**UNIVERSIDAD DE SAN CARLOS DE GUATEMALA**
**FACULTAD DE INGENIERÍA**
**ESCUELA DE CIENCIAS**
**PRIMER SEMESTRE 2022**

OLC C

# Laboratorio Organización de Lenguajes y Compiladores 1 – Proyecto 1

**Manual técnico**

Nombre: José David Panaza Batres
Carné: 202111478
Auxiliar: José Diego Pérez Toralla
Fecha: 20 de marzo de 2023

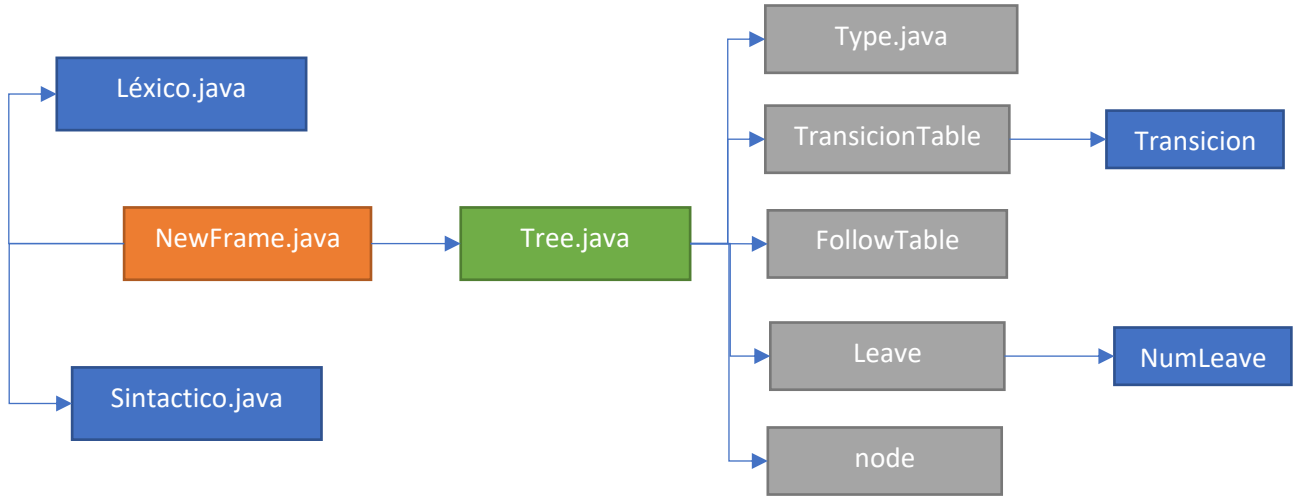
# Manual técnico

Este programa fue desarrollado enteramente en Python aplicando el paradigma de programación orientada a objetos (POO). Esto se debe a que se tenía en mente realizar una aplicación de escritorio, que fuese intuitiva y fácil de usar a cualquier usuario. Todo esto con el fin de poder ayudar al estudiante a organizar su pensum según la información de sus cursos.

## Patrón de diseño

Debido a la simplicidad de las funciones, se decidió utilizar un patrón de diseño orientado a lo creacional. Especificando de mejor manera, se utilizó un patrón Singleton. Este se refiere a un diseño que restringe la creación de instancias de una clase a un único objeto con varias funciones. Estas se utilizaron para crear los árboles y sus respectivos componentes.

## Diagrama de clases



## Métodos principales

- Generar Automata Button4(): Se obtienen los objetos del parser del archivo cup.

```
Sintactico sintactico = new Sintactico(new Lexico(new BufferedReader(new StringReader(jTextArea1.getText()))));
sintactico.parse();
// CONVERTIR LAS LISTAS A ARRAYS PARA EL MANEJOR
listaconjuntos = new Sintactico.Conjunto[sintactico.conj.size()];
listaexpresiones = new Sintactico.Expresion[sintactico.expr.size()];
listaevaluaciones = new Sintactico.Evaluar[sintactico.eval.size()];
estados =  new ArrayList<>();
// PASAR CONJUNTOS
for (int i = 0; i < listaconjuntos.length; i++) {
    listaconjuntos[i] = sintactico.conj.get(i);
}
// PASAR EXPRESIONES
for (int i = 0; i < listaexpresiones.length; i++) {
    listaexpresiones[i] = sintactico.expr.get(i);
}
// PASAR EVALUACIONES
for (int i = 0; i < listaevaluaciones.length; i++) {
    listaevaluaciones[i] = sintactico.eval.get(i);
```

- ExpandirConjuntos(): Se evalúan los conjuntos definidos y se llenas a conveniencia para después realizar un index of.

```
String letrasmin = "abcdefghijklmnñopqrstuvwxyz";
String letrasmayus = "ABCDEFGHIJKLMNÑOPQRSTUVWXYZ";
String chars = " !\"#$%&'()*+,-./:;<=>?@[\\]^_`{|}~";
String numeros = "0123456789";
for (int i = 0; i < listaconjuntos.length; i++) {
    var rango = listaconjuntos[i].rango.trim();
    var rarray = rango.toCharArray();
    if (rango.length() == 3 && (rarray[1] == '-' || rarray[1] == '~')) {
        if (letrasmin.indexOf(rarray[0]) != -1 && letrasmin.indexOf(rarray[2]) != -1) {
            var limiteinf = letrasmin.indexOf(rarray[0]);
            var limitesup = letrasmin.indexOf(rarray[2]);
            String rangof = "";
            if (limitesup > limiteinf) {
                var lista = letrasmin.toCharArray();
                for (int j = 0; j < letrasmin.length(); j++) {
                    if (j >= limiteinf && j <= limitesup) {
                        rangof = rangof + lista[j];
                    }
                }
                listaconjuntos[i].rango = rangof;
            } else {
                listaconjuntos[i].rango = "-1";
                continue;
            }
```

- DividirExpresion(): Se verifica si los títulos utilizados existen. De no ser así se elimina la expresión y se agrega una coma en cada expresión para dividirlas posteriormente

```
for (int i = 0; i < listaexpresiones.length; i++) {
    ArrayList<String> titulosexp = new ArrayList<>();
    var expresion = listaexpresiones[i].exp;
    var arrayexp = expresion.toCharArray();
    var titulos = "";
    boolean title = false;
    for (int j = 0; j < arrayexp.length; j++) {
        if (arrayexp[j] == '{') {
            title = true;
        } else if (arrayexp[j] == '}') {
            title = false;
            titulosexp.add(titulos);
            titulos = "";
        }
        if (title && arrayexp[j] != '{') {
            titulos = titulos + arrayexp[j];
        }
    }
    for (int j = 0; j < titulosexp.size(); j++) {
        boolean goodconj = false;
        for (int w = 0; w < listaconjuntos.length; w++) {
            if (titulosexp.get(j).equals(listaconjuntos[w].titulo)) {
                goodconj = true;
                continue;
            }
        }
        if (!goodconj) {
            listaexpresiones[i].exp = "-1";
            break;
        }
    }

}
```

```
String divisores = ".|*?+";
for (int i = 0; i < listaexpresiones.length; i++) {
    var expresion = listaexpresiones[i].exp;
    var arrayexp = expresion.toCharArray();
    var nuevaexp = "";
    for (int j = 0; j < arrayexp.length; j++){
        if(divisores.indexOf(arrayexp[j]) != -1 ){
            nuevaexp = nuevaexp + arrayexp[j] + ",";
        }
        else{
            if(arrayexp[j] == '\"'){
                if(arrayexp[j+1] == '\"'){
                    nuevaexp = nuevaexp + arrayexp[j]+ " " + arrayexp[j+1] + ",";
                    j = j + 1;
                }
                else{
                    nuevaexp = nuevaexp + arrayexp[j] + arrayexp[j+1] + arrayexp[j+2] + ",";
                    j = j + 2;
                }
            }
            else if(arrayexp[j] == '\\'){
                nuevaexp = nuevaexp + arrayexp[j] + arrayexp[j+1] + ",";
                j = j + 1;
            }
            else if(arrayexp[j] == '}'){
                nuevaexp = nuevaexp + ",";
            }
            else{
                if(arrayexp[j] != '{'){
                    nuevaexp = nuevaexp + arrayexp[j];
                }
            }
        }
    }
    listaexpresiones[i].exp = nuevaexp;
```

- CrearArboles(String titulo, String er): Se crear el objeto árbol. Se grafican los nodos según la expresión y se van graficando los arboles, las tablas, y los autómatas.

```
private void crearArboles(String titulo,String er){
    ArrayList<node> leaves = new ArrayList();
    ArrayList<ArrayList> table = new ArrayList();
    er = "." + er + "#";

    Tree arbol = new Tree(er, leaves, table);
    node raiz = arbol.getRoot();
    raiz.getNode();
    raiz.follow();

    arbol.GraficarArbol(titulo); // GRAFICAR ARBOLES

    followTable ft = new followTable();
    ft.printTable(table, titulo); // GRAFICAR TABLA SIGUIENTES

    transitionTable tran = new transitionTable(raiz, table, leaves);
    tran.impTable(titulo); // GRAFICAR TABLA TRANSICIONES
    tran.impGraph(titulo); // GRAFICAR AUTOMATA
    estados.add(tran.states);
}
```

- Clase arbol: Se van colocando los nodos y hojas

```
numLeave numHoja =  new numLeave(conthojas);
Stack<node> pila = new Stack();


ArrayList<String> strList = new ArrayList<>(Arrays.asList(erSplit));
Collections.reverse(strList);
cont = 0;
strList.forEach((character) -> {
    switch(character){
        case "|":
            node lefto = (node) pila.pop();
            node righto = (node) pila.pop();

            node no = new node(character, Types.OR, number:0, cont, lefto, righto, leaves, table);
            cont++;
            pila.push(no);
            break;
        case ".":
            node lefta = (node) pila.pop();
            node righta = (node) pila.pop();

            node na = new node(character, Types.AND, number:0, cont,lefta, righta, leaves, table);
            cont++;
            pila.push(na);
            break;
        case "*":
            node unario = (node) pila.pop();
            node ncom = new node(character, Types.CERO_MAS, number:0,cont, unario, right:null, leaves, table);
            cont++;
            pila.push(ncom);
            break;
        case "+":
            node unario1 = (node) pila.pop();
            node nuom = new node(character, Types.UNO_MAS, number:0, cont, unario1, right:null, leaves, table);
            cont++;
            pila.push(nuom);
            break;
        case "?":
            node unario2 = (node) pila.pop();
            node ncou = new node(character, Types.CERO_UNO, number:0, cont, unario2, right:null, leaves, table);
            cont++;
            pila.push(ncou);
            break;
        default:
            node nd = new node(character, Types.HOJA, numHoja.getNum(), cont, left:null, right:null, leaves, table);
            cont++;
            pila.push(nd); //construir el arbol
            leave hoja = new leave();
            hoja.addLeave(nd, leaves); // TABLA DE SIGUIENTES O TRANSICIONES
            break;
    }
});
this.root = (node) pila.pop();
```

- GraficarArbol(): Se recorre recursivamente el árbol y se van creando los objetos de graphviz para convertirlo a jpg

```
private String GraficaNodos(node nodo){
    String etiqueta = "";
    String primeros = "";
    String ultimos = "";
    String anulable = "";
    if(nodo.anullable){
        anulable = "A";
    }
    else{
        anulable = "N";
    }
    if(nodo.first.size() > 0){
        for(int i = 0; i<nodo.first.size();i++){
            if(i == nodo.first.size() - 1){
                primeros += nodo.first.get(i);
            }
            else{
                primeros += nodo.first.get(i) + ",";
            }
        }
    }
    if(nodo.last.size() > 0){
        for(int i = 0; i<nodo.last.size();i++){
            if(i == nodo.last.size() - 1){
                ultimos += nodo.last.get(i);
            }
            else{
                ultimos += nodo.last.get(i) + ",";
            }
        }
    }
    var arraylex = nodo.lexeme.toCharArray();
    if (nodo.left == null && nodo.right == null){
        if(arraylex[0] == '\\'){
            etiqueta += "nodo" + nodo.id + "[label = \"" + primeros + "|{" + anulable + "|\\" + nodo.lexeme + "|" + nodo.number + "}|" + ultimos + "\"];\n" ;
            //etiqueta += "nodo" + nodo.id + "[label = \"\\" + nodo.lexeme +  "\"];\n";
        }
        else if(arraylex[0] == '\"'){
            etiqueta += "nodo" + nodo.id + "[label = \"" + primeros + "|{" + anulable + "|\\" + arraylex[0] + arraylex[1] + "\\" + arraylex[2] + "|" + nodo.number + "}|" + ultimos + "\"];\n" ;
            //etiqueta += "nodo" + nodo.id + "[label = " + nodo.lexeme +  "];\n";
        }
        else if(arraylex[0] == '|'){
            etiqueta += "nodo" + nodo.id + "[label = \"" + primeros + "|{" + anulable + "|\\" + nodo.lexeme + "|" + nodo.number + "}|" + ultimos + "\"];\n" ;
        }
        else{
            etiqueta += "nodo" + nodo.id + "[label = \"" + primeros + "|{" + anulable + "|" + nodo.lexeme + "|" + nodo.number + "}|" + ultimos + "\"];\n" ;
            //etiqueta += "nodo" + nodo.id + "[label = \"" + nodo.lexeme +  "\"];\n";
        }
    }
```

- Clase FollowTable: de igual manera se analizan las posiciones siguientes que se definieron en el árbol para después crear los objetos en una tabla

```
public ArrayList next(int numNode, ArrayList<ArrayList> table){
    ArrayList result = new ArrayList();
    for(ArrayList item : table){
        if( (int) item.get(index:0) == numNode ){
            result.add(item.get(index:1));
            result.add(((ArrayList)item.get(index:2)).clone());
            return result;
        }
    }
    result.add(e:"");
    result.add(new ArrayList());
return result;
}

public void printTable(ArrayList<ArrayList> table, String titulo){//TABLA SIGUIENTES
    String tabla = "digraph G { \n node[shape=none fontname=Helvetica]\n";
    tabla += "n1[label = < \n <table>";
    tabla += "<tr><td colspan=\"2\">Hojas</td><td>Siguientes</td></tr>\n";
    String[] sigs = new String[table.size()];
    for(ArrayList item : table){
        sigs[(int)item.get(index:0)] = "<tr><td>" + item.get(index:1) + "</td><td>" + item.get(index:0) + "</td><td>" + item.get(index:2) + "</td></tr>\n";
    }
    for(int i = 0; i < sigs.length; i++){
        tabla += sigs[i];
    }
    tabla += "</table>\n>]\n}";
    crearTabla(titulo, tabla);
}
```

- Clase TransitionTable : Se analizan las transiciones de cada estado. A donde se va y con qué lexema se va.

```
    ArrayList<String> terminales =new ArrayList();

    for(ArrayList state : states){

        for(Object tr : (ArrayList)state.get(index:2)){
            transicion t = (transicion) tr;
            if(terminales.size() == 0){
                terminales.add(t.transition);
            }
            else{
                boolean repe = false;
                for (int i=0;i<terminales.size();i++){
                    if(t.transition.equals(terminales.get(i))){
                        repe = true;
                    }
                }
                if(!repe){
                    terminales.add(t.transition);
                }
            }
        }
    }
    String tabla = "digraph G { \n node[shape=none fontname=Helvetica]\n";
    tabla += "n1[label = < \n <table>";
    tabla += "<tr><td rowspan=\"2\">Estado</td><td colspan=\""+terminales.size()+"\">Terminales</td></tr>\n";
    tabla += "<tr>\n";
    for (int i=0;i<terminales.size();i++){
        tabla += "<td>" + terminales.get(i) + "</td>\n";
    }
    tabla += "</tr>\n";
    String[] trans = new String[terminales.size()];
    for(ArrayList state : states){
        for(int i= 0; i<trans.length;i++){
            trans[i] = "--";
        }
        tabla += "<tr>\n";
        tabla += "<td>" + state.get(index:0) + " " + state.get(index:1) + "</td>\n";
        for(Object tr : (ArrayList)state.get(index:2)){
            transicion t = (transicion) tr;
            for (int i = 0; i< terminales.size();i++){
                if(t.transition.equals(terminales.get(i))){
                    trans[i] = t.finalState;
                }
            }
        }
        for(int i =0; i<trans.length;i++){
            tabla += "<td>" + trans[i] + "</td>\n";
        }
        tabla += "</tr>\n";
    }
    tabla += "</table>\n>]\n}";
    crearTabla(titulo, tabla);
}
```

- impGraph(): se crea la grafica del AFD con los estados y transiciones

```
public void impGraph(String titulo){//AFD
    String graph = "digraph G { \n rankdir=\"LR\"";
    for(ArrayList state : states){
        if((boolean)state.get(index:3) == true){
            graph += state.get(index:0) + "[shape = doublecircle];" + "\n";
        }
        for(Object tr : (ArrayList)state.get(index:2)){
            transicion t = (transicion) tr;
            graph += t.graph() + "\n";
        }
    }
    graph += "}";
    crearGrafica(titulo, graph);
}
```

- GraficaANDF(): se recorre el árbol recursivamente y se van generando los objetos según las operaciones y los hijos que vienen por la izquierda o derecha.

```
int conts = 0;
private String GraficaANDF(node nodo){
    String etiqueta = "";
    var arraylex = nodo.lexeme.toCharArray();
    if (nodo.left == null && nodo.right == null){
        if(arraylex[0] == '\\'){
            etiqueta +=  "[label=\"\\" + nodo.lexeme + "\"]";
        }
        else if(arraylex[0] == '\"'){
            etiqueta +=  "[label=\"\\" + arraylex[0] + arraylex[1] + "\\" + arraylex[2] +"\"]";
        }
        else{
            etiqueta +=  "[label=\"" + nodo.lexeme + "\"]";
        }
    }
    else{
        if(arraylex[0] == '.'){
            if(nohijos((node) nodo.left)){
                etiqueta += "\nS" + conts + " -> S" + (++conts) + GraficaANDF((node)nodo.left);
            }
            else{
                etiqueta += GraficaANDF((node) nodo.left);
            }
            if(nohijos((node) nodo.right)){
                etiqueta += "\nS" + conts + " -> S" + (++conts) + GraficaANDF((node)nodo.right);
            }
            else{
                etiqueta += GraficaANDF((node) nodo.right);
            }

        }
```

```
    else if(arraylex[0] == '|'){
        var contor = conts;
        etiqueta += "\nS" + contor + " -> S" + (++conts) + "[label=\"e\"]";
        if(nohijos((node) nodo.left)){
            etiqueta += "\nS" + conts + " -> S" + (++conts) + GraficaANDF((node)nodo.left);
        }
        else{
            etiqueta += GraficaANDF((node) nodo.left);
        }
        var contfinal = conts;
        etiqueta += "\nS" + contor + " -> S" + (++conts) + "[label=\"e\"]";
        if(nohijos((node) nodo.right)){
            etiqueta += "\nS" + conts + " -> S" + (++conts) + GraficaANDF((node)nodo.right);
        }
        else{
            etiqueta += GraficaANDF((node) nodo.right);
        }
        etiqueta += "\nS" + conts + " -> S" + (++conts) + "[label=\"e\"]";

        etiqueta += "\nS" + contfinal + " -> S" + conts + "[label=\"e\"]";
    }
    else if(arraylex[0] == '*'){
        var contcom = conts;
        etiqueta += "\nS" + contcom + " -> S" + (++conts) + "[label=\"e\"]";
        var contini = conts;
        if(nohijos((node) nodo.left)){
            etiqueta += "\nS" + conts + " -> S" + (++conts) + GraficaANDF((node)nodo.left);
        }
        else{
            etiqueta += GraficaANDF((node) nodo.left);
        }
        etiqueta += "\nS" + conts + " -> S" + contini + "[label=\"e\"]";
        etiqueta += "\nS" + conts + " -> S" + (++conts) + "[label=\"e\"]";
        etiqueta += "\nS" + contcom + " -> S" + conts + "[label=\"e\"]";
    }
    else if(arraylex[0] == '+'){
        etiqueta += "\nS" + conts + " -> S" + (++conts) + "[label=\"e\"]";
        var contini = conts;
        if(nohijos((node)nodo.left)){
            etiqueta += "\nS" + conts + " -> S" + (++conts) + GraficaANDF((node)nodo.left);
        }
        else{
            etiqueta += GraficaANDF((node) nodo.left);
        }
        etiqueta += "\nS" + conts + " -> S" + contini + "[label=\"e\"]";
        etiqueta += "\nS" + conts + " -> S" + (++conts) + "[label=\"e\"]";
    }
    else if(arraylex[0] == '?'){
        var contcou = conts;
        etiqueta += "\nS" + contcou + " -> S" + (++conts) + "[label=\"e\"]";
        if(nohijos((node)nodo.left)){
            etiqueta += "\nS" + conts + " -> S" + (++conts) + GraficaANDF((node) nodo.left);
        }
        else{
            etiqueta += GraficaANDF((node) nodo.left);
        }
        etiqueta += "\nS" + conts + " -> S" + (++conts) + "[label=\"e\"]";
        etiqueta += "\nS" + contcou + " -> S"+ conts + "[label=\"e\"]";
    }
```

- CONVERTIR GRAPHVIZ: se crear un archivo dot en la carpeta correspondiente, y se realiza la conversión a través de comandos de consola. Ejemplo de como se grafica el árbol, se repite el mismo proceso en cada gráfica generada.

```java
private void crearGraficaArbol(String titulo, String grafica){
    FileWriter fichero = null;
    PrintWriter pw = null;
    try {
        fichero = new FileWriter("src/main/java/Reportes/ARBOLES_202111478/" + titulo + ".dot");
        pw = new PrintWriter(fichero);
        pw.println(grafica);
    } catch (Exception e) {
        System.out.println("error, no se realizo el archivo"+e);
    } finally {
        try {
            if (null != fichero) {
                fichero.close();
                System.out.println(x:"METODO DEL ARBOL GENERADO CORRECTAMENTE");
            }
        } catch (Exception e2) {
            e2.printStackTrace();
        }
    }
    //para compilar el archivo dot y obtener la imagen
    try {
        //"C:\\Program Files\\Graphviz\\bin\\dot.exe"
        //dirección doonde se ecnuentra el compilador de graphviz
        String dotPath = "C:\\Program Files\\Graphviz\\bin\\dot.exe";
        //dirección del archivo dot
        String fileInputPath = "src/main/java/Reportes/ARBOLES_202111478/" + titulo + ".dot";
        //dirección donde se creara la magen
        String fileOutputPath = "src/main/java/Reportes/ARBOLES_202111478/" + titulo + ".jpg";
        //tipo de conversón
        String tParam = "-Tjpg";
        String tOParam = "-o";

        String[] cmd = new String[5];
        cmd[0] = dotPath;
        cmd[1] = tParam;
        cmd[2] = fileInputPath;
        cmd[3] = tOParam;
        cmd[4] = fileOutputPath;

        Runtime rt = Runtime.getRuntime();

        rt.exec(cmd);

    } catch (Exception ex) {
        ex.printStackTrace();
    } finally {
    }
}
```