

Nombre de la práctica	ANALIZADOR LEXICO (UNIDAD 4)			No.	4
Asignatura:	LENGUAJES Y AUTÓMATAS I	Carrera:	INGENIERÍA EN SISTEMAS COMPUTACIONALES-3501	Duración de la práctica (Hrs)	10 horas

NOMBRE DEL ALUMNO: Ana Edith Hernández Hernández

GRUPO: 3501

## I. Competencia(s) específica(s):

Construye un analizador léxico a partir de un lenguaje de programación.

Encuadre con CACEI: Registra el (los) atributo(s) de egreso y los criterios de desempeño que se evaluarán en la materia.

No. atributo	Atributos de egreso del PE que impactan en la asignatura	No. Criterio	Criterios de desempeño	No. Indicador	Indicadores
2	El estudiante diseñará esquemas de trabajo y procesos, usando metodologías congruentes en la resolución de problemas de Ingeniería en Sistemas Computacionales	CD1	Identifica metodologías y procesos empleados en la resolución de problemas	I1	Identificación y reconocimiento de distintas metodologías para la resolución de problemas
		CD2	Diseña soluciones a problemas, empleando metodologías apropiadas al área	I1	Uso de metodologías para el modelado de la solución de sistemas y aplicaciones
				I2	Diseño algorítmico (Representación de diagramas de transiciones)
3	El estudiante plantea soluciones basadas en tecnologías empleando su juicio ingenieril para valorar necesidades, recursos y resultados esperados.	CD1	Emplea los conocimientos adquiridos para el desarrollar soluciones	I1	Elección de metodologías, técnicas y/o herramientas para el desarrollo de soluciones
				I2	Uso de metodologías adecuadas para el desarrollo de proyectos
				I3	Generación de productos y/o proyectos
		CD2	Analiza y comprueba resultados	I1	Realizar pruebas a los productos obtenidos
				I2	Documentar información de las pruebas realizadas y los resultados

## II. Lugar de realización de la práctica (laboratorio, taller, aula u otro):

Laboratorio de cómputo y equipo de cómputo personal.

## III. Material empleado:

- Equipo de cómputo
- Software para desarrollo NetBeans

## IV. Desarrollo de la práctica:

### ANALIZADOR LÉXICO

#### DESCRIPCION DEL PROBLEMA

Diseñar un Análisis Léxico de un compilador que traduzca estructuras en español.

#### EXPLICACION DEL CONTENIDO DE LA TABLA DE TOKENS

Una tabla de tokens como la presentada se utiliza en el análisis léxico de un compilador o intérprete para categorizar y reconocer los elementos básicos del lenguaje fuente C. Cada fila define un token (identificador único para una categoría léxica), su correspondiente número de token (un valor entero usado internamente) y su lexema (la representación textual o el significado asociado). Por ejemplo, palabras clave como if y else se mapean a tokens únicos para facilitar su manejo en fases posteriores del análisis. Además, operadores, delimitadores y tipos de datos también se registran, estandarizando su tratamiento en el código fuente. Esto permite que el compilador procese el programa de manera estructurada y eficiente.

#### TABLA DE TOKENS

Token	Token#	Lexema
if	1	"si"
else	2	"no"
switch	3	"lista"
case	4	"coso"
break	5	"terminar"
return	6	"devolver"
while	7	"cuando"
do	8	"hacer"
do while	9	"cuandoHacer"
int	10	"entero"
char	11	"caracter"
string	12	"cadena"
float	13	"decimalRestringido"
double	14	"decimalExacto"
void	15	"vacio"
printf	16	"imprimir"
scan	17	"leer"
puts	18	"texto"
for	19	"para"
else if	20	"sino"
opAritmetico	21	'+'
opAritmetico	22	'-'
opAritmetico	23	'*'
opAritmetico	24	'/'

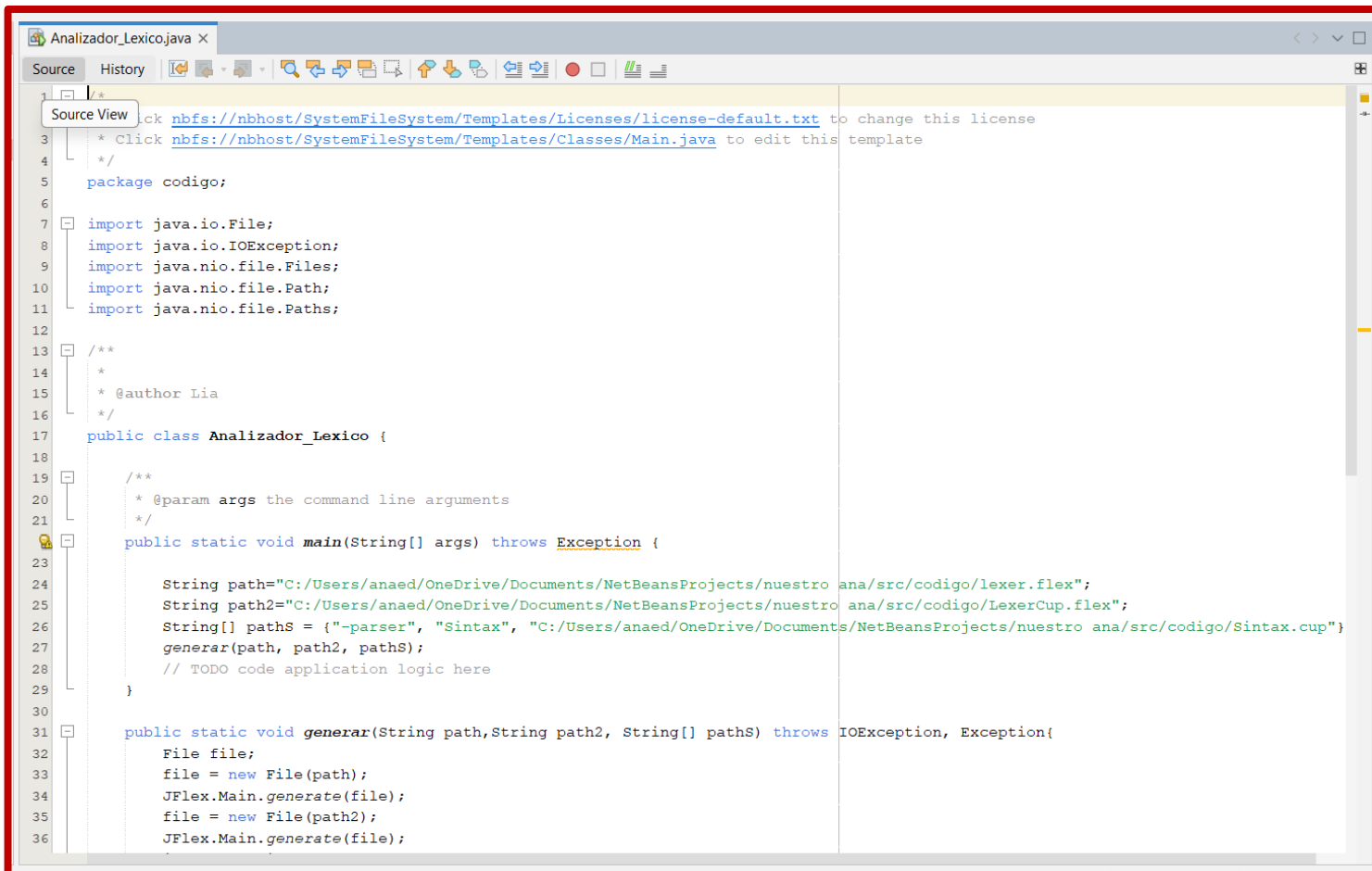


opAritmetico	25	"%"
opAsignacion	26	"="
opAsignacion	27	"+="
opAsignacion	28	"-="
opAsignacion	29	"*="
opAsignacion	30	"/="
opAsignacion	31	"%="
opRelacionales	32	"=="
opRelacionales	33	"!="
opRelacionales	34	">"
opRelacionales	35	"<"
opRelacionales	36	">="
opRelacionales	37	"<="
opLogicos	38	"&&"
opLogicos	39	"  "
opIncremento	40	"++"
opDecremento	41	"--"
parentesisIzq	42	"("
parentesisDer	43	")"
llavelzq	44	"{"
llaveder	45	"}"
corcheteIzq	46	"["
corcheteDer	47	"]"
puntoYcoma	48	","
coma	49	","
dosPuntos	50	":"
punto	51	."
comillaDoble	52	"
comilla	53	'
comentarioLinea	54	"//"
comentarioBloqueIzq	55	"*/"
comentarioBloqueDer	56	"/*"
class	57	"Clase"
true	58	"verdadero"
false	59	"falso"

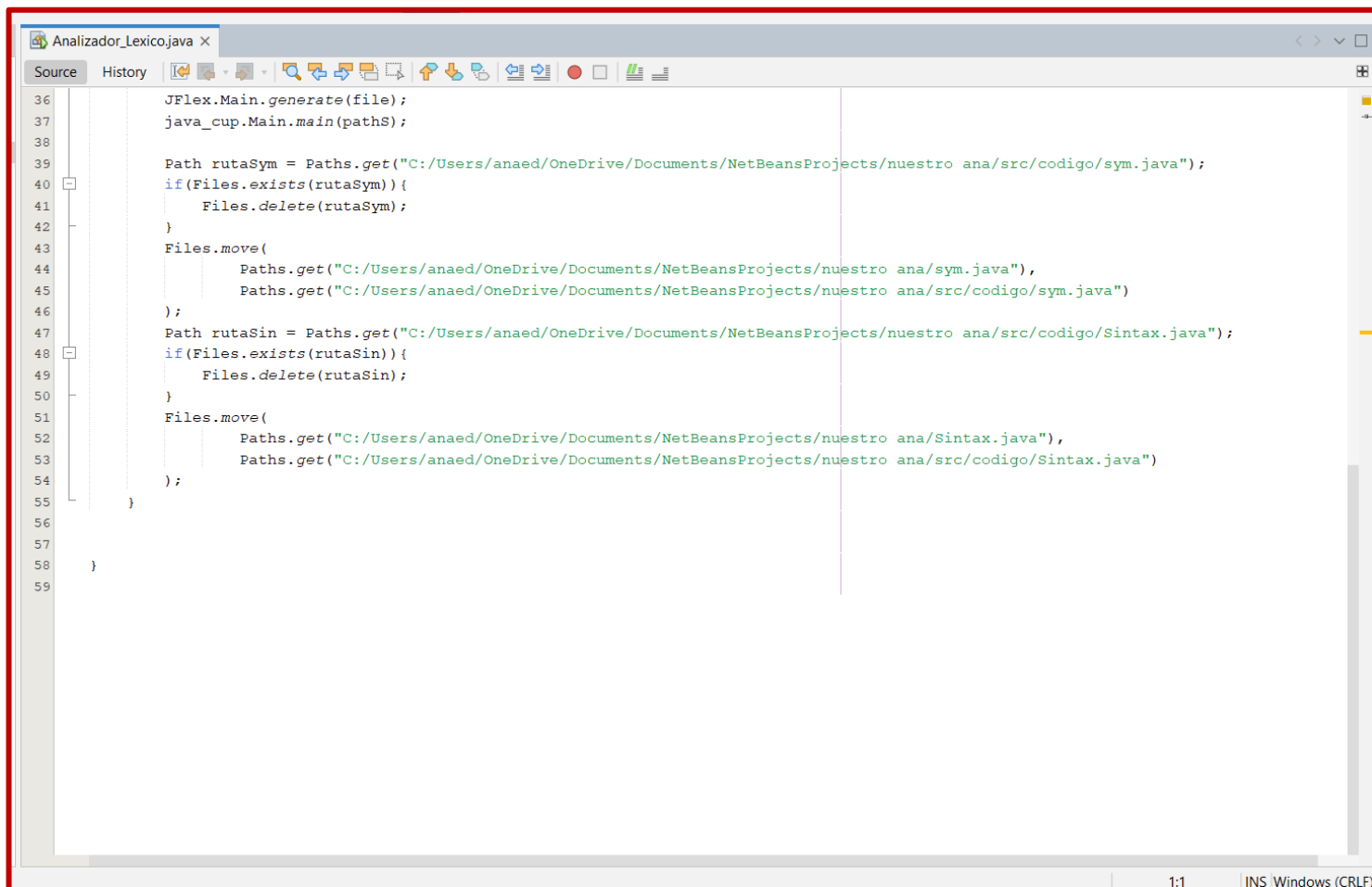
**DESCRIPCION DE CADA UNO DE LOS ARCHIVOS GENERADOS EN EL PROYECTO – (1 A 2 PARRAFOS)  
CONTINUAR COLOCANDO EL CODIGO, COLOCAR UNA LINEA DE SEPARACION ENTRE CADA ARCHIVO  
A EXPLICAR  
(NOTA: COLOCAR BORDES AL CODIGO, Y SE DEBE OBSERVAR LAS LINEAS DE CODIGO, TODO EL  
CODIGO Y LETRA VISIBLE)**

## Analizador\_Lexico

El código en Java implementa un analizador léxico y sintáctico para un proyecto utilizando herramientas como JFlex y CUP. La clase principal Analizador\_Lexico contiene el método main, donde se definen las rutas de dos archivos .flex utilizados por JFlex para generar analizadores léxicos y un archivo .cup que especifica la gramática para el analizador sintáctico. Estos archivos son procesados por el método estático generar, el cual usa JFlex.Main y java\_cup.Main para generar automáticamente los códigos correspondientes. Una vez creados, los archivos generados (sym.java y Sintax.java) son reubicados a una carpeta específica del proyecto (src/codigo), previa eliminación de las versiones existentes para evitar conflictos. Esto permite mantener el código fuente del proyecto organizado y actualizado con los cambios en la gramática o las reglas léxicas definidas. El programa también utiliza las clases Path y Files para manipular archivos y garantizar que las rutas de destino estén correctamente estructuradas.



```
1  package codigo;
2
3  import java.io.File;
4  import java.io.IOException;
5  import java.nio.file.Files;
6  import java.nio.file.Path;
7  import java.nio.file.Paths;
8
9  /**
10   * @author Lia
11   */
12
13  public class Analizador_Lexico {
14
15      /**
16       * @param args the command line arguments
17       */
18      public static void main(String[] args) throws Exception {
19
20          String path="C:/Users/anaed/OneDrive/Documents/NetBeansProjects/nuestro ana/src/codigo/lexer.flex";
21          String path2="C:/Users/anaed/OneDrive/Documents/NetBeansProjects/nuestro ana/src/codigo/LexerCup.flex";
22          String[] pathS = {"-parser", "Syntax", "C:/Users/anaed/OneDrive/Documents/NetBeansProjects/nuestro ana/src/codigo/Syntax.cup"};
23          generar(path, path2, pathS);
24          // TODO code application logic here
25      }
26
27      public static void generar(String path,String path2, String[] pathS) throws IOException, Exception{
28          File file;
29          file = new File(path);
30          JFlex.Main.generate(file);
31          file = new File(path2);
32          JFlex.Main.generate(file);
33      }
34  }
```



```
36 JFlex.Main.generate(file);
37 java_cup.Main.main(pathS);
38
39 Path rutaSym = Paths.get("C:/Users/anaed/OneDrive/Documents/NetBeansProjects/nuestro ana/src/codigo/sym.java");
40 if (Files.exists(rutaSym)) {
41     Files.delete(rutaSym);
42 }
43 Files.move(
44     Paths.get("C:/Users/anaed/OneDrive/Documents/NetBeansProjects/nuestro ana/sym.java"),
45     Paths.get("C:/Users/anaed/OneDrive/Documents/NetBeansProjects/nuestro ana/src/codigo/sym.java")
46 );
47 Path rutaSin = Paths.get("C:/Users/anaed/OneDrive/Documents/NetBeansProjects/nuestro ana/src/codigo/Sintax.java");
48 if (Files.exists(rutaSin)) {
49     Files.delete(rutaSin);
50 }
51 Files.move(
52     Paths.get("C:/Users/anaed/OneDrive/Documents/NetBeansProjects/nuestro ana/Sintax.java"),
53     Paths.get("C:/Users/anaed/OneDrive/Documents/NetBeansProjects/nuestro ana/src/codigo/Sintax.java")
54 );
55 }
56
57
58 }
59
```

## lexer.flex

Este código define las reglas de un analizador léxico utilizando JFlex para identificar los tokens en un lenguaje personalizado. Establece una serie de expresiones regulares para reconocer elementos léxicos como operadores aritméticos, relacionales y lógicos, palabras clave (como si, mientras, clase), identificadores, números, cadenas y símbolos especiales como paréntesis, corchetes y llaves. También incluye reglas para ignorar espacios y tabulaciones, reconocer comentarios de línea y bloque, y manejar errores al identificar caracteres no esperados. Cada token es devuelto con un valor asociado que corresponde a una constante previamente definida en Tokens. Esto facilita la generación del árbol sintáctico en etapas posteriores del análisis.



```
Analizador_Lexico.java x lexer.flex x
Source History
1 package codigo;
2 import static codigo.Tokens.*;
3 %%
4 %class Lexer
5 %type Tokens
6
7 L = [a-zA-Z_]
8 D = [0-9]
9 WHITE = [ \t\r\n]
10
11 %{
12 public String tipo;
13 %}
14
15 %%
16
17 {WHITE} { /* Ignora espacios y tabulaciones */ }
18 "/*.*" { /* Comentario de línea */ return COMENTARIOLINEA; }
19 "/*([^\n]|[\r\n]|\"\"[^\n])*\"/" { /* Comentario de bloque */ return COMENTARIOBLOQUE; }
20
21 "+"|"-"|"*"|" "/"|"^"|"%" { return OPARITMETICO; }
22 "="|"+="|"-="|"*=|" /=" { return OPASIGNACION; }
23 "=="|"!="|">"|<"|>="|<=" { return OPRELACIONALES; }
24 "&&"|"||" { return OPLOGICOS; }
25 "++"|"--" { return OPINCREMENTO; }
26 "verdadero"|"falso" { return BOOLEANO; }
27 "entero" | "caracter" | "cadena" | "decimalLargo" | "decimalCorto" { return TDATO; }
28 "si" { return IF; }
29 "sino" { return ELSEIF; }
30 "no" { return ELSE; }
31 "lista" { return SWITCH; }
32 "caso" { return CASE; }
33 "terminar" { return BREAK; }
34 "devolver" { return RETURN; }
35 "mientras" { return WHILE; }
36 "hacer" { return DO; }
37 "vacio" { return VOID; }
```

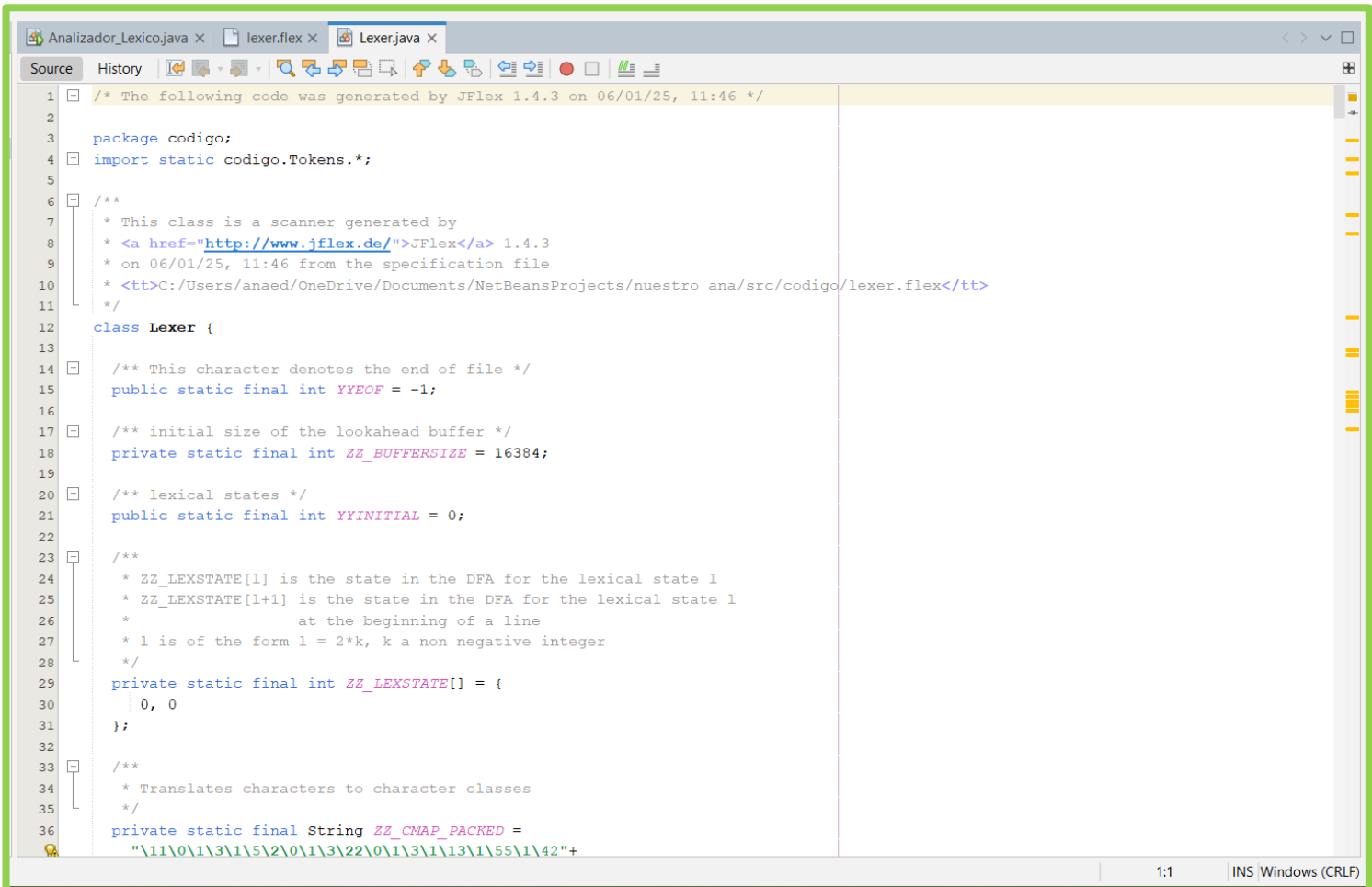


```
Analizador_Lexico.java x lexer.flex x
Source History
37 "vacio" { return VOID; }
38 "imprimir" { return PRINTF; }
39 "para" { return FOR; }
40 "clase" { return CLASS; }
41 "#" {return GATO; }
42 "predeterminado" { return DEFAULT; }
43
44 "(" { return PARENTESISIZQ; }
45 ")" { return PARENTESISDER; }
46 "{" { return LLAVEIZQ; }
47 "}" { return LLAVEDER; }
48 "[" { return CORCHETEIZQ; }
49 "]" { return CORCHETEDER; }
50 ";" { return PUNTOYCOMA; }
51 "," { return COMA; }
52 ":" { return DOSPUNTOS; }
53 "." { return PUNTO; }
54 "\"\" { return COMILLADOBLE; }
55 "'" { return COMILLA; }
56
57 {L}({L}|{D})* { return ID; }
58 ("-"?{D}+"."{D}*)|("-"?(D)+) { return NUM; }
59 \"[^\"]*\" { return CADENA; }
60 \".*\" { return ERROR; }
61
62
```

1:1 | INS | Unix (LF)

## Lexer.java

El Lexer.java se genera mediante el lexer.flex lo que este genera es un autómata finito, con las tablas de transiciones para que pueda reconocer los tokens y los convierta en un lexema y lo pueda reconocer el analizador.



```
1  /* The following code was generated by JFlex 1.4.3 on 06/01/25, 11:46 */
2
3  package codigo;
4  import static codigo.Tokens.*;
5
6  /**
7   * This class is a scanner generated by
8   * <a href="http://www.jflex.de/">JFlex</a> 1.4.3
9   * on 06/01/25, 11:46 from the specification file
10  * <tt>C:/Users/anaed/OneDrive/Documents/NetBeansProjects/nuestro ana/src/codigo/lexer.flex</tt>
11  */
12  class Lexer {
13
14      /** This character denotes the end of file */
15      public static final int YYEOF = -1;
16
17      /** initial size of the lookahead buffer */
18      private static final int ZZ_BUFFER_SIZE = 16384;
19
20      /** lexical states */
21      public static final int YYINITIAL = 0;
22
23      /**
24       * ZZ_LEXSTATE[l] is the state in the DFA for the lexical state l
25       * ZZ_LEXSTATE[l+1] is the state in the DFA for the lexical state l
26       * at the beginning of a line
27       * l is of the form l = 2*k, k a non negative integer
28       */
29      private static final int ZZ_LEXSTATE[] = {
30          0, 0
31      };
32
33      /**
34       * Translates characters to character classes
35       */
36      private static final String ZZ_CMAP_PACKED =
37          "\11\0\13\1\5\2\0\1\3\22\0\1\3\1\13\1\55\1\42"+
```

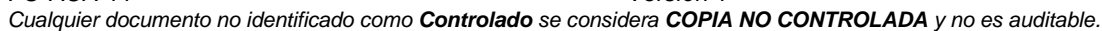




```
Analizador_Lexico.java x lexer.flex x Lexer.java x
Source History
38  "\1\0\1\1\1\1\15\1\56\1\43\1\44\1\6\1\7\1\52\1\10"+
39  "\1\54\1\4\1\2\1\53\1\51\1\14\1\12\1\14\2\0\2\1"+
40  "\1\37\1\0\1\1\35\1\6\1\1\47\1\0\1\50\1\1\1\1\1\0"+
41  "\1\23\1\1\1\32\1\22\1\20\1\25\1\36\1\40\1\33\2\1"+
42  "\1\26\1\34\1\30\1\24\1\41\1\1\1\21\1\27\1\31\1\1"+
43  "\1\17\4\1\1\45\1\16\1\46\uff82\0";
44
45  /**
46   * Translates characters to character classes
47   */
48  private static final char [] ZZ_CMAP = zzUnpackCMap(ZZ_CMAP_PACKED);
49
50  /**
51   * Translates DFA states to action switch labels.
52   */
53  private static final int [] ZZ_ACTION = zzUnpackAction();
54
55  private static final String ZZ_ACTION_PACKED_0 =
56  "\1\0\1\1\1\2\1\3\5\4\1\5\1\0\1\6"+
57  "\2\0\15\1\1\7\1\10\1\11\1\12\1\13\1\14"+
58  "\1\15\1\16\1\17\1\20\1\21\1\22\1\23\1\2"+
59  "\1\24\1\0\1\5\1\25\1\6\1\26\6\1\1\27"+
60  "\1\30\1\0\1\1\31\1\0\1\32\1\0\22\1\1\33"+
61  "\7\1\1\34\3\1\1\35\5\1\1\36\1\1\1\37"+
62  "\3\1\1\40\1\41\3\1\1\42\2\1\1\43\2\1"+
63  "\1\44\17\1\1\45\2\1\1\46\1\47\1\50\12\1"+
64  "\1\51";
65
66  private static int [] zzUnpackAction() {
67      int [] result = new int[153];
68      int offset = 0;
69      offset = zzUnpackAction(ZZ_ACTION_PACKED_0, offset, result);
70      return result;
71  }
72
73  private static int zzUnpackAction(String packed, int offset, int [] result) {
74      int i = 0; /* index in packed string */
```



```
Analizador_Lexico.java x lexer.flex x Lexer.java x
Source History
74     int i = 0;          /* index in packed string */
75     int j = offset;    /* index in unpacked array */
76     int l = packed.length();
77     while (i < l) {
78         int count = packed.charAt(i++);
79         int value = packed.charAt(i++);
80         do result[j++] = value; while (--count > 0);
81     }
82     return j;
83 }
84
85
86 /**
87  * Translates a state to a row index in the transition table
88  */
89 private static final int [] ZZ_ROWMAP = zzUnpackRowMap();
90
91 private static final String ZZ_ROWMAP_PACKED_0 =
92     "\0\0\0\0\57\0\136\0\215\0\274\0\353\0\011a\0\0149"+
93     "\0\215\0\0178\0\0178\0\0178\0\01a7\0\01d6\0\0205\0\0234"+
94     "\0\0263\0\0292\0\02c1\0\02f0\0\031f\0\034e\0\037d\0\03ac"+
95     "\0\03db\0\040a\0\0439\0\215\0\215\0\215\0\215\0\215"+
96     "\0\215\0\215\0\215\0\215\0\215\0\0468\0\0497\0\215"+
97     "\0\04c6\0\04f5\0\0524\0\215\0\215\0\215\0\215\0\0553"+
98     "\0\0582\0\05b1\0\05e0\0\060f\0\063e\0\066d\0\57\0\069c"+
99     "\0\06cb\0\06fa\0\0729\0\0758\0\0787\0\07b6\0\07e5\0\215"+
100    "\0\0497\0\215\0\0814\0\0843\0\0872\0\08a1\0\08d0\0\08ff"+
101    "\0\092e\0\095d\0\098c\0\09bb\0\09ea\0\0a19\0\0a48\0\0a77"+
102    "\0\0aa6\0\0ad5\0\0b04\0\0b33\0\0b62\0\215\0\0b91\0\0bcb"+
103    "\0\0bef\0\0c1e\0\0c4d\0\0c7c\0\0cab\0\57\0\0cda\0\0d09"+
104    "\0\0d38\0\57\0\0d67\0\0d96\0\0dc5\0\0df4\0\0e23\0\57"+
105    "\0\0e52\0\57\0\0e81\0\0eb0\0\0edf\0\57\0\57\0\0f0e"+
106    "\0\0f3d\0\0f6c\0\57\0\0f9b\0\0fca\0\57\0\0ff9\0\01028"+
107    "\0\57\0\01057\0\01086\0\010b5\0\010e4\0\01113\0\01142\0\01171"+
108    "\0\011a0\0\011cf\0\011fe\0\0122d\0\0125c\0\0128b\0\012ba\0\012e9"+
109    "\0\57\0\01318\0\01347\0\57\0\57\0\57\0\01376\0\013a5"+
110    "\0\013d4\0\01403\0\01432\0\01461\0\01490\0\014bf\0\014ee\0\0151d"+
```





```
Analizador_Lexico.java x lexer.flex x Lexer.java x
Source History
146 "\"61\\0\\1\\57\\57\\0\\1\\57\\41\\0\\2\\2\\14\\0\\1\\2\""+
147 "\"1\\60\\2\\2\\1\\61\\16\\2\\16\\0\\2\\2\\14\\0\\1\\2\""+
148 "\"1\\62\\1\\2\\16\\0\\2\\2\\14\\0\\1\\2\\1\\63\\2\\1\\2\""+
149 "\"1\\6\\0\\2\\2\\14\\0\\4\\2\\1\\64\\16\\2\\16\\0\\2\\2\""+
150 "\"14\\0\\14\\2\\1\\65\\6\\2\\16\\0\\2\\2\\14\\0\\14\\2\""+
151 "\"1\\66\\6\\2\\16\\0\\2\\2\\14\\0\\5\\2\\1\\67\\15\\2\""+
152 "\"1\\6\\0\\2\\2\\14\\0\\3\\2\\1\\70\\2\\1\\2\\16\\0\\2\\2\""+
153 "\"14\\0\\4\\2\\1\\71\\2\\2\\1\\72\\13\\2\\16\\0\\2\\2\""+
154 "\"14\\0\\15\\2\\1\\73\\5\\2\\16\\0\\2\\2\\14\\0\\14\\2\""+
155 "\"1\\74\\6\\2\\16\\0\\2\\2\\14\\0\\4\\2\\1\\75\\16\\2\""+
156 "\"1\\6\\0\\2\\2\\14\\0\\2\\2\\1\\76\\1\\2\\1\\77\\16\\2\""+
157 "\"23\\0\\1\\100\\50\\0\\55\\101\\1\\102\\1\\101\\2\\0\\1\\51\""+
158 "\"54\\0\\5\\52\\1\\0\\51\\52\\6\\53\\1\\103\\50\\53\\1\\0\""+
159 "\"2\\2\\14\\0\\2\\2\\1\\104\\20\\2\\16\\0\\2\\2\\14\\0\""+
160 "\"13\\2\\1\\105\\7\\2\\16\\0\\2\\2\\14\\0\\12\\2\\1\\106\""+
161 "\"10\\2\\16\\0\\2\\2\\14\\0\\1\\107\\12\\2\\1\\110\\7\\2\""+
162 "\"16\\0\\2\\2\\14\\0\\7\\2\\1\\111\\13\\2\\16\\0\\2\\2\""+
163 "\"14\\0\\10\\2\\1\\112\\12\\2\\16\\0\\2\\2\\14\\0\\11\\2\""+
164 "\"1\\113\\11\\2\\16\\0\\2\\2\\14\\0\\2\\2\\1\\114\\20\\2\""+
165 "\"16\\0\\2\\2\\14\\0\\2\\2\\1\\115\\1\\116\\4\\2\\1\\117\""+
166 "\"12\\2\\16\\0\\2\\2\\14\\0\\4\\2\\1\\120\\16\\2\\16\\0\""+
167 "\"2\\2\\14\\0\\22\\2\\1\\121\\16\\0\\2\\2\\14\\0\\1\\2\""+
168 "\"1\\122\\21\\2\\16\\0\\2\\2\\14\\0\\13\\2\\1\\123\\7\\2\""+
169 "\"16\\0\\2\\2\\14\\0\\1\\2\\1\\124\\21\\2\\16\\0\\2\\2\""+
170 "\"14\\0\\2\\2\\1\\125\\20\\2\\15\\0\\4\\53\\1\\126\\52\\53\""+
171 "\"1\\0\\2\\2\\14\\0\\3\\2\\1\\127\\17\\2\\16\\0\\2\\2\""+
172 "\"14\\0\\14\\2\\1\\130\\6\\2\\16\\0\\2\\2\\14\\0\\1\\2\""+
173 "\"1\\131\\21\\2\\16\\0\\2\\2\\14\\0\\5\\2\\1\\132\\15\\2\""+
174 "\"16\\0\\2\\2\\14\\0\\14\\2\\1\\133\\6\\2\\16\\0\\2\\2\""+
175 "\"14\\0\\10\\2\\1\\134\\12\\2\\16\\0\\2\\2\\14\\0\\12\\2\""+
176 "\"1\\135\\10\\2\\16\\0\\2\\2\\14\\0\\5\\2\\1\\136\\15\\2\""+
177 "\"16\\0\\2\\2\\14\\0\\15\\2\\1\\137\\5\\2\\16\\0\\2\\2\""+
178 "\"14\\0\\4\\2\\1\\140\\16\\2\\16\\0\\2\\2\\14\\0\\1\\2\""+
179 "\"1\\141\\21\\2\\16\\0\\2\\2\\14\\0\\5\\2\\1\\142\\15\\2\""+
180 "\"16\\0\\2\\2\\14\\0\\10\\2\\1\\143\\12\\2\\16\\0\\2\\2\""+
181 "\"14\\0\\2\\2\\1\\144\\20\\2\\16\\0\\2\\2\\14\\0\\11\\2\""+
182 "\"1\\145\\11\\2\\16\\0\\2\\2\\14\\0\\1\\2\\1\\146\\21\\2\""+
```



```
Analizador_Lexico.java x lexer.flex x Lexer.java x
Source History
183 "\16\0\2\2\14\0\3\2\1\147\17\2\16\0\2\2"+
184 "\14\0\4\2\1\150\16\2\16\0\2\2\14\0\4\2"+
185 "\1\151\16\2\16\0\2\2\14\0\5\2\1\152\15\2"+
186 "\16\0\2\2\14\0\2\2\1\153\20\2\16\0\2\2"+
187 "\14\0\7\2\1\154\13\2\16\0\2\2\14\0\15\2"+
188 "\1\155\5\2\16\0\2\2\14\0\5\2\1\156\15\2"+
189 "\16\0\2\2\14\0\4\2\1\157\16\2\16\0\2\2"+
190 "\14\0\14\2\1\160\6\2\16\0\2\2\14\0\13\2"+
191 "\1\161\7\2\16\0\2\2\14\0\11\2\1\162\11\2"+
192 "\16\0\2\2\14\0\1\2\1\163\21\2\16\0\2\2"+
193 "\14\0\14\2\1\164\6\2\16\0\2\2\14\0\12\2"+
194 "\1\165\10\2\16\0\2\2\14\0\2\2\1\166\20\2"+
195 "\16\0\2\2\14\0\1\2\1\167\21\2\16\0\2\2"+
196 "\14\0\3\2\1\170\17\2\16\0\2\2\14\0\5\2"+
197 "\1\171\15\2\16\0\2\2\14\0\1\172\22\2\16\0"+
198 "\2\2\14\0\4\2\1\173\16\2\16\0\2\2\14\0"+
199 "\11\2\1\174\11\2\16\0\2\2\14\0\12\2\1\175"+
200 "\10\2\16\0\2\2\14\0\4\2\1\171\16\2\16\0"+
201 "\2\2\14\0\15\2\1\176\5\2\16\0\2\2\14\0"+
202 "\2\2\1\177\20\2\16\0\2\2\14\0\12\2\1\200"+
203 "\10\2\16\0\2\2\14\0\1\2\1\201\21\2\16\0"+
204 "\2\2\14\0\1\2\1\202\21\2\16\0\2\2\14\0"+
205 "\7\2\1\203\13\2\16\0\2\2\14\0\4\2\1\204"+
206 "\16\2\16\0\2\2\14\0\1\2\1\205\21\2\16\0"+
207 "\2\2\14\0\14\2\1\206\6\2\16\0\2\2\14\0"+
208 "\4\2\1\207\16\2\16\0\2\2\14\0\1\2\1\210"+
209 "\21\2\16\0\2\2\14\0\2\2\1\134\20\2\16\0"+
210 "\2\2\14\0\2\2\1\211\20\2\16\0\2\2\14\0"+
211 "\16\2\1\212\1\2\1\213\2\2\16\0\2\2\14\0"+
212 "\2\2\1\214\20\2\16\0\2\2\14\0\2\2\1\171"+
213 "\20\2\16\0\2\2\14\0\2\2\1\215\20\2\16\0"+
214 "\2\2\14\0\10\2\1\216\12\2\16\0\2\2\14\0"+
215 "\2\2\1\217\20\2\16\0\2\2\14\0\4\2\1\220"+
216 "\16\2\16\0\2\2\14\0\5\2\1\221\15\2\16\0"+
217 "\2\2\14\0\15\2\1\222\5\2\16\0\2\2\14\0"+
218 "\2\2\1\223\20\2\16\0\2\2\14\0\2\2\1\224"+
219 "\20\2\16\0\2\2\14\0\14\2\1\225\6\2\16\0"+
```



```
Analizador_Lexico.java x lexer.flex x Lexer.java x
Source History
220 "\2\2\14\0\17\2\1\153\3\2\16\0\2\2\14\0"+
221 "\12\2\1\153\10\2\16\0\2\2\14\0\11\2\1\226"+
222 "\11\2\16\0\2\2\14\0\4\2\1\227\16\2\16\0"+
223 "\2\2\14\0\3\2\1\230\17\2\16\0\2\2\14\0"+
224 "\5\2\1\231\15\2\15\0";
225
226 private static int [] zzUnpackTrans() {
227     int [] result = new int[5452];
228     int offset = 0;
229     offset = zzUnpackTrans(zzTransPacked0, offset, result);
230     return result;
231 }
232
233 private static int zzUnpackTrans(String packed, int offset, int [] result) {
234     int i = 0; /* index in packed string */
235     int j = offset; /* index in unpacked array */
236     int l = packed.length();
237     while (i < l) {
238         int count = packed.charAt(i++);
239         int value = packed.charAt(i++);
240         value--;
241         do result[j++] = value; while (--count > 0);
242     }
243     return j;
244 }
245
246
247 /* error codes */
248 private static final int zz_UNKNOWN_ERROR = 0;
249 private static final int zz_NO_MATCH = 1;
250 private static final int zz_PUSHBACK_2BIG = 2;
251
252 /* error messages for the codes above */
253 private static final String zz_ERROR_MSG[] = {
254     "Unkown internal scanner error",
255     "Error: could not match input",
256     "Error: pushback value was too large"
257 }
```



```
Analizador_Lexico.java x lexer.flex x Lexer.java x
Source History
257 };
258
259 /**
260  * ZZ_ATTRIBUTE[aState] contains the attributes of state <code>aState</code>
261  */
262 private static final int [] ZZ_ATTRIBUTE = zzUnpackAttribute();
263
264 private static final String ZZ_ATTRIBUTE_PACKED_0 =
265     "\1\0\2\1\1\11\4\1\1\11\1\1\1\0\1\1"+
266     "\2\0\15\1\12\11\2\1\11\2\1\1\0\4\11"+
267     "\20\1\1\11\1\0\1\11\1\0\22\1\1\11\103\1";
268
269 private static int [] zzUnpackAttribute() {
270     int [] result = new int[153];
271     int offset = 0;
272     offset = zzUnpackAttribute(ZZ_ATTRIBUTE_PACKED_0, offset, result);
273     return result;
274 }
275
276 private static int zzUnpackAttribute(String packed, int offset, int [] result) {
277     int i = 0; /* index in packed string */
278     int j = offset; /* index in unpacked array */
279     int l = packed.length();
280     while (i < l) {
281         int count = packed.charAt(i++);
282         int value = packed.charAt(i++);
283         do result[j++] = value; while (--count > 0);
284     }
285     return j;
286 }
287
288 /** the input device */
289 private java.io.Reader zzReader;
290
291 /** the current state of the DFA */
292 private int zzState;
293
```



```
Analizador_Lexico.java x lexer.flex x Lexer.java x
Source History
294 /** the current lexical state */
295 private int zzLexicalState = YYINITIAL;
296
297 /** this buffer contains the current text to be matched and is
298  * the source of the yytext() string */
299 private char zzBuffer[] = new char[ZZ_BUFFER_SIZE];
300
301 /** the textposition at the last accepting state */
302 private int zzMarkedPos;
303
304 /** the current text position in the buffer */
305 private int zzCurrentPos;
306
307 /** startRead marks the beginning of the yytext() string in the buffer */
308 private int zzStartRead;
309
310 /** endRead marks the last character in the buffer, that has been read
311  * from input */
312 private int zzEndRead;
313
314 /** number of newlines encountered up to the start of the matched text */
315 private int yyline;
316
317 /** the number of characters up to the start of the matched text */
318 private int yychar;
319
320 /**
321  * the number of characters from the last newline up to the start of the
322  * matched text
323  */
324 private int yycolumn;
325
326 /**
327  * zzAtBOL == true <=> the scanner is currently at the beginning of a line
328  */
329 private boolean zzAtBOL = true;
330
```





```
Analizador_Lexico.java x lexer.flex x Lexer.java x
Source History
331 /** zzAtEOF == true <=> the scanner is at the EOF */
332 private boolean zzAtEOF;
333
334 /** denotes if the user-EOF-code has already been executed */
335 private boolean zzEOFDone;
336
337
338 /**
339  * Creates a new scanner
340  * There is also a java.io.InputStream version of this constructor.
341  *
342  * @param in the java.io.Reader to read input from.
343  */
344 Lexer(java.io.Reader in) {
345     this.zzReader = in;
346 }
347
348 /**
349  * Creates a new scanner.
350  * There is also java.io.Reader version of this constructor.
351  *
352  * @param in the java.io.InputStream to read input from.
353  */
354 Lexer(java.io.InputStream in) {
355     this(new java.io.InputStreamReader(in));
356 }
357
358 /**
359  * Unpacks the compressed character translation table.
360  *
361  * @param packed the packed character translation table
362  * @return the unpacked character translation table
363  */
364 private static char [] zzUnpackCMap(String packed) {
365     char [] map = new char[0x10000];
366     int i = 0; /* index in packed string */
367     int j = 0; /* index in unpacked array */
```



```
Analizador_Lexico.java x lexer.flex x Lexer.java x
Source History
368 while (i < 132) {
369     int count = packed.charAt(i++);
370     char value = packed.charAt(i++);
371     do map[j++] = value; while (--count > 0);
372 }
373 return map;
374 }
375
376
377 /**
378  * Refills the input buffer.
379  *
380  * @return <code>false</code>, iff there was new input.
381  *
382  * @exception java.io.IOException if any I/O-Error occurs
383  */
384 private boolean zzRefill() throws java.io.IOException {
385
386     /* first: make room (if you can) */
387     if (zzStartRead > 0) {
388         System.arraycopy(zzBuffer, zzStartRead,
389             zzBuffer, 0,
390             zzEndRead-zzStartRead);
391
392         /* translate stored positions */
393         zzEndRead-= zzStartRead;
394         zzCurrentPos-= zzStartRead;
395         zzMarkedPos-= zzStartRead;
396         zzStartRead = 0;
397     }
398
399     /* is the buffer big enough? */
400     if (zzCurrentPos >= zzBuffer.length) {
401         /* if not: blow it up */
402         char newBuffer[] = new char[zzCurrentPos*2];
403         System.arraycopy(zzBuffer, 0, newBuffer, 0, zzBuffer.length);
404         zzBuffer = newBuffer;
```



```
Analizador_Lexico.java x lexer.flex x Lexer.java x
Source History
404     zzBuffer = newBuffer;
405 }
406
407 /* finally: fill the buffer with new input */
408 int numRead = zzReader.read(zzBuffer, zzEndRead,
409                             zzBuffer.length-zzEndRead);
410
411 if (numRead > 0) {
412     zzEndRead+= numRead;
413     return false;
414 }
415 // unlikely but not impossible: read 0 characters, but not at end of stream
416 if (numRead == 0) {
417     int c = zzReader.read();
418     if (c == -1) {
419         return true;
420     } else {
421         zzBuffer[zzEndRead++] = (char) c;
422         return false;
423     }
424 }
425
426 // numRead < 0
427 return true;
428 }
429
430
431 /**
432  * Closes the input stream.
433  */
434 public final void yyclose() throws java.io.IOException {
435     zzAtEOF = true; /* indicate end of file */
436     zzEndRead = zzStartRead; /* invalidate buffer */
437
438     if (zzReader != null)
439         zzReader.close();
440 }
```

1:1 | INS Windows (CRLF)



```
Analizador_Lexico.java x lexer.flex x Lexer.java x
Source History
440 }
441
442
443 /**
444  * Resets the scanner to read from a new input stream.
445  * Does not close the old reader.
446  *
447  * All internal variables are reset, the old input stream
448  * <b>cannot</b> be reused (internal buffer is discarded and lost).
449  * Lexical state is set to <tt>ZZ_INITIAL</tt>.
450  *
451  * @param reader the new input stream
452  */
453 public final void yyreset(java.io.Reader reader) {
454     zzReader = reader;
455     zzAtBOL = true;
456     zzAtEOF = false;
457     zzEOFDone = false;
458     zzEndRead = zzStartRead = 0;
459     zzCurrentPos = zzMarkedPos = 0;
460     yyline = yychar = yycolumn = 0;
461     zzLexicalState = YYINITIAL;
462 }
463
464
465 /**
466  * Returns the current lexical state.
467  */
468 public final int yystate() {
469     return zzLexicalState;
470 }
471
472
473 /**
474  * Enters a new lexical state
475  *
476  * @param newState the new lexical state
```



```
Analizador_Lexico.java x lexer.flex x Lexer.java x
Source History
477 */
478 public final void yybegin(int newState) {
479     zzLexicalState = newState;
480 }
481
482
483 /**
484  * Returns the text matched by the current regular expression.
485  */
486 public final String yytext() {
487     return new String( zzBuffer, zzStartRead, zzMarkedPos-zzStartRead );
488 }
489
490
491 /**
492  * Returns the character at position <tt>pos</tt> from the
493  * matched text.
494  *
495  * It is equivalent to yytext().charAt(pos), but faster
496  *
497  * @param pos the position of the character to fetch.
498  *           A value from 0 to yylength()-1.
499  *
500  * @return the character at position pos
501  */
502 public final char yycharat(int pos) {
503     return zzBuffer[zzStartRead+pos];
504 }
505
506
507 /**
508  * Returns the length of the matched text region.
509  */
510 public final int yylength() {
511     return zzMarkedPos-zzStartRead;
512 }
513
```



```
Analizador_Lexico.java x lexer.flex x Lexer.java x
Source History
514
515 /**
516  * Reports an error that occurred while scanning.
517  *
518  * In a wellformed scanner (no or only correct usage of
519  * yypushback(int) and a match-all fallback rule) this method
520  * will only be called with things that "Can't Possibly Happen".
521  * If this method is called, something is seriously wrong
522  * (e.g. a JFlex bug producing a faulty scanner etc.).
523  *
524  * Usual syntax/scanner level error handling should be done
525  * in error fallback rules.
526  *
527  * @param errorCode the code of the error message to display
528  */
529 private void zzScanError(int errorCode) {
530     String message;
531     try {
532         message = ZZ_ERROR_MSG[errorCode];
533     }
534     catch (ArrayIndexOutOfBoundsException e) {
535         message = ZZ_ERROR_MSG[ZZ_UNKNOWN_ERROR];
536     }
537
538     throw new Error(message);
539 }
540
541
542 /**
543  * Pushes the specified amount of characters back into the input stream.
544  *
545  * They will be read again by then next call of the scanning method
546  *
547  * @param number the number of characters to be read again.
548  *               This number must not be greater than yylength()!
549  */
550 public void yypushback(int number) {
```

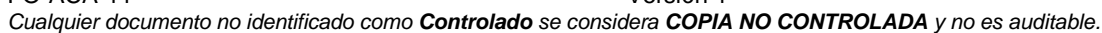


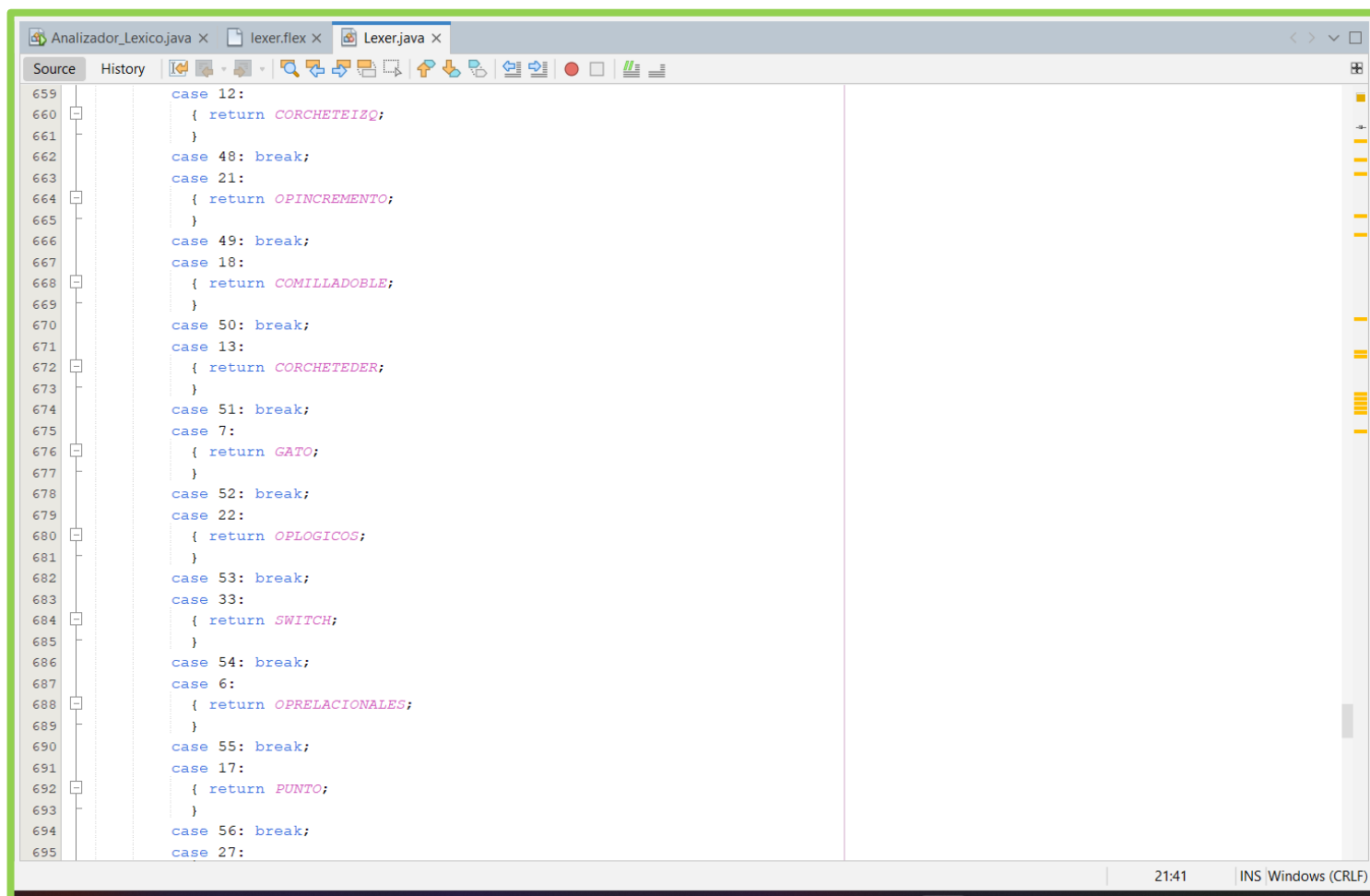
```
Analizador_Lexico.java x lexer.flex x Lexer.java x
Source History
550 public void yypushback(int number) {
551     if ( number > yylength() )
552         zzScanError(ZZ_PUSHBACK_2BIG);
553
554     zzMarkedPos -= number;
555 }
556
557
558 /**
559  * Resumes scanning until the next regular expression is matched,
560  * the end of input is encountered or an I/O-Error occurs.
561  *
562  * @return     the next token
563  * @exception  java.io.IOException if any I/O-Error occurs
564  */
565 public Tokens yylex() throws java.io.IOException {
566     int zzInput;
567     int zzAction;
568
569     // cached fields:
570     int zzCurrentPosL;
571     int zzMarkedPosL;
572     int zzEndReadL = zzEndRead;
573     char [] zzBufferL = zzBuffer;
574     char [] zzCMapL = ZZ_CMAP;
575
576     int [] zzTransL = ZZ_TRANS;
577     int [] zzRowMapL = ZZ_ROWMAP;
578     int [] zzAttrL = ZZ_ATTRIBUTE;
579
580     while (true) {
581         zzMarkedPosL = zzMarkedPos;
582
583         zzAction = -1;
584
585         zzCurrentPosL = zzCurrentPos = zzStartRead = zzMarkedPosL;
586     }
```



```
Analizador_Lexico.java x lexer.flex x Lexer.java x
Source History
587 zzState = ZZ_LEXSTATE[zzLexicalState];
588
589
590 zzForAction: {
591     while (true) {
592
593         if (zzCurrentPosL < zzEndReadL)
594             zzInput = zzBufferL[zzCurrentPosL++];
595         else if (zzAtEOF) {
596             zzInput = YYEOF;
597             break zzForAction;
598         }
599         else {
600             // store back cached positions
601             zzCurrentPos = zzCurrentPosL;
602             zzMarkedPos = zzMarkedPosL;
603             boolean eof = zzRefill();
604             // get translated positions and possibly new buffer
605             zzCurrentPosL = zzCurrentPos;
606             zzMarkedPosL = zzMarkedPos;
607             zzBufferL = zzBuffer;
608             zzEndReadL = zzEndRead;
609             if (eof) {
610                 zzInput = YYEOF;
611                 break zzForAction;
612             }
613             else {
614                 zzInput = zzBufferL[zzCurrentPosL++];
615             }
616         }
617         int zzNext = zzTransL[ zzRowMapL[zzState] + zzCMapL[zzInput] ];
618         if (zzNext == -1) break zzForAction;
619         zzState = zzNext;
620
621         int zzAttributes = zzAttrL[zzState];
622         if ( (zzAttributes & 1) == 1 ) {
623             zzAction = zzState;
```







```
659 case 12:
660     { return CORCHETEIZQ;
661     }
662 case 48: break;
663 case 21:
664     { return OPINCREMENTO;
665     }
666 case 49: break;
667 case 18:
668     { return COMILLADOBLE;
669     }
670 case 50: break;
671 case 13:
672     { return CORCHETEDER;
673     }
674 case 51: break;
675 case 7:
676     { return GATO;
677     }
678 case 52: break;
679 case 22:
680     { return OPLOGICOS;
681     }
682 case 53: break;
683 case 33:
684     { return SWITCH;
685     }
686 case 54: break;
687 case 6:
688     { return OPRELACIONALES;
689     }
690 case 55: break;
691 case 17:
692     { return PUNTO;
693     }
694 case 56: break;
695 case 27:
```



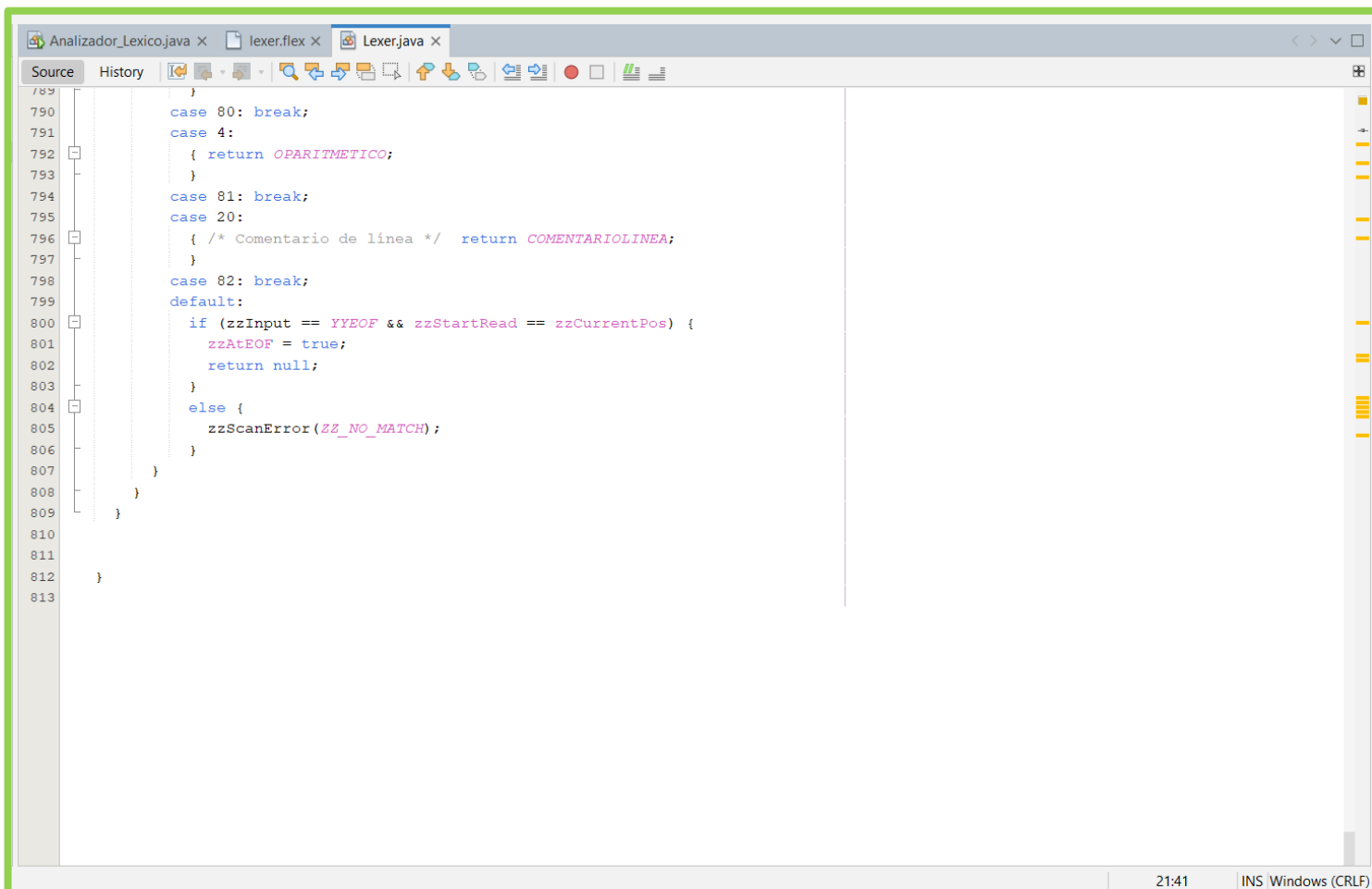
```
Analizador_Lexico.java x lexer.flex x Lexer.java x
Source History
696 { /* Comentario de bloque */ return COMENTARIOBLOQUE;
697 }
698 case 57: break;
699 case 15:
700 { return COMA;
701 }
702 case 58: break;
703 case 10:
704 { return LLAVEIZQ;
705 }
706 case 59: break;
707 case 11:
708 { return LLAVEDER;
709 }
710 case 60: break;
711 case 28:
712 { return ELSEIF;
713 }
714 case 61: break;
715 case 35:
716 { return DO;
717 }
718 case 62: break;
719 case 1:
720 { return ID;
721 }
722 case 63: break;
723 case 34:
724 { return CLASS;
725 }
726 case 64: break;
727 case 5:
728 { return OPASIGNACION;
729 }
730 case 65: break;
731 case 24:
732 { return ELSE;
```



```
732 { return ELSE;
733 }
734 case 66: break;
735 case 37:
736 { return RETURN;
737 }
738 case 67: break;
739 case 40:
740 { return WHILE;
741 }
742 case 68: break;
743 case 29:
744 { return CASE;
745 }
746 case 69: break;
747 case 19:
748 { return COMILLA;
749 }
750 case 70: break;
751 case 31:
752 { return VOID;
753 }
754 case 71: break;
755 case 3:
756 { /* Ignora espacios y tabulaciones */
757 }
758 case 72: break;
759 case 36:
760 { return TDATO;
761 }
762 case 73: break;
763 case 25:
764 { return ERROR;
765 }
766 case 74: break;
767 case 30:
768 { return FOR;
```



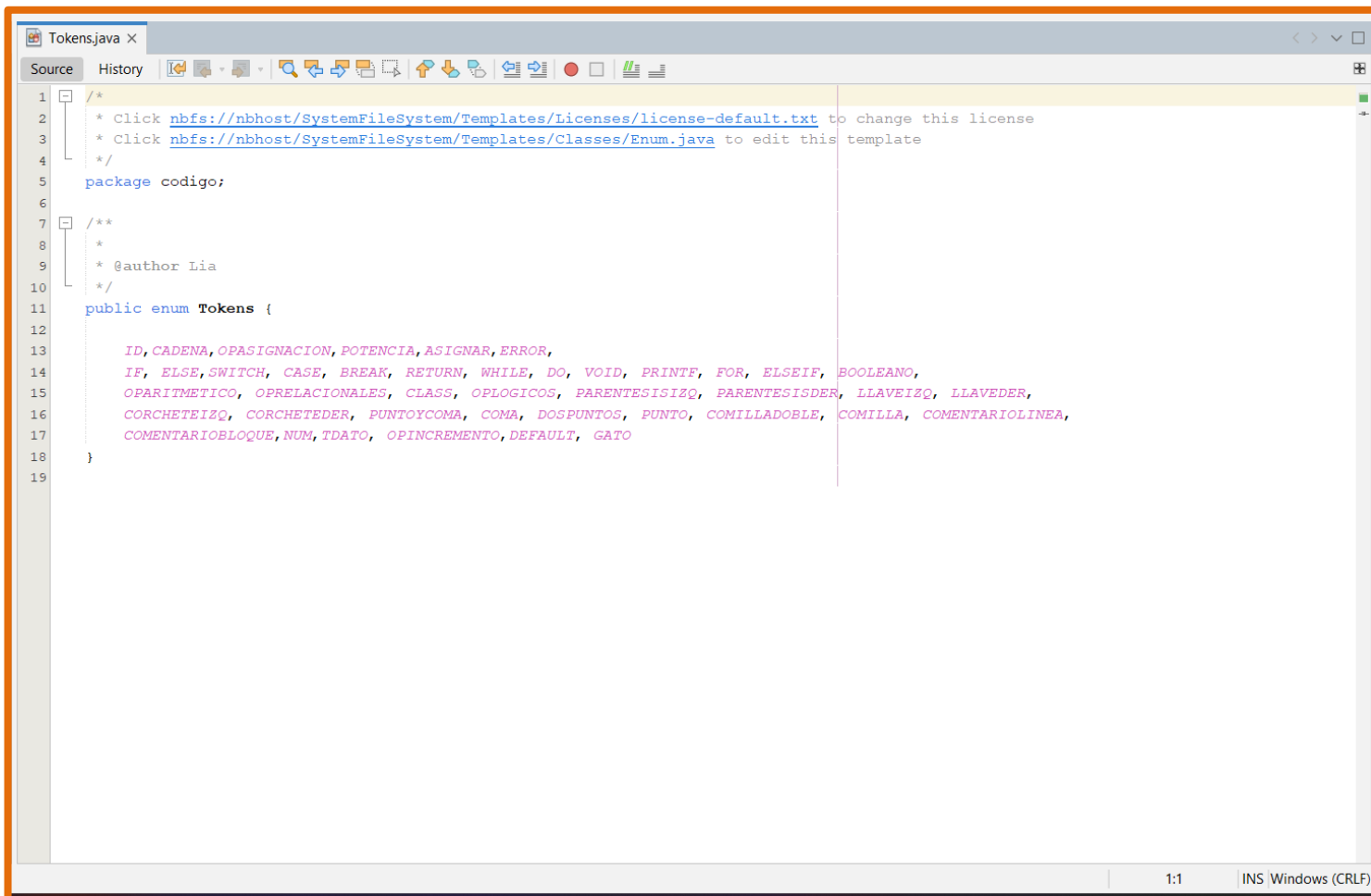
```
769     }
770     case 75: break;
771     case 38:
772     { return BREAK;
773     }
774     case 76: break;
775     case 41:
776     { return DEFAULT;
777     }
778     case 77: break;
779     case 14:
780     { return PUNTOYCOMA;
781     }
782     case 78: break;
783     case 39:
784     { return PRINTF;
785     }
786     case 79: break;
787     case 16:
788     { return DOSPUNTOS;
789     }
790     case 80: break;
791     case 4:
792     { return OPARITMETICO;
793     }
794     case 81: break;
795     case 20:
796     { /* Comentario de línea */ return COMENTARIOLINEA;
797     }
798     case 82: break;
799     default:
800     if (zzInput == YYEOF && zzStartRead == zzCurrentPos) {
801         zzAtEOF = true;
802         return null;
803     }
804     else {
805         zzScanError(ZZ_NO_MATCH);
```



```
789  
790     case 80: break;  
791     case 4:  
792         { return OPARITMETICO;  
793         }  
794     case 81: break;  
795     case 20:  
796         { /* Comentario de línea */ return COMENTARIOLINEA;  
797         }  
798     case 82: break;  
799     default:  
800         if (zzInput == YYEOF && zzStartRead == zzCurrentPos) {  
801             zzAtEOF = true;  
802             return null;  
803         }  
804         else {  
805             zzScanError(ZZ_NO_MATCH);  
806         }  
807     }  
808 }  
809 }  
810  
811 }  
812  
813 }
```

## Tokens.java

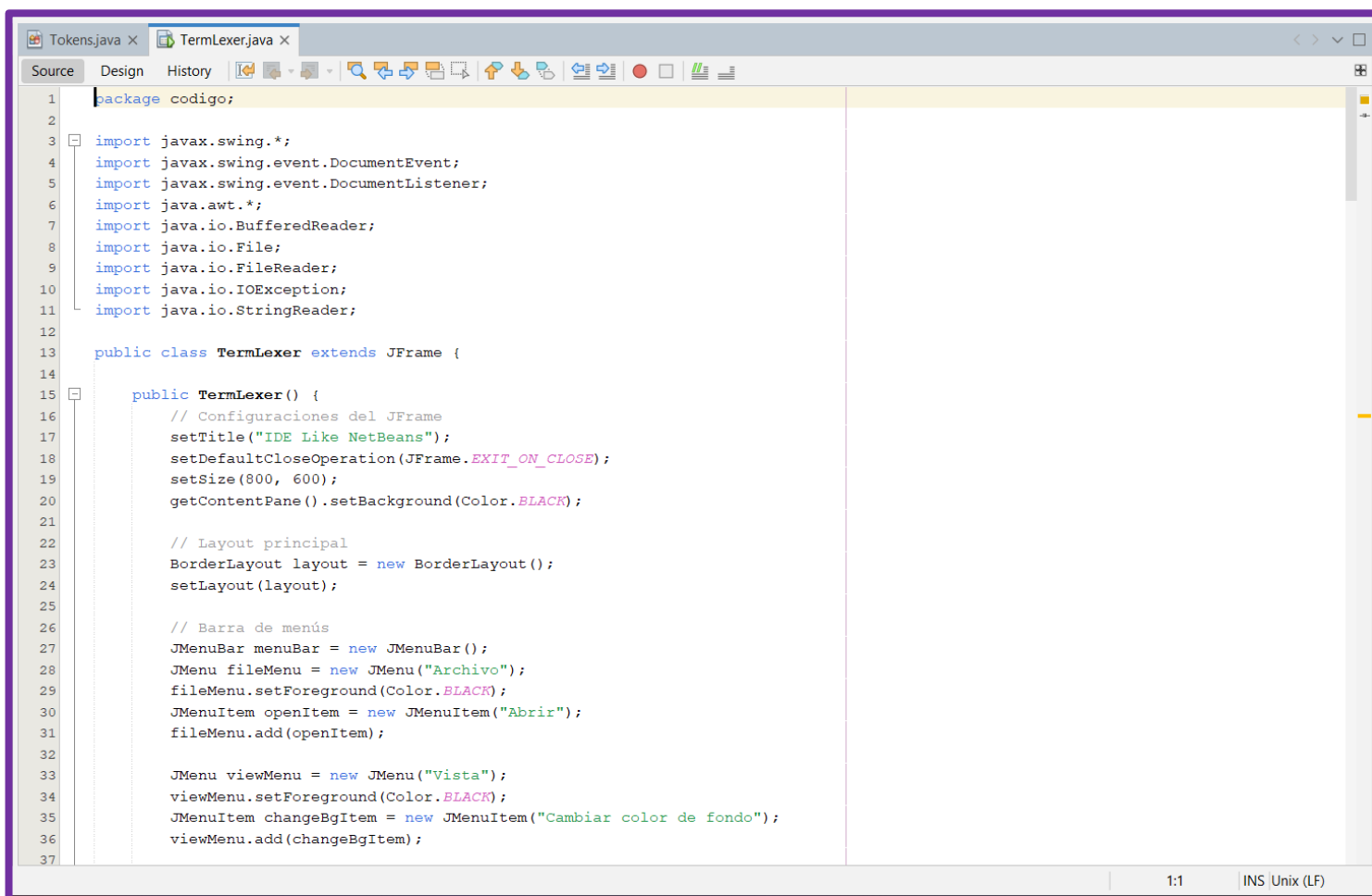
Este código define una enumeración llamada Tokens en Java, que lista los tipos de tokens reconocidos por un analizador léxico. Incluye identificadores (ID), cadenas (CADENA), operadores de asignación, relacionales, aritméticos y lógicos, palabras clave (como IF, WHILE, CLASS), símbolos especiales (paréntesis, llaves, corchetes, punto y coma), tipos de datos (TDATO), incrementos, comentarios, números y errores. Esta enumeración facilita la categorización y manejo de los elementos léxicos durante el análisis de un lenguaje.



```
1  /*
2  * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
3  * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Enum.java to edit this template
4  */
5  package codigo;
6
7  /**
8   *
9   * @author Lia
10  */
11  public enum Tokens {
12
13      ID, CADENA, OPASIGNACION, POTENCIA, ASIGNAR, ERROR,
14      IF, ELSE, SWITCH, CASE, BREAK, RETURN, WHILE, DO, VOID, PRINTF, FOR, ELSEIF, BOOLEANO,
15      OPARITMETICO, OPRELACIONALES, CLASS, OPLOGICOS, PARENTESISIZQ, PARENTESISDER, LLAVEIZQ, LLAVEDER,
16      CORCHETEIZQ, CORCHETEDER, PUNTOYCOMA, COMA, DOSPUNTOS, PUNTO, COMILLADOBLE, COMILLA, COMENTARIOLINEA,
17      COMENTARIOBLOQUE, NUM, TDATO, OPINCREMENTO, DEFAULT, GATO
18  }
19
```

## TermLexer.java

El código define una aplicación de escritorio en Java que emula un entorno de desarrollo integrado (IDE) para análisis léxico. Utilizando la biblioteca Swing, la interfaz incluye un área de edición de texto con numeración de líneas, una consola para mostrar los resultados del análisis, y una barra de menús para abrir archivos o cambiar el color de fondo. También hay botones para ejecutar análisis léxico y sintáctico, y un cuadro de diálogo de ayuda. El análisis léxico procesa el texto ingresado, generando una lista de tokens y sus lexemas, que se muestran en la consola. Los elementos visuales, como botones y áreas de texto, están estilizados, y los cambios en el contenido del editor actualizan dinámicamente la numeración de líneas. La ventana también permite cargar archivos externos y limpiar la consola.



```
1 package codigo;
2
3 import javax.swing.*;
4 import javax.swing.event.DocumentEvent;
5 import javax.swing.event.DocumentListener;
6 import java.awt.*;
7 import java.io.BufferedReader;
8 import java.io.File;
9 import java.io.FileReader;
10 import java.io.IOException;
11 import java.io.StringReader;
12
13 public class TermLexer extends JFrame {
14
15     public TermLexer() {
16         // Configuraciones del JFrame
17         setTitle("IDE Like NetBeans");
18         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
19         setSize(800, 600);
20         getContentPane().setBackground(Color.BLACK);
21
22         // Layout principal
23         BorderLayout layout = new BorderLayout();
24         setLayout(layout);
25
26         // Barra de menús
27         JMenuBar menuBar = new JMenuBar();
28         JMenu fileMenu = new JMenu("Archivo");
29         fileMenu.setForeground(Color.BLACK);
30         JMenuItem openItem = new JMenuItem("Abrir");
31         fileMenu.add(openItem);
32
33         JMenu viewMenu = new JMenu("Vista");
34         viewMenu.setForeground(Color.BLACK);
35         JMenuItem changeBgItem = new JMenuItem("Cambiar color de fondo");
36         viewMenu.add(changeBgItem);
37     }
38 }
```





```
Tokens.java x TermLexer.java x
Source Design History
37
38 menuBar.add(fileMenu);
39 menuBar.add(viewMenu);
40
41 setJMenuBar(menuBar);
42
43 // Barra de herramientas
44 JToolBar toolBar = new JToolBar();
45 toolBar.setBackground(Color.BLACK);
46
47 // Panel de edición (área de texto) con numeración de líneas
48 JTextArea editorArea = new JTextArea("");
49 editorArea.setFont(new Font("Monospaced", Font.PLAIN, 14));
50 editorArea.setBackground(Color.BLACK);
51 editorArea.setForeground(Color.WHITE);
52 editorArea.setCaretColor(Color.WHITE);
53 JScrollPane editorScrollPane = new JScrollPane(editorArea);
54 editorScrollPane.setBorder(BorderFactory.createTitledBorder(
55     BorderFactory.createLineBorder(Color.BLACK), "Editor de código",
56     0, 0, new Font("Calibri", Font.BOLD, 16), Color.BLACK));
57
58 JTextArea lineNumberArea = new JTextArea();
59 lineNumberArea.setEditable(false);
60 lineNumberArea.setFont(new Font("Monospaced", Font.PLAIN, 14));
61 lineNumberArea.setBackground(Color.DARK_GRAY);
62 lineNumberArea.setForeground(Color.WHITE);
63 JScrollPane lineNumberScrollPane = new JScrollPane(lineNumberArea);
64 lineNumberScrollPane.setBorder(BorderFactory.createTitledBorder(
65     BorderFactory.createLineBorder(Color.WHITE), "\n",
66     0, 0, new Font("Calibri", Font.BOLD, 16), Color.WHITE));
67
68 // Consola de salida
69 JTextArea consoleArea = new JTextArea();
70 consoleArea.setFont(new Font("Monospaced", Font.PLAIN, 12));
71 consoleArea.setEditable(false);
72 consoleArea.setBackground(Color.BLACK);
73 consoleArea.setForeground(Color.WHITE);
```

1:1 | INS | Unix (LF)



```
Tokens.java x TermLexer.java x
Source Design History
73 consoleArea.setForeground(Color.WHITE);
74 consoleArea.setCaretColor(Color.WHITE);
75 JScrollPane consoleScrollPane = new JScrollPane(consoleArea);
76 consoleScrollPane.setBorder(BorderFactory.createTitledBorder(
77     BorderFactory.createLineBorder(Color.BLACK, "Salida de análisis",
78     0, 0, new Font("Calibri", Font.BOLD, 16), Color.BLACK));
79
80 // Panel para analizar sintácticamente
81 JPanel syntaxPanel = new JPanel();
82 syntaxPanel.setBackground(Color.BLACK);
83 JButton analyzeButton = new JButton("Analizar");
84 JButton clearButton = new JButton("Limpiar");
85 JButton syntacticButton = new JButton("Sintáctico");
86 styleButton(analyzeButton);
87 styleButton(clearButton);
88 styleButton(syntacticButton);
89 syntaxPanel.add(analyzeButton);
90 syntaxPanel.add(clearButton);
91 syntaxPanel.add(syntacticButton);
92
93 // Botón "Tips"
94 JButton tipsButton = new JButton("Ayuda");
95 styleButton(tipsButton);
96 syntaxPanel.add(tipsButton);
97
98 // Acción del botón "Tips"
99 tipsButton.addActionListener(e -> {
100     // Muestra un cuadro de diálogo con tips para el uso del analizador
101     JOptionPane.showMessageDialog(this,
102         "Ayuda para el uso del analizador:\n" +
103         "- Asegúrate de que el código esté correctamente escrito.\n" +
104         "- Usa el botón 'Analizar' para realizar un análisis léxico.\n" +
105         "- Usa el botón 'Sintáctico' para verificar la gramática.",
106         "Tips de uso",
107         JOptionPane.INFORMATION_MESSAGE);
108     });
109
```



```
Tokens.java x TermLexer.java x
Source Design History
// Crear un panel central para centrar los elementos
JPanel centerPanel = new JPanel();
centerPanel.setLayout(new BorderLayout());

// Crear JSplitPane para dividir el editor y la consola
JSplitPane splitPane = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT, editorScrollPane, consoleScrollPane);
splitPane.setDividerLocation(0.47); // 80% del espacio para el editor, 20% para la consola
splitPane.setResizeWeight(0.47); // El editor toma el 80% del espacio

// Crear otro JSplitPane que divide la numeración de líneas y el panel con el editor y consola
JSplitPane editorSplitPane = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT, lineNumberScrollPane, splitPane);
editorSplitPane.setDividerLocation(40); // Ajustamos para que el lineNumberScrollPane ocupe un 10% del ancho
editorSplitPane.setResizeWeight(0.1); // Dejar que el editor se redimensione, pero mantener fija la numeración de líneas

// Añadir todo al panel central
centerPanel.add(editorSplitPane, BorderLayout.CENTER);

add(toolBar, BorderLayout.NORTH);
add(centerPanel, BorderLayout.CENTER); // Usamos centerPanel para que se centre
add(syntaxPanel, BorderLayout.SOUTH);

changeBgItem.addActionListener(e -> {
    Color currentBgColor = getContentPane().getBackground();
    if (currentBgColor.equals(Color.BLACK)) {
        getContentPane().setBackground(Color.WHITE);
        editorArea.setBackground(Color.WHITE);
        consoleArea.setBackground(Color.WHITE);
        lineNumberArea.setBackground(Color.LIGHT_GRAY);

        editorArea.setForeground(Color.BLACK);
        consoleArea.setForeground(Color.BLACK);
        lineNumberArea.setForeground(Color.BLACK);
    } else {
        getContentPane().setBackground(Color.BLACK);
        editorArea.setBackground(Color.BLACK);
        consoleArea.setBackground(Color.BLACK);
    }
});
```



```
146         consoleArea.setBackground(Color.BLACK);
147         lineNumberArea.setBackground(Color.DARK_GRAY);
148
149         editorArea.setForeground(Color.WHITE);
150         consoleArea.setForeground(Color.WHITE);
151         lineNumberArea.setForeground(Color.WHITE);
152     }
153 });
154
155 openItem.addActionListener(e -> {
156     JFileChooser fileChooser = new JFileChooser();
157     int option = fileChooser.showOpenDialog(this);
158     if (option == JFileChooser.APPROVE_OPTION) {
159         File file = fileChooser.getSelectedFile();
160         try (BufferedReader reader = new BufferedReader(new FileReader(file))) {
161             editorArea.read(reader, null);
162         } catch (IOException ex) {
163             JOptionPane.showMessageDialog(this, "Error al abrir el archivo: " + ex.getMessage(),
164                 "Error", JOptionPane.ERROR_MESSAGE);
165         }
166     }
167 });
168
169 editorArea.getDocument().addDocumentListener(new DocumentListener() {
170     @Override
171     public void insertUpdate(DocumentEvent e) {
172         updateLineNumbers(editorArea, lineNumberArea);
173     }
174
175     @Override
176     public void removeUpdate(DocumentEvent e) {
177         updateLineNumbers(editorArea, lineNumberArea);
178     }
179
180     @Override
181     public void changedUpdate(DocumentEvent e) {
182         updateLineNumbers(editorArea, lineNumberArea);
183     }
184 });
```

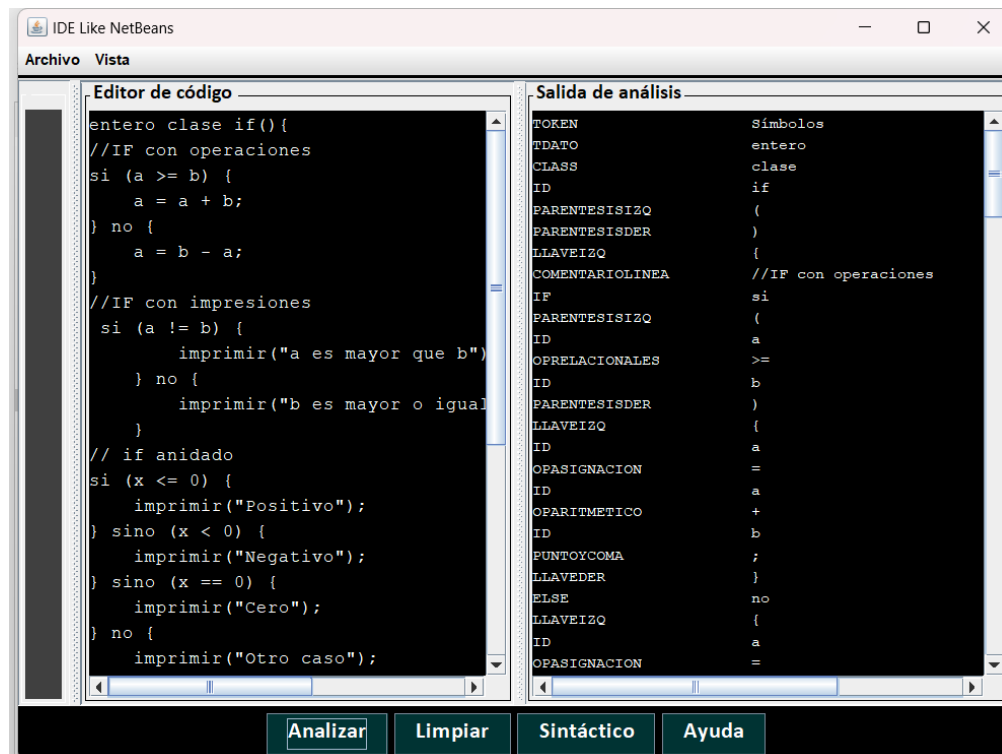
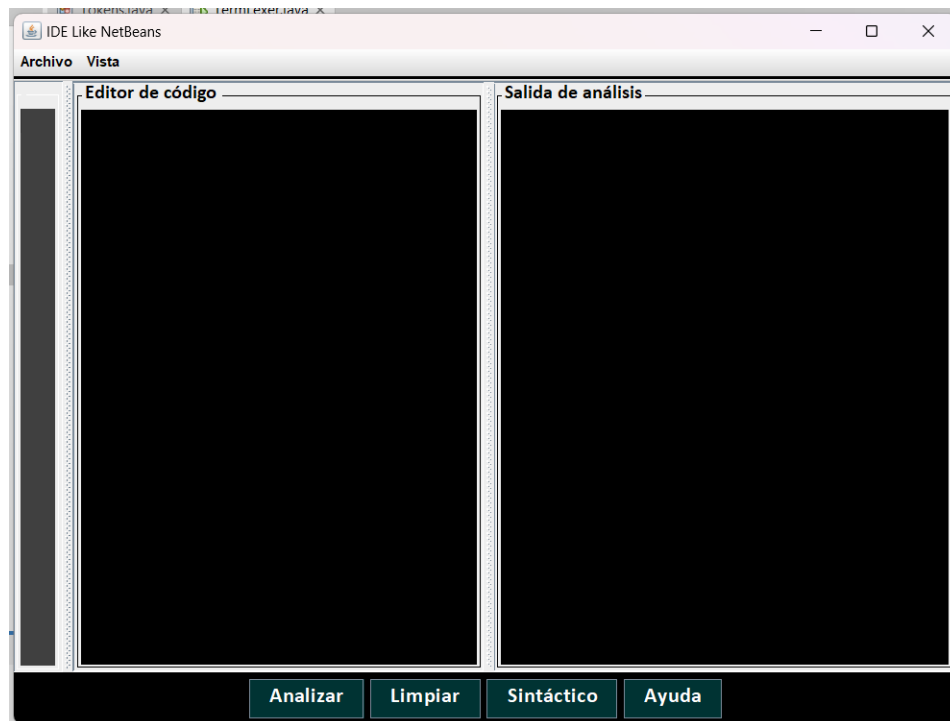


```
Tokens.java x TermLexer.java x
Source Design History
182 updateLineNumbers(editorArea, lineNumberArea);
183 }
184 });
185
186 analyzeButton.addActionListener(e -> {
187     String code = editorArea.getText();
188     try {
189         Lexer lexer = new Lexer(new StringReader(code));
190         StringBuilder result = new StringBuilder("TOKEN\t\t\tSimbolos\n");
191         Tokens token;
192         while ((token = lexer.yylex()) != null) {
193             // Obtener el tipo y el lexema del token
194             String tokenType = token.toString(); // Tipo de token (nombre del token)
195             String lexeme = lexer.yytext(); // Lexema (valor analizado del token)
196             result.append(tokenType).append("\t\t\t").append(lexeme).append("\n");
197         }
198         consoleArea.setText(result.toString());
199     } catch (IOException ex) {
200         consoleArea.setText("Error léxico.");
201     }
202 });
203
204
205
206
207 clearButton.addActionListener(e -> consoleArea.setText(""));
208
209 syntacticButton.addActionListener(e -> {
210     // Crear y mostrar la ventana TermSintac
211     new TermSintac().setVisible(true); // Aquí se crea y muestra la ventana TermSintac
212     dispose();
213 });
214 }
215
216 private static void styleButton(JButton button) {
217     button.setBackground(new Color(0, 51, 51));
218     button.setFont(new Font("Calibri", Font.BOLD, 18));
219 }
```



```
215
216 private static void styleButton(JButton button) {
217     button.setBackground(new Color(0, 51, 51));
218     button.setFont(new Font("Calibri", Font.BOLD, 18));
219     button.setForeground(Color.WHITE);
220 }
221
222 private static void updateLineNumbers(JTextArea editorArea, JTextArea lineNumberArea) {
223     String text = editorArea.getText();
224     String[] lines = text.split("\n");
225     StringBuilder lineNumbers = new StringBuilder();
226     for (int i = 1; i <= lines.length; i++) {
227         lineNumbers.append(i).append("\n");
228     }
229     lineNumberArea.setText(lineNumbers.toString());
230 }
231
232 public static void main(String[] args) {
233     SwingUtilities.invokeLater(() -> {
234         TermLexer termLexer = new TermLexer(); // Crear la instancia de la ventana
235         termLexer.setVisible(true);           // Mostrarla
236     });
237 }
238
239
```

## PANTALLAS RESULTANTES CON PRUEBAS DEL RECONOCIMIENTO DE LOS TOKENS



## V. Conclusiones:

EXPLICAR EN (MEDIA CUARTILLA A 1 CUARTILLA) EL OBJETIVO DEL DESARROLLO DEL PROYECTO, SOFTWARE Y LIBRERIAS QUE UTILIZARON Y SU FUNCION, PROBLEMATICAS PERSONALES QUE ENFRENTARON.

El proyecto se centra en el desarrollo de un analizador léxico, cuyo propósito es descomponer el código fuente de un lenguaje de programación como en C en componentes básicos llamados tokens, como palabras reservadas, identificadores, operadores, números y otros símbolos reconocibles. Esto se logró mediante el uso de JFlex, una herramienta que permite definir reglas léxicas a través de expresiones regulares. Estas reglas permiten identificar con precisión los elementos del lenguaje, transformando el texto en una secuencia estructurada que sirve como entrada para fases posteriores del procesamiento, como el análisis sintáctico. Durante el desarrollo, el analizador fue diseñado para manejar errores léxicos, como caracteres o estructuras no reconocidas, y brindar retroalimentación clara al usuario a través de una interfaz gráfica. Entre los principales desafíos enfrentados estuvo el diseño de patrones suficientemente específicos para evitar ambigüedades entre tokens similares, además de integrar el analizador con una consola interactiva en tiempo real que mostrara los resultados de manera clara. A pesar de estas dificultades, el analizador léxico cumple exitosamente con su objetivo de ser una herramienta que permite a los usuarios entender cómo un compilador procesa el texto fuente en esta primera etapa, contribuyendo al aprendizaje práctico sobre los fundamentos de los compiladores.