

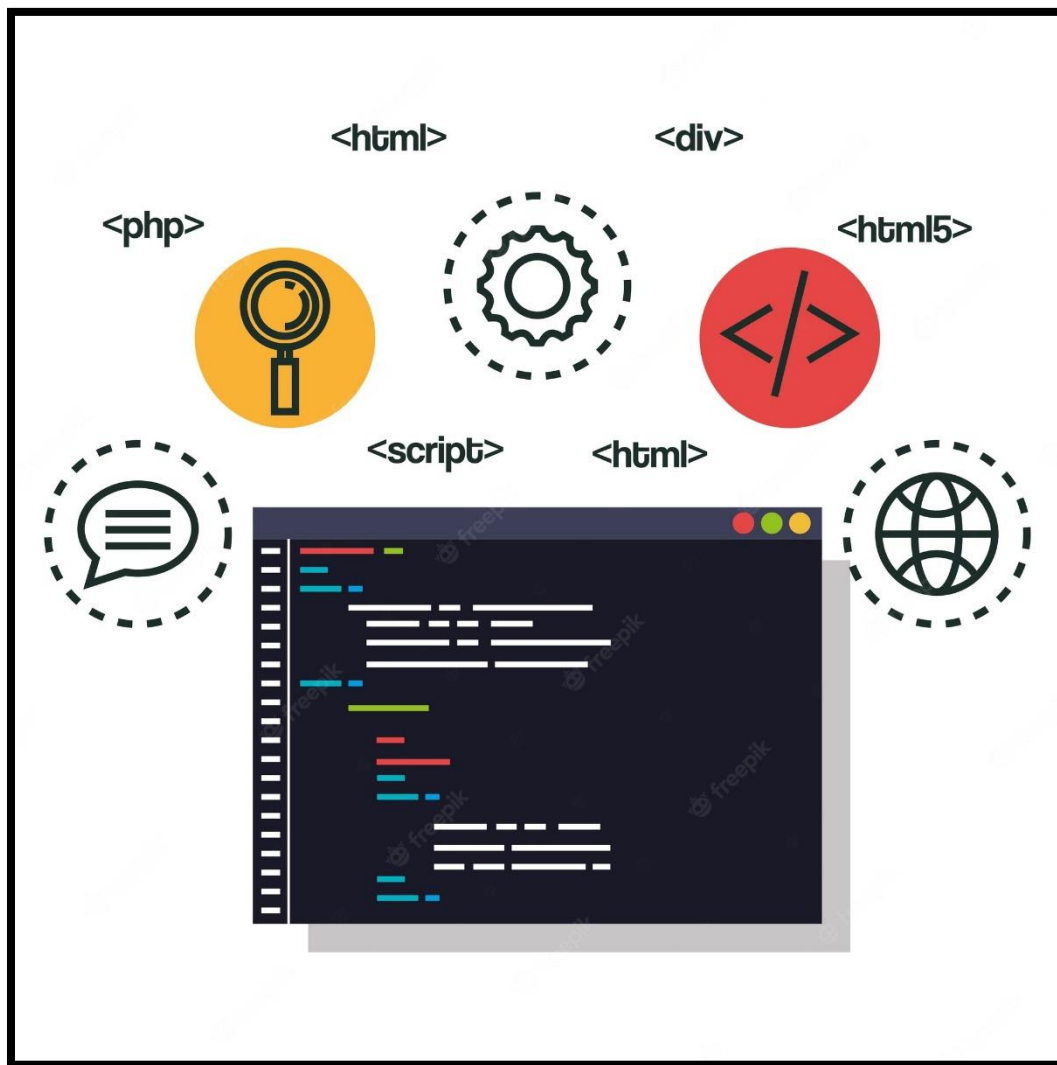
Universidad de San Carlos de Guatemala  
Facultad de ingeniería en Ciencias y Sistemas

OLC1-2023

Aux. Kevin Lajpop

Ing. Mario Jose Bautista Fuentes

## Manual Técnico



Nombre: Piter Angel Esaú Valiente De León

Carnet: 201902301

## **¿Qué es JDK?**

El JDK (Java Development Kit) es un conjunto de herramientas y recursos esenciales para desarrolladores de software que les permite crear, compilar y ejecutar aplicaciones Java. Incluye un compilador para convertir el código fuente Java en bytecode, una máquina virtual para ejecutar las aplicaciones en diferentes sistemas operativos, bibliotecas de clases estándar para tareas comunes, herramientas de línea de comandos para el desarrollo y depuración, y muchas otras utilidades que facilitan la programación en Java. En resumen, el JDK es fundamental para el desarrollo de aplicaciones Java, ya que proporciona todo lo necesario para construir software robusto y multiplataforma en este lenguaje de programación.

Version JDK utilizada: 19.0

## **¿Qué es Netbeans?**

NetBeans es un entorno de desarrollo integrado (IDE, por sus siglas en inglés) de código abierto ampliamente utilizado para la programación en varios lenguajes, con un enfoque particular en Java.

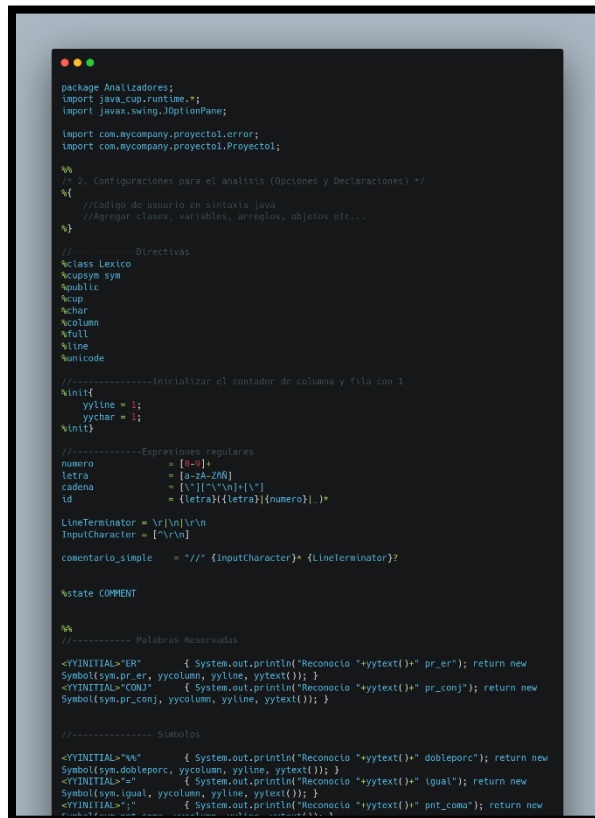
Versión de Netbeans utilizada: 16.1

## **¿Qué es el método de Thompson?**

El método de Thompson se refiere a un algoritmo utilizado para la construcción de autómatas finitos no deterministas (AFN) a partir de expresiones regulares. Fue propuesto por Ken Thompson, un informático estadounidense, en la década de 1960. Este método es fundamental en el diseño de compiladores y procesadores de lenguajes de programación.

## Estructura Jflex

JFlex (también conocido como JavaCC) es una herramienta utilizada para generar analizadores léxicos (scanners) en Java. Está diseñado para trabajar en conjunto con JavaCC (Java Compiler Compiler) para construir analizadores léxicos y sintácticos para procesadores de lenguajes de programación y compiladores



```
package Analizadores;
import java.cup.runtime.*;
import javax.swing.JOptionPane;
import com.mycompany.proyectol.error;
import com.mycompany.proyectol.Proyectol;

%{
    // 2. Configuraciones para el analisis (Opciones y Declaraciones) %}

    //Codigo de usuario en sintaxis java
    //Agregar clases, variables, arreglos, objetos etc...
%}

// Directivas
%class Lexico
%cupsym sym
%public
%cup
%char
%column
%full
%line
%unicode

//-----Inicializar el contador de columna y fila con 1
%init{
    yyline = 1;
    yychar = 1;
%init}

//-----Expresiones regulares
numero      = [0-9]+
letra       = [a-zA-Z0-9]
cadena      = [^"]*"
id          = (letra){(letra){(numero){}}

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

comentario_simple = "/*" {InputCharacter}* {LineTerminator}?

%state COMMENT

%{
    //----- Palabras Reservadas

    <%INITIAL="ER" { System.out.println("Reconocio "+yytext()+" pr_er"); return new
    Symbol(sym.pr_er, yycolumn, yyline, yytext()); }
    <%INITIAL="CONJ" { System.out.println("Reconocio "+yytext()+" pr_conj"); return new
    Symbol(sym.pr_conj, yycolumn, yyline, yytext()); }

    //----- Simbolos

    <%INITIAL="%%" { System.out.println("Reconocio "+yytext()+" dobleporc"); return new
    Symbol(sym.dobleporc, yycolumn, yyline, yytext()); }
    <%INITIAL="==" { System.out.println("Reconocio "+yytext()+" igual"); return new
    Symbol(sym.igual, yycolumn, yyline, yytext()); }
    <%INITIAL="," { System.out.println("Reconocio "+yytext()+" pnt_coma"); return new
```

## Estructura CUP

Un archivo con extensión .cup se asocia comúnmente con CUP, que es un generador de analizadores sintácticos para Java (Constructor of Useful Parsers). CUP es utilizado junto con JFlex (Java Fast Lexical Analyzer Generator) para construir analizadores léxicos y sintácticos para procesadores de lenguajes de programación y compiladores en Java.





```
package Generador;

import java.io.File;

public class G_Lexico
{
    public static void main(String[] args)
    {
        String path="src/main/java/Analizadores/Lexico.jflex";
        generarLexer(path);
    }

    public static void generarLexer(String path)
    {
        File file=new File(path);
        jflex.Main.generate(file);
    }
}
```

## Generador sintactico

Un generador sintáctico, también conocido como generador de analizadores sintácticos o generador de parsers, es una herramienta de software que facilita la creación de analizadores sintácticos para un lenguaje de programación específico. Estos analizadores sintácticos son parte fundamental de un compilador o intérprete, encargándose de analizar la estructura gramatical del código fuente y construir un árbol de análisis sintáctico.

```
package Generador;

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

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

        //Dirección de carpeta donde se va a generar el parser.java & el simbolosxxx.java
        opciones[1] = "src/main/java/Analizadores";

        opciones[2] = "-symbols";
        opciones[3] = "sym"; // nombre que tendrá la clase de símbolos

        opciones[4] = "--parser";
        opciones[5] = "Sintactico"; // nombre a la clase del paso anterior

        //Ubicación archivo .cup
        opciones[6] = "src/main/java/Analizadores/Sintactico.cup";
        try
        {
            java_cup.Main.main(opciones);
        }
        catch (Exception ex)
        {
            System.out.print(ex);
        }
    }
}
```

## Rango de árbol

El método Arreglo en la clase Rango tiene como objetivo generar un arreglo de caracteres que representan un rango específico. Este rango es definido por dos caracteres, el inicio y el fin. La función realiza lo siguiente:

Crea una lista enlazada de caracteres llamada caracteres.

Verifica si el primer carácter (primerChar) es mayor que el segundo carácter (segundoChar). Si es así, intercambia los valores para asegurarse de que primerChar sea menor o igual que segundoChar. Si ambos caracteres son iguales, devuelve la lista de caracteres en ese momento.

Verifica si el primer carácter es un símbolo, es decir, si su valor ASCII está fuera de ciertos rangos específicos. Si es un símbolo, se establece la variable arr\_simbol en true.

Utiliza un bucle for que itera desde el primerChar hasta el segundoChar. Dentro del bucle:

Excluye el primerChar y el segundoChar de la lista de caracteres.

Si la variable `arr_simbol` es `true`, realiza algunas conversiones específicas para tratar con símbolos numéricos y letras. Por ejemplo, si el `primerChar` es 'O', entonces el bucle se ajusta para incluir todos los dígitos numéricos.

Agrega cada caracter al arreglo `caracteres`.

Agrega el `segundoChar` a la lista de caracteres.

Devuelve la lista enlazada de caracteres resultante.

```
package Simbolos;

import java.util.LinkedList;

public class Rango {
    private String inicio;
    private String fin;

    public Rango(String inicio, String fin) {
        this.inicio = inicio;
        this.fin = fin;
    }

    public String getInicio() {
        return inicio;
    }

    public void setInicio(String inicio) {
        this.inicio = inicio;
    }

    public String getFin() {
        return fin;
    }

    public void setFin(String fin) {
        this.fin = fin;
    }

    public LinkedList<Character> Arreglo() {
        //Función para crear arreglo con el rango de caracteres
        boolean arr_simbol = false;
        LinkedList<Character> caracteres = new LinkedList<>();
        char primerChar = this.inicio.charAt(0);
        char segundoChar = this.fin.charAt(0);
        caracteres.add(primerChar);
        //Colocar el caracter mas grande al inicio
        if (primerChar > segundoChar) {
            char temp = primerChar;
            primerChar = segundoChar;
            segundoChar = temp;
        } else if (primerChar == segundoChar) {
            return caracteres;
        }

        if (primerChar <= 47 || (primerChar >= 58 && primerChar <= 64)
            || (primerChar >= 91 && primerChar <= 96) || primerChar >= 123) {
            arr_simbol = true;
        }

        for (char c = primerChar; c <= segundoChar; c++) {
            if (c != primerChar && c != segundoChar) {
                if (arr_simbol) {
                    if (c == 48) {
                        c = 57;
                        continue;
                    } else if (c == 65) {
                        c = 90;
                        continue;
                    } else if (c == 97) {
                        c = 122;
                        continue;
                    }
                }
                caracteres.add(c);
            }
        }
        caracteres.add(segundoChar);
        return caracteres;
    }
}
```

## Conjunto

La clase Conjunto en el paquete Simbolos parece ser una representación de un conjunto de caracteres en algún contexto específico. Aquí está la explicación de la funcionalidad de la clase:

Atributos:

id: Representa la identificación o nombre del conjunto.

caracteres: Almacena los caracteres que pertenecen al conjunto en una lista enlazada.

Constructor:

Conjunto(String id, LinkedList<Character> caracteres): El constructor de la clase toma dos parámetros, un id que identifica al conjunto y una lista de caracteres que pertenecen al conjunto. Asigna estos valores a los atributos correspondientes.

Métodos de Acceso (Getters y Setters):

getId(): Devuelve el identificador del conjunto.

setId(String id): Establece el identificador del conjunto.

getCaracteres(): Devuelve la lista de caracteres que pertenecen al conjunto.

setCaracteres(LinkedList<Character> caracteres): Establece la lista de caracteres del conjunto.



```

package Simbolos;
import java.util.LinkedList;

public class Conjunto {
    private String id;
    private LinkedList<Character> caracteres;

    public Conjunto(String id, LinkedList<Character> caracteres) {
        this.id = id;
        this.caracteres = caracteres;
    }

    public String getId() {
        return id;
    }

    public void setId(String id) {
        this.id = id;
    }

    public LinkedList<Character> getCaracteres() {
        return caracteres;
    }

    public void setCaracteres(LinkedList<Character> caracteres) {
        this.caracteres = caracteres;
    }
}

```

## Graficar Thompson

El método GraficaNodosThompson pertenece a una clase y parece ser utilizado para generar representaciones gráficas de nodos en un árbol Thompson, que es una estructura utilizada en la construcción de autómatas finitos no deterministas (AFN) a partir de expresiones regulares. Aquí está una explicación de lo que hace el método:

**Parámetros:**

**Nodo nodo:** Representa un nodo en el árbol Thompson.

**String i:** Un identificador que se utiliza para nombrar los nodos en la representación gráfica.

**Variables Locales:**

**int k:** Una variable local que parece no ser utilizada efectivamente en el código.

**String r:** Una cadena que almacena la representación gráfica de los nodos.

Proceso:

Se inicializan las variables locales k y r.

Se extrae el token del nodo (nodo.token).

Se crea un nombre único para el nodo (nodoName) utilizando el identificador i.

Se realiza una serie de verificaciones y ajustes en el token y se asignan valores a las propiedades first\_thomp y last\_thomp del nodo.

Dependiendo del tipo de nodo (determinado por el token), se realiza una acción específica:

Para . (concatenación), se generan nodos y conexiones apropiadas.

Para | (unión), se generan nodos y conexiones con transiciones  $\epsilon$ .

Para + (cierre positivo), se generan nodos y conexiones con transiciones  $\epsilon$ .

Para \* (cierre de Kleene), se generan nodos y conexiones con transiciones  $\epsilon$ .

Para ? (opcional), se generan nodos y conexiones con transiciones  $\epsilon$ .

Se incrementa la variable k.

Se devuelve la cadena resultante r que contiene la representación gráfica.

```

private String GraficaNodosThompson(Nodo nodo, String l){
    int k=0;
    String r = "";
    String nodoTerm = nodo.token;
    String nodoName = "node" + i;
    nodoTerm = nodoTerm.replace("\\", "");
    //Primer nodo = "" utilizo nodo = _1
    if(nodo.tipo != TIPOS.SIMBOLO){
        if("." equals(nodo.first_thomp)){
            nodo.first_thomp = "nodeh" + nodo.id + "_0";
        }
        nodo.last_thomp = "nodeh" + nodo.id + "_1";
        r += nodo.first_thomp + "[label= \"\"] \n";
        r += nodo.last_thomp + "[label= \"\"] \n";
        r += nodo.first_thomp + "->" + nodo.last_thomp + "[label= \"\" + nodoTerm + \"\"]";
    }
    \n";
}

if(nodo.tipo == TIPOS.SIMBOLO) switch(nodo.token){
    case "+":
        if("." equals(nodo.first_thomp)){
            nodo.hijoIzq.first_thomp = nodo.first_thomp;
        }
        r += GraficaNodosThompson(nodo.hijoIzq, i+"0");
        nodo.hijoDer.first_thomp = nodo.hijoIzq.last_thomp;
        r += GraficaNodosThompson(nodo.hijoDer, i+"1");

        nodo.first_thomp = nodo.hijoIzq.first_thomp;
        nodo.last_thomp = nodo.hijoDer.last_thomp;
        break;

    case "|":
        if("." equals(nodo.first_thomp)){
            nodo.first_thomp = "nodo" + i + "_0";
        }
        r += GraficaNodosThompson(nodo.hijoIzq, i+"0");
        r += GraficaNodosThompson(nodo.hijoDer, i+"1");

        r += nodo.first_thomp + "[label= \"\"] \n";
        r += "nodo" + i + "_1[label= \"\"] \n";
        r += nodo.first_thomp + "->" + nodo.hijoIzq.first_thomp + "[label= \"e\"]";
        r += nodo.first_thomp + "->" + nodo.hijoDer.first_thomp + "[label= \"e\"]";

        r += nodo.hijoIzq.last_thomp + "->" + "nodo" + i + "_1" + "[label= \"e\"]";
        r += nodo.hijoDer.last_thomp + "->" + "nodo" + i + "_1" + "[label= \"e\"]";

        nodo.last_thomp = "nodo" + i + "_1";
        break;

    case "*":
        if("." equals(nodo.first_thomp)){
            nodo.first_thomp = "nodo" + i + "_0";
        }
        r += GraficaNodosThompson(nodo.hijoIzq, i+"0");

        r += nodo.first_thomp + "[label= \"\"] \n";
        r += "nodo" + i + "_1[label= \"\"] \n";
        r += nodo.first_thomp + "->" + nodo.hijoIzq.first_thomp + "[label= \"e\"]";
        r += nodo.hijoIzq.last_thomp + "->" + "nodo" + i + "_1" + "[label= \"e\"]";

        r += nodo.hijoIzq.last_thomp + "->" + nodo.hijoIzq.first_thomp + "[label= \"e\", constraint=false]"; \n";
        nodo.last_thomp = "nodo" + i + "_1";
        break;

    case "^":
        if("." equals(nodo.first_thomp)){
            nodo.first_thomp = "nodo" + i + "_0";
        }
        r += GraficaNodosThompson(nodo.hijoIzq, i+"0");

        r += nodo.first_thomp + "[label= \"\"] \n";
        r += "nodo" + i + "_1[label= \"\"] \n";
        r += nodo.first_thomp + "->" + nodo.hijoIzq.first_thomp + "[label= \"e\"]";
        r += nodo.hijoIzq.last_thomp + "->" + "nodo" + i + "_1" + "[label= \"e\"]";

        r += nodo.hijoIzq.last_thomp + "->" + nodo.hijoIzq.first_thomp + "[label= \"e\", constraint=false]"; \n";
        r += nodo.first_thomp + "->" + "nodo" + i + "_1" + "[label= \"e\" , constraint=false]"; \n";
        nodo.last_thomp = "nodo" + i + "_1";
        break;

    case "?":
        if("." equals(nodo.first_thomp)){
            nodo.first_thomp = "nodo" + i + "_0";
        }
        r += GraficaNodosThompson(nodo.hijoIzq, i+"0");

        r += nodo.first_thomp + "[label= \"\"] \n";
        r += "nodo" + i + "_1[label= \"\"] \n";
        r += nodo.first_thomp + "->" + nodo.hijoIzq.first_thomp + "[label= \"e\"]";
        r += nodo.hijoIzq.last_thomp + "->" + "nodo" + i + "_1" + "[label= \"e\"]";

        r += nodo.first_thomp + "->" + "nodo" + i + "_1" + "[label= \"e\" , constraint=false]"; \n";
        nodo.last_thomp = "nodo" + i + "_1";
        break;

    default:
        break;
}
k++;
return r;
}

```

### Método GraficarThompson()

Este método genera la representación gráfica de un árbol Thompson utilizando el formato DOT, que es un lenguaje de descripción de gráficos. El método realiza las siguientes acciones:

Construye una cadena grafica que contiene la descripción del grafo en formato DOT.

Llama al método GraficaNodosThompson con el nodo raíz del árbol Thompson y un identificador inicial "O" para obtener la descripción de los nodos en el grafo.

Agrega un nodo adicional correspondiente al último nodo del árbol (doble círculo) con una etiqueta vacía.

Llama al método GenerarDot para generar un archivo DOT y, posteriormente, generar una imagen del grafo utilizando Graphviz.

### Método AnalizarCadena(String text, Object sim)

Este método utiliza un objeto de tipo Transition para realizar el análisis de una cadena de texto en el contexto del autómata finito representado por el árbol. Devuelve un valor booleano que indica si la cadena es aceptada por el autómata.

### Método privado GraficaNodos(Nodo nodo, String i)

Este método es utilizado por el método GraficarThompson para generar la descripción de los nodos en el grafo DOT. Realiza lo siguiente:

Inicializa la variable k a 0 y la cadena r a una cadena vacía.

Determina un valor adicional para la forma y el color de los nodos en función de propiedades específicas del nodo (aceptado, anulable, tipo, etc.).

Construye la descripción del nodo actual (nodei) con su etiqueta y propiedades adicionales.

Itera sobre los hijos del nodo actual y genera conexiones y descripciones de los nodos hijos llamando recursivamente a GraficaNodos.

Devuelve la cadena resultante que representa la descripción del nodo y sus conexiones en formato DOT.

```

    public void GraficarThompson() throws InterruptedException{

        String grafica = "digraph AFD{\n" +
        "\n" +
        " rankdir=LR;\n" +
        "node [shape = circle]\n" +
            GraficaNodosThompson(this.raiz.hijoIzq, "0") +
            this.raiz.hijoIzq.last_thomp + "[label = \"\", shape=doublecircle ] \n}";
        GenerarDot("AFND_201902301/AFND", grafica);

    }

    public boolean AnalizarCadena(String text, Object sim) {
        Transition tran = new Transition(raiz, table, leaves);
        return tran.analizar(text, sim);
    }

    private String GraficaNodos(Nodo nodo, String i){
        int k=0;
        String r = "";
        String nodoTerm = nodo.token;

        String extra = "";

        if(nodo.tipo != TIPOS.SIMBOLO){
            extra = ", shape = oval";
        }
        if(nodo.aceptado){
            extra = ",shape=doublecircle, color=\"#2BFF1D\"";
        }
        if(nodo.anulable){
            extra = "shape=doublecircle, color=red";
        }

        nodoTerm = nodoTerm.replace("\\", "");
        r= "node" + i + "[label = \"\" + nodoTerm + \"\" + extra + \" ];\n";

        for(int j =0 ; j<=nodo.hijos.size()-1; j++){
            r = r + "node" + i + " -- node" + i + k + "\n";

            r = r + GraficaNodos(nodo.hijos.get(j), ""+i+k);
            k++;
        }

        return r;
    }
}

```

Método addLeave(Nodo nodo, ArrayList<Nodo> leaves)

Este método agrega un nodo (Nodo) a una lista de nodos (ArrayList<Nodo>). La lista leaves se utiliza para almacenar nodos específicos en algún contexto no proporcionado en el código. La función simplemente añade el nodo pasado como argumento a la lista de nodos.

Método getLeave(int numLeave, ArrayList<Nodo> leaves)

Este método busca y devuelve un nodo específico de la lista de nodos (leaves) basándose en su identificador (numLeave). Itera sobre la lista de nodos y devuelve el nodo cuyo identificador coincide con el valor proporcionado (numLeave). Si no se encuentra ningún nodo con ese identificador, devuelve null.

Método isAccept(int numLeave, ArrayList<Nodo> leaves)

Este método verifica si un nodo específico en la lista de nodos (leaves) es aceptado. Utiliza el identificador (numLeave) para encontrar el nodo correspondiente y devuelve true si ese nodo está marcado como aceptado (aceptado). En caso contrario, devuelve false.

Método getTipo(int numLeave, ArrayList<Nodo> leaves)

Este método devuelve el tipo (TIPOS) de un nodo específico en la lista de nodos (leaves). Utiliza el identificador (numLeave) para encontrar el nodo correspondiente y devuelve el tipo del nodo. Si no se encuentra el nodo con ese identificador, devuelve TIPOS.CADENA.

```

package com.mycompany.proyecto1;

import java.util.ArrayList;

public class Hojas {
    public void addLeave(Nodo nodo, ArrayList<Nodo> leaves){
        leaves.add(nodo);
    }

    public Nodo getLeave(int numLeave, ArrayList<Nodo> leaves){
        for (Nodo item : leaves) {
            if(item.id == numLeave) return item;
        }
        return null;
    }

    public boolean isAccept(int numLeave, ArrayList<Nodo> leaves){
        for (Nodo item : leaves) {
            if(item.id == numLeave) return item.aceptado;
        }
        return false;
    }

    public TIPOS getTipo(int numLeave, ArrayList<Nodo> leaves){
        for (Nodo item : leaves) {
            if(item.id == numLeave) return item.tipo;
        }
        return TIPOS.CADENA;
    }
}

```

## Atributos de la Clase Nodo

prim\_pos y ulti\_pos: Listas de enteros que representan conjuntos de posiciones primeras y últimas, respectivamente.

token: Una cadena que almacena el token asociado al nodo.

tipo: Enumeración (TIPOS) que representa el tipo del nodo (por ejemplo, ESCAPE, CADENA, ID, SIMBOLO).

lexema: Una cadena que almacena el lexema asociado al nodo.

id: Un entero que representa el identificador único del nodo.

aceptado: Un booleano que indica si el nodo es aceptado.

anulable: Un booleano que indica si el nodo es anulable.

hijolq y hijoDer: Referencias a los nodos hijos izquierdo y derecho, respectivamente.

first\_thomp y last\_thomp: Cadenas que almacenan información sobre el primer y último conjunto de posiciones, específicamente utilizadas en el contexto de la construcción de autómatas finitos de Thompson.

hijos: Una lista de nodos hijos.

tabla: Una lista de listas utilizada para almacenar información adicional sobre el nodo.

hojas: Una lista de nodos que representan las hojas del árbol.

### Constructor de la Clase Nodo

El constructor de la clase inicializa varios atributos del nodo, incluyendo las listas prim\_pos y ulti\_pos, el token, el tipo, el lexema, el identificador, los nodos hijos, la lista de hojas y la tabla.

### Método getNode()

Este método realiza una serie de operaciones para calcular y establecer propiedades como anulable, prim\_pos, y ulti\_pos del nodo y sus hijos, según el tipo de nodo y su token (por ejemplo, "." para concatenación, "|" para unión, "\*" para cierre de Kleene, etc.).

### Método follow()

Este método también realiza operaciones específicas en función del tipo de nodo y su token para calcular y establecer conjuntos de seguimiento (Follow). Este método parece estar relacionado con la construcción de la tabla de análisis sintáctico.



[illegible]