;
        this.posY += 1;
        this.iArchivo -= 1;
    } else if (this.buffer.equals("connect")) {
        this.nuevoToken(this.buffer, "PRConnect", posX, posY);
        this.estado = 0;
        this.posY += 1;
        this.iArchivo -= 1;
    } else if (this.buffer.equals("to")) {
        this.nuevoToken(this.buffer, "PRTto", posX, posY);
    }

```



```

    }

    public void q1(char caracter){
        if(caracter != '"'){
            this.buffer+=caracter;
            this.posY+=1;

        }else{
            this.buffer+=caracter;
            this.nuevoToken(this.buffer, "Cadena de texto", posX, posY);
            this.posY+=1;
            this.estado=0;
        }
    }

    public void imprimirTokens(){
        for(Token token: this.ListaTokens){
            System.out.println(token);
        }
    }

    public void imprimirErrores(){
        for(ErrorSL error: this.ListaErrores){
            System.out.println(error);
        }
    }

    public List<Token> getTokens(){
        return this.ListaTokens;
    }

    public void generarReporteHTML(String rutaArchivo) {
        try {

```

Luego definimos que en nuestro estado q uno se aceptarían todo tipo de cadenas de texto y también colocamos más métodos para poder imprimir tanto los toques como los errores y retornar tanto su lista de tokens y errores.

```

public void generarReporteHTML(String rutaArchivo) {
    try {
        FileWriter escritorHTMLToken = new FileWriter(rutaArchivo);

        escritorHTMLToken.write("<!DOCTYPE html>\n");
        escritorHTMLToken.write("<html>\n");
        escritorHTMLToken.write("<head>\n");
        escritorHTMLToken.write("<title>Reporte de Tokens</title>\n");
        escritorHTMLToken.write("<style>\n");
        escritorHTMLToken.write("body { font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif; background-color: #1e1e2e; color: #00ff7f; text-align: center; text-shadow: 2px 2px 5px rgba(0, 255, 127, 0.5); }\n");
        escritorHTMLToken.write("table { width: 80%; margin: 20px auto; border-collapse: collapse; background-color: #2a2a3a; border: 2px solid #00ff7f; transition: all 0.3s ease-in-out;\n");
        escritorHTMLToken.write("tr { background-color: #00ff7f; color: #1e1e2e; font-weight: bold; text-transform: uppercase; }\n");
        escritorHTMLToken.write("tr:nth-child(even) { background-color: #2f2f3f; }\n");
        escritorHTMLToken.write("</style>\n");
        escritorHTMLToken.write("</head>\n");
        escritorHTMLToken.write("<body>\n");
        escritorHTMLToken.write("<h1>Reporte de Tokens</h1>\n");
        escritorHTMLToken.write("<table>\n");
        escritorHTMLToken.write("<tr><th>Token</th><th>Lexema</th><th>Línea</th><th>Columna</th></tr>\n");
        for (Token token : this.ListaTokens) {
            escritorHTMLToken.write("<tr>\n");
            escritorHTMLToken.write("<td>" + descriphHTML(token.getTipoToken()) + "</td>\n");
            escritorHTMLToken.write("<td>" + descriphHTML(token.getLexema()) + "</td>\n");
            escritorHTMLToken.write("<td>" + (token.getPosX() + 1) + "</td>\n");
            escritorHTMLToken.write("<td>" + (token.getPosY() + 1) + "</td>\n");
            escritorHTMLToken.write("</tr>\n");
        }
        escritorHTMLToken.write("</table>\n");
        escritorHTMLToken.write("</body>\n");
        escritorHTMLToken.write("</html>\n");
        escritorHTMLToken.close();
    }
}

```

Se creó toda la estructura para poder generar un archivo html tanto para reporte de error léxico y de token.

## AnaSintactico

al igual que con nuestro analizador léxico antes de empezar a programar algo en nuestro analizador sintáctico primero debemos de definir nuestra gramática nuestra gramática nos indicará cuántos procedimientos debemos de crear y de cómo será la transición entre ellos y la gramática nos ayudará a verificar que todos los lexemas vengan en el orden correspondiente, la gramática que definimos es la siguiente.

Nombre: Álvaro Gabriel Gálvez G.

Carnet: 202300673

### Tarea 5

#### Tabla de tokens

Token	EB
+world	world
cadena	"cadena"
llave abrir	{
+place	place
identificador	$[a-zA-Z][a-zA-Z][0-9]^*$
dosp	
locacion	Playa / Cueva / templo / jungla / montaña / pantano / isla / ro lucrant pantano
preservat	at
parentesis Abir	(
num	$[0-9]^+$
coma	,
parentesis Cerrar	)
pr connect	connect
identificador	$[a-zA-Z][a-zA-Z][0-9]^*$
pr to	to
identificador	identificador
pr with	with
cadena	"cadena"

Token	ER
proyect	object
cadena	"cadena"
dosp	:
}	llave cerrar

~~proyecto~~ ~~proyecto~~ ~~trabajo~~ ~~llave~~ ~~arma~~ ~~objetomage~~ ~~porca~~ ~~trabajo~~ ~~libro~~ ~~llave~~ ~~trabajo~~  
Simbolos terminales = { +work, cadena, llave abrir, +place, identificador, dosp, locacion, presentador, parentesis abrir, num, coma, parentesis cerrar, preconnect, pto, pto.h, proyect, llave cerrar }

No terminales = { INICIO, MUNDO, MUNDO', MUNDOV, LPLACES, LPLACES', LPLACE, LCONNECTS, LCONNECTS', LCONNECT, LOBJECTS, LOBJECTS', LOBJECT

Simbolo inicial = { INICIO }

INICIO  $\rightarrow$  MUNDOS

MUNDOS  $\rightarrow$  MUNDOSV MUNDOS'

MUNDOS'  $\rightarrow$  coma MUNDOSV MUNDOS' /  $\epsilon$

MUNDOSV  $\rightarrow$  +world cadena | +objeto | +LPLACES LCONNECTS LOBJECTS  
| +LCONNECTS LPLACES LOBJECTS

LPLACES  $\rightarrow$  LPLACE LPLACES'

LPLACES'  $\rightarrow$  LPLACE LPLACES' /  $\epsilon$

LPLACE  $\rightarrow$  +place identificador dosp locacion presentat  
| +parentesisAbir num coma num parentesisCerrar

LCONNECTS  $\rightarrow$  LCONNECT LCONNECTS'

LCONNECTS'  $\rightarrow$  LCONNECT LCONNECTS' /  $\epsilon$

LCONNECT  $\rightarrow$  +preconnect identificador pito identificador  
| +prewith cadena

LOBJECTS  $\rightarrow$  LOBJECT LOBJECTS'

LOBJECTS'  $\rightarrow$  LOBJECT LOBJECTS' /  $\epsilon$

POSOBJ  $\rightarrow$  identificador | +parentesisAbir num coma num parentesisCerrar

LOBJECT  $\rightarrow$  +proyecto cadena dosp proyecto praf POSOBJ

Una vez definidas nuestra gramática ya podemos empezar a programar nuestra clase de analizador sintáctico

```

* @author aceba
*/
public class AnaSintactico {
    private List<Token> ListaTokens;
    private List<ErrorSintactico> ListaErrores;
    HashMap<String, MundoCis> HMMundo;
    private int posX;
    private int posY;

    String nombreMundoA;

    public AnaSintactico(List<Token> ListaTokens){
        this.ListaTokens = ListaTokens;
        this.ListaErrores = new ArrayList<>();
        this.HMMundo = new HashMap<>();
        this.nombreMundoA = "";
        this.posX = 0;
        this.posY = 0;
    }

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

    public void agregarError(Token token){
        this.nuevoError("No se esperaba el token "+token.getLexema(), token.getPosY(), token.getPosX(), "Error de tipo sintactico");
    }

    public void agregarErrorEspecifico(Token token, String mensaje) {
        this.ListaErrores.add(new ErrorSintactico(
            token.getLexema(),
            mensaje
        ));
    }
}

```

Al igual que con nuestro analizador léxico, hoy definimos tanto listas como hashtags para poder ir almacenando todos los datos correspondientes al archivo de entrada.

```

1 token.getPosY(),
2 "Error Sintáctico"
3 ));
4 }
5
6 public void analizar(){
7     this.INICIO();
8 }
9
10 public void INICIO(){
11     System.out.println("INICIO");
12     this.MUNDOS();
13 }
14
15 public void MUNDOS(){
16     System.out.println("MUNDOS()");
17     this.MUNDOU();
18     this.MUNDOSP();
19 }
20
21 public void MUNDOSP(){
22     System.out.println("MUNDOS PRIMA");
23     try{
24         Token tokenTemporal = this.ListaTokens.get(0);
25         if(tokenTemporal.getTipoToken().equals("Coma")){
26             this.ListaTokens.removeFirst();
27             this.MUNDOU();
28             this.MUNDOSP();
29         }else{
30
31         }
32     }
33 }

```

En hoy esta parte es donde definimos nuestra gramática, hola inicialmente se crea un método llamado analizar el cual contiene el inicio y el inicio será como lo dice el nombre

y el inicio de nuestra gramática, como definimos en nuestra gramática inicialmente se movería a mundos y mundos contendría mundo un que corresponde a mundo único y mundo p que es mundo es prima esto lo hicimos para eliminar la recursividad por la derecha, hoy y ya luego mundos prima lo que hicimos fue que colocamos hubo una variable temporal que fue la que iremos removiendo para ir verificando el orden de los tokens, hola primero verificamos de que el primer token encontrado según la coma esto porque así se vivió la gramática y así para saber si vienen más mundos o no si bien una coma volvemos a aplicar recursividad para que pueda aceptar uno más mundos pero en caso de que no lo haga se termina.

```
public void MUNDOU() {
    System.out.println("MUNDO UNICO");
    Token tokenTemporal = this.ListaTokens.removeFirst();
    if (tokenTemporal.getTipoToken().equals("PRWorld")) {
        //Continuamos
        tokenTemporal = this.ListaTokens.removeFirst();

        if (tokenTemporal.getTipoToken().equals("Cadena de texto")) {
            this.nombreMundoA = tokenTemporal.getLexema();
            this.HMmundo.put(nombreMundoA, new MundoCls());
            tokenTemporal = this.ListaTokens.removeFirst();

            if (tokenTemporal.getTipoToken().equals("LlaveAbrir")) {
                this.LPLACES();
                this.LCONNECTS();
                this.LOBJECTS();

                tokenTemporal = this.ListaTokens.removeFirst();

                if (tokenTemporal.getTipoToken().equals("LlaveCerrar")) {
                    return;
                } else {
                    agregarErrorEspecifico(tokenTemporal, "Se esperaba llave para cerrar '}', no " + tokenTemporal.getLexema());
                }
            } else {
                agregarErrorEspecifico(tokenTemporal, "Se esperaba llave de apertura '{', no " + tokenTemporal.getLexema());
            }
        } else {
            agregarErrorEspecifico(tokenTemporal, "Se esperaba cadena de texto, no " + tokenTemporal.getLexema());
        }
    }
}
```

Ahora en este pedazo de código definimos mundo u que es un mundo único el cual por como lo definimos primero sacamos las variables temporales para ir verificando el orden y luego verificamos de que primero venga una palabra reservada a Word seguida de una cadena de texto seguida de una llave de apertura y seguida de una llave de cierre y que en caso de que no se encuentre se reporta un error y se agrega la tabla de errores sintácticos.

```

    }
    public void LPLACES() {
        System.out.println("LPLACES()");
        this.LPLACE();
        this.LPLACESP();
    }

    public void LPLACESP() {
        System.out.println("LPLACESP");
        if(this.ListaTokens.get(0).getTipoToken().equals("PRPlace")) {
            this.LPLACE();
            this.LPLACESP();
        }else{
            return;
        }
    }

    public void LPLACE() {
        System.out.println("LPLACE");
        String nombreLugar;
        String tipo;
        String LposX;
        String LposY;

        Token tokenTemporal = this.ListaTokens.removeFirst();

        if (tokenTemporal.getTipoToken().equals("PRPlace")) {

            tokenTemporal = this.ListaTokens.removeFirst();

```

Se aplicó una lógica similar para definir nuestras plazas ya que primero se define que pueden venir muchos países y que dentro de esos países puede venir o un solo lugar o lugares prima luego colocamos en lugares prima que si viene un replace que es palabra reservada a place se volverá a apuntar a sí mismo esto para que pueda aceptar uno o más países y de lo contrario se retornará para poder seguir leyendo la gramática y luego ya en el enlace colocamos de que primero se remueve una variable temporal para ir verificando el orden de los tokens.



```

Token tokenTemporal = this.ListaTokens.removeFirst();

if (tokenTemporal.getTipoToken().equals("PRPlace")) {

    tokenTemporal = this.ListaTokens.removeFirst();

    if (tokenTemporal.getTipoToken().equals("Identificador")) {
        nombreLugar = tokenTemporal.getLexema();
        tokenTemporal = this.ListaTokens.removeFirst();

        if (tokenTemporal.getTipoToken().equals("DosPuntos")) {

            tokenTemporal = this.ListaTokens.removeFirst();

            if (tokenTemporal.getTipoToken().equals("PRLugar")) {
                tipo = tokenTemporal.getLexema();
                tokenTemporal = this.ListaTokens.removeFirst();

                if (tokenTemporal.getTipoToken().equals("PRAt")) {

                    tokenTemporal = this.ListaTokens.removeFirst();

                    if (tokenTemporal.getTipoToken().equals("ParentesisAbrir")) {

                        tokenTemporal = this.ListaTokens.removeFirst();

                        if (tokenTemporal.getTipoToken().equals("Numero")) {
                            LposX = tokenTemporal.getLexema();
                            tokenTemporal = this.ListaTokens.removeFirst();

                            if (tokenTemporal.getTipoToken().equals("Coma")) {

                                tokenTemporal = this.ListaTokens.removeFirst();

                                if (tokenTemporal.getTipoToken().equals("Numero")) {
                                    LposX = tokenTemporal.getLexema();
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}

```

Ya estando en nuestro procedimiento de países lo que hicimos fue que verificamos que primero venga una palabra reservada place seguida de un identificador seguida de: seguida de la palabra reservada lugar seguida de la palabra reservada at seguida de (seguida de un número seguida de una coma seguido de un número seguida de una llave de cierre.

```

public void LCONNECTS () {
    System.out.println("LCONNECTS()");
    this.LCONNECT();
    this.LCONNECTSP();
}

public void LCONNECTSP() {
    System.out.println("LCONNECTSP");
    if(this.ListaTokens.get(0).getTipoToken().equals("PRConnect")) {
        this.LCONNECT();
        this.LCONNECTSP();
    }else{
        return;
    }
}

public void LCONNECT() {
    System.out.println("LCONNECT");

    Token tokenTemporal = this.ListaTokens.removeFirst();

    if (tokenTemporal.getTipoToken().equals("PRConnect")) {

        tokenTemporal = this.ListaTokens.removeFirst();

        if (tokenTemporal.getTipoToken().equals("Identificador")) {

            tokenTemporal = this.ListaTokens.removeFirst();

            if (tokenTemporal.getTipoToken().equals("PRWith")) {

```

Para connect se hizo lo mismo se creó que puedan venir más de un connect con connect individual y connect prima ya en conecte individual se verificó que primero venga la palabra reservada connect seguida de un identificador seguida de la palabra reservada tu seguida del identificador seguida de la palabra reservada with y seguida de una cadena de texto y que en caso que algo no venga en ese orden se notificará que hubo un error sintáctico.

```

}

public void LOBJECTS() {
    System.out.println("LOBJECTS()");
    this.LOBJECT();
    this.LOBJECTSP();
}

public void LOBJECTSP() {
    System.out.println("LOBJECTSP");
    if(this.ListaTokens.get(0).getTipoToken().equals("PalabraObjeto")) {
        this.LOBJECT();
        this.LOBJECTSP();
    } else {
        return;
    }
}

public void OBJECTCASE() {
    Token tokenTemporal1 = this.ListaTokens.get(0);

    if (tokenTemporal1.getTipoToken().equals("Identificador")) {

        Token tokenTemporal = this.ListaTokens.removeFirst();

        System.out.println("IDENTIFICADOR");
        return;
    } else if (tokenTemporal1.getTipoToken().equals("ParentesisAbrir")) {

        Token tokenTemporal = this.ListaTokens.removeFirst();

        tokenTemporal = this.ListaTokens.removeFirst();
        if (tokenTemporal.getTipoToken().equals("Numero")) {
            tokenTemporal = this.ListaTokens.removeFirst();

```

Para los procedimientos de object se crearon varios procedimientos el primero que podrá aceptar uno o más objetos el segundo que será objetos prima que es para que igualmente acepte uno más objetos y luego el object que hace que lo que hace es de que verifica de que si viene o un identificador o una variable en coordenadas esto porque el objeto puede venir ya sea en un lugar en específico o en un lugar de coordenadas para que se cree de cero entonces por eso creamos una producción que acepta o un identificador o una coordenada.

```

public void LOBJECT() {
    System.out.println("LOBJECT");
    Token tokenTemporal = this.ListaTokens.removeFirst();

    if (tokenTemporal.getTipoToken().equals("PalabraObjeto")) {
        tokenTemporal = this.ListaTokens.removeFirst();

        if (tokenTemporal.getTipoToken().equals("Cadena de texto")) {
            tokenTemporal = this.ListaTokens.removeFirst();

            if (tokenTemporal.getTipoToken().equals("DosPuntos")) {
                tokenTemporal = this.ListaTokens.removeFirst();

                if (tokenTemporal.getTipoToken().equals("PRObjeto")) {
                    tokenTemporal = this.ListaTokens.removeFirst();

                    if (tokenTemporal.getTipoToken().equals("PRAt")) {
                        this.OBJECTCASE();
                        return;
                    } else {
                        agregarErrorEspecifico(tokenTemporal, "Se esperaba palabra at, no " + tokenTemporal.getLexema());
                    }
                } else {
                    agregarErrorEspecifico(tokenTemporal, "Se esperaba uno de los posibles objetos, no " + tokenTemporal.getI
                }
            } else {
                agregarErrorEspecifico(tokenTemporal, "Se esperaban dos puntos, no " + tokenTemporal.getLexema());
            }
        } else {
    }
}

```

ya en nuestro objeto individual verificamos de que primero venga la palabra reservada objeto seguida de una cadena de texto seguida de: seguida de otra palabra reservada de una lista de objetos seguida de la palabra at para luego llamar object case ya que en este nos verifica si viene o un identificador o una coordenada.

```

    public Set<String> getNombresMundos() {
        return this.HMmundo.keySet();
    }

    public List<ErrorSintactico> getErrores() {
        return this.ListaErrores;
    }

    private Token getTokenActual() {
        if (!this.ListaTokens.isEmpty()) {
            return this.ListaTokens.getFirst();
        }
        return new Token("EOF", "FinDeArchivo", this.posX, this.posY);
    }

    public void agregarErrorSintactico(String mensajeError) {
        Token tokenActual = getTokenActual();
        this.ListaErrores.add(new ErrorSintactico(
            tokenActual.getLexema(),
            mensajeError,
            tokenActual.getPosX(),
            tokenActual.getPosY(),
            "Error Sintáctico"
        ));
    }

    public void generarReporteErroresHTML(String rutaArchivo) {
        try {
            FileWriter escritorError = new FileWriter(rutaArchivo);

            escritorError.write("<!DOCTYPE html>\n");
            escritorError.write("<html>\n");
            escritorError.write("<head>\n");
            escritorError.write("<title>Reporte de Tokens</title>\n");

```

Luego se retorna en nuestras listas de errores y de nombre es mundo con nuestro #esto para poder graficarlo en un futuro y también colocamos un procedimiento para poder agregar el error sintáctico sin ningún tipo de problema conjunto con su posición en XY el lexema y mensaje de error y un procedimiento para generar reportes html que es exactamente igual a que hicimos en analizador léxico.

## Clase de MundoCIs



## LugarCls

```
2      * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this licens
3      * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
4      */
5      package com.mycompany.proyecto21fp;
6
7      /**
8       *
9       * @author aceba
10      */
11      public class LugarCls {
12          String identificador;
13          String tipo;
14          String LposX;
15          String LposY;
16
17
18          public LugarCls(String identificador, String tipo, String LposX, String LposY){
19              this.identificador = identificador;
20              this.tipo = tipo;
21              this.LposX = LposX;
22              this.LposY = LposY;
23          }
24      }
25  }
```

Esta es una clase simple que es capaz de almacenar los lugares con su identificador tipo de lugar posición en XY posición y.

**INterfazGrafica.java**



```

public class InterfazGrafica extends javax.swing.JFrame {

    private JTextArea txtAreaContenido;
    private JPanel panelImagen;
    private JComboBox<String> comboMundos;
    private JButton btnCargarArchivo, btnLimpiarArea, btnAnalizarArchivo, btnGenerarReportes, btnAcercaDe;
    private AnaLexico analizadorLexico;
    private AnaSintactico analizadorSintactico;
    private StringBuilder contenidoArchivo;

    /**
     * Creates new form InterfazGrafica
     */
    public InterfazGrafica() {
        analizadorLexico = new AnaLexico();
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(800, 600);
        setLocationRelativeTo(null);
        initComponents();
        initComponents2();
    }

    private void initComponents2() {
        JPanel panelPrincipal = new JPanel(new BorderLayout(10, 10));
        panelPrincipal.setBorder(BorderFactory.createEmptyBorder(20, 20, 20, 20));
        panelPrincipal.setBackground(new Color(70, 130, 180));
        JPanel panelCentral = new JPanel(new GridLayout(1, 2, 10, 0));
        txtAreaContenido = new JTextArea();
        txtAreaContenido.setFont(new Font("Monospaced", Font.PLAIN, 12));
        JScrollPane scrollTexto = new JScrollPane(txtAreaContenido);
        scrollTexto.setBorder(BorderFactory.createTitledBorder(
            BorderFactory.createEtchedBorder(),
            "Área de texto",
            TitledBorder.CENTER,

```

Ya en nuestra interfaz gráfica se encuentran todos los elementos que hacen posible que el menú sea interactivo para poder realizar todas las operaciones necesarias primero se crean todos los contenidos que se irían en el área donde se verá todo el texto junto con su panel para la imagen con un combobox para el mundo con todos sus botones junto con el botón de analizador léxico sintáctico y de reportes.

```

private void initComponents2() {
    JPanel panelPrincipal = new JPanel(new BorderLayout(10, 10));
    panelPrincipal.setBorder(BorderFactory.createEmptyBorder(20, 20, 20, 20));
    panelPrincipal.setBackground(new Color(70, 130, 180));
    JPanel panelCentral = new JPanel(new GridLayout(1, 2, 10, 0));
    txtAreaContenido = new JTextArea();
    txtAreaContenido.setFont(new Font("Monospaced", Font.PLAIN, 12));
    JScrollPane scrollTexto = new JScrollPane(txtAreaContenido);
    scrollTexto.setBorder(BorderFactory.createTitledBorder(
        BorderFactory.createEtchedBorder(),
        "Área de texto",
        TitledBorder.CENTER,
        TitledBorder.TOP));
    panelCentral.add(scrollTexto);
    panelImagen = new JPanel();
    panelImagen.setLayout(new BorderLayout());
    panelImagen.setBorder(BorderFactory.createTitledBorder(
        BorderFactory.createEtchedBorder(),
        "Área de imagen",
        TitledBorder.CENTER,
        TitledBorder.TOP));
    panelImagen.setBackground(Color.LIGHT_GRAY);
    panelCentral.add(panelImagen);
    panelPrincipal.add(panelCentral, BorderLayout.CENTER);
    JPanel panelSuperior = new JPanel(new BorderLayout());
    String[] mundos = {};
    comboMundos = new JComboBox<>(mundos);
    comboMundos.setPreferredSize(new Dimension(200, 30));
    panelSuperior.add(comboMundos, BorderLayout.EAST);
}

```

Jain int components to si le dan todas las características a todos los elementos que queremos al anteriormente para que se puedan visualizar en nuestra interfaz gráfica cómo los botones el panel de texto entre otros.

```

private void configurarEventos() {
    btnCargarArchivo.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            cargarArchivo();
        }
    });

    btnLimpiarArea.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            limpiarArea();
        }
    });

    btnAnalizarArchivo.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            analizarArchivo();
        }
    });

    btnGenerarReportes.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            generarReportes();
        }
    });
}

```

En configurar el evento se mandan a llamar todos los procedimientos para que al momento de presionar un botón se haga una acción.

Nuestra función de graficar también contiene muchos procedimientos como por ejemplo el de graficar mundo seleccionado no se verifica el mundo que seleccionó se seleccionó en el combobox para luego tomar ese valor y graficarlo dependiendo de lo que es seleccionado en el combobox también contiene uno para mostrar la imagen generada que es exactamente igual al que se hizo en la práctica pasada por lo que no se le tomará mucho énfasis también uno para cargar el archivo el cual abre un file chaser que igualmente es un tema visto en la práctica pasada por lo que es no se hola profundizará tanto, también contiene un procedimiento para limpiar el área que lo único que hace es de que sobrescribe un texto vacío, un procedimiento para poder actualizar los combos de los mundos el cual con un for itera la lista de los nombres de los mundos y los actualiza, también hay un procedimiento para poder generar los reportes el cual verifica que si está vacío indicará que no está lleno y en caso de que tenga hh

contenido se notificará que sí mandaron a realizar todo lo necesario.

Hola ahora haremos énfasis en el procedimiento más importante que es el de analizar archivo ya que en este de analizar archivo donde ocurre toda la magia.

## Procedimiento analizarArchivo

```
private void analizarArchivo() {
    String textoActual = txtAreaContenido.getText();

    if (textoActual != null && !textoActual.isEmpty()) {
        try {
            contenidoArchivo = new StringBuilder(textoActual);
            analizadorLexico.analizarArchivo(contenidoArchivo);
            analizadorLexico.imprimirTokens();
            System.out.println("");
            analizadorLexico.imprimirErrores();
            analizadorLexico.generarReporteHTML("C:\\Users\\aceba\\OneDrive\\Desktop\\Practical\\-LFP-202300673-\\Proyecto 2\\reporte.html");
            analizadorLexico.generarReporteErroresHTML("C:\\Users\\aceba\\OneDrive\\Desktop\\Practical\\-LFP-202300673-\\Proyecto 2\\reporteErrores.html");
            analizadorSintactico = new AnaSintactico(analizadorLexico.getTokens());
            analizadorSintactico.analizar();
            analizadorSintactico.generarReporteErroresHTML("C:\\Users\\aceba\\OneDrive\\Desktop\\Practical\\-LFP-202300673-\\Proyecto 2\\reporteSintactico.html");
            actualizarComboBoxMundos();
            JOptionPane.showMessageDialog(this,
                "Análisis completado exitosamente.",
                "Análisis Completado",
                JOptionPane.INFORMATION_MESSAGE);
        } catch (Exception ex) {
            JOptionPane.showMessageDialog(this,
                "Error durante el análisis: " + ex.getMessage(),
                "Error de Análisis",
                JOptionPane.ERROR_MESSAGE);
        }
    } else {
        JOptionPane.showMessageDialog(this,
            "No hay contenido para analizar. Por favor escriba o cargue texto primero.",
            "Sin Contenido",
            JOptionPane.WARNING_MESSAGE);
    }
}
```

En analizar archivo lo que hace es de que se crea primero un string de texto actual esto para que al momento de analizar el archivo se tome en cuenta lo que está en ese texto y no lo que se encontró en el archivo de entrada también se crea un try catch para el manejo de errores dentro de ese try catch hoy se coloca el contenido del archivo y también se manda a llamar la clase de analizador léxico esto con el fin de poder hacer uso de nuestra función de analizador léxico y hacerlo en base al contenido del archivo también se coloca el procedimiento de imprimir tokens de

analizador léxico con un espacio para que en consola podemos visualizar todos los tokens impresos y también con su método para poder imprimir los errores. Hoy en este procedimiento también se mandan a generar los reportes html de errores sintácticos léxicos y de tokens donde se coloca la ruta exacta donde se guardarán todos archivos y luego se crea una nuevos objeto del analizador sintáctico esto con el fin de poder mandarlo a analizar y que está formada todos los producciones necesarios para poder graficar también se actualiza el combobox de mundos una vez sea seleccionado el archivo y esta forma tendrá un mejor manejo de errores ya una vez graficado y analizado todo se puede hacer uso del botón de graficar el cual no funciona si no anteriormente se realizó un análisis del archivo.