UNIVERSIDAD SAN CARLOS DE GUATEMALA
CENTRO UNIVERSITARIO DE OCCIDENTE
DIVISION DE CIENCIAS DE LA INGENIERIA
INGENIERIA EN CIENCIAS Y SISTEMAS
LABORATORIO DE COMPILADORES 1

MANUAL DE TECNICO

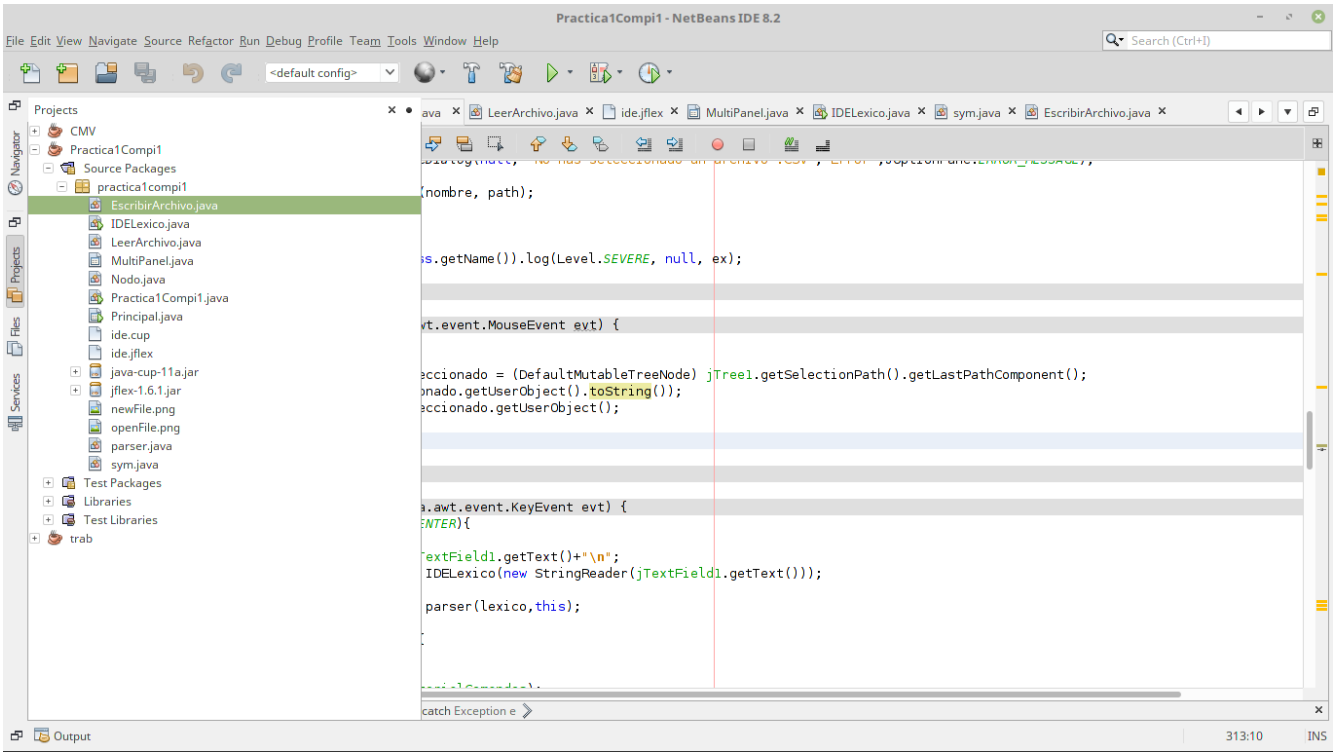
LUIS FERNANDO RODRIGUEZ LIMA
201730879

CARACTERISITICAS DEL PROYECTO:

IDE: NETBEANS 8.2
JAVA: JAVA 8

JFLEX: JFLEX 1.6.1
CUP: JAVA CUP 11 a

ESCTURCTURA DEL PROYECTO:



ANALIZADOR LEXICO DEL GENERAL

```
package practica1compi1;
import java_cup.runtime.Symbol;
%%//Separador de area

%class IDELexico
%cup
%cupdebug
%line
%column

/*Identificadores*/
Letra = [a-zA-Z]
```

```
Signo = [-_@+*#]
Numero = [0123456789]
%{
   private Symbol symbol(int type){
      return new Symbol(type, yyline, yycolumn);
   }
   private Symbol symbol(int type,Object value){
       return new Symbol(type, yyline, yycolumn, value);
   }
%}
%%//Separador de area.

<YYINITIAL> {
          ("<")       { /*System.out.println("Token: MENOR_QUE con lexema: "+yytext());*/  return symbol(sym.MENOR_QUE);}
          (">")       { /*System.out.println("Token: MAYOR_QUE con lexema: "+yytext());*/ return symbol(sym.MAYOR_QUE);}
        (PROYECTO)  { /*System.out.println("Token: PROYECTO con lexema: "+yytext());*/ return symbol(sym.PROYECTO);}
          (ARCHIVO)    { /*System.out.println("Token: ARCHIVO con lexema: "+yytext());*/ return symbol(sym.ARCHIVO);}
          (CARPETA)    { /*System.out.println("Token: CARPETA con lexema: "+yytext());*/ return symbol(sym.CARPETA);}
          (ubicacion) { /*System.out.println("Token: UBICACION con lexema: "+yytext());*/ return symbol(sym.UBICACION);}
          (nombre)       { /*System.out.println("Token: NOMBRE con lexema: "+yytext());*/ return symbol(sym.NOMBRE);}
        ("<=")      { /*System.out.println("Token: MENOR_IGUAL con lexema: "+yytext());*/  return symbol(sym.MENOR_IGUAL);}
         (">=")      { /*System.out.println("Token: MAYOR_IGUAL con lexema: "+yytext());*/ return symbol(sym.MAYOR_IGUAL);}
          ("<>")       { /*System.out.println("Token: DIFERENTE con lexema: "+yytext());*/ return symbol(sym.DIFERENTE);}
       (SELECCIONAR)  { /*System.out.println("Token: SELECCIONAR con lexema: "+yytext());*/ return symbol(sym.SELECCIONAR);}
          (FILTRAR)    { /*System.out.println("Token: FILTRAR con lexema: "+yytext());*/ return symbol(sym.FILTRAR);}
         (INSERTAR)    { /*System.out.println("Token: INSERTAR con lexema: "+yytext());*/ return symbol(sym.INSERTAR);}
          (ACTUALIZAR) { /*System.out.println("Token: ACTUALIZAR con lexema: "+yytext());*/ return symbol(sym.ACTUALIZAR);}
          (ASIGNAR) { /*System.out.println("Token: ASIGANAR con lexema: "+yytext());*/ return symbol(sym.ASIGNAR);}
            (VALORES) { /*System.out.println("Token: VALORES con lexema: "+yytext());*/ return symbol(sym.VALORES);}
          (ELIMINAR) { /*System.out.println("Token: ELIMINAR con lexema: "+yytext());*/ return symbol(sym.ELIMINAR);}
      (EN) { /*System.out.println("Token: EN con lexema: "+yytext());*/ return symbol(sym.EN);}
```

```
(AND)  { /*System.out.println("Token: AND con lexema: "+yytext());*/ return symbol(sym.AND);}
(OR)  { /*System.out.println("Token: OR con lexema: "+yytext());*/ return symbol(sym.OR);}
(sql>)  { /*System.out.println("Token: SQL con lexema: "+yytext()); return symbol(sym.SQL);*/}
("=")  { /*System.out.println("Token: IGUAL con lexema: "+yytext());*/ return symbol(sym.IGUAL);}
("\"")  { /*System.out.println("Token: COMILLAS con lexema: "+yytext());*/ return symbol(sym.COMILLAS);}
("(")  { /*System.out.println("Token: PABIERTO con lexema: "+yytext());*/ return symbol(sym.PABIERTO);}
(")")  { /*System.out.println("Token: PCERRADO con lexema: "+yytext());*/ return symbol(sym.PCERRADO);}
(",")  { /*System.out.println("Token: COMA con lexema: "+yytext());*/ return symbol(sym.COMA);}
(" ")+  { /*System.out.println("Token: ESPACIO con lexema: "+yytext()); return symbol(sym.ESPACIO);*/}
("\n")+  { /*System.out.println("Token: SALTO_LINEA con lexema: "+yytext()); return symbol(sym.SALTO_LINEA);*/}
("\t")+  {/*Ignorar*/}
(";")  { /*System.out.println("Token: PUNTOYCOMA con lexema: "+yytext());*/ return symbol(sym.PUNTOYCOMA);}
("*")  { /*System.out.println("Token: POR con lexema: "+yytext());*/ return symbol(sym.POR);}
({Letra}|{Signo}|{Numero})({Letra}|{Numero}|{Signo})*  { /*System.out.println("Token: IDNOMBRE con lexema: "+yytext());*/ return symbol(sym.IDNOMBRE, new String(yytext()));}
(".")  { /*System.out.println("Token: PUNTO con lexema: "+yytext());*/ return symbol(sym.PUNTO);}
("/")  { /*System.out.println("Token: DIAGONAL con lexema: "+yytext());*/ return symbol(sym.DIAGONAL);}
(.csv)  { /*System.out.println("Token: EXTENSION_CSV con lexema: "+yytext());*/ return symbol(sym.EXTENSION_CSV);}
.  { System.out.println("ERROR LEXICO con lexema: "+yytext()); return symbol(sym.ERROR, new String(yytext()));}
}
```

ANALIZADOR SINTACTICO GENERAL

```
package practica1compi1;
import java_cup.runtime.Symbol;
import java.util.ArrayList;
import javax.swing.tree.DefaultMutableTreeNode;
import javax.swing.tree.DefaultTreeModel;
import javax.swing.JTree;
import javax.swing.JOptionPane;

parser code{:
```

```
    ArrayList<DefaultMutableTreeNode> nodos = new ArrayList<>();
    ArrayList<DefaultMutableTreeNode> nodosTemporales = new ArrayList<>();

    Principal principal;

    public void mandarMensaje(String mensaje){
       principal.agregarHistorial(mensaje);
    }

       public void llenarArbol(ArrayList<DefaultMutableTreeNode> nodo, DefaultMutableTreeNode
nodoraiz){
       for (int i =0; i<nodo.size();i++){
          Nodo n = (Nodo) nodo.get(i).getUserObject();
           nodoraiz.add(nodo.get(i));
           }
        DefaultTreeModel modelo = new DefaultTreeModel(nodoraiz);
       principal.mostarArbol(modelo);
       principal.setVisible(true);
       principal.desactivarAbrirProyecto();
    }

    public parser(IDELexico idelex, Principal p){
       super(idelex);
       this.principal = p;

    }
    public void setAccion(String tipo,String mensaje,String accion){
       System.out.println(tipo+"      "+mensaje);
    }
    public void syntax_error(Symbol s){
       setAccion("ERROR","Error Sintaxis columna: "+(s.right+1)+" Y Linea: "+(s.left+1),"");
         //JOptionPane.showMessageDialog(null,"Error Sintaxis columna: "+(s.right+1)+" Y Linea: "+
(s.left+1),"Error",JOptionPane.ERROR_MESSAGE);
       principal.agregarHistorial("Error Sintaxis columna: "+(s.right+1)+" Y Linea: "+(s.left+1));
    }

    String var="";

    public void variable(String path){
       var = var + path;
    }
:}

/* Terminales (tokens renviados por el scanner). */
terminal          MENOR_QUE, MAYOR_QUE, PROYECTO, PUNTO, COMA, POR,  ARCHIVO,
CARPETA, UBICACION, NOMBRE, IGUAL, COMILLAS, DIAGONAL, EXTENSION_CSV
,MENOR_IGUAL, MAYOR_IGUAL, DIFERENTE, PUNTOYCOMA, SELECCIONAR, FILTRAR,
INSERTAR, ACTUALIZAR, ASIGNAR, ELIMINAR, VALORES, EN, AND, OR, PABIERTO,
PCERRADO;
```

```
terminal String IDNOMBRE, ERROR, PATH;

/*No terminales*/
non terminal        s;
non terminal        l;
non terminal        sql;
non terminal        seleccionarSql;
non terminal        insertarSql;
non terminal        actualizarSql;
non terminal        eliminarSql;
non terminal        pathSql;
non terminal        valoresSql;
non terminal        columnasSql;
non terminal        condicionesSql;
non terminal        filtroSql;
non terminal        expCuerpo;
non terminal        expArchivo;
non terminal        expCarpeta;
non terminal        path;
non terminal        pathCuerpo;

/*La Gramatica*/
start with l;

l ::= s
     |sql
     ;

sql ::= sql insertarSql
       | insertarSql
       | sql seleccionarSql
       | seleccionarSql
       | sql actualizarSql
       | actualizarSql
       | sql eliminarSql
       | eliminarSql
       ;


seleccionarSql ::= SELECCIONAR columnasSql EN pathSql:a FILTRAR IDNOMBRE IGUAL COMILLAS IDNOMBRE COMILLAS PUNTOYCOMA {: parser.setAccion("Comando seleccionar con columnas","","");
                parser.mandarMensaje("Valores filtrados correctamente");
:}
                    | SELECCIONAR POR EN pathSql:a FILTRAR IDNOMBRE IGUAL COMILLAS IDNOMBRE COMILLAS PUNTOYCOMA {: parser.setAccion("Comando seleccionar simple","","");
                 parser.mandarMensaje("Todos los campos filtrados correctamente");
:}
                ;
```

```
eliminarSql ::= ELIMINAR EN pathSql:a PUNTOYCOMA {: parser.setAccion("eliminar simple","","");
            parser.mandarMensaje("Todos los valores eliminados correctamente");
            :}
                                | ELIMINAR EN pathSql:a FILTRAR filtroSql PUNTOYCOMA {: parser.setAccion("eliminar con filtro","","");
              parser.mandarMensaje("Valores filtrados eliminados correctamente");
:}
            ;

actualizarSql ::= ACTUALIZAR EN pathSql:a ASIGNAR  condicionesSql FILTRAR filtroSql PUNTOYCOMA {: parser.setAccion("Comando actualizar","","");
              parser.mandarMensaje("Valores Actualizados correctamente");
:}
                        | ACTUALIZAR EN pathSql:a ASIGNAR condicionesSql PUNTOYCOMA {: parser.setAccion("Comando actualizar sin filtro","","");
              parser.mandarMensaje("Valores actualizados correctamente");
:}
             ;

insertarSql ::=  INSERTAR EN pathSql:a VALORES PABIERTO valoresSql PCERRADO PUNTOYCOMA {: parser.setAccion("Entro al sql Insertar con exito con path "+(String)a,"","");

        parser.mandarMensaje("Valores Insertados a la ruta: "+a);
:}
              | INSERTAR EN pathSql:a PABIERTO columnasSql PCERRADO VALORES PABIERTO valoresSql PCERRADO PUNTOYCOMA {:

            parser.mandarMensaje("Valores Insertados a la ruta: "+a);
:}

      ;

pathSql ::= IDNOMBRE:i PUNTO IDNOMBRE:n {: parser.setAccion("pathSQL con valor:"+i+"."+n,"",""); String valor = i+"."+n; RESULT = valor;:}
                      | pathSql:q PUNTO IDNOMBRE:s {: parser.setAccion("pathSQL con vloar:"+(String)q+"."+s,"",""); String valor = (String) q + "."+s; RESULT = valor;:}
          ;

valoresSql ::= COMILLAS IDNOMBRE COMILLAS {::}
            | IDNOMBRE {::}
            | valoresSql COMA COMILLAS IDNOMBRE COMILLAS {::}
            | valoresSql COMA IDNOMBRE {::}
            ;

columnasSql ::= IDNOMBRE {::}
             | columnasSql COMA IDNOMBRE {::}
             ;
```

```
filtroSql ::= IDNOMBRE IGUAL IDNOMBRE {::}
            | IDNOMBRE IGUAL COMILLAS IDNOMBRE COMILLAS {::}
            | IDNOMBRE MENOR_QUE IDNOMBRE {::}
            | IDNOMBRE MAYOR_QUE IDNOMBRE {::}
            | IDNOMBRE MENOR_IGUAL IDNOMBRE {::}
            | IDNOMBRE MAYOR_IGUAL IDNOMBRE {::}
            | filtroSql AND IDNOMBRE IGUAL IDNOMBRE {::}
            | filtroSql AND IDNOMBRE IGUAL COMILLAS IDNOMBRE COMILLAS {::}
            | filtroSql AND IDNOMBRE MENOR_QUE IDNOMBRE {::}
            | filtroSql AND IDNOMBRE MAYOR_QUE IDNOMBRE {::}
            | filtroSql AND IDNOMBRE MENOR_IGUAL IDNOMBRE {::}
            | filtroSql AND IDNOMBRE MAYOR_IGUAL IDNOMBRE {::}
            | filtroSql AND IDNOMBRE DIFERENTE IDNOMBRE {::}
            ;

condicionesSql ::= IDNOMBRE:c IGUAL IDNOMBRE:s {::}
                 | IDNOMBRE:c IGUAL COMILLAS IDNOMBRE COMILLAS {::}
                 | condicionesSql COMA IDNOMBRE:c IGUAL IDNOMBRE:s {::}
                   | condicionesSql COMA IDNOMBRE:c IGUAL COMILLAS IDNOMBRE COMILLAS
{::}
                 ;

s ::=    MENOR_QUE PROYECTO NOMBRE IGUAL COMILLAS IDNOMBRE:p COMILLAS
MAYOR_QUE    MENOR_QUE    DIAGONAL    PROYECTO    MAYOR_QUE    {:
parser.setAccion("Comando para Proyecto abierto y cerrado sin cuerpo con nombre: "+p,"","");

        Nodo simple = new Nodo(p,null,false);
       DefaultMutableTreeNode s = new DefaultMutableTreeNode(simple);
       parser.llenarArbol(parser.nodos,s);
                                    JOptionPane.showMessageDialog(null,"Proyecto    analizado    con
exito","Info",JOptionPane.INFORMATION_MESSAGE);
:}
           |MENOR_QUE PROYECTO NOMBRE IGUAL COMILLAS IDNOMBRE:q COMILLAS
MAYOR_QUE expCuerpo:e MENOR_QUE DIAGONAL PROYECTO MAYOR_QUE
{: parser.setAccion("Comando para Proyecto abierto y cerrado","","");

        Nodo simple = new Nodo(q,null,false);
       DefaultMutableTreeNode s = new DefaultMutableTreeNode(simple);

       if ( e instanceof DefaultMutableTreeNode){
       parser.nodos.add((DefaultMutableTreeNode)e);
      }else{
          ArrayList<DefaultMutableTreeNode> ha = (ArrayList) e;
          for (int i=0; i<ha.size(); i++){
              parser.nodos.add(ha.get(i));
          }
      }
       parser.llenarArbol(parser.nodos,s);
```

```
                                    JOptionPane.showMessageDialog(null,"Proyecto analizado con exito","Info",JOptionPane.INFORMATION_MESSAGE);

:}
      | error:e {:parser.syntax_error((Symbol) e); :}
      ;

expCuerpo ::= expCuerpo:l expArchivo:s {: parser.setAccion("Entrada archivo mas salto linea","","");

                ArrayList<DefaultMutableTreeNode> hermanos = new ArrayList<>();

                if(l instanceof DefaultMutableTreeNode){
                  DefaultMutableTreeNode cuerpo = (DefaultMutableTreeNode) l;

                 hermanos.add(cuerpo);


}else{
                ArrayList<DefaultMutableTreeNode> ui = (ArrayList) l;
                for(int i=0; i<ui.size(); i++){
                      hermanos.add(ui.get(i));
                }
}

               if(s instanceof DefaultMutableTreeNode){
                  DefaultMutableTreeNode j = (DefaultMutableTreeNode) s;

                 hermanos.add(j);


}else{
                ArrayList<DefaultMutableTreeNode> usi = (ArrayList) s;
                for(int i=0; i<usi.size(); i++){
                      hermanos.add(usi.get(i));
                }
}



                 RESULT = hermanos;

            :}
                  |expArchivo:a {: parser.setAccion("Solo archivo","","");DefaultMutableTreeNode ss = (DefaultMutableTreeNode) a; RESULT = ss;:}
             |expCuerpo:y expCarpeta:u {: parser.setAccion("carpeta +","",""); DefaultMutableTreeNode lodelCuerpo = (DefaultMutableTreeNode) y; DefaultMutableTreeNode lodelaCarpeta = (DefaultMutableTreeNode) u; ArrayList<DefaultMutableTreeNode> hermanos = new ArrayList<>();hermanos.add(lodelCuerpo); hermanos.add(lodelaCarpeta);RESULT = hermanos; :}
```

```
            |expCarpeta:w {: parser.setAccion("Solo Carpeta","",""); DefaultMutableTreeNode car = (DefaultMutableTreeNode) w; RESULT = car; :}
            ;

expArchivo ::= MENOR_QUE ARCHIVO NOMBRE IGUAL COMILLAS IDNOMBRE:n COMILLAS UBICACION IGUAL COMILLAS path:p COMILLAS DIAGONAL MAYOR_QUE {: parser.setAccion("Comando para archivo con nombre: "+n+" Y path: "+p,"","");
            Nodo nodo = new Nodo(n,(String) p,true);
            DefaultMutableTreeNode archivo = new DefaultMutableTreeNode(nodo);
            RESULT = archivo;
:}
              | error:e {:parser.syntax_error((Symbol) e); :}
              ;

expCarpeta ::=  MENOR_QUE CARPETA NOMBRE IGUAL COMILLAS IDNOMBRE:o COMILLAS MAYOR_QUE MENOR_QUE DIAGONAL CARPETA MAYOR_QUE  {: parser.setAccion("Comando para carpeta vacia","","");

            Nodo carpeta = new Nodo(o,null,false);
            DefaultMutableTreeNode carpetas = new DefaultMutableTreeNode(carpeta);
            carpetas.add(new DefaultMutableTreeNode(""));
            RESULT = carpetas;

:}
              |MENOR_QUE CARPETA NOMBRE IGUAL COMILLAS IDNOMBRE:c COMILLAS MAYOR_QUE expCuerpo:p MENOR_QUE DIAGONAL CARPETA MAYOR_QUE
{: parser.setAccion("Comando para carpeta con mas cosas dentro","","");
            Nodo carper = new Nodo(c,null,false);
            DefaultMutableTreeNode carpeta = new DefaultMutableTreeNode(carper);
            if( p instanceof DefaultMutableTreeNode){
               carpeta.add((DefaultMutableTreeNode)p);
            }else{
               ArrayList<DefaultMutableTreeNode> la = (ArrayList) p;
               for (int i=0; i<la.size(); i++){
                  carpeta.add(la.get(i));
               }
            }

            RESULT = carpeta;

:}
            ;

path ::= pathCuerpo:p EXTENSION_CSV  {: parser.variable(".csv");

  String s = (String) p;
            s = p+".csv";
            RESULT = s;
```

```
:}


;

pathCuerpo ::= DIAGONAL IDNOMBRE:i {: parser.variable("/"+i);

                String valor = "/"+i;
                RESULT = valor;

 :}
               |pathCuerpo:p DIAGONAL IDNOMBRE:i {: parser.variable("/"+i);

               String j = (String) p;
               j = p+"/"+i;
               RESULT = j;
:}
                 ;
```