2023

EXREGAN USAC

MANUAL TÉCNICO

ALDO SAÚL VÁSQUEZ MOREIRA

Para el desarrollo de este software se utilizó:

- **Lenguaje de programación: JAVA JDK 8**
- **IDE: IntelliJ IDEA**
- **Paradigma de programación: Programación Orientada a Objetos**
- **Interfaz gráfica: JAVA Swing**
- **Generador de Lexer: JFlex**
- **Generador de Parser: CUP**

Sin embargo, en este manual nos centraremos en las clases principales así como en las herramientas utilizadas para generar el compilador, dado que las demás herramientas son más conocidas.

Inicialmente, para el análizador léxico tenemos el archivo "AnalizadorLexico.jflex" en el cual se definieron los tokens que se envían al analizador sintáctico o parser.

Cuenta con la siguiente configuración, la cual indica el nombre del archivo, que tomará en cuenta las líneas y columnas y que no será sensible a las mayúsculas y minúsculas, ya que así lo requiere el lenguaje definido.

```
%class scanner
%unicode
%cup
%line
%column
%public
%ignorecase
```

Luego en el área que permite código JAVA se instanció un ArrayList denominado "erroresLexicos" el cual será de utilidad para el almacenamiento de los errores léxicos encontrados en el archivo.

```
%{
    public static ArrayList<Excepcion> erroresLexicos = new ArrayList<Excepcion>();
%}
```

Seguidamente, se encuentran cada una de las expresiones regulares necesarias en el lenguaje, así como el resto de tokens:

```
/*Expresiones regulares*/
ESCAPADOS = "\\n"|"\\\""|"\\\'"
NO_ESCAPADOS = [^\'\"]
CARACTER = (\" {NO_ESCAPADOS} \") | {ESCAPADOS}
CADENA = \" ([^\"]|"\\\"")+ \"
ENTER = \r|\n|\r\n
ESPACIO = [\ \r\t\f\t]
NUMERO = [0-9]
NUMERO_DECIMAL = [0-9]+("."[  |0-9]+)?
LETRA =[a-zA-ZÑñ]
IDENTIFICADOR = {LETRA}({LETRA}|{NUMERO}|"_")*
CARACTER_ESPECIAL = [!-\/:-@\[-`{-~]
COMENTARIO_LINEAL = "//" .*
COMENTARIO_MULTILINEAL ="<!"{ENTER}*({LETRA}|{ESPACIO}|{NUMERO}|{NUMERO_DECIMAL}|{ENTER}|{CARACTER_ESPECIAL})+"!>"{
COMENTARIO = ({COMENTARIO_LINEAL}|{COMENTARIO_MULTILINEAL})

/*simbolos*/
MENOR_QUE = "<"
MAYOR_QUE = ">"
ADMIRACION = "!"
DIAGONAL = "/"
```

```
/*simbolos*/
MENOR_QUE = "<"
MAYOR_QUE = ">"
ADMIRACION = "!"
DIAGONAL = "/"
LLAVE_IZQUIERDA = "{"
LLAVE_DERECHA = "}"
PORCENTAJE = "%"
PUNTO_COMA = ";"
DOS_PUNTOS = ":"
PUNTO = "."
MENOS = "-"
COMA = ","
INTERROGACION = "?"
SUMA = "+"
ASTERISCO = "*"
TILDE = "~"
BARRA_VERTICAL = "|"
COMILLAS = \"

/*palabras reservadas*/
RESERVADA_CONJUNTO = "Conj"

/*caracteres especiales*/

SALTO_LINEA = \\n
COMILLA_SIMPLE = \\'
COMILLA_DOBLE = \\\"

%%
```

Luego de esto se tiene el área que devuelve los tokens y los retorna al analizador sintáctico:

```
/*Simbolos*/
{MENOR_QUE} {System.out.println("Reconocio : "+yytext()+" MENOR QUE"); return new Symbol(sym.MENOR_QUE, yyline, yyc
{MAYOR_QUE} {System.out.println("Reconocio : "+yytext()+" MAYOR QUE");return new Symbol(sym.MAYOR_QUE, yyline, yyco
{ADMIRACION} {System.out.println("Reconocio : "+yytext()+" ADMIRACION");return new Symbol(sym.ADMIRACION, yyline, y
{DIAGONAL} {System.out.println("Reconocio : "+yytext()+" DIAGONAL");return new Symbol(sym.DIAGONAL, yyline, yycolum
{LLAVE_IZQUIERDA} {System.out.println("Reconocio : "+yytext()+" LLAVE IZQUIERDA"); return new Symbol(sym.LLAVE_IZQU
{LLAVE_DERECHA} {System.out.println("Reconocio : "+yytext()+" LLAVE DERECHA");return new Symbol(sym.LLAVE_DERECHA,
{PORCENTAJE} {System.out.println("Reconocio : "+yytext()+" PORCENTAJE");return new Symbol(sym.PORCENTAJE, yyline, y
{PUNTO_COMA} {System.out.println("Reconocio : "+yytext()+" PUNTO Y COMA");return new Symbol(sym.PUNTO_COMA, yyline,
{DOS_PUNTOS} {System.out.println("Reconocio : "+yytext()+" DOS PUNTOS");return new Symbol(sym.DOS_PUNTOS, yyline, y
{PUNTO} {System.out.println("Reconocio : "+yytext()+" PUNTO");return new Symbol(sym.PUNTO, yyline, yycolumn, yytext
{MENOS} {System.out.println("Reconocio : "+yytext()+" MENOS");return new Symbol(sym.MENOS, yyline, yycolumn, yytext
{COMA} {System.out.println("Reconocio : "+yytext()+" COMA");return new Symbol(sym.COMA, yyline, yycolumn, yytext())
{INTERROGACION} {System.out.println("Reconocio : "+yytext()+" INTERROGACION");return new Symbol(sym.INTERROGACION,
{SUMA} {System.out.println("Reconocio : "+yytext()+" SUMA");return new Symbol(sym.SUMA, yyline, yycolumn, yytext())
{ASTERISCO} {System.out.println("Reconocio : "+yytext()+" ASTERISCO");return new Symbol(sym.ASTERISCO, yyline, yyco
{TILDE} {System.out.println("Reconocio : "+yytext()+" TILDE");return new Symbol(sym.TILDE, yyline, yycolumn, yytext
{BARRA_VERTICAL} {System.out.println("Reconocio : "+yytext()+" BARRA VERTICAL");return new Symbol(sym.BARRA_VERTICA
{COMILLAS} {System.out.println("Reconocio : "+yytext()+" COMILLAS");return new Symbol(sym.COMILLAS, yyline, yycolum
{RESERVADA_CONJUNTO} {System.out.println("Reconocio : "+yytext()+" PR CONJUNTO");return new Symbol(sym.RESERVADA_CO

/*caracteres especiales*/
{SALTO_LINEA} {System.out.println("Reconocio : "+yytext()+" SALTO LINEA");return new Symbol(sym.SALTO_LINEA, yyline
{COMILLA_SIMPLE} {System.out.println("Reconocio : "+yytext()+" COMILLA SIMPLE");return new Symbol(sym.COMILLA_SIMPL
{COMILLA_DOBLE} {System.out.println("Reconocio : "+yytext()+" COMILLA DOBLE");return new Symbol(sym.COMILLA_DOBLE,

/*expresiones regulares*/
{NUMERO} {System.out.println("Reconocio : "+yytext()+" NUMERO");return new Symbol(sym.NUMERO, yyline, yycolumn, yyt
{NUMERO_DECIMAL} {System.out.println("Reconocio : "+yytext()+" NUMERO DECIMAL");return new Symbol(sym.NUMERO_DECIMA
```

La parte del analizador sintáctico se explicará de mejor manera en el archivo de gramática.

Las clases principales para el desarrollo del sistema se denominan:

- **Arbol: Esta clase es de mucha importancia ya que instancia el resto de clases que tienen que ver con el árbol. Por lo cual, obtiene los datos del árbol tal como, los siguientes, los estados, si los nodos son anulables o no, etc. Así como la generación del grafo y diversas funciones más.**

```
2 usages  Aldo Vasquez
public void insertar(String valor, String tipo) {
    if (raiz == null) {
        this.raiz = new NodoArbol(valor, tipo, contador);
    } else {
        insertarNodo(valor, tipo, raiz);
    }
    contador++;
}

1 usage  Aldo Vasquez
private void insertarNodo(String valor, String tipo, NodoArbol nodo) {
    if (nodo == this.raiz && this.raiz.getHijoIzquierdo() == null) {
        insertarHijoIzquierda(valor, tipo, nodo);
    } //Verificando el tipo de nodo e insertarlo
    else { //Tipo de nodo que esta actualmente
        insertarEstado = false;
        insertarEnPreOrden(valor, tipo, nodo);
    }
}

3 usages  Aldo Vasquez
private void insertarHijoIzquierda(String valor, String tipo, NodoArbol nodo) {
    NodoArbol nodoInsert = new NodoArbol(valor, tipo, contador++);
    if (tipo.equals("valor")) {
        nodoInsert.setId(hojas++);
    }
    nodo.setHijoIzquierdo(nodoInsert);
}

2 usages  Aldo Vasquez
private void insertarHijoDerecha(String valor, String tipo, NodoArbol nodo) {
    NodoArbol nodoInsert = new NodoArbol(valor, tipo, contador++);
    if (tipo.equals("valor")) {
        nodoInsert.setId(hojas++);
    }
    nodo.setHijoDerecho(nodoInsert);
}
```

```java
public void obtenerGraficaTree(String numero) throws IOException {
    if (!arbolVacio()) {
        //creacion de la carpeta que contiene los arboles
        String ruta = new File( pathname: ".").getAbsolutePath();
        String ruta_absoluta = ruta;
        crear_carpeta( nombre: "ARBOLES_202109754");
        ruta += File.separator + "ARBOLES_202109754" + File.separator + "Arbol" + numero + ".dot";
        File archivo = new File(ruta);
        if (!archivo.exists()) {
            archivo.createNewFile();
        }
        //Escribimos dentro del archivo .dot
        try (PrintWriter write = new PrintWriter(ruta, csn: "UTF-8")) {
            write.println("digraph Arbol{");
            write.println("node [shape=record, height=.1];");
            write.close();
        } catch (FileNotFoundException | UnsupportedEncodingException e) {
            JOptionPane.showMessageDialog( parentComponent: null, message: "Error al crear el reporte de archivos." + e
        }
```

- **NodoArbol: Declara las propiedades de un nodo de un autómata finito determinista.**

- **ExpresionRegular: Esta contiene el funcionamiento principal ya que implementa la clase del árbol, así como las que la clase Arbol instancia. Esto dado que al analizar la expresión regular crea el AFD. Asimismo, obtiene cada uno de los conjuntos declarados, identifica la notación y obtiene los elementos pertenecientes a dicho conjunto. Al obtener los elementos, verifica las expresiones regulares e implementa las funciones de manipulación de información que se encuentran en la clase ManipuladorData para validación de las cadenas exitosamente.**

  Algunos de los métodos principales de esta clase son:

```java
1 usage  Aldo Vasquez
public void crearTabST() {
    ArrayList<Estados> tabla_Estados = new ArrayList<>();
    estados = 0;

    //Obtener el Estado S0 (Primeros del nodo raiz)
    tabla_Estados.add(new Estados( id: "S0", arbolExpresion.getRaiz().getPrimeros(), estadoAceptacion(arbolExpresion

    int repetirFor = 1;

    //Insertar los encabezados de estados
    while (repetirFor != 0) {
        repetirFor--;
        //Desglozar todos los estados
        for (int i = 0; i < tabla_Estados.size(); i++) {
            Estados actualEstado = tabla_Estados.get(i);

            String numConjunto = actualEstado.getNumContenidos();
            String[] numerosID = numConjunto.split( regex: ",");
```

```java
2 usages  Aldo Vasquez *
private boolean estadoAceptacion(String numConjunto) {
    int ultimoIdTablaSiguientes = this.tablaSiguientes.get(this.tablaSiguientes.size() - 1).getId();
    return Arrays.stream(numConjunto.split( regex: ",")) Stream<String>
            .map(Integer::parseInt) Stream<Integer>
            .anyMatch(id -> id == ultimoIdTablaSiguientes);
}

1 usage  Aldo Vasquez *
public int obtenerPosicionEstadoAct(String idEstado) {
    for (int i = 0; i < tablaEstados.size(); i++) {
        if (tablaEstados.get(i).getId().equals(idEstado)) {
            return i;
        }
    }
    return -1;
}
```

```java
1 usage  Aldo Vasquez
public void crearGraficaTablaEstados() throws IOException {
if (!tablaSiguientes.isEmpty()) {
    //creacion de la carpeta que contiene los arboles
    String ruta = new File( pathname: ".").getAbsolutePath();
    String ruta_absoluta = ruta;
    crear_carpeta( nombre: "TRANSICIONES_202109754");
    ruta += File.separator + "TRANSICIONES_202109754" + File.separator + "TablaEstados" + getNumero() + ".dot"
    File archivo = new File(ruta);
    if (!archivo.exists()) {
        archivo.createNewFile();
    }

    int countSimbolos = tablaSiguientes.size();
    //Escribimos dentro del archivo .dot
    try (PrintWriter write = new PrintWriter(ruta,  csn: "UTF-8")) {
        write.println("digraph TablaEstados{");
        write.println("tbl [");
        write.println("shape = plaintext");
        write.println("label = <");
```

```
1 usage  Aldo Vasquez                                                    22 1 213
public void crearGraficoTablaSiguientes() throws IOException {
    if (!tablaSiguientes.isEmpty()) {
        //creacion de la carpeta que contiene la tabla de siguientes
        String ruta = new File( pathname: ".").getAbsolutePath();
        String ruta_absoluta = ruta;
        crear_carpeta( nombre: "SIGUIENTES_202109754");
        ruta += File.separator + "SIGUIENTES_202109754" + File.separator + "TablaSiguientes" + getNumero() + ".d
        File archivo = new File(ruta);
        if (!archivo.exists()) {
            archivo.createNewFile();
        }
        //Escribimos dentro del archivo .dot
        try (PrintWriter write = new PrintWriter(ruta, csn: "UTF-8")) {
            write.println("digraph TablaSiguientes{");
            write.println("tbl [");
            write.println("shape = plaintext");
            write.println("label = <");
            write.println("<table border='0' cellborder='1' color='black' cellspacing='0'>");
            write.println("<tr><td>Valor</td><td>Id</td><td>Siguientes</td></tr>");
            //Codigo HTML Tabla
            Iterator<Siguientes> iteradorSiguientes = tablaSiguientes.iterator();
```

```
1 usage  Aldo Vasquez
public void crearGraficoAutomataFinito() throws IOException {
    if (!tablaEstados.isEmpty()) {
        String ruta = new File( pathname: ".").getAbsolutePath();
        String ruta_absoluta = ruta;
        crear_carpeta( nombre: "AFD_202109754");
        ruta += File.separator + "AFD_202109754"+ File.separator + "AFD" + getNumero() + ".dot";
        File archivo = new File(ruta);
        if (!archivo.exists()) {
            archivo.createNewFile();
        }
        //Escribimos dentro del archivo .dot
        try (PrintWriter write = new PrintWriter(ruta, csn: "UTF-8")) {
            write.println("digraph AFD{");
            write.println("rankdir=LR;");
            write.println("size=\"13\"");

            //Crear nodo de aceptacion
            for (int i = 0; i <= this.tablaEstados.size() - 1; i++) {
                if (this.tablaEstados.get(i).isEstadoAceptacion()) {
                    write.println(this.tablaEstados.get(i).getId() + "[peripheries = 2, shape=circle];");
                }
```

- **Conjunto: Se encarga de declarar las propiedades y funciones principales de un conjunto.**
- **Estados: Obtiene los estados para el AFD.**
- **Siguientes: Obtiene los siguientes de un nodo.**

- **ManipuladorData: Implementa todas las funciones para establecer el funcionamiento básico del programa. Por lo cual, dicha clase se instancia en la ventana principal.**

- **Cadena: Básicamente se encarga de establecer las propiedades de importancia para la validación de las cadenas.**

Se tienen otras clases, sin embargo, su funcionamiento es secundario ya que únicamente cumplen funciones complementarias o de menor importancia en el sistema.

- **Excepcion: Esta clase define las propiedades de un error del lenguaje, ya sea léxico o sintáctico.**

- **GeneradorReporteErrores: Obtiene la lista de errores léxicos y sintácticos, luego genera el reporte web.**