MANUAL TÉCNICO:

Proyecto Número 1

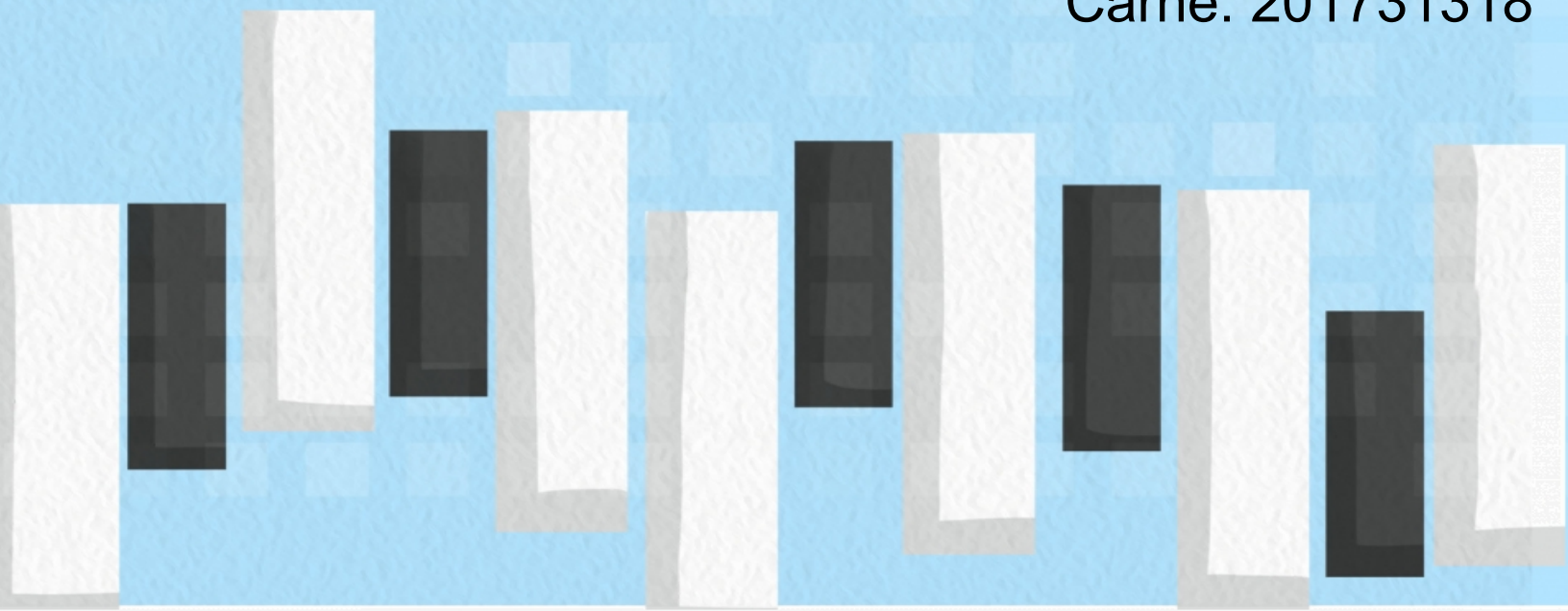
Organizacion Lenguajes y Compiladores 2

Universidad De San Carlos De Guatemala
Centro Universitario De Occidente
División De Ciencias De La Ingeniería

CREADOR REPRODUCTOR DE MÚSICA

Astrid Gabriela Martínez Castillo

Carne: 201731318



INDICE

INTRODUCCION .....**1 2.**

OBJETIVOS **2** 2.1

Objetivos Específicos.....,.....2 2.2

Objetivos Generales.....2 **3.**

ALCANCE.....**2 4.**

REQUERIMIENTOS TECNICOS.....**2** 4.1

Requerimientos Mínimos de Hardware 2 4.2

Requerimientos Mínimos de Software .....2 **5.**

HERRAMIENTAS PARA EL DESARROLLO.....**3 6.**

GRAMATICA ..... **7-30 .**

ANALIZADORES LÉXICOS.....**30-39**

INTRODUCCIÓN

Este documento describe cada una de las herramientas que se utilizaron para el desarrollo de un creador reproductor de música, tanto aspectos relacionados con el hardware como el software esperando sirva de referencia para especificar la creación del proyecto , así como también los algoritmos y diagramas de flujo de cada uno de los programas y del programa en general.

2. OBJETIVOS

2.1. Objetivos Específicos

- Familiarizar al estudiante con la herramienta JFlex
- Familiarizar al estudiante con la herramienta CUP
- Aplicar conocimientos de análisis lexico y sintáctico..

2.2. Objetivos Generales

- Aplicar el concepto de compiladores como una alternativa
- para la resolución de problemas.
- Que el estudiante entienda el funcionamiento de los
- Analizadores de sintaxis ascendentes.
- Combinar la funcionalidad de JFlex y Cup en aplicaciones
- reales.
- Implementar las fases de análisis léxico, sintáctico y
- semántico de un compilador.
- Manejar errores léxicos, sintácticos y semánticos.

3. REQUERIMIENTOS TECNICOS

Software

• Máquina virtual de JAVA (tenerlo instalado).

Hardware

• Una computadora con sus respectivos accesorios (mouse y teclado).

3.1. Requerimientos Mínimos de Software y Hardware

• **Windows:** Windows 10 (8u51 y superiores) Windows 8.x (escritorio) Windows 7 SP1 Windows Vista SP2 Windows Server 2008 R2 SP1 (64 bits) Windows Server 2012 y 2012 R2 (64 bits).

• **Memoria RAM:** 128 MB.

• **Espacio en disco:** 124 MB para JRE; 2 MB para Java

Update. • **Procesador Mínimo:** Pentium 2 a 266 MHz

• **Linux**

Oracle Linux 5.5+1
Oracle Linux 6.x (32 bits), 6.x (64 bits)2
Oracle Linux 7.x (64 bits)2 (8u20 y superiores)
Ubuntu Linux 12.04 LTS, 13.x

Ubuntu Linux 14.x (8u25 y superiores)
Ubuntu Linux 15.04 (8u45 y superiores)
Ubuntu Linux 15.10 (8u65 y superiores)
Exploradores: Firefox

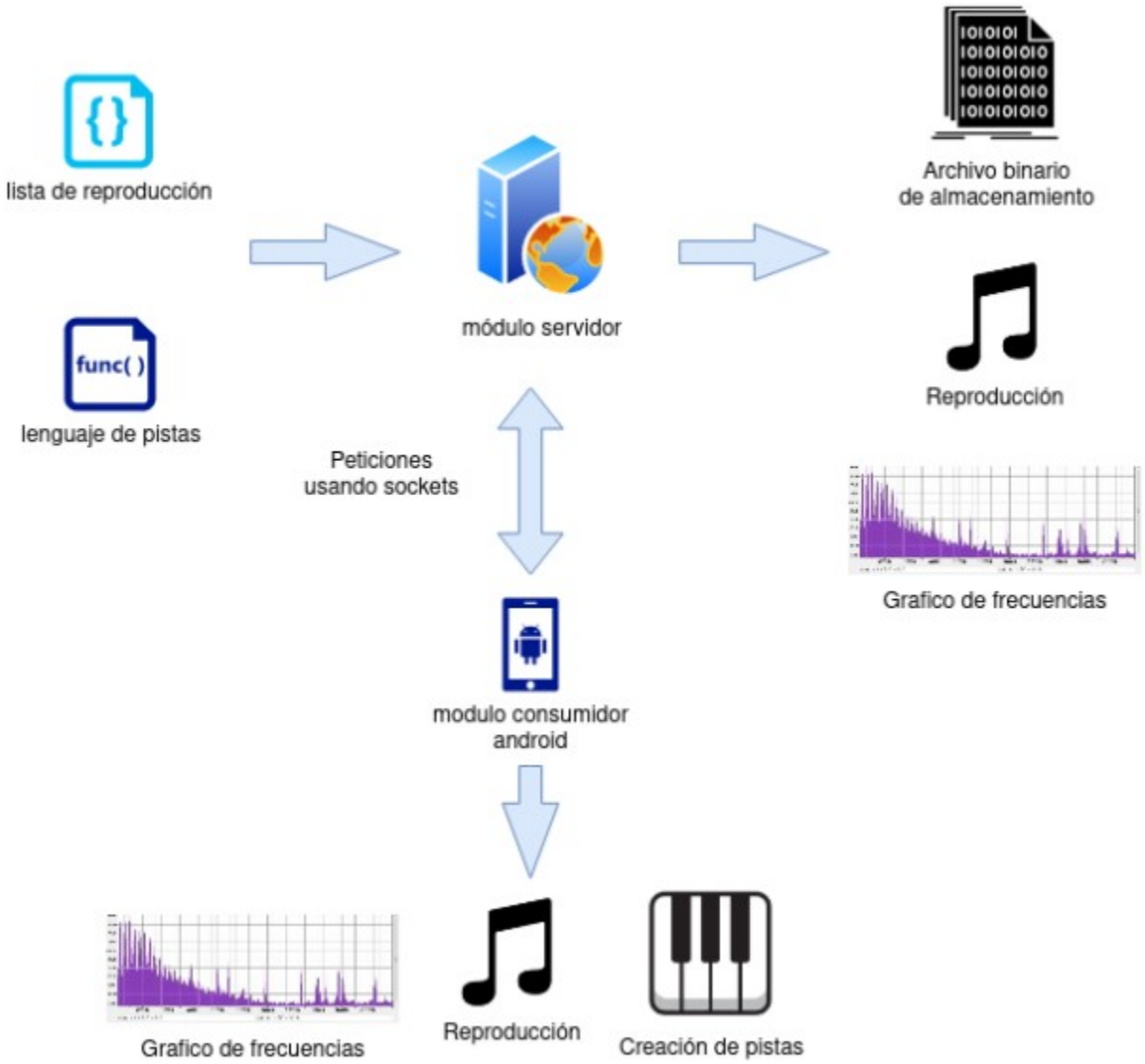
4. HERRAMIENTAS UTILIZADAS PARA EL DESARROLLO

- Java JDK. 8
- Netbeans 8.2
- jflex1.6.1
- cup 11
- Linux Ubuntu 17.04
- AndroidJDK
- Flutter

5. CONFIGURACIÓN

ver el manual de instalación

6. **ARQUITECTURA DEL SISTEMA** .



7. **GRAMÁTICA**

CREACION DE PISTAS

start with S

S ::=

pista:s {:parser.instruccion=s; RESULT= s; System.out.println("s");:}

| ERRORSENTENCE S:s {:parser.instruccion=s; RESULT= s; :}

| ;

pista::= PISTA IDENTIFICADOR:id CORCHETEIZ EXTIENDE extiende:ext CORCHETEDER

LLAVEABRE cuerpo:cPista {:RESULT= new PistaInst(id.toString(),cPista, ext);:} LLAVECIERRA

|PISTA IDENTIFICADOR:id EXTIENDE extiende:ext

LLAVEABRE cuerpo:cPista {:RESULT= new PistaInst(id.toString(),cPista, ext);:} LLAVECIERRA

| PISTA IDENTIFICADOR:id LLAVEABRE cuerpo:cPista LLAVECIERRA {:RESULT= new PistaInst(id.toString(),cPista, null);:}

//| PISTA ERRORSENTENCE EXTIENDE extiende:ext LN cuerpo:cPista {:RESULT= new PistaInst(null,cPista, ext);:}

| PISTA ERRORSENTENCE LLAVEABRE cuerpo:cPista LLAVECIERRA {:RESULT= new PistaInst(null,cPista, null);:}

;

extiende::= extiende:ext COMA IDENTIFICADOR:id {: ext.add(id.toString());

RESULT= ext;:}

| IDENTIFICADOR:id {:ArrayList<String> ids = new ArrayList();

ids.add(id.toString());

RESULT = ids; :}

;

```
cuerpo::=               cuerpoPista: cuerpo {:RESULT= cuerpo;:}

                |       {:RESULT= null;:}

;


cuerpoPista::=  cuerpoPista:cuerpo cuerpPistaP:cuerpoP     {:

                                                    cuerpo.add(cuerpoP);

                                                    RESULT= cuerpo;

                                                    :}

                |cuerpPistaP:cuerpoP                        {:ArrayList<Instruccion> sentencias
= new ArrayList();

                                                    sentencias.add(cuerpoP);

                                                    RESULT = sentencias;

                                                    :}


;

cuerpPistaP ::=     MetodoFuncioDec:fun

                        {:System.out.println("cuerpPistaP-MetodoFuncioDec");
RESULT= fun;:}

                    | declaracionVar:decVar     PUNTOYCOMA

                        {:System.out.println("cuerpPistaP-declaracionVar"); RESULT=
decVar; :}

;


arreglo ::=             declaraArreglo:arr  IGUAL elementosArreglo:elem

                    {:arr.setValores(elem); RESULT= arr;:}
```

```
                    |declaraArreglo:arr   {:RESULT= arr;:}

;


elementosArreglo::=         elementosArreglo:e LLAVEABRE listElementos:elem
LLAVECIERRA

                                {: e.add(elem); RESULT= e;:}

                            |LLAVEABRE listElementos:e LLAVECIERRA

                                {: ArrayList<ArrayList<Expresion>> elem= new
ArrayList();

                                elem.add(e);

                                RESULT= elem;:}

;

listElementos::=        listElementos:elementos COMA expresion: e

            {:elementos.add(e); RESULT= elementos;:}

            |expresion:e

            {:ArrayList<Expresion> elementos = new ArrayList();

                elementos.add(e);

                RESULT = elementos;

            :}

;

declaraArreglo::=       VAR tipos:tipo ARREGLO listaID:id  dimensionesArr:dimension

                        {:RESULT= new Arreglo(tipo, id.get(0), dimension.size(), dimension);:};


dimensionesArr::=   dimensionesArr:dimensiones CORCHETEIZ expresion:e
CORCHETEDER

                        {:dimensiones.add(e); RESULT= dimensiones;:}
```

```
                |CORCHETEIZ expresion:e CORCHETEDER
                        {:ArrayList<Expresion> dimensiones = new ArrayList();
                        dimensiones.add(e);
                        RESULT = dimensiones;
                        :}
;
MetodoFuncioDec::=      KEEP MetodoFuncion:met
                        {:  Class c = met.getClass();
                                if(c.getName().contains("Metodo")){
                                ((Metodo)met).setIsKeep(true);
                                RESULT= met;
                                }else if(c.getName().contains("Funcion")){
                                ((Funcion)met).setIsKeep(true);
                                RESULT= met;
                                }
                                :}
                |MetodoFuncion:met      {:RESULT= met;:}
                |VOID MetodoFuncion:met         {:RESULT= met;:}
;



tipos::=        SENTERO         {:RESULT=Simbolo.Tipo.STRING;:}
```

```
        |SDOBLE           {:RESULT=Simbolo.Tipo.DOUBLE;:}

        |SBOOLEAN        {:RESULT=Simbolo.Tipo.BOOL;:}

        |SCARACTER       {:RESULT=Simbolo.Tipo.CHAR;:}

        |SCADENA          {:RESULT=Simbolo.Tipo.STRING;:}

;



decrementosuma ::=          expresion:exp SUMASUMA        {: RESULT = new
DecIncremento(exp, DecIncremento.TipoDecIncremento.masmas); :}

                | expresion:exp MENOSMENOS  {: RESULT = new DecIncremento(exp,
DecIncremento.TipoDecIncremento.menosmenos); :}

                | expresion:exp               {: RESULT = new DecIncremento(exp, null); :}

;

sentencias ::=



                arreglo:sent PUNTOYCOMA{:RESULT = sent; :}

                |declaracionVar:sent  PUNTOYCOMA

                        {:RESULT = sent; :}

                | llamadaAsignacionSentencias:sent PUNTOYCOMA

                {:RESULT = sent; :}

                | decrementosuma:sent PUNTOYCOMA

                {:RESULT = sent; :}

                | sentenciaIfElse:sent

                {:RESULT = sent; :}

                | switchsentencia:sent

                {: RESULT = sent; :}

                | forSentencia:sent
```

```
                {:RESULT = sent; :}
                | mientrasSentecia:sent
                {:RESULT = sent; :}
                | hacerMientrasSentencia:sent
                {:RESULT = sent; :}
                | CONTINUESENTENCE:sentencia PUNTOYCOMA
                {: RESULT =(Instruccion) sentencia; :}
                | BREAKSENTENCE:sentencia PUNTOYCOMA
                {: RESULT = (Instruccion) sentencia; :}
                | RETURNSENTENCE:ret PUNTOYCOMA
                {: RESULT = (Instruccion)ret; :}
                | MENSAJE PARABRE expresion:exp PARCIERRA PUNTOYCOMA
                {:  Instruccion ins = new Mensaje(exp);
                RESULT = ins; :}
                |MetodoFuncionNative:funNatva PUNTOYCOMA
                {: RESULT = funNatva; ;:}
                |error sentencias:sen {: RESULT = sen; ;:}
;

listaSentencias::=    listaSentencias: instrucciones  sentencias: sent
                        {:
                                instrucciones.add(sent);
                                RESULT = instrucciones;
                        :}
```

```
                        |sentencias: sent
                                {:
                                        ArrayList<Instruccion> instrucciones = new ArrayList();
                                        instrucciones.add(sent);
                                        RESULT = instrucciones;
                                :}

;

// VARIABLES

declaraVar::=   VAR  tipos:tip listaID:idList          {:RESULT = new Declaracion(idList,tip, false );:}
                |KEEP VAR tipos:tip listaID:idList  {:RESULT = new Declaracion(idList,tip, true );:}
                |tipos:tip listaID:idList  {:RESULT = new Declaracion(idList,tip, true );:}
                |KEEP tipos:tip listaID:idList  {:RESULT = new Declaracion(idList,tip, true );:}
;

declaracionVar ::=    declaraVar:dec  IGUAL  expresion:exp
                                {:
                                        Declaracion decl= (Declaracion) dec;
                                        RESULT = new Asignacion(decl.getIds(), exp, expleft, expright);
                                :}
                        |declaraVar:dec  IGUAL  MetodoFuncionNative:funcionNativa
                                {:  Declaracion decl= (Declaracion) dec;
                                        RESULT = new Asignacion(decl.getIds(), funcionNativa);
:}
```

```
                        |declaraVar:dec

                                {:RESULT = (Declaracion)dec;:}

;

listaID::=  listaID:lsIds COMA IDENTIFICADOR:id {:lsIds.add(id.toString());

                                        RESULT = lsIds;:}

        |IDENTIFICADOR:id  {:ArrayList<String> ids = new ArrayList();

                        ids.add(id.toString());

                        RESULT = ids;:}

;


llamadaAsignacionSentencias::=  IDENTIFICADOR:id IGUAL expresion:exp

                                                {: RESULT = new Asignacion(id.toString(),
(Expresion)exp); :}

                                                |IDENTIFICADOR:id IGUAL
MetodoFuncionNative:funcionNativa

                                                {: RESULT = new Asignacion(id.toString(),
funcionNativa); :}

                                                |IDENTIFICADOR:id dimensionesArr:e IGUAL
 expresion:exp

                                                {: RESULT = new
ReasignarValorArreglo(id.toString(), exp, e); :}



;

//----------------------


/*  INSTRUCCIONES CONDICIONALES */
```

```
sentenciaIfElse ::=   IF PARABRE expresion:exp PARCIERRA  LLAVEABRE
listaSentencias:sent LLAVECIERRA

                                {: RESULT =  new SentenciaIf(exp, sent, null, null); :}

                            | IF PARABRE expresion:exp PARCIERRA LLAVEABRE
listaSentencias:sent LLAVECIERRA  ELSE LLAVEABRE listaSentencias:sentElse
LLAVECIERRA

                                {: RESULT =  new SentenciaIf( exp, sent,new
SentenciaElse(sentElse), null); :}

                            | IF PARABRE expresion:exp PARCIERRA LLAVEABRE
listaSentencias:sent LLAVECIERRA ELSE   sentenciaIfElse:sentElseIf

                                {: RESULT = new SentenciaIf(exp, sent, null,new
SentenciaIfElse(sentElseIf)); :}

;

//------------

CONTINUESENTENCE ::= CONTINUAR

        ;

BREAKSENTENCE ::= SALIR  {:RESULT= new Salir();:}

        ;



RETURNSENTENCE ::=    RETORNA expresion:exp   {:RESULT =new Return(exp);:}

                                | RETORNA                   {:RESULT = new Return(null);:}

;

 //SWITCH

switchsentencia ::=           SWITCH PARABRE expresion:exp PARCIERRA LLAVEABRE
LLAVECIERRA

                                        {: RESULT = new SwitchInstruccion(exp, null, null); :}

                                |SWITCH PARABRE expresion:exp PARCIERRA LLAVEABRE
caselist:casels defaultOp:def  LLAVECIERRA

                                        {: RESULT =  new SwitchInstruccion(exp, casels, def); :}
```

```
                                | SWITCH PARABRE expresion:exp PARCIERRA LLAVEABRE
caselist:casels  LLAVECIERRA

                                        {: RESULT =  new SwitchInstruccion(exp, casels, null); :}

                                | SWITCH PARABRE expresion:exp PARCIERRA  LLAVEABRE
defaultOp:def          LLAVECIERRA

                                        {: RESULT = new SwitchInstruccion(exp, null, def); :}

;



caselist::= caselist:casesF caseFinal:casef       {:

                                casesF.add(casef);

                                RESULT = casesF;

                        :}

        | caseFinal:casef               {: ArrayList<Case> cases = new ArrayList();

                                cases.add(casef);

                                RESULT = cases; :}

        ;



caseFinal::=  CASO expresion:exp DOSPUNTOS  listaSentencias:sent     {: RESULT =
new Case(exp, sent); :}

                |CASO expresion:exp DOSPUNTOS

                ;



defaultOp::=  DEFAULT DOSPUNTOS listaSentencias:sent             {: RESULT = new
Default(sent); :}

                |DEFAULT DOSPUNTOS



;
```

```
//------------------------------

//CICLOS

//FOR

 forSentencia ::=               PARA PARABRE asignacionesFor:asign PUNTOYCOMA
expresion:expr PUNTOYCOMA parFor:asignFor PARCIERRA

                                LLAVEABRE listaSentencias:sent LLAVECIERRA

                                        {: RESULT = new For(asign, expr, asign, sent); :}

                                |PARA PARABRE asignacionesFor:asign PUNTOYCOMA
expresion:expr PUNTOYCOMA parFor:asignFor PARCIERRA

                                LLAVEABRE  LLAVECIERRA

                                        {: RESULT = new For(asign, expr, asign, null); :}

          ;



parFor::=       declaracionVar:dec {:RESULT = dec;:}

                        |decrementosuma:dec {:RESULT = dec;:}

;



asignacionesFor::=              tipos:tipo IDENTIFICADOR:id IGUAL expresion:exp

                                        {: RESULT = new DeclaracionFor((Simbolo.Tipo)tipo,
id.toString(), (Expresion)exp); :}

                                | IDENTIFICADOR:id IGUAL expresion:exp

                                        {: RESULT = new DeclaracionFor(null, id.toString(),
(Expresion)exp); :}

                                |VAR tipos:tipo IDENTIFICADOR:id IGUAL expresion:exp

                                        {: RESULT = new DeclaracionFor((Simbolo.Tipo)tipo,
```

```
id.toString(), (Expresion)exp); :}

        ;



//MIENTRAS WHILE

mientrasSentecia::= MIENTRAS PARABRE expresion:expr PARCIERRA LLAVEABRE
listaSentencias:list LLAVECIERRA

                    {: RESULT = new While( expr,list); :}

                    |MIENTRAS PARABRE expresion:expr PARCIERRA LLAVEABRE
LLAVECIERRA

                    {: RESULT = new While( expr,null); :}

        ;



//DOWHILE



hacerMientrasSentencia::=

                                HACER LLAVEABRE listaSentencias:list LLAVECIERRA
MIENTRAS PARABRE expresion:expr PARCIERRA

                                        {: RESULT = new DoWhile(expr,list);  :}

                                |HACER LLAVEABRE LLAVECIERRA  MIENTRAS PARABRE
expresion:expr PARCIERRA

                                        {: RESULT = new DoWhile(expr,null);  :}

;

//declaracion METODO FUNCION

 MetodoFuncion::=   tipos:tipo IDENTIFICADOR:id PARABRE declaracionParametro:param
PARCIERRA  LLAVEABRE listaSentenciasMetodo:ls  LLAVECIERRA

                                {: RESULT = new Funcion(id.toString(), param,
(ArrayList<Instruccion>)ls, (Simbolo.Tipo)tipo); :}
```

```
                        | tipos:tipo IDENTIFICADOR:id PARABRE PARCIERRA   LLAVEABRE
listaSentenciasMetodo:ls  LLAVECIERRA

                                {: RESULT = new Funcion(id.toString(),
null,(ArrayList<Instruccion>) ls, (Simbolo.Tipo)tipo);:}



                        | IDENTIFICADOR:id PARABRE declaracionParametro:params
PARCIERRA  LLAVEABRE listaSentenciasMetodo:ls   LLAVECIERRA

                                {: RESULT = new Metodo(id.toString(),
params,(ArrayList<Instruccion>)ls ,false); :}



                        | IDENTIFICADOR:id PARABRE PARCIERRA  LLAVEABRE
listaSentenciasMetodo:ls    LLAVECIERRA

                                {: RESULT = new Metodo(id.toString(), null,
(ArrayList<Instruccion>)ls,false); :}



                        | PRINCIPAL:id PARABRE PARCIERRA   LLAVEABRE
listaSentenciasMetodo:ls    LLAVECIERRA

                                {: RESULT = new Metodo(id.toString(), null,
(ArrayList<Instruccion>)ls, true); :}



;

listaSentenciasMetodo::=            listaSentencias:list {:RESULT= list;:}

                                    |{:RESULT=null;:}

;

MetodoFuncionNative::=  REPRODUCIR PARABRE notas:nota COMA expresion:exp1
COMA expresion:exp2 COMA expresion:exp3 PARCIERRA

                                {:RESULT = new Reproducir(nota.toString(), exp1,exp2,
exp3 );:}

                        |ESPERAR PARABRE num:ms COMA num:canal PARCIERRA
```

```
                                {:RESULT = new Esperar((Integer)ms, (Integer)canal);:}

                            |ORDENAR PARABRE parSumarizar:exp COMA
formaOrdenar:f PARCIERRA

                                {:RESULT = new Ordenar((Expresion)exp,f.toString());:}

                            |SUMARIZAR PARABRE parSumarizar:sum PARCIERRA


                                {: RESULT = new Sumarizar((Expresion) sum); :}

                            |LONGITUD PARABRE parLongitud:sum PARCIERRA


                                {:RESULT = new Longitud(sum);:}

                            |error PUNTOYCOMA

;



formaOrdenar::=     ASCENDENTE      {:RESULT = "ascendente";:}

                |DESCENDENTE    {:RESULT = "descendente";:}

                |PARES      {:RESULT = "pares";:}

                |IMPARES    {:RESULT = "impares";:}

                |PRIMOS     {:RESULT = "primos";:}

;



parSumarizar::=     IDENTIFICADOR:p         {:RESULT = new
Identificador(p.toString(),pleft, pright);:}

                |declaracionesArr:arr   {:RESULT = arr;:}

;



parLongitud::=      IDENTIFICADOR:p         {: RESULT = new
Identificador(p.toString(),pleft, pright);:}
```

```
                |CADENA:cadena              {: RESULT = new Primitivo(Simbolo.Tipo.STRING,
cadena.toString(),cadenaleft, cadenaright);:}

                |declaracionesArr:arr   {:  RESULT = (Expresion) arr;:}

;



num::=  ENTERO:num       {:RESULT=  Integer. parseInt(num.toString());:}

        |DECIMAL:num       {:RESULT=  Integer. parseInt(num.toString());:}

;



notas::=        IDENTIFICADOR:id {:

                String nota = Nota.ComprobarNota(id.toString());

                if(nota!= null){

                RESULT = nota;

                }else{

                RESULT = null;

                }

        :}

;

declaracionParametro ::=  parametro:param  COMA declaracionParametro:params        {:
if(params!=null){

                                                        params.add((Parametro)param);

                                                        RESULT = params;

                                                        }else{

                                                        RESULT= null;

                                                        }

                                                :}
```

```
                | parametro:parametro                     {:  ArrayList<Parametro> params =
new ArrayList();

                                                        params.add((Parametro)parametro);

                                                        RESULT = params;  :}

                | ERRORSENTENCE COMA declaracionParametro:params  {: RESULT =
params; :}

                | ERRORSENTENCE  {: RESULT = null; :}

;



parametro::= tipos:tipo IDENTIFICADOR:id {: RESULT = new
Parametro((Simbolo.Tipo)tipo, id.toString()); :}

        ;



//----------------------------------------

expresion::=  expr:exp                  {:RESULT= (Expresion) exp;:}

                |primitivas:exp                 {:RESULT= (Expresion) exp;:}

                |IDENTIFICADOR:id dimensionesArr:dim

                {: RESULT = (Expresion)new LlamadaArreglo(id.toString(), "Arreglo", dim); :}

                |IDENTIFICADOR:id {: RESULT = new Primitivo(Simbolo.Tipo.STRING,new
String(id.toString()), idleft, idright); :}

;



expr ::=        MENOS expresion:der

                {: RESULT = new Operacion(der, der,
Operacion.Operador.MENOS_UNARIO, derleft, derright );  :} %prec UMENOS

                | NOT expresion:der

                {: RESULT = new Operacion(der, der, Operacion.Operador.NOT, derleft,
derright );  :}
```

```
                | ESNULO expresion:der

                {: RESULT = new Operacion(der, der, Operacion.Operador.NULO, derleft,
derright );  :}



                | expresion:der MAS expresion:iz

                {: RESULT = new Operacion(der, iz, Operacion.Operador.SUMA, derleft,
derright );  :}

                | expresion:der MENOS expresion:iz

                {: RESULT = new Operacion(der, iz, Operacion.Operador.RESTA, derleft,
derright );  :}

                | expresion:der ASTERISCO expresion:iz

                {: RESULT = new Operacion(der, iz, Operacion.Operador.MULTIPLICACION,
derleft, derright );  :}

                | expresion:der DIV expresion:iz

                {: RESULT = new Operacion(der, iz, Operacion.Operador.DIVISION, derleft,
derright ); :}

                | expresion:der POT expresion:iz

                {: RESULT = new Operacion(der, iz, Operacion.Operador.POTENCIA, derleft,
derright );  :}

                | expresion:der MODULO expresion:iz

                {: RESULT = new Operacion(der, iz, Operacion.Operador.MODULO, derleft,
derright );  :}



                | expresion:der MAYOR expresion:iz      {: RESULT = new Operacion(der, iz,
Operacion.Operador.MAYOR_QUE, derleft, derright ); :}

                | expresion:der MENOR expresion:iz      {: RESULT = new Operacion(der, iz,
Operacion.Operador.MENOR_QUE, derleft, derright );  :}

                | expresion:der MAYORIGUAL expresion:iz   {: RESULT = new
Operacion(der, iz, Operacion.Operador.MAYOR_IGUA_QUE, derleft, derright );  :}

                | expresion:der MENORIGUAL expresion:iz   {: RESULT = new
```

```
Operacion(der, iz, Operacion.Operador.MENOR_IGUA_QUE, derleft, derright );  :}


                | expresion:der IGUAL IGUAL expresion:iz  {: RESULT = new Operacion(der,
iz, Operacion.Operador.IGUAL_IGUAL, derleft, derright );  :}

                | expresion:der NOIGUAL expresion:iz    {: RESULT = new Operacion(der, iz,
Operacion.Operador.NOI_GUAL, derleft, derright );  :}



                | expresion:der AND expresion:iz          {: RESULT = new Operacion(der, iz,
Operacion.Operador.AND, derleft, derright ); :}

                | expresion:der NAND expresion :iz        {: RESULT = new Operacion(der, iz,
Operacion.Operador.NAND,derleft, derright ); :}

                |  expresion:der OR expresion :iz          {: RESULT = new Operacion(der, iz,
Operacion.Operador.OR,  derleft, derright ); :}

                | expresion:der NOR expresion:iz          {: RESULT = new Operacion(der, iz,
Operacion.Operador.NOR,derleft, derright ); :}

                | expresion:der XOR expresion:iz   {: RESULT = new Operacion(der, iz,
Operacion.Operador.XOR,derleft, derright ); :}



                | PARABRE expresion:expr PARCIERRA

                {: RESULT = expr; :}

                |IDENTIFICADOR:id MASIGUAL expresion:e

                {:RESULT = new Operacion(new Identificador(id.toString(), idleft, idright),e,
Operacion.Operador.SUMA, idleft, idright );  :}

                | IDENTIFICADOR:id PARABRE listaExpresion:e PARCIERRA

                {: RESULT = (Expresion)new LlamadaFuncion(id.toString(), e);  :}

                |IDENTIFICADOR:id PARABRE PARCIERRA

                {: RESULT = (Expresion)new LlamadaFuncion(id.toString(), null); :}

;
```

```
primitivas::=

                DECIMAL:p           {: RESULT = new Primitivo(Simbolo.Tipo.DOUBLE,new
Double(p.toString()), pleft, pright); :}

                | ENTERO:p           {: RESULT = new Primitivo(Simbolo.Tipo.INT,
Integer.valueOf(p.toString()), pleft,pright);          :}

                | CADENA:p          {: RESULT = new Primitivo(Simbolo.Tipo.STRING,
p.toString(), pleft, pright); :}

                | CHAR:p            {: RESULT = new
Primitivo(Simbolo.Tipo.CHAR,p.toString().charAt(0), pleft, pright);     :}

                | BOOLF:p           {: RESULT = new Primitivo(Simbolo.Tipo.BOOL,false,
pleft, pright);           :}

                | BOOLT:p           {: RESULT = new Primitivo(Simbolo.Tipo.BOOL,true,
pleft, pright);           :}

   ;

listaExpresion::=      listaExpresion:lsexp COMA expresion:exp

                                {:lsexp.add(exp);

                                RESULT= lsexp;:}

                        |expresion:exp        PUNTOYCOMA

                                {:ArrayList<Expresion> ex= new ArrayList();

                                ex.add(exp);

                                RESULT= ex;

                                :}

;

ERRORSENTENCE ::=      FINALERROR   ;

FINALERROR ::=  error

        ;
```

CREACIÓN LISTAS DE REPRODUCCIÓN

```
start with inicio ;


inicio::=     cuerpoInicio:listas
                    {:parser.listas=listas; RESULT= listas;:}
;


cuerpoInicio::=          cuerpoInicio:listas nuevaLista:l
                              {:
                              listas.add(l);
                              RESULT= listas;
                              :}
                         |nuevaLista:L
                              {:
                              ArrayList<Lista> listas = new ArrayList();
                              listas.add(L);
                              RESULT= listas;
                              :}
;


nuevaLista::=     LLAVEABRE LISTA DOSPUNTOS LLAVEABRE cuerpoL:lista LLAVECIERRA
LLAVECIERRA
                         {:RESULT=lista; :}
;
cuerpoL::=        NOMBRE DOSPUNTOS CADENA:nombre
                    {:
```

```
                    Lista lista=new Lista();
                    AtributoLista at= new AtributoLista("nombre", nombre);
                    lista.Asignar(at);
                    RESULT= lista;
                :}
                |NOMBRE DOSPUNTOS CADENA:nombre COMA cuerpoLista:lista
                {:
                    AtributoLista at= new AtributoLista("nombre", nombre);
                    lista.Asignar(at);
                    RESULT= lista;
                :}
;
cuerpoLista::=  cuerpoLista:lista COMA cuerpo:c
                    {:
                        lista.Asignar(c);
                        RESULT= lista;
                    :}
                |cuerpo:c
                    {:
                        Lista lista=new Lista();
                        lista.Asignar(c);
                        RESULT= lista;
                    :}
;
```

```
cuerpo::=
        ALEATORIA DOSPUNTOS verdaderoFalso :fv
            {:
                AtributoLista at= new AtributoLista("random", fv);
                RESULT = at;
            :}
        |CIRCULAR DOSPUNTOS verdaderoFalso:fv
            {:
                AtributoLista at= new AtributoLista("circular", fv);
                RESULT = at;
            :}
        |PISTAS DOSPUNTOS CORCHETEIZ pistas:p CORCHETEDER
            {:
                AtributoLista at= new AtributoLista("pistas", p);
                RESULT = at;
            :}
        |error cuerpo:c {:RESULT = c;:}
;

verdaderoFalso::=       BOOLF:e  {:RESULT = false;:}
                        |BOOLT:e {:RESULT = true;:}
;
```

```
pistas::=           pistas:p  COMA IDENTIFICADOR:id
                        {:p.add(id.toString()); RESULT = p;:}
                    |IDENTIFICADOR : id
                        {:
                            ArrayList<String> identificadores=new ArrayList();
                            identificadores.add(id.toString());
                            RESULT= identificadores;
                        :}
;
```

8. ANALIZADORES LÉXICOS

ANALIZADOR CREADOR DE PISTAS

```
BLANCOS=[ \r\t\n]+
D = [0-9]+
LetraS = [A-Za-zÑñ]
simbol= [_@#$-]
Comilla = [\"]
ID = [A-Za-z|Ñ|ñ|0-9]*
OR = [\|]
%state COMENT_MULTI
%state COMENT_SIMPLE
%state CADENA
%state CHAR
%%
```

```
<YYINITIAL> "<-"        {yybegin(COMENT_MULTI);pintar.pintaGris(yychar,yylength());}

<COMENT_MULTI> "->"        { pintar.pintaGris(yychar,yylength()); yybegin(YYINITIAL);}

<COMENT_MULTI>[^\n]        { pintar.pintaGris(yychar,yylength()); }

<COMENT_MULTI> [\r|\r\n\f] { pintar.pintaGris(yychar,yylength()); }


<YYINITIAL> ("\'"|"\’")            {yybegin(CHAR);pintar.pintaNara(yychar,yylength());}

<CHAR>  ("\'"|"\’") { pintar.pintaNara(yychar,yylength()); yybegin(YYINITIAL);return
symbol(sym.CHAR , yytext(), yyline, yycolumn);}

<CHAR>({LetraS}|{D}|"#t"|"#n"|simbol)        { pintar.pintaNara(yychar,yylength()); }


<YYINITIAL> ("\""|"\“")            {yybegin(CADENA);pintar.pintaNara(yychar,yylength());}

<CADENA>  ("\"")  { pintar.pintaNara(yychar,yylength()); yybegin(YYINITIAL);return
symbol(sym.CADENA , yytext(), yyline, yycolumn);}

<CADENA>[^\n]          { pintar.pintaNara(yychar,yylength()); }

<CADENA> [\r|\r\n\f] { pintar.pintaNara(yychar,yylength()); }


<YYINITIAL> ">>"                {yybegin(COMENT_SIMPLE); pintar.pintaGris(yychar,yylength());}

<COMENT_SIMPLE> .          { pintar.pintaGris(yychar,yylength()); }

<COMENT_SIMPLE> "\n"      { pintar.pintaGris(yychar,yylength()); yybegin(YYINITIAL);}


<YYINITIAL> {


    ({OR})({OR})                        {System.out.println( "OR"+ yytext()); return symbol(sym.OR,
```

```
yytext(), yyline, yycolumn);}

    ({BLANCOS})                 {}



    ("entero")                  {  pintar.pintaAzul(yychar,yylength());  return symbol(sym.SENTERO,
yytext(), yyline, yycolumn);}

    ("doble")                   {  pintar.pintaAzul(yychar,yylength());  return symbol(sym.SDOBLE,
yytext(), yyline, yycolumn);}

    ("boolean")                 {  pintar.pintaAzul(yychar,yylength());  return symbol(sym.SBOOLEAN,
yytext(), yyline, yycolumn);}

    ("caracter")                {  pintar.pintaAzul(yychar,yylength());  return
symbol(sym.SCARACTER, yytext(), yyline, yycolumn);}

    ("cadena")                          {  pintar.pintaAzul(yychar,yylength());  return
symbol(sym.SCADENA, yytext(), yyline, yycolumn);}




    ("Extiende"|"extiende")     {  pintar.pintaAzul(yychar,yylength()); return symbol(sym.EXTIENDE,
yytext(), yyline, yycolumn);}

    ("Pista"|"pista")                   {  pintar.pintaAzul(yychar,yylength());  return
symbol(sym.PISTA, yytext(), yyline, yycolumn);}

    ("falso"|"false")                   {  pintar.pintaAzul(yychar,yylength()); return
symbol(sym.BOOLF, yytext(), yyline, yycolumn);}

    ("verdadero"|"true")        {  pintar.pintaAzul(yychar,yylength()); return symbol(sym.BOOLT,
yytext(), yyline, yycolumn);}

    ("keep"|"Keep")                     {  pintar.pintaAzul(yychar,yylength()); return symbol(sym.KEEP,
yytext(), yyline, yycolumn);}



    ("var"|"Var")               { System.out.println( "VAR"+ yytext());
pintar.pintaAzul(yychar,yylength()); return symbol(sym.VAR, yytext(), yyline, yycolumn);}

    ("arreglo"|"Arreglo")               {  pintar.pintaAzul(yychar,yylength()); return
symbol(sym.ARREGLO, yytext(), yyline, yycolumn);}

    ("switch"|"Switch")                 {  pintar.pintaAzul(yychar,yylength()); return
symbol(sym.SWITCH, yytext(), yyline, yycolumn);}
```

```
    ("default"|"Default")                { pintar.pintaAzul(yychar,yylength()); return
symbol(sym.DEFAULT, yytext(), yyline, yycolumn);}

    ("caso"|"Caso")                      { pintar.pintaAzul(yychar,yylength()); return symbol(sym.CASO,
yytext(), yyline, yycolumn);}

    ("salir"|"Salir")                { pintar.pintaAzul(yychar,yylength()); return symbol(sym.SALIR,
yytext(), yyline, yycolumn);}

    ("para"|"Para")                      { pintar.pintaAzul(yychar,yylength()); return symbol(sym.PARA,
yytext(), yyline, yycolumn);}

    ("mientras"|"Mientras")              { pintar.pintaAzul(yychar,yylength()); return
symbol(sym.MIENTRAS, yytext(), yyline, yycolumn);}

    ("hacer"|"Hacer")                    { pintar.pintaAzul(yychar,yylength()); return
symbol(sym.HACER, yytext(), yyline, yycolumn);}

    ("continuar"|"Continuar")            { pintar.pintaAzul(yychar,yylength()); return
symbol(sym.CONTINUAR, yytext(), yyline, yycolumn);}

    ("void"|"Void")                  { pintar.pintaAzul(yychar,yylength()); return symbol(sym.VOID,
yytext(), yyline, yycolumn);}

    ("retorna"|"Retorna")                { pintar.pintaAzul(yychar,yylength()); return
symbol(sym.RETORNA, yytext(), yyline, yycolumn);}




    ("Principal")                    { pintar.pintaAzul(yychar,yylength()); return symbol(sym.PRINCIPAL,
yytext(), yyline, yycolumn);}

    ("reproducir"|"Reproducir")          { pintar.pintaAzul(yychar,yylength()); return
symbol(sym.REPRODUCIR, yytext(), yyline, yycolumn);}

    ("esperar"|"Esperar")                { pintar.pintaAzul(yychar,yylength()); return
symbol(sym.ESPERAR, yytext(), yyline, yycolumn);}

    ("ordenar"|"Ordenar")                { pintar.pintaAzul(yychar,yylength()); return
symbol(sym.ORDENAR, yytext(), yyline, yycolumn);}

    ("sumarizar"|"Sumarizar")            { pintar.pintaAzul(yychar,yylength()); return
symbol(sym.SUMARIZAR, yytext(), yyline, yycolumn);}

    ("longitud"|"Longitud")              { pintar.pintaAzul(yychar,yylength()); return
symbol(sym.LONGITUD, yytext(), yyline, yycolumn);}

    ("mensaje"|"Mensaje")                { pintar.pintaAzul(yychar,yylength()); return
```

```
symbol(sym.MENSAJE, yytext(), yyline, yycolumn);}


   ("ascendente"|"Ascendente")          {  pintar.pintaAzul(yychar,yylength()); return
symbol(sym.ASCENDENTE, yytext(), yyline, yycolumn);}

   ("descendente"|"Descendente")        {  pintar.pintaAzul(yychar,yylength()); return
symbol(sym.DESCENDENTE, yytext(), yyline, yycolumn);}

   ("pares"|"Pares")                    {  pintar.pintaAzul(yychar,yylength()); return
symbol(sym.PARES, yytext(), yyline, yycolumn);}

   ("impares"|"Impares")                {  pintar.pintaAzul(yychar,yylength()); return
symbol(sym.IMPARES, yytext(), yyline, yycolumn);}

   ("primos"|"Primos")                  {  pintar.pintaAzul(yychar,yylength()); return
symbol(sym.PRIMOS, yytext(), yyline, yycolumn);}


   ("si"|"Si")          {  pintar.pintaAzul(yychar,yylength()); return symbol(sym.IF, yytext(), yyline,
yycolumn);}

   ("sino"|"Sino")              {  pintar.pintaAzul(yychar,yylength()); return symbol(sym.ELSE,
yytext(), yyline, yycolumn);}



   "("                          { return symbol(sym.PARABRE, yytext(), yyline, yycolumn);}

   ")"                          { return symbol(sym.PARCIERRA, yytext(), yyline, yycolumn);}

   "["                          { return symbol(sym.CORCHETEIZ, yytext(), yyline, yycolumn);}

   "]"                          { return symbol(sym.CORCHETEDER, yytext(), yyline, yycolumn);}

   "{"                          { return symbol(sym.LLAVEABRE, yytext(), yyline, yycolumn);}

   "}"                          { return symbol(sym.LLAVECIERRA, yytext(), yyline, yycolumn);}



   ({LetraS})({ID})             {System.out.println( "ID"); pintar.pintaVerde(yychar,yylength());
```

```
System.out.println( "id"); return symbol(sym.IDENTIFICADOR , yytext(), yyline, yycolumn);}

    ({D})(".")({D})                 { pintar.pintaMora(yychar,yylength()); return
symbol(sym.DECIMAL , yytext(), yyline, yycolumn);}

    ({D})                           { pintar.pintaMora(yychar,yylength()); System.out.println( "int"); return
symbol(sym.ENTERO , yytext(), yyline, yycolumn);}



    "+="                            { return symbol(sym.MASIGUAL, yytext(), yyline, yycolumn);}

    "++"                            { return symbol(sym.SUMASUMA, yytext(), yyline, yycolumn);}

    "--"                            { return symbol(sym.MENOSMENOS, yytext(), yyline, yycolumn);}


    "+"                             { return symbol(sym.MAS, yytext(), yyline, yycolumn);}

    "*"                             { return symbol(sym.ASTERISCO, yytext(), yyline, yycolumn);}

    "-"                             { return symbol(sym.MENOS, yytext(), yyline, yycolumn);}

    "/"                             { return symbol(sym.DIV, yytext(), yyline, yycolumn);}

    "%"                             { return symbol(sym.MODULO, yytext(), yyline, yycolumn);}

    "^"                             { return symbol(sym.POT, yytext(), yyline, yycolumn);}


    "!="                            { return symbol(sym.NOIGUAL, yytext(), yyline, yycolumn);}

    ">"                             { return symbol(sym.MAYOR, yytext(), yyline, yycolumn);}

    "<"                             { return symbol(sym.MENOR, yytext(), yyline, yycolumn);}

    ">="                            { return symbol(sym.MAYORIGUAL, yytext(), yyline, yycolumn);}

    "<="                            { return symbol(sym.MENORIGUAL, yytext(), yyline, yycolumn);}

    "!!"                            { return symbol(sym.ESNULO, yytext(), yyline, yycolumn);}

    "="                             { return symbol(sym.IGUAL, yytext(), yyline, yycolumn);}
```

```
    "&&"                        { return symbol(sym.AND, yytext(), yyline, yycolumn);}

    "!&&"                       { System.out.println( "NAND"+ yytext()); return symbol(sym.NAND,
yytext(), yyline, yycolumn);}

    ("!")({OR})({OR})                       { System.out.println( "NOR"+ yytext()); return
symbol(sym.NOR, yytext(), yyline, yycolumn);}

    ("&")({OR})                     { System.out.println( "XOR"+ yytext());return symbol(sym.XOR,
yytext(), yyline, yycolumn);}

    ("!")                       { return symbol(sym.NOT, yytext(), yyline, yycolumn);}



    ";"                         { return symbol(sym.PUNTOYCOMA, yytext(), yyline+1, yycolumn+1);}

    ":"                         { return symbol(sym.DOSPUNTOS, yytext(), yyline+1, yycolumn+1);}

    ","                         { return symbol(sym.COMA, yytext(), yyline, yycolumn);}

    ( \t\r\n\f)+                {/*IGNORAR*/}

 . {ERROR}
```

ANALIZADOR CREADOR DE LISTAS

```
BLANCOS=[ \r\t\n]+

LetraS = [A-Za-zÑñ]

Comilla = [\"]

ID = [A-Za-z|Ñ|ñ|0-9]*

OR = [\|]

%state COMENT_MULTI

%state COMENT_SIMPLE

%%
```

```
<YYINITIAL> "<-"        {yybegin(COMENT_MULTI);}

<COMENT_MULTI> "->"     {pintar.pintaGris(yychar,yylength()); yybegin(YYINITIAL);  }

<COMENT_MULTI>[^\n]     {}

<COMENT_MULTI> [\r|\r\n\f]  {}



<YYINITIAL> "//"        {yybegin(COMENT_SIMPLE);}

<COMENT_SIMPLE> .       { pintar.pintaGris(yychar,yylength()); }

<COMENT_SIMPLE> "\n"    { pintar.pintaGris(yychar,yylength()); yybegin(YYINITIAL);}


//simbolos

<YYINITIAL> {


 ({BLANCOS})                {}



//palabras reservadas

    ("lista")                       {  pintar.pintaAzul(yychar,yylength());  return symbol(sym.LISTA,
yytext(), yyline, yycolumn);}

    ("nombre")                  {  pintar.pintaAzul(yychar,yylength());  return symbol(sym.NOMBRE,
yytext(), yyline, yycolumn);}

    ("pistas")                      {  pintar.pintaAzul(yychar,yylength());  return
symbol(sym.PISTAS, yytext(), yyline, yycolumn);}

    ("random"|"aleatoria")              {  pintar.pintaAzul(yychar,yylength());  return
symbol(sym.ALEATORIA, yytext(), yyline, yycolumn);}
```

```
        ("circuar")                         {  pintar.pintaAzul(yychar,yylength());  return
symbol(sym.CIRCULAR, yytext(), yyline, yycolumn);}

        ("falso"|"false")               {  pintar.pintaAzul(yychar,yylength()); return symbol(sym.BOOLF,
yytext(), yyline, yycolumn);}

        ("verdadero"|"true")            {  pintar.pintaAzul(yychar,yylength()); return symbol(sym.BOOLT,
yytext(), yyline, yycolumn);}




    // signos de agrupacion

      "["                               { return symbol(sym.CORCHETEIZ, yytext(), yyline, yycolumn);}

       "]"                              { return symbol(sym.CORCHETEDER, yytext(), yyline, yycolumn);}

       "{"                              { return symbol(sym.LLAVEABRE, yytext(), yyline, yycolumn);}

       "}"                              { return symbol(sym.LLAVECIERRA, yytext(), yyline, yycolumn);}



       "/"                              { return symbol(sym.DIV, yytext(), yyline, yycolumn);}

       ":"                              { return symbol(sym.DOSPUNTOS, yytext(), yyline, yycolumn);}

       ","                              { return symbol(sym.COMA, yytext(), yyline, yycolumn);}

    // digitos

        ({Comilla})({LetraS}({LetraS}{1,9})|({LetraS}))({Comilla})      {pintar.pintaNara(yychar,yylength());
System.out.println( "cadena"); return symbol(sym.CADENA , yytext(), yyline, yycolumn);}

        ({LetraS})({ID})                    {System.out.println( "ID"); pintar.pintaVerde(yychar,yylength());
System.out.println( "id"); return symbol(sym.IDENTIFICADOR , yytext(), yyline, yycolumn);}



    // caracteres de escape

      ( \t\r\n\f)+                      {/*IGNORAR*/}

       . {
```

```
    ErrorL nuevoError= new ErrorL(yytext(),yyline,yychar,"lexico");
    ErrorL.tablaErrores.add(nuevoError);
   System.out.println("Este es un error lexico: "+yytext()+
    ", en la linea: "+yyline+", en la columna: "+yychar);
    }
}
```