```
package analizadores;
import java_cup.runtime.Symbol;
import excepciones.Errores;
import analizadores.TokenLista; // IMPORTANTE: Para guardar los tokens
import analizadores.Tokens;     // IMPORTANTE: Tu enum de tokens

%%

%class Scanner
%public
%line
%column
%cup
%ignorecase

%{
   // --- MÉTODOS AUXILIARES PARA GUARDAR EN LA LISTA ---

   // 1. Para tokens con valor (como números o IDs)
   private Symbol symbol(int type, Object value, Tokens tokenTipo) {
      TokenLista.guardar(tokenTipo, yytext(), yyline+1, yycolumn+1);
      return new Symbol(type, yyline+1, yycolumn+1, value);
   }

   // 2. Para tokens simples (palabras reservadas, signos)
   private Symbol symbol(int type, Tokens tokenTipo) {
      TokenLista.guardar(tokenTipo, yytext(), yyline+1, yycolumn+1);
      return new Symbol(type, yyline+1, yycolumn+1);
   }
%}

// --- EXPRESIONES REGULARES ---
ENTERO  = [0-9]+
DECIMAL = [0-9]+\.[0-9]+
ID      = [a-zA-Z][a-zA-Z0-9_]*
CADENA  = \"([^\\\"]|\\.)*\"
CHAR    = \'([^\\\']|\\.)\'
ESPACIO = [ \t\r\n\f]+
COMENTARIO_LINEA = "//".*
COMENTARIO_BLOQUE = "/*" [^]*? "*/"

%%

<YYINITIAL> {
   // --- PALABRAS RESERVADAS ---
   "var"      { return symbol(sym.VAR, Tokens.VAR); }
   "int"      { return symbol(sym.INT, Tokens.INT); }
   "double"   { return symbol(sym.DOUBLE, Tokens.DOUBLE); }
   "bool"     { return symbol(sym.BOOL, Tokens.BOOL); }
   "char"     { return symbol(sym.CHAR, Tokens.CHAR); }
   "string"   { return symbol(sym.STRING, Tokens.STRING); }
   "if"       { return symbol(sym.IF, Tokens.IF); }
   "else"     { return symbol(sym.ELSE, Tokens.ELSE); }
   "switch"   { return symbol(sym.SWITCH, Tokens.SWITCH); }
   "case"     { return symbol(sym.CASE, Tokens.CASE); }
   "default"  { return symbol(sym.DEFAULT, Tokens.DEFAULT); }
   "break"    { return symbol(sym.BREAK, Tokens.BREAK); }
```

```
"while"    { return symbol(sym.WHILE, Tokens.WHILE); }
"for"      { return symbol(sym.FOR, Tokens.FOR); }
"do"       { return symbol(sym.DO, Tokens.DO); }
"print"    { return symbol(sym.PRINT, Tokens.PRINT); }
"true"     { return symbol(sym.TRUE, Tokens.TRUE); }
"false"    { return symbol(sym.FALSE, Tokens.FALSE); }
"new"      { return symbol(sym.NEW, Tokens.NEW); }
"list"     { return symbol(sym.LIST, Tokens.LIST); }
"append"   { return symbol(sym.APPEND, Tokens.APPEND); }
"return"   { return symbol(sym.RETURN, Tokens.RETURN); }
"continue" { return symbol(sym.CONTINUE, Tokens.CONTINUE); }
"start"    { return symbol(sym.START, Tokens.START); }

// --- OPERADORES ---
"++"       { return symbol(sym.MASMAS, Tokens.INCREMENTO); }
"--"       { return symbol(sym.MENOSMENOS, Tokens.DECREMENTO); }
"+"        { return symbol(sym.MAS, Tokens.SUMA); }
"-"        { return symbol(sym.MENOS, Tokens.RESTA); }
"**"       { return symbol(sym.POTENCIA, Tokens.POT); }
"*"        { return symbol(sym.POR, Tokens.MULT); }
"/"        { return symbol(sym.DIV, Tokens.DIV); }
"%"        { return symbol(sym.MOD, Tokens.MOD); }

// --- SIGNOS DE AGRUPACIÓN ---
"("        { return symbol(sym.PAR_A, Tokens.PAR_A); }
")"        { return symbol(sym.PAR_C, Tokens.PAR_C); }
"{"        { return symbol(sym.LLAVE_A, Tokens.LLAVE_A); }
"}"        { return symbol(sym.LLAVE_C, Tokens.LLAVE_C); }
"["        { return symbol(sym.COR_IZQ, Tokens.COR_IZQ); }
"]"        { return symbol(sym.COR_DER, Tokens.COR_DER); }

// --- SIGNOS DE PUNTUACIÓN ---
";"        { return symbol(sym.PTCOMA, Tokens.PUNTO_COMA); }
":"        { return symbol(sym.DOSPTOS, Tokens.DOS_PUNTOS); }
"="        { return symbol(sym.IGUAL, Tokens.ASIGN); }
","        { return symbol(sym.COMA, Tokens.COMA); }
"."        { return symbol(sym.PUNTO, Tokens.PUNTO); }

// --- OPERADORES RELACIONALES Y LÓGICOS ---
"=="       { return symbol(sym.IGUALIGUAL, Tokens.IGUAL); }
"!="       { return symbol(sym.DIFERENTE, Tokens.DIF); }
"<="       { return symbol(sym.MENORIGUAL, Tokens.MENORIGUAL); }
">="       { return symbol(sym.MAYORIGUAL, Tokens.MAYORIGUAL); }
"<"        { return symbol(sym.MENOR, Tokens.MENOR); }
">"        { return symbol(sym.MAYOR, Tokens.MAYOR); }
"||"       { return symbol(sym.OR, Tokens.OR); }
"&&"       { return symbol(sym.AND, Tokens.AND); }
"!"        { return symbol(sym.NOT, Tokens.NOT); }
"^"        { return symbol(sym.XOR, Tokens.XOR); }

// --- VALORES ---
{ENTERO}    { return symbol(sym.ENTERO, yytext(), Tokens.ENTERO); }
{DECIMAL}   { return symbol(sym.DECIMAL, yytext(), Tokens.DECIMAL); }
{ID}        { return symbol(sym.ID, yytext(), Tokens.ID); }

{CADENA}    {
   String val = yytext();
```

```
        // Guardamos las comillas y todo en el reporte
        TokenLista.guardar(Tokens.LIT_STRING, val, yyline+1, yycolumn+1);
        // Al parser le mandamos el valor limpio (sin comillas)
        return new Symbol(sym.LIT_STRING, yyline+1, yycolumn+1, val.substring(1, val.length()-1));
    }

    {CHAR}     {
        String val = yytext();
        TokenLista.guardar(Tokens.LIT_CHAR, val, yyline+1, yycolumn+1);
        return new Symbol(sym.LIT_CHAR, yyline+1, yycolumn+1, val.substring(1, val.length()-1));
    }

    // --- ESPACIOS Y COMENTARIOS ---
    {ESPACIO}          { /* Ignorar */ }
    {COMENTARIO_LINEA}  { /* Ignorar */ }
    {COMENTARIO_BLOQUE} { /* Ignorar */ }
}

// --- ERRORES ---
[^] {
    Errores.agregar("Lexico", "Caracter no reconocido: " + yytext(), yyline+1, yycolumn+1);
    TokenLista.guardar(Tokens.ERROR, yytext(), yyline+1, yycolumn+1); // Guardamos el error también
    System.err.println("Error léxico: " + yytext() + " en línea " + (yyline+1));
}
```