Manual Técnico

Alberto Gabriel Reyes Ning
201612174

Introducción:

En el siguiente manual se describe a detalle las clases. Así como lo que contiene cada una y porque fue necesaria la creación de estas, para que se usó cada método, variable y lista esto con el fin de saber que pasaba al ejecutar el programa, explicando de forma breve la lógica aplicada en este proyecto. Cabe mencionar que en la parte del análisis léxico y sintáctico se utilizó las librerías jflex y Jcup respectivamente.

Packages:

- **Analizadores**
- **Errores**
- **Estructuras**
- **Instruction**
- **Interfaz**
- **OLC1-PROYECTO1-201612174**

Clases:

Analizadores:

- **Generardor**
- **Parser.cup**
- **Parser.java**
- **Scanner**
- **Lex.jflex**
- **Sym**

Errores:

- **Error_**
- **LinkendListErrror**

Estructuras:

- **Árbol**
- **Conjunto**
- **Estados**
- **Lexema**
- **Lista**
- **Nodo**
- **NodoArbol**
- **Nodo**
- **NodoPila**
- **Pila**
- **REPORTES**
- **Siguientes**

- **Transiciones**

Interfaz:

- **Principal**

OLC1-PROYECTO1-201612174:

- **OLC1-PROYECTO1-201612174**
- **CargaMasiva**

Clase Main:

Esta clase contiene su propio método main, con el cual se inicializan los analizadores léxicos y sintácticos. También se guardan los archivos que se generan posteriormente en el mismo directorio del paquete.

```
/**
 *
 * @author Alberto Gabriel Reyes Ning, 201612174
 */
public class Main {

    public static void main(String[] args) {
        try {
            // Analisis lexico y poder utilizar Jflex
            String ruta = "src/Analizador/";
            String openJflex[] = {ruta + "lexico.jflex", "-d", ruta};
            jflex.Main.generate(argv: openJflex);
            //Para utilizar el analizado sintatico Jcup
            String openCup[] = {"-destdir", ruta, "-parser", "Parser", ruta + "Parser.cup"};
            java_cup.Main.main(argv: openCup);

        } catch (Exception e) {
            System.err.println(x: "Error al acceder a lexico.jflex y parser.cup");
        }
        //System.out.println("");
    }
}
```

Parser.cup

Es un archivo EmptyFile, en este archivo se escribe la gramática de tipo 3. Se definen las expresiones regulares y se retornan lo tokens que se envían al analizador sintáctico en Jcup. También se recupera de los errores léxicos y se guardan en un ArrayList.

```
//____________Expresiones regulares_____________
IGNORAR=[ \r\t\n]+
LETTERS=[a-zA-Z]
DIGIT=[0-9]
PC=";"
CP="+" //cerradura positiva
CK="*"
OR="|"
I="?"
P="."
V="~"
ESPECIALES=(\\n|\\\'|\\\")
DPORCENTAJE=%%
COM_MTL=\<\!\<*([^<!]|[^!]"<"|"!"[^>])*\!*\!\>
LLAVEIZ=\{
LLAVED=\}
ID={LETTERS}({LETTERS}|{DIGIT}|"_")*
DOS_P=":"
ASIGNACION="-"[ ]*">"
SIMBOLOS_CONJ=([ -\/]|[\:-\@]|[\[-\´]|[\{-\}])
SIMBOLOS={LETTERS}|{DIGIT}|{SIMBOLOS_CONJ}|{ESPECIALES}
LISTA_SIMBOLOS={SIMBOLOS}([ ]*","[ ]*{SIMBOLOS})*
CADENA=\"([^\"]|{ESPECIALES})*\"
CONJUNTO= ({LLAVEIZ}{ID}{LLAVED})
COMENT=("/""/"([^\n]*))
CONJ="CONJ"
```

Parser.cup

Es un archivo EmptyFile, en este archivo se escribe la gramática de tipo 2. Se van colcando las producciones en el orden correspondiente. En la parte del la Expresión regular se van agregando los nodos al árbol y de esta forma crear el árbol, esto para sus uso posterior. También se recupera de los errores sintácticos y se guardan en un ArrayList.

```
inicio1::=
            TK_LLAVEIZ inicio  TK_LLAVED inicio1
            |TK_LLAVEIZ inicio  TK_LLAVED
            |comentarios2 inicio1
            |comentarios2
;
inicio::=
        instructions  inicio
       |  instructions
;

comentarios2 ::=
             TK_COMENT:c {: System.out.println("Comentario: "+c); :}
             |TK_COM_MTL:c {: System.out.println("Comentario: "+c); :}
;
instructions::=
       TK_COMENT:c {: System.out.println("Comentario: "+c); :}
       |conjuntos TK_PC
       |TK_DPORCENTAJE
       |expresion_regular TK_PC
       |validar_cadena TK_PC
       |TK_COM_MTL:c {: System.out.println("Comentario: "+c); :}
;

conjuntos::=
      TK_CONJ TK_DOS_P  TK_ID TK_ASIGNACION TK_SIMBOLOS:a  TK_V TK_SIMBOLOS:c {: System.out.println(a+"~"+c); :}
      | TK_CONJ TK_DOS_P  TK_ID TK_ASIGNACION TK_LETTERS:a TK_V TK_LETTERS:c {: System.out.println(a+"~"+c); :}
      | TK_CONJ TK_DOS_P  TK_ID TK_ASIGNACION TK_DIGIT:a TK_V TK_DIGIT:c {: System.out.println(a+"~"+c); :}
      |   TK_CONJ TK_DOS_P  TK_ID TK_ASIGNACION TK_LISTA_SIMBOLOS:c {: System.out.println("conjunto: "+c); :}
;
```

Árbol

En esta clase se hacen todos los métodos relacionados con el árbol binario, ya que se utilizo un árbol binario para almacenar los datos de la expresión regular.

```java
public class Arbol {

    Nodo root;
    int cont;
    int contEst = 1;
    String name;
    //________listas__________
    ArrayList<Temp_siguietes> lista_siguienes = new ArrayList<>();
    ArrayList<Siguientes> lista_tabla_sig = new ArrayList<>();
    ArrayList<String[]> lista_numeros = new ArrayList<>();
    ArrayList<Transiciones> transiciones = new ArrayList<>();

    //Esta lista va a almacenar los estados nuevos que se vayan generando
    ArrayList<Estado> estados_nuevos = new ArrayList<>();
    // Esta lista almacenara lo estados que ya existen
    ArrayList<Estado> estados_existentes = new ArrayList<>();
    //
    ArrayList<String> listaSiguientes = new ArrayList<>();
    ArrayList<String> terminales = new ArrayList<>();
    ArrayList<String> terminalesNoRep = new ArrayList<>();
    ArrayList<Estado> lista_estados = new ArrayList<>();
    ArrayList<String[]> temporal = new ArrayList<>();
    // ArrayList
    ArrayList<String[]> transiciones2 = new ArrayList<>();
    ArrayList<String> estadoAceptacion = new ArrayList<>();
    private String nombreEstado = "";

    public Arbol() {
        root = null;
    }

    public Arbol(Nodo root) {
        this.root = root;
        this.name = "";
        //contador para enumerar las hojas
        preorder(node: root);
        this.cont = 0;

    }
```

Para asignar la numeración a cada hoja se utilizó el método preorden del árbol binario para recorrerlo.

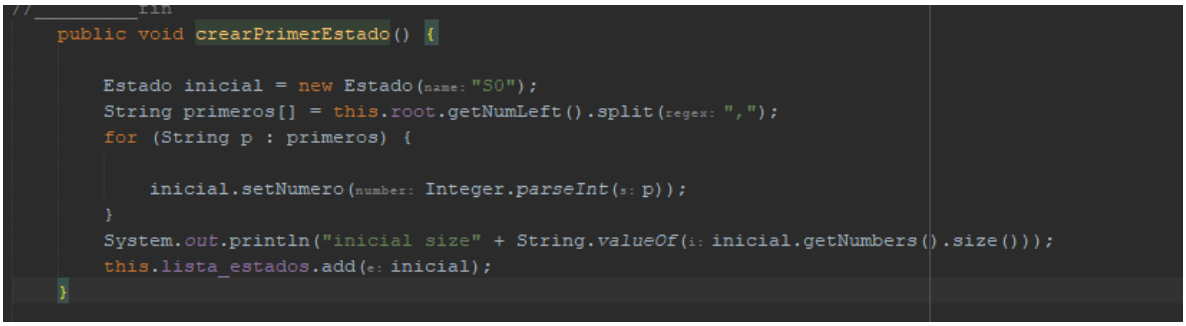
```
public void preorder(Nodo node) {

    if (node != null) {

        if (!node.getObject().equals(anObject: ".") && !node.getObject().equals(anObject: "\\|") && !node.getObject().equals(anObject: "*") && !node.getObject().
            //aumentar el contador
            this.cont++;
            //asignar el numero del lado izquierdo
            node.setNumLeft(numLeft:String.valueOf(i: this.cont));
            //asignar el numero del lado drecho
            node.setNumRight(numRight:String.valueOf(i: this.cont));
            String numeros[] = new String[2];
            numeros[0] = String.valueOf(i: this.cont);
            numeros[1] = node.getObject();
            lista_numeros.add(e: numeros);
        }
        preorder(node: node.getNode_Left());
        preorder(node: node.getNode_Right());
    }
}
//_______enumerar y colocar la anulabilidad__________
```

Método crearPrimerEstado()

Este método se utilizo para poder obtener le primer estado de cada árbol.



```
//_________fin
    public void crearPrimerEstado() {

        Estado inicial = new Estado(name: "S0");
        String primeros[] = this.root.getNumLeft().split(regex: ",");
        for (String p : primeros) {

            inicial.setNumero(number: Integer.parseInt(s: p));
        }
        System.out.println("inicial size" + String.valueOf(i: inicial.getNumbers().size()));
        this.lista_estados.add(e: inicial);
    }
```

Estado

Esta clase se utilizo para crear objetos de tipo estado. Se almacenaban los estados y transiciones con que se realizaban entre ellos y con el símbolo correspondiente.

```java
public class Estado {

    private String name;
    private ArrayList<Integer> datos=new ArrayList<>();

    public Estado(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }


    public void setNumero(int number){
        if (!this.datos.isEmpty()) {
            if (!existente(number)) {
                this.datos.add(e: number);
                Collections.sort(list: this.datos);
            }
        }else{
            this.datos.add(e: number);
        }
    }
```

Transiciones

Clase para crear objetos de tipo Transiciones, los cuales van a almacenar las transiciones que realizan los estados para llegar a otro junto con su lexema.

```java
public class Transiciones {
    private Estado actual;
    private String lexema;
    private Estado next;

    public Transiciones(Estado actual, String lexema, Estado next) {
        this.actual = actual;
        this.lexema = lexema;
        this.next = next;
    }

    public Estado getActual() {
        return actual;
    }

    public void setActual(Estado actual) {
        this.actual = actual;
    }

    public String getLexema() {
        return lexema;
    }

    public void setLexema(String lexema) {
        this.lexema = lexema;
    }

    public Estado getNext() {
        return next;
    }

    public void setNext(Estado next) {
        this.next = next;
    }
}
```

Principal

En esta clase se realizan los métodos necesarios para poder mostrar y darle funcionalidad a la ventana, la cual sirve para que el usuario pueda interactuar con el programa.

```java
public class VentanaPrincipal extends javax.swing.JFrame {

    JFileChooser jFile = new JFileChooser();
    File file;
    int contArbol = 1;
    int contSig = 1;
    int contTabla = 1;
    int cont_afd = 1;
    String ruta = "";
    int contRepo = 0;
    // String cadenaAnilar = "";

    public VentanaPrincipal() {
        initComponents();
    }
//                          metodos referentes al archivo
```

OLC1-PROYECTO1-201612174

Contiene el método principal del programa.

```
public class OLC1_PROYECTO1_201612174 {

    /**
     * @param args the command line arguments
     */
    public static ArrayList<Arbol> arboles= new ArrayList<>();
    public static void main(String[] args) {
        // TODO code application logic here

        VentanaPrincipal ven=new VentanaPrincipal();
        ven.setVisible(b: true);


    }

   public static String OpenArchivo(File file,String ruta) {
        String direccion=ruta;
        String cadena="";
        String cadena2="";
        try {
             file=new File(pathname: direccion);
            BufferedReader lectura= new BufferedReader(new FileReader(file));
            while((cadena=lectura.readLine())!=null){
                cadena2+=cadena+"\n";
            }
            lectura.close();
        } catch (Exception e) {
            System.err.println(x: "No se pudo abrir el archivo");
        }
        return cadena2;
    }

}
```

Conclusión

El programa se logró estructurar y acoplar a los requerimientos que se pedían por parte del cliente. Utilizando el paradigma de programación orientada a objetos se obtuvo una mejor organización del código que hacía más eficaz con una mejor organización a la hora de llamar clases y métodos. Debido a que se tiene un mejor orden y percepción de lo que se busca realizar. El uso de las librerías Jflex y Jcup son de gran utilidad para el manejo de la parte léxica y sintáctica, ya que estas facilitan el análisis de distintos lenguajes.