

Nombre de la práctica	ANALIZADOR SINTACTICO (UNIDAD 5)			No.	5
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 sintáctico 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	"**"



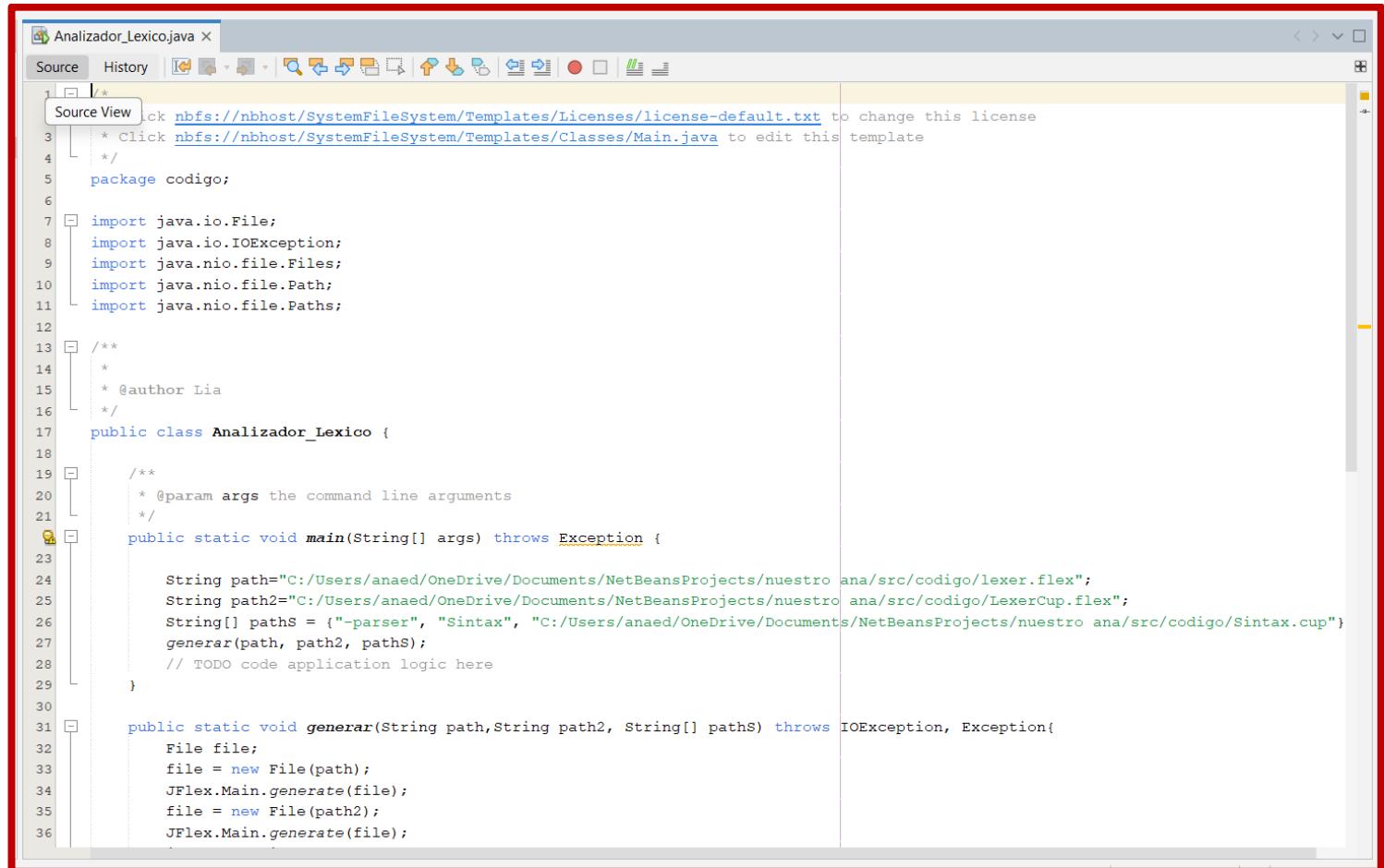
opAritmetico	24	" / "
opAritmetico	25	" % "
opAsignacion	26	" = "
opAsignacion	27	" += "
opAsignacion	28	" -= "
opAsignacion	29	" *= "
opAsignacion	30	" /= "
opAsignacion	31	" %= "
opRelacionales	32	" == "
opRelacionales	33	" != "
opRelacionales	34	" > "
opRelacionales	35	" < "
opRelacionales	36	" >= "
opRelacionales	37	" <= "
opLogicos	38	" && "
opLogicos	39	" "
opIncremento	40	" ++ "
opDecremento	41	" -- "
parentesisIzq	42	" ("
parentesisDer	43	") "
llaveIzq	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/código), 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.



```
Analizador_Lexico.java X
Source History |< > □
Source View Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
            * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Main.java to edit this template
1  /*
2  */
3  package codigo;
4
5  import java.io.File;
6  import java.io.IOException;
7  import java.nio.file.Files;
8  import java.nio.file.Path;
9  import java.nio.file.Paths;
10
11 /**
12  *
13  * @author Lia
14  */
15 public class Analizador_Lexico {
16
17     /**
18      * @param args the command line arguments
19     */
20     public static void main(String[] args) throws Exception {
21
22         String path="C:/Users/anaed/OneDrive/Documents/NetBeansProjects/nuestro ana/src/codigo/lexer.flex";
23         String path2="C:/Users/anaed/OneDrive/Documents/NetBeansProjects/nuestro ana/src/codigo/LexerCup.flex";
24         String[] pathS = {"-parser", "Sintax", "C:/Users/anaed/OneDrive/Documents/NetBeansProjects/nuestro ana/src/codigo/Sintax.cup"};
25         generar(path, path2, pathS);
26         // TODO code application logic here
27     }
28
29
30
31     public static void generar(String path, String path2, String[] paths) throws IOException, Exception{
32         File file;
33         file = new File(path);
34         JFlex.Main.generate(file);
35         file = new File(path2);
36         JFlex.Main.generate(file);
}
```



The screenshot shows a NetBeans IDE window with the title "Analizador_Lexico.java X". The code is written in Java and uses the JFlex library to generate tokens. It includes logic to delete existing symbol and syntax files and move them to new locations. The code is as follows:

```
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 }
```

1:1

INS Windows (CRLF)

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, relaciones 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.



The screenshot shows a Java code editor with two tabs: "Analizador_Lexico.java" and "lexer.flex". The "lexer.flex" tab is active, displaying the following Java code:

```
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 linea */ return COMENTARIOLINEA; }
19 /*/*([^\n]|[\r\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; }
```

The code defines a lexer grammar for a programming language, using tokens like OPARITMETICO, OPASIGNACION, OPRELACIONALES, OPLOGICOS, OPINCREMENTO, BOOLEANO, TDATO, IF, ELSEIF, ELSE, SWITCH, CASE, BREAK, RETURN, WHILE, DO, and VOID.



MANUAL DE PRÁCTICAS



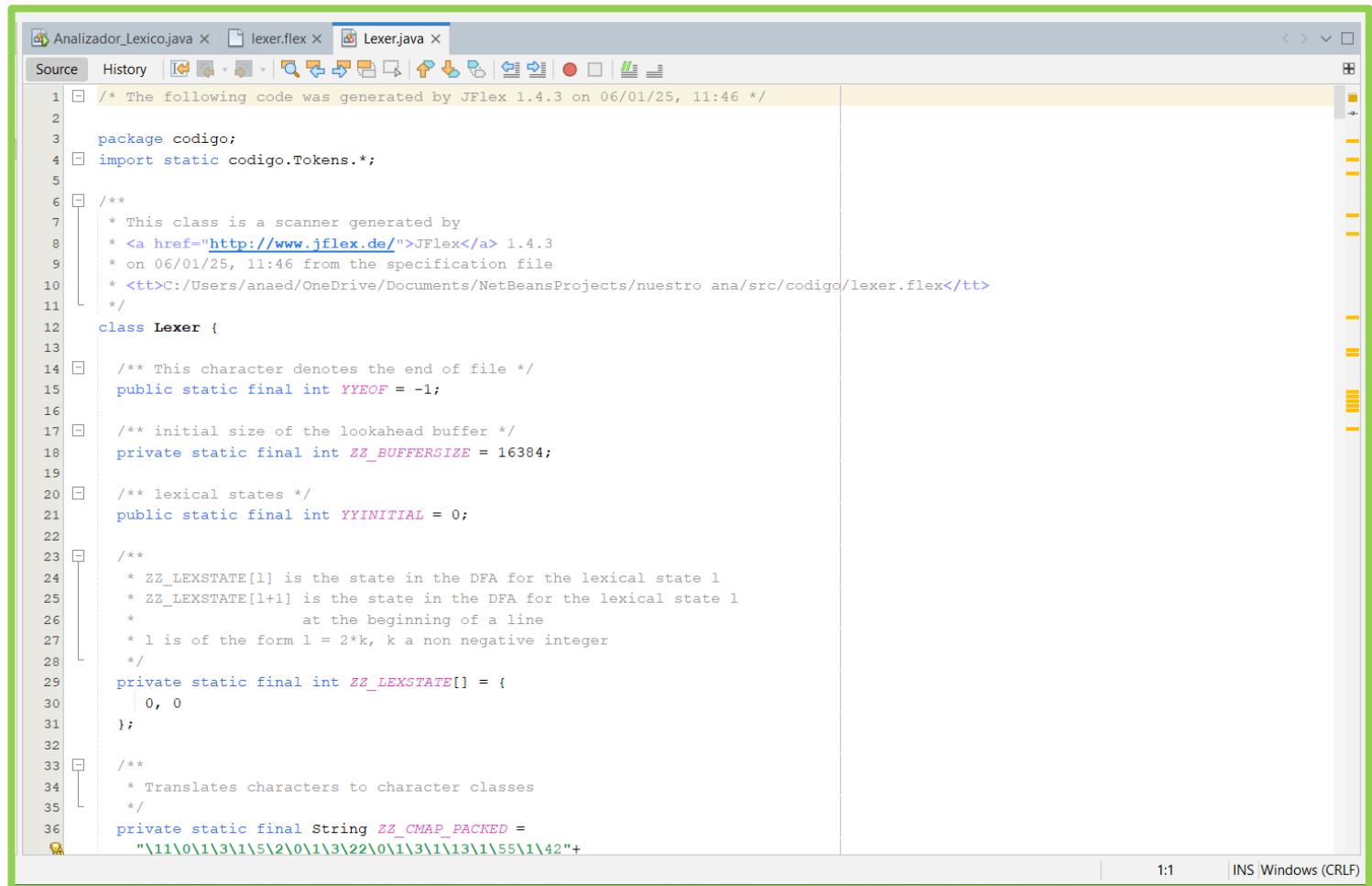
The screenshot shows a Java code editor interface with two tabs: "Analizador_Lexico.java" and "lexer.flex". The "Analizador_Lexico.java" tab is active, displaying the following code:

```
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)|(D)}* { return ID; }
58 ("-"?{D}+.{D}*|("-"?{D}+) { return NUM; }
59 \"[^"]*\\" { return CADENA; }
60 ".*" { return ERROR; }
61
62
```

The code is a lexical analyzer implementation in Java, utilizing regular expressions to define tokens such as identifiers, numbers, strings, and punctuation. The "lexer.flex" tab is visible at the top but contains no visible text.

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.



Analizador_Lexico.java x lexer.flex x Lexer.java x

Source History

```
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_BUFSIZE = 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          "\\"1\\0\\1\\3\\1\\5\\2\\0\\1\\3\\22\\0\\1\\3\\1\\13\\1\\55\\1\\42"+
```

1:1 INS Windows (CRLF)



MANUAL DE PRÁCTICAS



```
38     "1\\0\\1\\1\\1\\15\\1\\56\\1\\43\\1\\44\\1\\6\\1\\7\\1\\52\\1\\10"+  
39     "1\\54\\1\\4\\12\\2\\1\\53\\1\\51\\1\\14\\1\\12\\1\\14\\2\\0\\2\\1"+  
40     "1\\37\\10\\1\\1\\35\\1\\16\\1\\1\\47\\1\\0\\1\\50\\1\\11\\1\\1\\0"+  
41     "1\\23\\1\\1\\1\\32\\1\\22\\1\\20\\1\\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\\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\\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\\10\\1\\1\\31\\1\\0\\1\\32\\1\\0\\22\\1\\1\\33"+  
61     "1\\7\\1\\1\\34\\1\\3\\1\\1\\35\\5\\1\\1\\36\\1\\1\\1\\37"+  
62     "1\\3\\1\\1\\40\\1\\41\\1\\3\\1\\1\\42\\2\\1\\1\\43\\2\\1"+  
63     "1\\1\\44\\1\\17\\1\\1\\45\\2\\1\\1\\46\\1\\47\\1\\50\\1\\2\\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 */
```



MANUAL DE PRACTICAS



The screenshot shows a Java code editor with the following details:

- Project Structure:** The top bar shows three tabs: "Analizador_Lexico.java X", "lexer.flex X", and "Lexer.java X".
- Code Editor:** The main area displays the content of the "Lexer.java" file. The code is heavily annotated with numerous escape sequences, likely representing characters from a packed string.
- Toolbars and Menus:** A toolbar at the top includes icons for Source, History, Find, Copy, Paste, and Save. A menu bar is visible at the very top.
- Code Folding:** The code is organized into several sections using curly braces {}, and some sections are collapsed, indicated by a minus sign -.
- Vertical Ruler:** A vertical ruler on the right side shows position markers and line numbers.

```
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     "\u0\u0\u57\u0\u136\u0\u215\u0\u274\u0\u353\u0\u011a\u0\u0149"+
93     "\u0\u215\u0\u0178\u0\u0178\u0\u0178\u0\u01a7\u0\u01d6\u0\u0205\u0\u0234"+
94     "\u0\u0263\u0\u0292\u0\u02c1\u0\u02f0\u0\u031f\u0\u034e\u0\u037d\u0\u03ac"+
95     "\u0\u03db\u0\u040a\u0\u0439\u0\u215\u0\u215\u0\u215\u0\u215\u0\u215"+
96     "\u0\u215\u0\u215\u0\u215\u0\u215\u0\u0468\u0\u0497\u0\u215"+
97     "\u0\u04c6\u0\u04f5\u0\u0524\u0\u215\u0\u215\u0\u215\u0\u215\u0\u0553"+
98     "\u0\u0582\u0\u05b1\u0\u05e0\u0\u060f\u0\u063e\u0\u066d\u0\u57\u0\u069c"+
99     "\u0\u06cb\u0\u06fa\u0\u0729\u0\u0758\u0\u0787\u0\u07b6\u0\u07e5\u0\u215"+
100    "\u0\u0497\u0\u215\u0\u0814\u0\u0843\u0\u0872\u0\u08a1\u0\u08d0\u0\u08ff"+
101    "\u0\u092e\u0\u095d\u0\u098c\u0\u09bb\u0\u09ea\u0\u0a19\u0\u0a48\u0\u0a77"+
102    "\u0\u0aa\u0\u0ad5\u0\u0b04\u0\u0b33\u0\u0b62\u0\u215\u0\u0b91\u0\u0bc0"+
103    "\u0\u0bef\u0\u0c1e\u0\u0c4d\u0\u0c7c\u0\u0cab\u0\u57\u0\u0cda\u0\u0d09"+
104    "\u0\u0d38\u0\u57\u0\u0d67\u0\u0d96\u0\u0dc5\u0\u0df4\u0\u0e23\u0\u57"+
105    "\u0\u0e52\u0\u57\u0\u0e81\u0\u0eb0\u0\u0edf\u0\u57\u0\u57\u0\u0fe0"+
106    "\u0\u0f3d\u0\u0f6c\u0\u57\u0\u0f9b\u0\u0fc4\u0\u57\u0\u0ff9\u0\u1028"+
107    "\u0\u57\u0\u1057\u0\u1086\u0\u10b5\u0\u10e4\u0\u1113\u0\u1142\u0\u1171"+
108    "\u0\u11a0\u0\u11cf\u0\u11fe\u0\u122d\u0\u125c\u0\u128b\u0\u12ba\u0\u12e9"+
109    "\u0\u57\u0\u1318\u0\u1347\u0\u57\u0\u57\u0\u1376\u0\u13a5"+
110    "\u0\u13d4\u0\u1403\u0\u1432\u0\u1461\u0\u1490\u0\u14bf\u0\u14ee\u0\u151d"+
```



MANUAL DE PRACTICAS



The screenshot shows a Java IDE interface with three tabs at the top: 'Analizador_Lexico.java X', 'lexer.flex X', and 'Lexer.java X'. The 'Lexer.java X' tab is active, displaying the following Java code:

```
110     "\u0\u13d4\u0\u1403\u0\u1432\u0\u1461\u0\u1490\u0\u14bf\u0\u14ee\u0\u151d"+  
111     "\u0\u57";  
112  
113     private static int [] zzUnpackRowMap() {  
114         int [] result = new int[153];  
115         int offset = 0;  
116         offset = zzUnpackRowMap(ZZ_ROWMAP_PACKED_0, offset, result);  
117         return result;  
118     }  
119  
120     private static int zzUnpackRowMap(String packed, int offset, int [] result) {  
121         int i = 0; /* index in packed string */  
122         int j = offset; /* index in unpacked array */  
123         int l = packed.length();  
124         while (i < l) {  
125             int high = packed.charAt(i++) << 16;  
126             result[j++] = high | packed.charAt(i++);  
127         }  
128         return j;  
129     }  
130  
131     /**  
132      * The transition table of the DFA  
133      */  
134     private static final int [] ZZ_TRANS = zzUnpackTrans();  
135  
136     private static final String ZZ_TRANS_PACKED_0 =  
137     "\u1\u0\u1\u2\u1\u3\u1\u4\u1\u5\u1\u4\u1\u6\u1\u7"+  
138     "\u1\u10\u1\u11\u1\u12\u1\u13\u1\u14\u1\u15\u1\u16\u1\u17"+  
139     "\u1\u20\u1\u21\u1\u22\u1\u23\u1\u24\u1\u25"+  
140     "\u1\u26\u1\u27\u1\u30\u1\u31\u3\u2\u1\u32\u1\u33\u1\u34"+  
141     "\u1\u35\u1\u36\u1\u37\u1\u40\u1\u41\u1\u42\u1\u43\u1\u44"+  
142     "\u1\u45\u1\u46\u1\u47\u1\u50\u1\u0\u2\u2\u14\u0\u23\u2"+  
143     "\u17\u0\u1\u3\u51\u0\u1\u51\u0\u1\u52\u1\u0\u1\u53"+  
144     "\u3\u0\u1\u54\u56\u0\u1\u54\u53\u0\u1\u55\u2\u0\u1\u54"+  
145     "\u46\u0\u1\u3\u5\u0\u1\u55\u1\u0\u1\u54\u56\u0\u1\u56"+  
146     "\u5\u6\u1\u0\u1\u57\u1\u57\u1\u1\u57\u1\u0\u1\u57\u1\u2"+
```



MANUAL DE PRÁCTICAS



```
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\1\2" +
148 " \1\62\11\2\16\0\2\2\14\0\1\2\1\63\21\2" +
149 " \16\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 " \16\0\2\2\14\0\1\2\1\70\21\2\16\0\2\2" +
153 " \14\0\4\2\1\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 " \16\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\0\53\1\103\50\53\1\0" +
159 " \2\2\14\0\2\2\1\104\20\2\16\0\2\2\14\0\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" +
```



MANUAL DE PRACTICAS





MANUAL DE PRACTICAS



The screenshot shows a Java code editor with the following details:

- File Tabs:** The tabs at the top are "Analizador_Lexico.java X", "lexer.flex X", and "Lexer.java X".
- Source View:** The main area displays the code for the `Lexer.java` class.
- Annotations:** There are several annotations in the code:
 - A yellow warning icon is located next to the line `private static int [] zzUnpackTrans()`.
 - A red error icon is located next to the line `private static final String ZZ_ERROR_MSG[] = {`.
- Code Content:** The code includes various string literals, method definitions for `zzUnpackTrans`, and error handling logic.

```
220     "\12\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(ZZ_TRANS_PACKED_0, 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         "Unknown internal scanner error",  
255         "Error: could not match input",  
256         "Error: pushback value was too large"
```



MANUAL DE PRACTICAS



The screenshot shows the Java code for the lexer. The code defines several static final variables and methods related to attribute unpacking. It includes a packed string constant and a method to unpack it into an array of integers.

```
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         "\\" + 0 + 2 + 1 + 1 + 4 + 1 + 1 + 1 + 1 + 0 + 1 + "+\n"
266         +"\\2\\0\\15\\1\\12\\11\\2\\1\\1\\11\\2\\1\\1\\0\\4\\11"+\n267         +"\\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 }
```



MANUAL DE PRACTICAS



The screenshot shows a Java IDE interface with the following details:

- Project Structure:** The project is named "Analizador_Lexico.java".
- Files:** The files shown are "lexer.flex" and "Lexer.java".
- Lexer.java Content:** The code defines several private variables:
 - `zzLexicalState` (initially `YYINITIAL`) representing the current lexical state.
 - `zzBuffer` (a new `char[ZZ_BUFSIZE]`) representing the buffer containing the current text to be matched.
 - `zzMarkedPos` representing the text position at the last accepting state.
 - `zzCurrentPos` representing the current text position in the buffer.
 - `zzStartRead` marking the beginning of the `yytext()` string in the buffer.
 - `zzEndRead` marking the last character in the buffer that has been read from input.
 - `yyline` representing the number of newlines encountered up to the start of the matched text.
 - `yychar` representing the number of characters up to the start of the matched text.
 - `yycolumn` representing the number of characters from the last newline up to the start of the matched text.
 - `zzAtBOL` (boolean) indicating if the scanner is currently at the beginning of a line.
- UI Elements:** The interface includes tabs for Source, History, and various icons for file operations like Open, Save, and Print. A status bar at the bottom right shows "1:1" and "INS Windows (CRLF)".



MANUAL DE PRÁCTICAS



The screenshot shows a Java code editor with the file `Lexer.java` open. The code is part of a lexical analyzer (lexer) for a parser generator like Flex. It includes constructors for `Reader` and `InputStream`, and a static method `zzUnpackCMap` for unpacking character translation tables. The code uses JavaDoc-style comments to describe the parameters and purpose of each method.

```
Analizador_Lexico.java X lexer.flex X Lexer.java X
Source History | I O V D F G B P S T C E L R M
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 */
```

1:1

INS Windows (CRLF)



MANUAL DE PRÁCTICAS



```
Analizador_Lexico.java X lexer.flex X Lexer.java X
Source History | I O S D F G B C E P R T L V Z X Y
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;
405 }
```

1:1

INS Windows (CRLF)



MANUAL DE PRÁCTICAS



```
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 yclose() 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)



MANUAL DE PRÁCTICAS



The screenshot shows a Java code editor with the file `Lexer.java` open. The code is part of a lexical analyzer implementation. It includes methods for resetting the scanner to a new input stream, returning the current lexical state, and entering a new lexical state. The code uses annotations like `@param` and `zz*` variables.

```
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     zzEODDone = 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
477 }
```

1:1

INS Windows (CRLF)



MANUAL DE PRÁCTICAS



```
Analizador_Lexico.java X lexer.flex X Lexer.java X
Source History | I O D S M F G B C E P A R L T Z Y
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 * Returns the character at position <tt>pos</tt> from the
492 * matched text.
493 *
494 * It is equivalent to yytext().charAt(pos), but faster
495 *
496 * @param pos the position of the character to fetch.
497 *             A value from 0 to yylength()-1.
498 *
499 * @return the character at position pos
500 */
501 public final char yycharat(int pos) {
502     return zzBuffer[zzStartRead+pos];
503 }
504
505
506 /**
507 * Returns the length of the matched text region.
508 */
509 public final int yylength() {
510     return zzMarkedPos-zzStartRead;
511 }
512
513
```



MANUAL DE PRÁCTICAS



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



MANUAL DE PRÁCTICAS



The screenshot shows a Java code editor with the following details:

- Title Bar:** Analizador_Lexico.java x lexer.flex x Lexer.java x
- Toolbar:** Source, History, and various icons for file operations.
- Code Area:** The code is for a lexer, specifically the `Lexer.java` file. It includes methods like `yypushback` and `yylex`. The code uses annotations like `@return` and `@exception`.
- Code Content:**

```
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;
```
- Status Bar:** 21:41 | INS Windows (CRLF)



MANUAL DE PRACTICAS



```
zzState = ZZ_LEXSTATE[zzLexicalState];

zzForAction: {
    while (true) {

        if (zzCurrentPosL < zzEndReadL)
            zzInput = zzBufferL[zzCurrentPosL++];
        else if (zzAtEOF) {
            zzInput = YYEOF;
            break zzForAction;
        }
        else {
            // store back cached positions
            zzCurrentPos = zzCurrentPosL;
            zzMarkedPos = zzMarkedPosL;
            boolean eof = zzRefill();
            // get translated positions and possibly new buffer
            zzCurrentPosL = zzCurrentPos;
            zzMarkedPosL = zzMarkedPos;
            zzBufferL = zzBuffer;
            zzEndReadL = zzEndRead;
            if (eof) {
                zzInput = YYEOF;
                break zzForAction;
            }
            else {
                zzInput = zzBufferL[zzCurrentPosL++];
            }
        }
        int zzNext = zzTransL[ zzRowMapL[zzState] + zzCMapL[zzInput] ];
        if (zzNext == -1) break zzForAction;
        zzState = zzNext;

        int zzAttributes = zzAttrL[zzState];
        if ( (zzAttributes & 1) == 1 ) {
            zzAction = zzState;
        }
    }
}
```



```
623     zzAction = zzState;
624     zzMarkedPosL = zzCurrentPosL;
625     if ( (zzAttributes & 8) == 8 ) break zzForAction;
626   }
627 }
628 }
629 }
630
631 // store back cached position
632 zzMarkedPos = zzMarkedPosL;
633
634 switch (zzAction < 0 ? zzAction : ZZ_ACTION[zzAction]) {
635   case 23:
636     { return IF;
637     }
638   case 42: break;
639   case 2:
640     { return NUM;
641     }
642   case 43: break;
643   case 8:
644     { return PARENTESISIZQ;
645     }
646   case 44: break;
647   case 26:
648     { return CADENA;
649     }
650   case 45: break;
651   case 32:
652     { return BOOLEANO;
653     }
654   case 46: break;
655   case 9:
656     { return PARENTESISDER;
657     }
658   case 47: break;
659   case 12:
```

21:41

INS Windows (CRLF)



MANUAL DE PRÁCTICAS



```
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:
```

21:41 | INS Windows (CRLF)



MANUAL DE PRÁCTICAS



The screenshot shows a Java code editor interface with the following details:

- Title Bar:** The title bar displays three tabs: "Analizador_Lexico.java", "lexer.flex", and "Lexer.java".
- Toolbar:** A toolbar at the top contains icons for Source, History, and various file operations like Open, Save, Print, and Copy.
- Code Area:** The main area displays the content of the "Lexer.java" file. The code is a switch statement handling various tokens. It includes comments, labels, and returns specific tokens like COMENTARIOBLOQUE, LLAVEIZQ, LLAVEDER, ELSEIF, DO, ID, CLASS, OPASIGNACION, and FALSE.

```
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 FALSE;
```



The screenshot shows a Java code editor with the Lexer.java file open. The code is a switch statement handling various tokens. The editor has syntax highlighting and a vertical toolbar on the right.

```
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;
```

21:41 | INS Windows (CRLF)



MANUAL DE PRÁCTICAS



```
769     }
770     case 75: break;
771     case 38:
772         {
773             return BREAK;
774         }
775     case 76: break;
776     case 41:
777         {
778             return DEFAULT;
779         }
780     case 77: break;
781     case 14:
782         {
783             return PUNTOYCOMA;
784         }
785     case 78: break;
786     case 39:
787         {
788             return PRINTF;
789         }
790     case 79: break;
791     case 16:
792         {
793             return DOSPUNTOS;
794         }
795     case 80: break;
796     case 4:
797         {
798             return OPARITMETICO;
799         }
800     case 81: break;
801     case 20:
802         {
803             /* Comentario de línea */ return COMENTARIOLINEA;
804         }
805     case 82: break;
806     default:
807         if (zzInput == YYEOF && zzStartRead == zzCurrentPos) {
808             zzAtEOF = true;
809             return null;
810         }
811         else {
812             zzScanError(ZZ_NO_MATCH);
813         }
814     }
815 }
```

21:41 | INS Windows (CRLF)

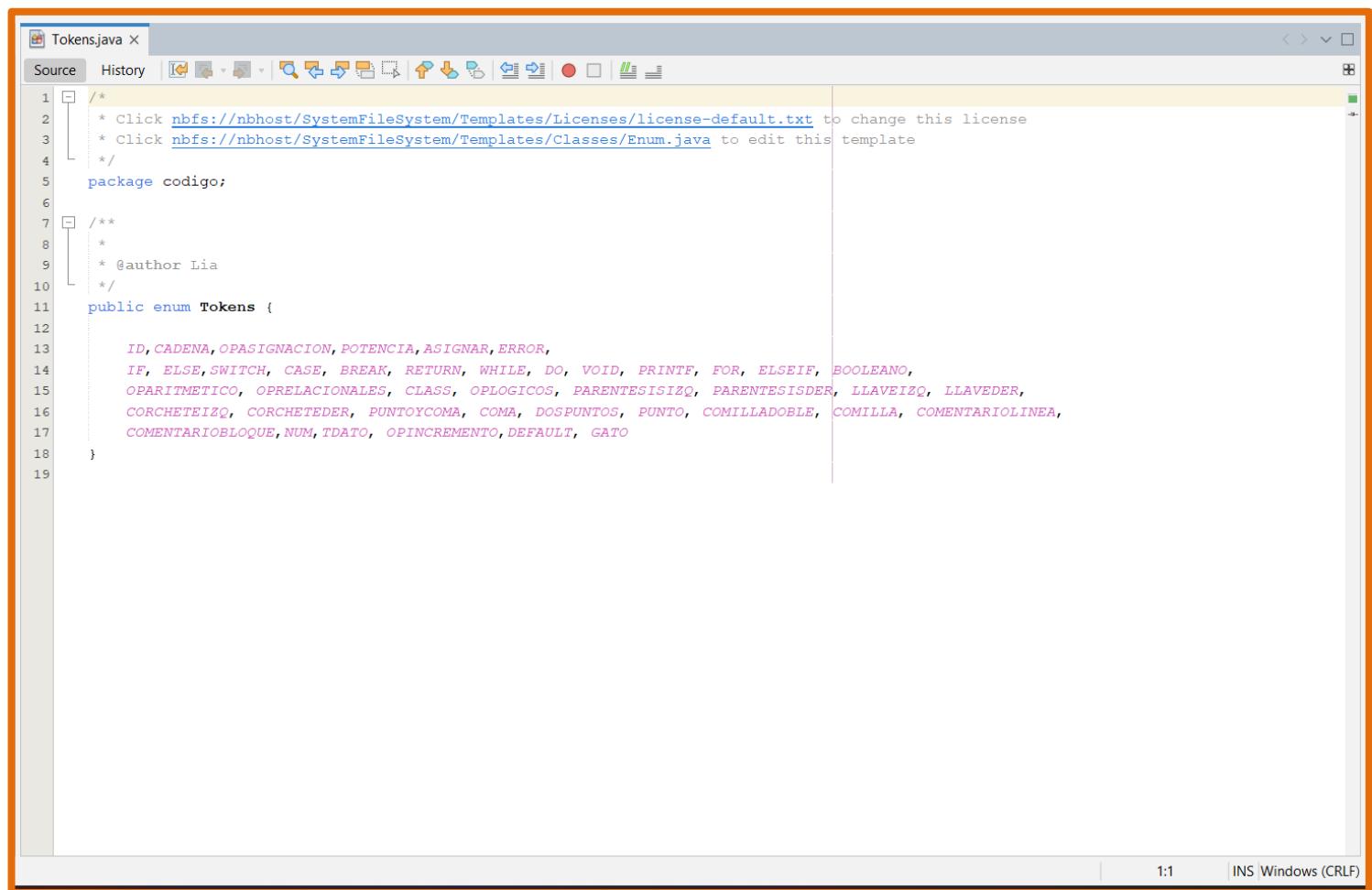


The screenshot shows a Java code editor interface. The main window displays the content of 'Analizador_Lexico.java'. The code is a switch statement handling various input cases, including arithmetic operators, comments, and EOF processing. The editor has syntax highlighting and a vertical scrollbar on the right. The status bar at the bottom right indicates the time as 21:41 and the file type as INS Windows (CRLF).

```
189 }  
190     case 80: break;  
191     case 4:  
192         { return OPARITMETICO;  
193     }  
194     case 81: break;  
195     case 20:  
196         { /* Comentario de línea */ return COMENTARIOLINEA;  
197     }  
198     case 82: break;  
199     default:  
200         if (zzInput == YYEOF && zzStartRead == zzCurrentPos) {  
201             zzAtEOF = true;  
202             return null;  
203         }  
204         else {  
205             zzScanError(ZZ_NO_MATCH);  
206         }  
207     }  
208 }  
209  
210  
211 }  
212  
213 }
```

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.



```
Tokens.java X
Source History | I O D F G H J K L P R T V Z E S
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.

La captura de pantalla muestra la interfaz de NetBeans IDE con el archivo TermLexer.java abierto. El código define una clase TermLexer que extiende JFrame. El constructor configura el frame con un tamaño de 800x600 y un fondo negro. Se establece un layout de BorderLayout. Una barra de menús es creada con dos menús principales: "Archivo" y "Vista". El menú "Archivo" contiene un ítem "Abrir". El menú "Vista" contiene un ítem "Cambiar color de fondo". El código también importa varias bibliotecas de Java Swing y AWT, así como BufferedReader y FileReader para manejo de archivos.

```
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 }
```



MANUAL DE PRÁCTICAS



The screenshot shows a Java code editor interface with two tabs: "Tokens.java" and "TermLexer.java". The "Tokens.java" tab is active, displaying the following code:

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

The code defines a Java application with a menu bar, a toolbar, and two panels: one for editing text with line numbers and another for a console output area.



MANUAL DE PRÁCTICAS



The screenshot shows a Java IDE interface with two tabs: "Tokens.java X" and "TermLexer.java X". The "Tokens.java X" tab is active, displaying the following Java code:

```
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 }
```

The code implements a basic user interface with a text area for output, three buttons for analysis, and a help button that displays a message dialog with usage instructions.



MANUAL DE PRACTICAS



```
110 // Crear un panel central para centrar los elementos
111 JPanel centerPanel = new JPanel();
112 centerPanel.setLayout(new BorderLayout());
113
114 // Crear JSplitPane para dividir el editor y la consola
115 JSplitPane splitPane = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT, editorScrollPane, consoleScrollPane);
116 splitPane.setDividerLocation(0.47); // 80% del espacio para el editor, 20% para la consola
117 splitPane.setResizeWeight(0.47); // El editor toma el 80% del espacio
118
119 // Crear otro JSplitPane que divide la numeración de líneas y el panel con el editor y consola
120 JSplitPane editorSplitPane = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT, lineNumberScrollPane, splitPane);
121 editorSplitPane.setDividerLocation(40); // Ajustamos para que el lineNumberScrollPane ocupe un 10% del ancho
122 editorSplitPane.setResizeWeight(0.1); // Dejar que el editor se redimensione, pero mantener fija la numeración de líneas
123
124 // Añadir todo al panel central
125 centerPanel.add(editorSplitPane, BorderLayout.CENTER);
126
127
128 add(toolBar, BorderLayout.NORTH);
129 add(centerPanel, BorderLayout.CENTER); // Usamos centerPanel para que se centre
130 add(syntaxPanel, BorderLayout.SOUTH);
131
132 changeBgItem.addActionListener(e -> {
133     Color currentBgColor = getContentPane().getBackground();
134     if (currentBgColor.equals(Color.BLACK)) {
135         getContentPane().setBackground(Color.WHITE);
136         editorArea.setBackground(Color.WHITE);
137         consoleArea.setBackground(Color.WHITE);
138         lineNumberArea.setBackground(Color.LIGHT_GRAY);
139
140         editorArea.setForeground(Color.BLACK);
141         consoleArea.setForeground(Color.BLACK);
142         lineNumberArea.setForeground(Color.BLACK);
143     } else {
144         getContentPane().setBackground(Color.BLACK);
145         editorArea.setBackground(Color.BLACK);
146         consoleArea.setBackground(Color.BLACK);
147     }
148 });
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
```



MANUAL DE PRÁCTICAS



The screenshot shows a Java code editor with two tabs: "Tokens.java" and "TermLexer.java". The "Tokens.java" tab is active, displaying the following code:

```
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 });
185 }
```

The code implements a file opening dialog and updates line numbers in two text areas (editorArea and lineNumberArea) whenever the document changes.



The screenshot shows a Java code editor with two tabs: "Tokens.java" and "TermLexer.java X". The "Tokens.java" tab is active, displaying the following code:

```
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\tSímbolos\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));
```

The code implements a lexer and parser for tokens and symbols. It uses a Lexer from the `new Lexer(new StringReader(code))` and appends token type and value to a `StringBuilder`. It also handles exceptions and styles buttons.

1:1

INS Unix (LF)



MANUAL DE PRÁCTICAS



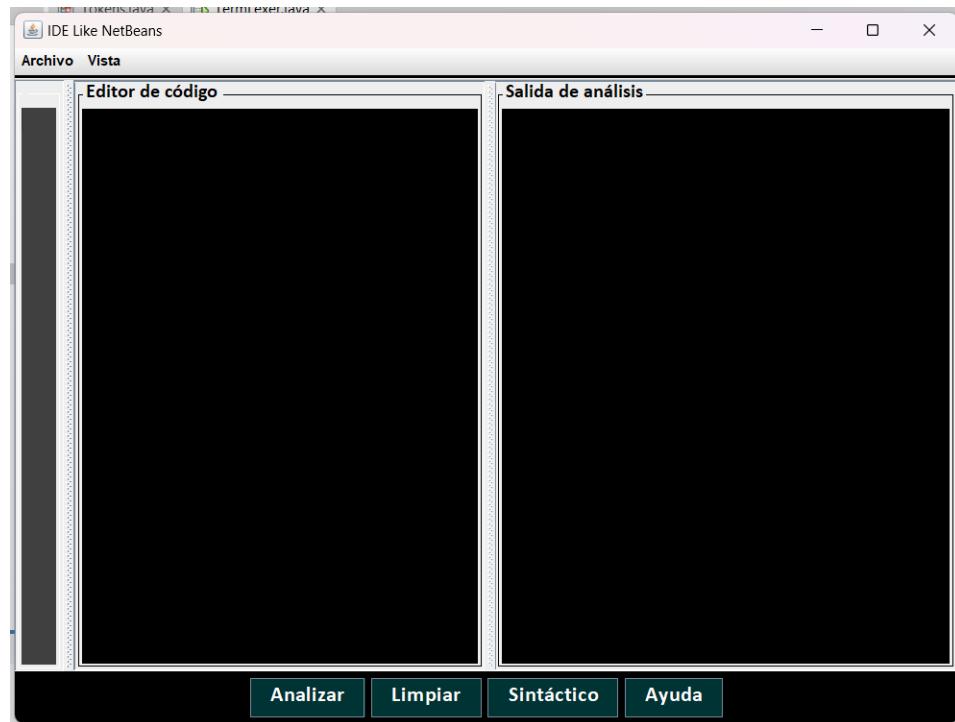
The screenshot shows a Java code editor interface with two tabs: "Tokens.java" and "TermLexer.java". The "Tokens.java" tab is active, displaying the following code:

```
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 }
```

The "TermLexer.java" tab is visible at the top. The bottom right corner of the editor shows "1:1" and "INS Unix (LF)".



PANTALLAS RESULTANTES CON PRUEBAS DEL RECONOCIMIENTO DE LOS TOKENS



```
entero clase if(){
//IF con operaciones
si (a >= b) {
    a = a + b;
} no {
    a = b - a;
}
//IF con impresiones
si (a != b) {
    imprimir("a es mayor que b")
} no {
    imprimir("b es mayor o igual")
}
// if anidado
si (x <= 0) {
    imprimir("Positivo");
} sino (x < 0) {
    imprimir("Negativo");
} sino (x == 0) {
    imprimir("Cero");
} no {
    imprimir("Otro caso");
```

Salida de análisis:

TOKEN	Símbolos
TDATO	entero
CLASS	clase
ID	if
PARENTESISIZQ	(
PARENTESISDER)
LLAVEIZQ	{
COMENTARIOLINEA	//IF con operaciones
IF	si
PARENTESISIZQ	(
ID	a
OPRELACIONALES	>=
ID	b
PARENTESISDER)
LLAVEIZQ	{
ID	a
OPASIGNACION	=
ID	a
OPARITMETICO	+
ID	b
PUNTOYCOMA	;
LLAVEDER	}
ELSE	no
LLAVEIZQ	{
ID	a
OPASIGNACION	=



ANALIZADOR SINTACTICO

DESCRIPCION DEL PROBLEMA

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

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)

LexerCup.flex

Este código define un analizador léxico (lexer) en Java utilizando JFlex, una herramienta que genera analizadores léxicos basados en expresiones regulares. El propósito principal del lexer es analizar un flujo de entrada (por ejemplo, un archivo de código fuente) y dividirlo en tokens reconocibles que posteriormente genera un archivo llamado sym.java para poder ser usado en analizador sintáctico y haya una conexión con los datos de entrada del Analizador Léxico

```
1 package codigo;
2 import java_cup.runtime.Symbol;
3 %%
4 %class LexerCup
5 %type java_cup.runtime.Symbol
6 %cup
7 %full
8 %line
9 %char
10
11 L = [a-zA-Z_]+
12 D = [0-9]+
13 WHITE = [ \t\r\n]+
14
15 %}
16 private Symbol symbol(int type, Object value){
17     return new Symbol(type, yyline, yycolumn, value);
18 }
19 private Symbol symbol(int type){
20     return new Symbol(type, yyline, yycolumn);
21 }
22 %}
23
24 %%
25
26 (WHITE) /* Ignora espacios y tabulaciones */
27 //".*" /* Comentario de linea */ return new Symbol(sym.COMENTARIOLINEA, yychar, yyline, yytext());
28 /*([^\r\n]|\r\n)*"[^\r\n]*"/* Comentario de bloque */ return new Symbol(sym.COMENTARIOBLOQUE, yychar, yyline, yytext());
29 "+|"-|*|"/|^~|^%" { return new Symbol(sym.OPARITMETICO, yychar, yyline, yytext()); }
30 "="|"+"|"=="|"/"=" { return new Symbol(sym.OPASIGNACION, yychar, yyline, yytext()); }
31 "==|"|"="|"<"|">"|"<=" { return new Symbol(sym.OPRELACIONALES, yychar, yyline, yytext()); }
32 "&&|!"|! { return new Symbol(sym.OPLOGICOS, yychar, yyline, yytext()); }
33 "+|"|-|" { return new Symbol(sym.OPINCREMENTO, yychar, yyline, yytext()); }
34 "entero"|"caracter"|"cadena"|"decimalLargo"|"decimalCorto" { return new Symbol(sym.TDATO, yychar, yyline, yytext()); }
35 "si" { return new Symbol(sym.IF, yychar, yyline, yytext()); }
36 "sino" { return new Symbol(sym.ELSEIF, yychar, yyline, yytext()); }
37 "no" { return new Symbol(sym.ELSE, yychar, yyline, yytext()); }
```



MANUAL DE PRÁCTICAS



```
38 "lista" { return new Symbol(sym.SWITCH, yychar, yyline, yytext()); }
39 "caso" { return new Symbol(sym.CASE, yychar, yyline, yytext()); }
40 "terminar" { return new Symbol(sym.BREAK, yychar, yyline, yytext()); }
41 "devolver" { return new Symbol(sym.RETURN, yychar, yyline, yytext()); }
42 "hacer" { return new Symbol(sym.DO, yychar, yyline, yytext()); }
43 "mientras" { return new Symbol(sym.WHILE, yychar, yyline, yytext()); }
44 "vacio" { return new Symbol(sym.VOID, yychar, yyline, yytext()); }
45 "imprimir" { return new Symbol(sym.PRINTF, yychar, yyline, yytext()); }
46 "para" { return new Symbol(sym.FOR, yychar, yyline, yytext()); }
47 "clase" { return new Symbol(sym.CLASS, yychar, yyline, yytext()); }
48 "verdadero"|"falso" { return new Symbol(sym.BOOLEANO, yychar, yyline, yytext()); }
49 "#" { return new Symbol(sym.GATO, yychar, yyline, yytext()); }
50 "predeterminado" { return new Symbol(sym.DEFAULT, yychar, yyline, yytext()); }
51 "(" { return new Symbol(sym.PARENTESISIZQ, yychar, yyline, yytext()); }
52 ")" { return new Symbol(sym.PARENTESISDER, yychar, yyline, yytext()); }
53 "{" { return new Symbol(sym.LLAVEIZQ, yychar, yyline, yytext()); }
54 ")" { return new Symbol(sym.LLAVEDER, yychar, yyline, yytext()); }
55 "[" { return new Symbol(sym.CORCHETEIZQ, yychar, yyline, yytext()); }
56 "]" { return new Symbol(sym.CORCHETEDER, yychar, yyline, yytext()); }
57 ";" { return new Symbol(sym.PUNTOYCOMA, yychar, yyline, yytext()); }
58 "," { return new Symbol(sym.COMA, yychar, yyline, yytext()); }
59 ":" { return new Symbol(sym.DOSPUNTOS, yychar, yyline, yytext()); }
60 "." { return new Symbol(sym.PUNTO, yychar, yyline, yytext()); }
61 "\"\"\" { return new Symbol(sym.COMILLADOBLE, yychar, yyline, yytext()); }
62 '\"' { return new Symbol(sym.COMILLA, yychar, yyline, yytext()); }

63

64 {L}({L}|{D})* { return new Symbol(sym.ID, yychar, yyline, yytext()); }
65 ("?"?{D}+)."({D}*)|("?"?{D}+) { return new Symbol(sym.NUM, yychar, yyline, yytext()); }
66 "\\"[^\\"]*\\" { return new Symbol(sym.CADENA, yychar, yyline, yytext()); }
67

68 ".*" { return new Symbol(sym.ERROR, yychar, yyline, yytext()); }
```



LexerCup.java

LexerCup.java se genera mediante el LexerCup.flex lo que este genera es un autómata finito, con las tablas de transiciones para que pueda reconocer los tokens definidos para que puedan ser usados en el sintáctico.

Analizador_Lexico.java X lexer.flex X Lexer.java X LexerCup.flex X LexerCup.java X

Source History

```
1  /* The following code was generated by JFlex 1.4.3 on 06/01/25, 11:46 */
2
3  package codigo;
4  import java_cup.runtime.Symbol;
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/LexerCup.flex</tt>
11  */
12  class LexerCup implements java_cup.runtime.Scanner {
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_BUFFERSIZE = 16384;
19
20  /** lexical states */
21  public static final int YYINITIAL = 0;
22
23  /**
24   * ZZ_LEXSTATE[1] 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 char [] ZZ_CMAP = {
37      0, 0, 0, 0, 0, 0, 0, 0, 3, 5, 0, 0, 3, 0, 0,
```

1:1 INS Windows (CRLF)



MANUAL DE PRACTICAS





MANUAL DE PRACTICAS





MANUAL DE PRACTICAS



The screenshot shows a Java code editor with the file `Analizador_Lexico.java` open. The code is part of a lexical analyzer implementation. It includes several static methods for unpacking row maps and a transition table. The code uses various escape sequences and character literals.

```
111     "\u0\u0fca\u0\u57\u0\u0ff9\u0\u57\u0\u57\u0\u1028\u0\u57\u0\u57"+  
112     "\u0\u057\u0\u108\u0\u105\u0\u10e4\u0\u113\u0\u1142\u0\u1171\u0\u11a0"+  
113     "\u0\u11cf\u0\u11fe\u0\u122d\u0\u125c\u0\u128b\u0\u12ba\u0\u12e9\u0\u1318"+  
114     "\u0\u57\u0\u1347\u0\u1376\u0\u57\u0\u57\u0\u13a5\u0\u13d4"+  
115     "\u0\u1403\u0\u1432\u0\u1461\u0\u1490\u0\u14bf\u0\u14ee\u0\u151d\u0\u154c"+  
116     "\u0\u57";  
117  
118     private static int [] zzUnpackRowMap() {  
119         int [] result = new int[153];  
120         int offset = 0;  
121         offset = zzUnpackRowMap(ZZ_ROWMAP_PACKED_0, offset, result);  
122         return result;  
123     }  
124  
125     private static int zzUnpackRowMap(String packed, int offset, int [] result) {  
126         int i = 0; /* index in packed string */  
127         int j = offset; /* index in unpacked array */  
128         int l = packed.length();  
129         while (i < l) {  
130             int high = packed.charAt(i++) << 16;  
131             result[j++] = high | packed.charAt(i++);  
132         }  
133         return j;  
134     }  
135  
136     /**  
137      * The transition table of the DFA  
138      */  
139     private static final int [] ZZ_TRANS = zzUnpackTrans();  
140  
141     private static final String ZZ_TRANS_PACKED_0 =  
142     "\u1\u0\u1\u2\u1\u3\u1\u4\u1\u5\u1\u4\u1\u6\u1\u7"+  
143     "\u1\u10\u1\u11\u1\u12\u1\u13\u1\u14\u1\u15\u1\u16\u1\u17"+  
144     "\u1\u20\u1\u21\u2\u21\u1\u22\u1\u2\u1\u23\u1\u24\u1\u25"+  
145     "\u1\u26\u3\u2\u1\u27\u1\u30\u1\u31\u1\u32\u1\u33\u1\u34"+  
146     "\u1\u35\u1\u36\u1\u37\u1\u40\u1\u41\u1\u42\u1\u43\u1\u44"+  
147     "\u1\u45\u1\u46\u1\u47\u1\u50\u1\u0\u2\u2\u1\u4\u0\u23\u2"+
```



MANUAL DE PRACTICAS



```
148 " \17\0\1\3\5\1\0\1\5\1\5\0\1\4\1\0\1\4 "+  
149 " \55\0\1\52\1\0\1\53\3\0\1\54\56\0\1\54 "+  
150 " \53\0\1\55\2\0\1\54\14\0\1\3\5\0\1\55 "+  
151 " \1\0\1\54\135\0\1\56\61\0\1\57\57\0\1\57 "+  
152 " \41\0\2\2\14\0\1\2\1\60\21\2\16\0\2\2 "+  
153 " \14\0\4\2\1\61\16\2\16\0\2\2\14\0\1\62 "+  
154 " \22\2\16\0\2\2\14\0\6\2\1\63\3\2\1\64 "+  
155 " \10\2\16\0\2\2\14\0\1\65\22\2\16\0\2\2 "+  
156 " \14\0\11\2\1\66\11\2\16\0\2\2\14\0\10\2 "+  
157 " \1\67\12\2\16\0\2\2\14\0\10\2\1\70\12\2 "+  
158 " \16\0\2\2\14\0\10\2\1\71\12\2\16\0\2\2 "+  
159 " \14\0\1\72\5\2\1\73\14\2\16\0\2\2\14\0 "+  
160 " \6\2\1\74\14\2\16\0\2\2\14\0\3\2\1\75 "+  
161 " \21\2\1\76\14\2\16\0\2\2\14\0\6\2\1\77 "+  
162 " \14\2\23\0\1\100\50\0\55\101\1\102\1\101\2\0 "+  
163 " \1\51\54\0\5\52\1\0\51\52\6\53\1\103\50\53 "+  
164 " \1\0\2\2\14\0\2\2\1\104\20\2\16\0\2\2 "+  
165 " \14\0\3\2\1\105\17\2\16\0\2\2\14\0\3\2 "+  
166 " \1\106\3\2\1\107\6\2\1\110\4\2\16\0\2\2 "+  
167 " \14\0\6\2\1\111\14\2\16\0\2\2\14\0\5\2 "+  
168 " \1\112\11\2\1\113\3\2\16\0\2\2\14\0\21\2 "+  
169 " \1\114\1\2\16\0\2\2\14\0\1\115\22\2\16\0 "+  
170 " \2\2\14\0\16\2\1\116\4\2\16\0\2\2\14\0 "+  
171 " \1\2\1\17\21\2\16\0\2\2\14\0\3\2\1\120 "+  
172 " \17\2\16\0\2\2\14\0\5\2\1\121\15\2\16\0 "+  
173 " \2\2\14\0\5\2\1\122\15\2\16\0\2\2\14\0 "+  
174 " \1\123\22\2\16\0\2\2\14\0\3\2\1\124\17\2 "+  
175 " \16\0\2\2\14\0\12\2\1\125\10\2\15\0\4\53 "+  
176 " \1\126\52\53\1\0\2\2\14\0\1\127\22\2\16\0 "+  
177 " \2\2\14\0\11\2\1\130\11\2\16\0\2\2\14\0 "+  
178 " \6\2\1\131\14\2\16\0\2\2\14\0\1\132\22\2 "+  
179 " \16\0\2\2\14\0\4\2\1\133\16\2\16\0\2\2 "+  
180 " \14\0\16\2\1\134\4\2\16\0\2\2\14\0\10\2 "+  
181 " \1\135\12\2\16\0\2\2\14\0\4\2\1\136\16\2 "+  
182 " \16\0\2\2\14\0\3\2\1\137\17\2\16\0\2\2 "+  
183 " \14\0\12\1\140\21\2\16\0\2\2\14\0\2\2 "+  
184 " \1\141\20\2\16\0\2\2\14\0\4\2\1\142\16\2 "+
```



MANUAL DE PRACTICAS



```
185      "\\"16\\0\\2\\2\\14\\0\\7\\2\\1\\143\\13\\2\\16\\0\\2\\2\\\"+
186      "\\"14\\0\\10\\2\\1\\144\\12\\2\\16\\0\\2\\2\\14\\0\\1\\145\\\"+
187      "\\"2\\2\\16\\0\\2\\2\\14\\0\\7\\2\\1\\146\\13\\2\\16\\0\\\"+
188      "\\"2\\2\\14\\0\\6\\2\\1\\147\\14\\2\\16\\0\\2\\2\\14\\0\\\"+
189      "\\"16\\2\\1\\150\\4\\2\\16\\0\\2\\2\\14\\0\\3\\2\\1\\151\\\"+
190      "\\"17\\2\\16\\0\\2\\2\\14\\0\\10\\2\\1\\152\\12\\2\\16\\0\\\"+
191      "\\"2\\2\\14\\0\\5\\2\\1\\153\\15\\2\\16\\0\\2\\2\\14\\0\\\"+
192      "\\"1\\2\\1\\154\\21\\2\\16\\0\\2\\2\\14\\0\\1\\155\\22\\2\\\"+
193      "\\"16\\0\\2\\2\\14\\0\\11\\2\\1\\156\\11\\2\\16\\0\\2\\2\\\"+
194      "\\"14\\0\\12\\2\\1\\157\\10\\2\\16\\0\\2\\2\\14\\0\\10\\2\\\"+
195      "\\"1\\160\\12\\2\\16\\0\\2\\2\\14\\0\\2\\2\\1\\161\\20\\2\\\"+
196      "\\"16\\0\\2\\2\\14\\0\\6\\2\\1\\162\\14\\2\\16\\0\\2\\2\\\"+
197      "\\"14\\0\\6\\2\\1\\163\\14\\2\\16\\0\\2\\2\\14\\0\\4\\2\\\"+
198      "\\"1\\164\\16\\2\\16\\0\\2\\2\\14\\0\\3\\2\\1\\165\\17\\2\\\"+
199      "\\"16\\0\\2\\2\\14\\0\\1\\166\\22\\2\\16\\0\\2\\2\\14\\0\\\"+
200      "\\"4\\2\\1\\167\\16\\2\\16\\0\\2\\2\\14\\0\\4\\2\\1\\170\\\"+
201      "\\"16\\2\\16\\0\\2\\2\\14\\0\\1\\2\\1\\171\\21\\2\\16\\0\\\"+
202      "\\"2\\2\\14\\0\\2\\1\\172\\20\\2\\16\\0\\2\\2\\14\\0\\\"+
203      "\\"6\\2\\1\\170\\14\\2\\16\\0\\2\\2\\14\\0\\6\\2\\1\\173\\\"+
204      "\\"14\\2\\16\\0\\2\\2\\14\\0\\17\\2\\1\\174\\3\\2\\16\\0\\\"+
205      "\\"2\\2\\14\\0\\11\\2\\1\\175\\11\\2\\16\\0\\2\\2\\14\\0\\\"+
206      "\\"3\\2\\1\\176\\17\\2\\16\\0\\2\\2\\14\\0\\7\\2\\1\\177\\\"+
207      "\\"13\\2\\16\\0\\2\\2\\14\\0\\2\\2\\1\\200\\20\\2\\16\\0\\\"+
208      "\\"2\\2\\14\\0\\6\\2\\1\\201\\14\\2\\16\\0\\2\\2\\14\\0\\\"+
209      "\\"1\\202\\22\\2\\16\\0\\2\\2\\14\\0\\12\\2\\1\\203\\10\\2\\\"+
210      "\\"16\\0\\2\\2\\14\\0\\1\\204\\22\\2\\16\\0\\2\\2\\14\\0\\\"+
211      "\\"10\\2\\1\\205\\12\\2\\16\\0\\2\\2\\14\\0\\6\\2\\1\\206\\\"+
212      "\\"14\\2\\16\\0\\2\\2\\14\\0\\1\\207\\22\\2\\16\\0\\2\\2\\\"+
213      "\\"14\\0\\1\\210\\22\\2\\16\\0\\2\\2\\14\\0\\3\\2\\1\\211\\\"+
214      "\\"17\\2\\16\\0\\2\\2\\14\\0\\3\\2\\1\\170\\17\\2\\16\\0\\\"+
215      "\\"2\\2\\14\\0\\3\\2\\1\\212\\1\\2\\1\\213\\5\\2\\16\\0\\\"+
216      "\\"2\\2\\14\\0\\3\\2\\1\\214\\17\\2\\16\\0\\2\\2\\14\\0\\\"+
217      "\\"3\\2\\1\\215\\17\\2\\16\\0\\2\\2\\14\\0\\16\\2\\1\\216\\\"+
218      "\\"4\\2\\16\\0\\2\\2\\14\\0\\3\\2\\1\\150\\17\\2\\16\\0\\\"+
219      "\\"2\\2\\14\\0\\3\\2\\1\\217\\17\\2\\16\\0\\2\\2\\14\\0\\\"+
220      "\\"6\\2\\1\\220\\14\\2\\16\\0\\2\\2\\14\\0\\4\\2\\1\\221\\\"+
221      "\\"16\\2\\16\\0\\2\\2\\14\\0\\11\\2\\1\\222\\11\\2\\16\\0\\\"+
```



MANUAL DE PRACTICAS





LexerCup.java X

Source History

```
259     "Error: could not match input",
260     "Error: pushback value was too large"
261 };
262
263 /**
264 * ZZ_ATTRIBUTE[aState] contains the attributes of state <code>aState</code>
265 */
266 private static final int [] ZZ_ATTRIBUTE = zzUnpackAttribute();
267
268 private static final String ZZ_ATTRIBUTE_PACKED_0 =
269     "\\"1\\0\\7\\1\\A\\1\\1\\1\\1\\0\\1\\1\\2\\0\\15\\1"+  

270     "\\"12\\11\\2\\1\\1\\1\\1\\2\\1\\1\\0\\4\\11\\20\\1\\1\\11"+  

271     "\\"1\\0\\1\\11\\1\\0\\22\\1\\1\\11\\103\\1";
272
273 private static int [] zzUnpackAttribute() {
274     int [] result = new int[153];
275     int offset = 0;
276     offset = zzUnpackAttribute(ZZ_ATTRIBUTE_PACKED_0, offset, result);
277     return result;
278 }
279
280 private static int zzUnpackAttribute(String packed, int offset, int [] result) {
281     int i = 0; /* index in packed string */
282     int j = offset; /* index in unpacked array */
283     int l = packed.length();
284     while (i < l) {
285         int count = packed.charAt(i++);
286         int value = packed.charAt(i++);
287         do result[j++] = value; while (--count > 0);
288     }
289     return j;
290 }
291
292 /** the input device */
293 private java.io.Reader zzReader;
294
295 /** the current state of the DFA */
```

1:1 | INS Windows (CRLF)



MANUAL DE PRACTICAS



The screenshot shows the Java code for `LexxCup.java` in an IDE. The code defines several private fields for managing the DFA state, lexical state, buffer, and text position. It also includes annotations for `yytext()`, `yyline`, and `yychar`. The code is annotated with numerous Javadoc-style comments describing the purpose of each variable and its usage.

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



MANUAL DE PRÁCTICAS



The screenshot shows a Java code editor window with the file "LexerCup.java" open. The code is a lexer implementation for a parser generator. It includes private fields for tracking the start of the line (zzAtBOL) and end-of-file (zzAtEOF), and a boolean for whether the user-EOF code has been executed (zzEOFDone). The code defines two symbol creation methods: one taking type and value, and another taking only type. It also contains two constructor implementations: one for a Reader input stream and one for an InputStream input stream, both initializing the zzReader field.

```
LexerCup.java X
Source History | I O V S M F G B D E L C P R T Z Y
332  private boolean zzAtBOL = true;
333  /**
334   * zzAtEOF == true <=> the scanner is at the EOF */
335  private boolean zzAtEOF;
336
337  /**
338   * ** denotes if the user-EOF-code has already been executed */
339  private boolean zzEOFDone;
340
341  /* user code: */
342  private Symbol symbol(int type, Object value){
343      return new Symbol(type, yyline, yycolumn, value);
344  }
345  private Symbol symbol(int type){
346      return new Symbol(type, yyline, yycolumn);
347  }
348
349
350 /**
351  * Creates a new scanner
352  * There is also a java.io.InputStream version of this constructor.
353  *
354  * @param in the java.io.Reader to read input from.
355  */
356 LexerCup(java.io.Reader in) {
357     this.zzReader = in;
358 }
359
360 /**
361  * Creates a new scanner.
362  * There is also java.io.Reader version of this constructor.
363  *
364  * @param in the java.io.InputStream to read input from.
365  */
366 LexerCup(java.io.InputStream in) {
367     this(new java.io.InputStreamReader(in));
368 }
```



The screenshot shows a Java code editor window with the file 'LexerCup.java' open. The code is a Java class with various methods and comments. The editor has a toolbar at the top with icons for file operations like new, open, save, and cut/paste. The status bar at the bottom right shows '1:1' and 'INS Windows (CRLF)'. The code itself is as follows:

```
376     * @exception    java.io.IOException  if any I/O-Error occurs
377     */
378     private boolean zzRefill() throws java.io.IOException {
379
380         /* first: make room (if you can) */
381         if (zzStartRead > 0) {
382             System.arraycopy(zzBuffer, zzStartRead,
383                             zzBuffer, 0,
384                             zzEndRead-zzStartRead);
385
386             /* translate stored positions */
387             zzEndRead-= zzStartRead;
388             zzCurrentPos-= zzStartRead;
389             zzMarkedPos-= zzStartRead;
390             zzStartRead = 0;
391         }
392
393         /* is the buffer big enough? */
394         if (zzCurrentPos >= zzBuffer.length) {
395             /* if not: blow it up */
396             char newBuffer[] = new char[zzCurrentPos*2];
397             System.arraycopy(zzBuffer, 0, newBuffer, 0, zzBuffer.length);
398             zzBuffer = newBuffer;
399         }
400
401         /* finally: fill the buffer with new input */
402         int numRead = zzReader.read(zzBuffer, zzEndRead,
403                                     zzBuffer.length-zzEndRead);
404
405         if (numRead > 0) {
406             zzEndRead+= numRead;
407             return false;
408         }
409         // unlikely but not impossible: read 0 characters, but not at end of stream
410         if (numRead == 0) {
411             int c = zzReader.read();
412             if (c < 0)
```



The screenshot shows a Java code editor window with the file "LexerCup.java" open. The code is a part of a lexer implementation, specifically for handling input streams. It includes methods for closing the input stream, resetting the scanner to a new input stream, and returning the current lexical state. The code uses annotations like `throws java.io.IOException` and `@param reader`.

```
424 /**
425  * Closes the input stream.
426 */
427 public final void yyclose() throws java.io.IOException {
428     zzAtEOF = true; /* indicate end of file */
429     zzEndRead = zzStartRead; /* invalidate buffer */
430
431     if (zzReader != null)
432         zzReader.close();
433 }
434
435
436
437 /**
438  * Resets the scanner to read from a new input stream.
439  * Does not close the old reader.
440  *
441  * All internal variables are reset, the old input stream
442  * <b>cannot</b> be reused (internal buffer is discarded and lost).
443  * Lexical state is set to <tt>ZZ_INITIAL</tt>.
444  *
445  * @param reader the new input stream
446 */
447 public final void yyreset(java.io.Reader reader) {
448     zzReader = reader;
449     zzAtBOL = true;
450     zzAtEOF = false;
451     zzEODDone = false;
452     zzEndRead = zzStartRead = 0;
453     zzCurrentPos = zzMarkedPos = 0;
454     yyline = yychar = yycolumn = 0;
455     zzLexicalState = YYINITIAL;
456 }
457
458
459 /**
460  * Returns the current lexical state.
```



MANUAL DE PRÁCTICAS



LexerCup.java X

Source History

```
461     */
462     public final int yystate() {
463         return zzLexicalState;
464     }
465
466
467     /**
468      * Enters a new lexical state
469      *
470      * @param newState the new lexical state
471      */
472     public final void yybegin(int newState) {
473         zzLexicalState = newState;
474     }
475
476
477     /**
478      * Returns the text matched by the current regular expression.
479      */
480     public final String yytext() {
481         return new String( zzBuffer, zzStartRead, zzMarkedPos-zzStartRead );
482     }
483
484
485     /**
486      * Returns the character at position <tt>pos</tt> from the
487      * matched text.
488      *
489      * It is equivalent to yytext().charAt(pos), but faster
490      *
491      * @param pos the position of the character to fetch.
492      *           A value from 0 to yylength()-1.
493      *
494      * @return the character at position pos
495      */
496     public final char yycharat(int pos) {
497         return zzBuffer[zzStartRead+pos];
498     }
499
500 }
```

1:1 INS Windows (CRLF)



LexerCup.java X

Source History

```
498 }
499
500
501 /**
502 * Returns the length of the matched text region.
503 */
504 public final int yylength() {
505     return zzMarkedPos-zzStartRead;
506 }
507
508
509 /**
510 * Reports an error that occurred while scanning.
511 *
512 * In a wellformed scanner (no or only correct usage of
513 * yypushback(int) and a match-all fallback rule) this method
514 * will only be called with things that "Can't Possibly Happen".
515 * If this method is called, something is seriously wrong
516 * (e.g. a JFlex bug producing a faulty scanner etc.).
517 *
518 * Usual syntax/scanner level error handling should be done
519 * in error fallback rules.
520 *
521 * @param errorCode the code of the errormessage to display
522 */
523 private void zzScanError(int errorCode) {
524     String message;
525     try {
526         message = ZZ_ERROR_MSG[errorCode];
527     }
528     catch (ArrayIndexOutOfBoundsException e) {
529         message = ZZ_ERROR_MSG[ZZ_UNKNOWN_ERROR];
530     }
531
532     throw new Error(message);
533 }
534 }
```



The screenshot shows a Java code editor window with the file "LexerCup.java" open. The code is part of a lexer implementation. It includes several methods: `yypushback`, `zzDoEOF`, and `yyread`. The code uses annotations like `@param` and `@return` to describe parameters and return values. The editor has a toolbar at the top and a status bar at the bottom indicating "INS Windows (CRLF)".

```
534
535
536 /**
537 * Pushes the specified amount of characters back into the input stream.
538 *
539 * They will be read again by then next call of the scanning method
540 *
541 * @param number the number of characters to be read again.
542 * This number must not be greater than yylength()!
543 */
544 public void yypushback(int number) {
545     if ( number > yylength() )
546         zzScanError(ZZ_PUSHBACK_2BIG);
547
548     zzMarkedPos -= number;
549 }
550
551
552 /**
553 * Contains user EOF-code, which will be executed exactly once,
554 * when the end of file is reached
555 */
556 private void zzDoEOF() throws java.io.IOException {
557     if (!zzEOFDone) {
558         zzEOFDone = true;
559         yyclose();
560     }
561 }
562
563
564 /**
565 * Resumes scanning until the next regular expression is matched,
566 * the end of input is encountered or an I/O-Error occurs.
567 *
568 * @return      the next token
569 * @exception   java.io.IOException if any I/O-Error occurs
570 */
```



LexerCup.java X

Source History

```
public java_cup.runtime.Symbol next_token() throws java.io.IOException {
    int zzInput;
    int zzAction;

    // cached fields:
    int zzCurrentPosL;
    int zzMarkedPosL;
    int zzEndReadL = zzEndRead;
    char [] zzBufferL = zzBuffer;
    char [] zzCMapL = ZZ_CMAP;

    int [] zzTransL = ZZ_TRANS;
    int [] zzRowMapL = ZZ_ROWMAP;
    int [] zzAttrL = ZZ_ATTRIBUTE;

    while (true) {
        zzMarkedPosL = zzMarkedPos;

        yychar+= zzMarkedPosL-zzStartRead;

        boolean zzR = false;
        for (zzCurrentPosL = zzStartRead; zzCurrentPosL < zzMarkedPosL;
             zzCurrentPosL++) {
            switch (zzBufferL[zzCurrentPosL]) {
                case '\u000B':
                case '\u000C':
                case '\u0085':
                case '\u2028':
                case '\u2029':
                    yyline++;
                    zzR = false;
                    break;
                case '\r':
                    yyline++;
                    zzR = true;
                    break;
                case '\n':

```



The screenshot shows a Java code editor window titled "LexerCup.java X". The code is a snippet from a lexer implementation, specifically handling character peeking and line counting. The code uses standard Java syntax with annotations for line numbers (608 to 644) and color-coded keywords and comments.

```
608     if (zzR)
609         zzR = false;
610     else {
611         yyline++;
612     }
613     break;
614 default:
615     zzR = false;
616 }
617 }

618 if (zzR) {
619     // peek one character ahead if it is \n (if we have counted one line too much)
620     boolean zzPeek;
621     if (zzMarkedPosL < zzEndReadL)
622         zzPeek = zzBufferL[zzMarkedPosL] == '\n';
623     else if (zzAtEOF)
624         zzPeek = false;
625     else {
626         boolean eof = zzRefill();
627         zzEndReadL = zzEndRead;
628         zzMarkedPosL = zzMarkedPos;
629         zzBufferL = zzBuffer;
630         if (eof)
631             zzPeek = false;
632         else
633             zzPeek = zzBufferL[zzMarkedPosL] == '\n';
634     }
635     if (zzPeek) yyline--;
636 }
637 zzAction = -1;

638 zzCurrentPosL = zzCurrentPos = zzStartRead = zzMarkedPosL;

639 zzState = ZZ_LEXSTATE[zzLexicalState];
640
641
642
643
644
```

1:1 | INS Windows (CRLF)



The screenshot shows a Java code editor window titled "LexerCup.java X". The code is a snippet from a lexer implementation, specifically the `zzForAction` loop. The code handles input reading, end-of-file detection, buffer management, and state transitions. It includes comments explaining the logic for handling cached positions and new buffers. The code uses standard Java syntax with imports and class definitions visible at the top of the editor window.

```
644
645     zzForAction: {
646         while (true) {
647
648             if (zzCurrentPosL < zzEndReadL)
649                 zzInput = zzBufferL[zzCurrentPosL++];
650             else if (zzAtEOF) {
651                 zzInput = YYEOF;
652                 break zzForAction;
653             }
654             else {
655                 // store back cached positions
656                 zzCurrentPos = zzCurrentPosL;
657                 zzMarkedPos = zzMarkedPosL;
658                 boolean eof = zzRefill();
659                 // get translated positions and possibly new buffer
660                 zzCurrentPosL = zzCurrentPos;
661                 zzMarkedPosL = zzMarkedPos;
662                 zzBufferL = zzBuffer;
663                 zzEndReadL = zzEndRead;
664                 if (eof) {
665                     zzInput = YYEOF;
666                     break zzForAction;
667                 }
668                 else {
669                     zzInput = zzBufferL[zzCurrentPosL++];
670                 }
671             }
672             int zzNext = zzTransL[ zzRowMapL[zzState] + zzCMapL[zzInput] ];
673             if (zzNext == -1) break zzForAction;
674             zzState = zzNext;
675
676             int zzAttributes = zzAttrL[zzState];
677             if ( (zzAttributes & 1) == 1 ) {
678                 zzAction = zzState;
679                 zzMarkedPosL = zzCurrentPosL;
680                 if ( (zzAttributes & 8) == 8 ) break zzForAction;
681             }
682         }
683     }
684 }
```

1:1 | INS Windows (CRLF)



The screenshot shows a Java code editor window titled "LexerCup.java X". The code is a lexer implementation, likely for a C-like language, using the Cup parser generator. It handles various tokens such as IF, BREAK, CLASS, TDATO, PRINTF, OPRELACIONALES, and DOSPUNTOS. The code uses a switch statement to map action numbers to symbol definitions. The editor interface includes tabs for "Source" and "History", and a toolbar with various icons for file operations.

```
681     }
682
683     }
684 }
685
686 // store back cached position
687 zzMarkedPos = zzMarkedPosL;
688
689 switch (zzAction < 0 ? zzAction : ZZ_ACTION[zzAction]) {
690     case 24:
691         { return new Symbol(sym.IF, yychar, yyline, yytext()); }
692     case 42: break;
693     case 37:
694         { return new Symbol(sym.BREAK, yychar, yyline, yytext()); }
695     case 43: break;
696     case 31:
697         { return new Symbol(sym.CLASS, yychar, yyline, yytext()); }
698     case 44: break;
699     case 36:
700         { return new Symbol(sym.TDATO, yychar, yyline, yytext()); }
701     case 45: break;
702     case 39:
703         { return new Symbol(sym.PRINTF, yychar, yyline, yytext()); }
704     case 46: break;
705     case 6:
706         { return new Symbol(sym.OPRELACIONALES, yychar, yyline, yytext()); }
707     case 47: break;
708     case 16:
709         { return new Symbol(sym.DOSPUNTOS, yychar, yyline, yytext()); }
710     case 48: break;
711 }
```

1:1 | INS Windows (CRLF)



```
LexerCup.java X
Source History |< > □
718     case 14:
719         { return new Symbol(sym.PUNTOYCOMA, yychar, yyline, yytext());
720     }
721     case 49: break;
722     case 28:
723         { return new Symbol(sym.CASE, yychar, yyline, yytext());
724     }
725     case 50: break;
726     case 38:
727         { return new Symbol(sym.RETURN, yychar, yyline, yytext());
728     }
729     case 51: break;
730     case 11:
731         { return new Symbol(sym.LLAVEDER, yychar, yyline, yytext());
732     }
733     case 52: break;
734     case 22:
735         { return new Symbol(sym.OPLOGICOS, yychar, yyline, yytext());
736     }
737     case 53: break;
738     case 4:
739         { return new Symbol(sym.OPARITMETICO, yychar, yyline, yytext());
740     }
741     case 54: break;
742     case 9:
743         { return new Symbol(sym.PARENTESISDER, yychar, yyline, yytext());
744     }
745     case 55: break;
746     case 2:
747         { return new Symbol(sym.NUM, yychar, yyline, yytext());
748     }
749     case 56: break;
750     case 10:
751         { return new Symbol(sym.LLAVEIZQ, yychar, yyline, yytext());
752     }
753     case 57: break;
754     case 8:
```



MANUAL DE PRACTICAS





MANUAL DE PRÁCTICAS



LexerCup.java X

Source History

```
791     { return new Symbol(sym.DO, yychar, yyline, yytext());  
792 }  
793 case 67: break;  
794 case 23:  
795     { return new Symbol(sym.ELSE, yychar, yyline, yytext());  
796 }  
797 case 68: break;  
798 case 26:  
799     { return new Symbol(sym.CADENA, yychar, yyline, yytext());  
800 }  
801 case 69: break;  
802 case 21:  
803     { return new Symbol(sym.OPINCREMENTO, yychar, yyline, yytext());  
804 }  
805 case 70: break;  
806 case 30:  
807     { return new Symbol(sym.FOR, yychar, yyline, yytext());  
808 }  
809 case 71: break;  
810 case 32:  
811     { return new Symbol(sym.SWITCH, yychar, yyline, yytext());  
812 }  
813 case 72: break;  
814 case 3:  
815     { /* Ignora espacios y tabulaciones */  
816 }  
817 case 73: break;  
818 case 25:  
819     { return new Symbol(sym.ERROR, yychar, yyline, yytext());  
820 }  
821 case 74: break;  
822 case 29:  
823     { return new Symbol(sym.ELSEIF, yychar, yyline, yytext());  
824 }  
825 case 75: break;  
826 case 40:  
827     { return new Symbol(sym.WHILE, yychar, yyline, yytext());  
}
```

1:1 INS Windows (CRLF)



LexerCup.java X

Source History

```
827     { return new Symbol(sym.WHILE, yychar, yyline, yytext());  
828 }  
829 case 76: break;  
830 case 17:  
831     { return new Symbol(sym.PUNTO, yychar, yyline, yytext());  
832 }  
833 case 77: break;  
834 case 13:  
835     { return new Symbol(sym.CORCHETEDER, yychar, yyline, yytext());  
836 }  
837 case 78: break;  
838 case 7:  
839     { return new Symbol(sym.GATO, yychar, yyline, yytext());  
840 }  
841 case 79: break;  
842 case 19:  
843     { return new Symbol(sym.COMILLA, yychar, yyline, yytext());  
844 }  
845 case 80: break;  
846 case 18:  
847     { return new Symbol(sym.COMILLADOBLE, yychar, yyline, yytext());  
848 }  
849 case 81: break;  
850 case 12:  
851     { return new Symbol(sym.CORCHETEIZQ, yychar, yyline, yytext());  
852 }  
853 case 82: break;  
854 default:  
855     if (zzInput == YYEOF && zzStartRead == zzCurrentPos) {  
856         zzAtEOF = true;  
857         zzDoEOF();  
858         { return new java_cup.runtime.Symbol(sym.EOF); }  
859     }  
860     else {  
861         zzScanError(ZZ_NO_MATCH);  
862     }  
863 }  
864 }  
865 }
```

LexerCup.java X

Source History

```
845 case 80: break;  
846 case 18:  
847     { return new Symbol(sym.COMILLADOBLE, yychar, yyline, yytext());  
848 }  
849 case 81: break;  
850 case 12:  
851     { return new Symbol(sym.CORCHETEIZQ, yychar, yyline, yytext());  
852 }  
853 case 82: break;  
854 default:  
855     if (zzInput == YYEOF && zzStartRead == zzCurrentPos) {  
856         zzAtEOF = true;  
857         zzDoEOF();  
858         { return new java_cup.runtime.Symbol(sym.EOF); }  
859     }  
860     else {  
861         zzScanError(ZZ_NO_MATCH);  
862     }  
863 }  
864 }  
865 }
```



Sym.java

Sym.java define las constantes de los tokens de nuestro lenguaje con valores enteros importante para que se puedan comunicar entre el léxico y el sintáctico

```
1 //-----
2 // The following code was generated by CUP v0.11b 20160615 (GIT 4ac7450)
3 //-----
4
5 package codigo;
6
7 /**
8  * CUP generated class containing symbol constants. */
9 public class sym {
10 	/* terminals */
11 	public static final int COMILLADOBLE = 35;
12 	public static final int OPARITMETICO = 21;
13 	public static final int DOSPUNTOS = 33;
14 	public static final int ASIGNAR = 6;
15 	public static final int PUNTOYCOMA = 31;
16 	public static final int COMENTARIOBLOQUE = 38;
17 	public static final int TDATO = 40;
18 	public static final int OPASIGNACION = 4;
19 	public static final int CASE = 11;
20 	public static final int LLAVEDER = 28;
21 	public static final int FOR = 18;
22 	public static final int COMENTARIOLINEA = 37;
23 	public static final int CORCHETEIZQ = 29;
24 	public static final int ELSEIF = 19;
25 	public static final int GATO = 43;
26 	public static final int CLASS = 23;
27 	public static final int NUM = 39;
28 	public static final int IF = 8;
29 	public static final int ID = 2;
30 	public static final int PARENTESISIZQ = 25;
31 	public static final int EOF = 0;
32 	public static final int RETURN = 13;
33 	public static final int COMILLA = 36;
34 	public static final int EFECH = 1;
35 	public static final int COMA = 32;
36 	public static final int CADENA = 3;
37 	public static final int BREAK = 12;
```



MANUAL DE PRÁCTICAS



The screenshot shows a Java code editor interface with two tabs: "LexerCup.java" and "sym.java". The "LexerCup.java" tab is active, displaying the following code:

```
38 public static final int VOID = 16;
39 public static final int PRINTF = 17;
40 public static final int ERROR = 7;
41 public static final int SWITCH = 10;
42 public static final int POTENCIA = 5;
43 public static final int CORCHETEDER = 30;
44 public static final int ELSE = 9;
45 public static final int PUNTO = 34;
46 public static final int WHILE = 14;
47 public static final int LLAVEIZQ = 27;
48 public static final int DEFAULT = 42;
49 public static final int OPRELACIONALES = 22;
50 public static final int BOOLEANO = 20;
51 public static final int PARENTESISDER = 26;
52 public static final int OPINCREMENTO = 41;
53 public static final int DO = 15;
54 public static final int OPLOGICOS = 24;
55 public static final String[] terminalNames = new String[] {
56     "EOF",
57     "error",
58     "ID",
59     "CADENA",
60     "OPASIGNACION",
61     "POTENCIA",
62     "ASIGNAR",
63     "ERROR",
64     "IF",
65     "ELSE",
66     "SWITCH",
67     "CASE",
68     "BREAK",
69     "RETURN",
70     "WHILE",
71     "DO",
72     "VOID",
73     "PRINTF",
74     "FOR",
```



The screenshot shows a Java code editor window with two tabs: 'LexerCup.java' and 'sym.java'. The 'LexerCup.java' tab is active, displaying the following Java code:

```
74 "FOR",
75 "ELSEIF",
76 "BOOLEANO",
77 "OPARITMETICO",
78 "OPRELACIONALES",
79 "CLASS",
80 "OPLOGICOS",
81 "PARENTESISIZQ",
82 "PARENTESISDER",
83 "LLAVEIZQ",
84 "LLAVEDER",
85 "CORCHETEIZQ",
86 "CORCHETEDER",
87 "PUNTOYCOMA",
88 "COMA",
89 "DOSPUNTOS",
90 "PUNTO",
91 "COMILLADOBLE",
92 "COMILLA",
93 "COMENTARIOLINEA",
94 "COMENTARIOBLOQUE",
95 "NUM",
96 "TDATO",
97 "OPINCREMENTO",
98 "DEFAULT",
99 "GATO"
};
```

The code defines various tokens (keywords and punctuation) for a parser. The 'sym.java' tab is visible in the background.

Sintax.cup

Este código implementa un parser (analizador sintáctico) en Java CUP para un lenguaje de programación personalizado. Define la gramática del lenguaje usando reglas que describen cómo deben estructurarse las clases, métodos, declaraciones, asignaciones, estructuras de control de flujo, y expresiones. También define terminales (tokens como identificadores, operadores, y palabras clave) y no terminales (estructuras sintácticas complejas). El parser comienza con la construcción INICIO, que describe cómo se declaran las clases y métodos principales, y permite analizar programas completos. Incluye soporte para estructuras de control como if, else, while, for, switch, así como declaraciones de variables y métodos con parámetros. Además, el código incluye manejo de errores sintácticos, rastreando la línea, columna, y el texto relacionado con el error.



MANUAL DE PRACTICAS



```
LexerCup.java x sym.java x Sintax.cup x
Source History | I D G S F T B C E P R M
1 package codigo;
2
3 import java_cup.runtime.Symbol;
4
5 parser code
6 {
7     private Symbol s;
8
9     public void syntax_error(Symbol s) {
10         this.s = s;
11         System.err.println("Syntax error at line: " + s.left + ", column: " + s.right + ", Text: \\" + s.value + "\\");
12     }
13
14     public Symbol getS() {
15         return this.s;
16     }
17 }
18
19 terminal ID,CADENA,OPASIGNACION,POTENCIA,ASIGNAR,ERROR,
20 IF,ELSE,SWITCH,CASE,BREAK,RETURN,WHILE,DO,VOID,PRINTF,FOR,ELSEIF,BOOLEANO,
21 OPARITMETICO,OPRELACIONALES,CLASS,OPLOGICOS,PARENTESISIZQ,PARENTESISDER,LLAVEIZQ,LLAVEDER,
22 CORCHETEIZQ,CORCHETEDER,PUNTOYCOMA,COMA,DOSPUNTOS,PUNTO,COMILLADOBLE,COMILLA,COMENTARIOLINEA,
23 COMENTARIOBLOQUE,NUM,TDATO,OPINCREMENTO,DEFAULT,GATO;
24
25 non terminal INICIO,ELSEIF_ANIDADO,SENTENCIAS,CASOS,CASO,SENTENCIA,DECLARACION,ASIGNACION,CONTROL_FLUJO,
26 EXPRESION,BLOQUE,METODO,METODO_LISTA,TIPO_RETORNO,PARAMETROS_METODO,LISTA_EXPRESIONES,INICIALIZACION_FOR,INCREMENTO_FOR;
27
28 precedence left OPARITMETICO,OPRELACIONALES,OPLOGICOS;
29
30 start with INICIO;
31
32 // INICIO: Clase con m閎odos y/o sentencias
33 INICIO ::= TDATO CLASS ID PARENTESISIZQ PARENTESISDER LLAVEIZQ METODO_LISTA SENTENCIAS LLAVEDER
34     | TDATO CLASS ID PARENTESISIZQ PARENTESISDER LLAVEIZQ SENTENCIAS LLAVEDER
35     | TDATO CLASS ID PARENTESISIZQ PARENTESISDER LLAVEIZQ LLAVEDER
36     | TDATO CLASS ID PARENTESISIZQ PARENTESISDER LLAVEIZQ METODO_LISTA LLAVEDER
37 ;
```



MANUAL DE PRACTICAS



```
// Método o declaración
METODO_LISTA ::= METODO
| METODO METODO_LISTA
;

// Sentencias generales
SENTENCIAS ::= SENTENCIA
| SENTENCIA SENTENCIAS
;
;

// Una sentencia puede ser cualquiera de las siguientes
SENTENCIA ::= COMENTARIOLINEA
| COMENTARIOBLOQUE
| DECLARACION
| ASIGNACION
| CONTROL_FLUJO
| RETURN EXPRESION PUNTOYCOMA
| PRINTF PARENTESISIZQ LISTA_EXPRESIONES PARENTESISDER PUNTOYCOMA
;
;

// Declaración de variables
DECLARACION ::= TDATO ID PUNTOYCOMA
| TDATO ID OPASIGNACION EXPRESION PUNTOYCOMA
;
;

// Declaración de métodos (con el asterisco al inicio)
METODO ::= GATO TIPO_RETORNO ID PARENTESISIZQ PARAMETROS_METODO PARENTESISDER BLOQUE
;
;

// Asignaciones de valores
ASIGNACION ::= ID OPASIGNACION EXPRESION PUNTOYCOMA
;
;

// Definición del tipo de retorno de un método
TIPO_RETORNO ::= TDATO
| VOID
;
;
```



MANUAL DE PRACTICAS



```
LexerCup.java X sym.java X Sintax.cup X
Source History | ☰ 🔍 ↻ ↺ ↻ ↺ 🔍 ☰
75 ;
76
77 // Control de flujo
78 CONTROL_FLUJO ::= IF PARENTESISIZQ EXPRESION PARENTESISDER BLOQUE
79           | IF PARENTESISIZQ EXPRESION PARENTESISDER BLOQUE ELSE BLOQUE
80           | IF PARENTESISIZQ EXPRESION PARENTESISDER BLOQUE ELSEIF_ANIDADO ELSE BLOQUE
81           | WHILE PARENTESISIZQ EXPRESION PARENTESISDER BLOQUE
82           | FOR PARENTESISIZQ INICIALIZACION_FOR EXPRESION PUNTOYCOMA INCREMENTO_FOR PARENTESISDER BLOQUE
83           | SWITCH PARENTESISIZQ EXPRESION PARENTESISDER LLAVEIZQ CASOS LLAVEDER
84           | DO BLOQUE WHILE PARENTESISIZQ EXPRESION PARENTESISDER PUNTOYCOMA
85 ;
86
87 // Lista de else if (sino)
88 ELSEIF_ANIDADO ::= ELSEIF PARENTESISIZQ EXPRESION PARENTESISDER BLOQUE
89           | ELSEIF PARENTESISIZQ EXPRESION PARENTESISDER BLOQUE ELSEIF_ANIDADO
90 ;
91
92 // Definición de los casos dentro de un switch
93 CASOS ::= CASO
94           | CASO CASOS
95           | DEFAULT DOSPUNTOS SENTENCIAS BREAK PUNTOYCOMA
96 ;
97
98 // Definición de un caso individual
99 CASO ::= CASE EXPRESION DOSPUNTOS SENTENCIAS BREAK PUNTOYCOMA
100 ;
101
102 // Inicialización en un bucle for
103 INICIALIZACION_FOR ::= ASIGNACION
104           | DECLARACION
105 ;
106
107 // Incremento en un bucle for
108 INCREMENTO_FOR ::= ID OPINCREMENTO
109 ;
110
111 // Expresiones generales
```

The screenshot shows the IntelliJ IDEA IDE with the Sintax.cup file open in the editor. The code defines a grammar for expressions, including rules for EXPRESION, LISTA_EXPRESIONES, BLOQUE, and PARAMETROS_METODO. The code uses CUP (JavaCC) syntax with annotations like // Se añade soporte para true y false and // Aquí ya está cubierto && y ||.

```
// Expresiones generales
EXPRESION ::= NUM
           | ID
           | CADENA
           | BOOLEANO          // Se añade soporte para true y false
           | EXPRESION OPARITMETICO EXPRESION
           | EXPRESION OPRELACIONALES EXPRESION
           | EXPRESION OPLOGICOS EXPRESION // Aquí ya está cubierto && y ||
           | PARENTESISIZQ EXPRESION PARENTESISDER

;
;

// Lista de expresiones separadas por comas
LISTA_EXPRESIONES ::= EXPRESION
                   | EXPRESION COMA LISTA_EXPRESIONES
;
;

// Bloques de código (cuerpo de sentencias)
BLOQUE ::= LLAVEIZQ SENTENCIAS LLAVEDER
;
;

// Parámetros del método
PARAMETROS_METODO ::= TIPO_RETORNO ID
                     | TIPO_RETORNO ID COMA PARAMETROS_METODO
                     | /* vacío */
```



Sintax.java

Este se genera mediante el Sintax.cup lo que este hace es reconocer las sentencias para que a la hora de colocar un código reconozca esa sintaxis o marque un error.



MANUAL DE PRACTICAS





MANUAL DE PRACTICAS

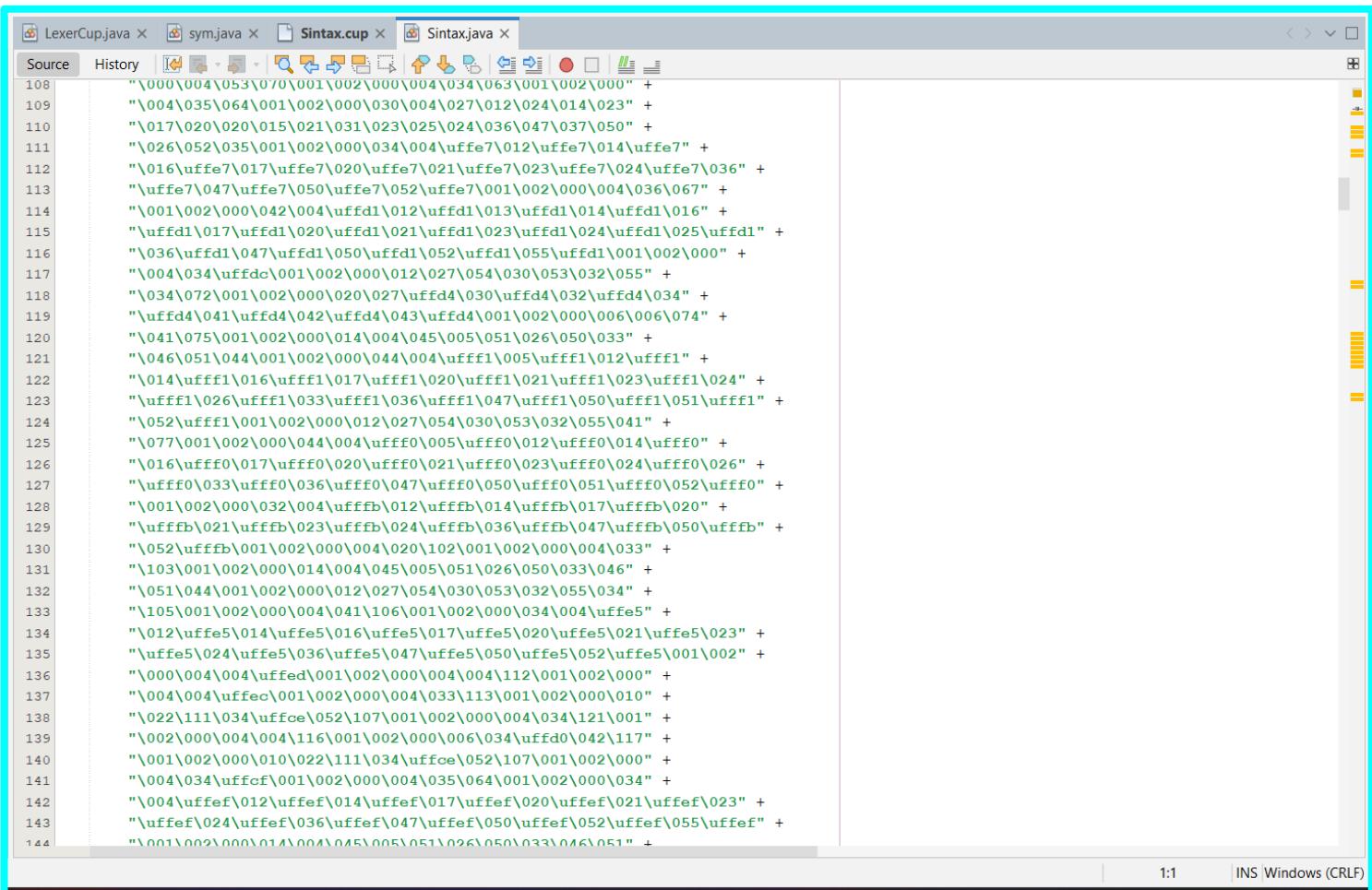


The screenshot shows a Java code editor with several tabs open. The tabs include 'LexerCup.java', 'sym.java', 'Sintax.cup', and 'Syntax.java'. The 'Syntax.java' tab is currently active, displaying a large block of Java code. The code consists of numerous lines of text, each starting with a line number from 72 to 107. The code is heavily annotated with various escape sequences and characters, such as '\u033\046\051\044\001\002\000\004\036\177\001\002\000' and '\u034\004\ufff4\012\ufff4\014\ufff4\016\ufff4\017\ufff4\020\ufff4'. The code is color-coded for readability, with different colors used for comments, strings, and other language constructs.

```
72  "\u033\046\051\044\001\002\000\004\036\177\001\002\000" +
73  "\u034\004\ufff4\012\ufff4\014\ufff4\016\ufff4\017\ufff4\020\ufff4" +
74  "\u021\ufff4\023\ufff4\024\ufff4\036\ufff4\047\ufff4\050\ufff4\052" +
75  "\ufff4\001\002\000\004\033\154\001\002\000\004\033\135" +
76  "\001\002\000\004\033\126\001\002\000\034\004\ufff7\012" +
77  "\ufff7\014\ufff7\016\ufff7\017\ufff7\020\ufff7\021\ufff7\023\ufff7" +
78  "\u024\ufff7\036\ufff7\047\ufff7\050\ufff7\052\ufff7\001\002\000" +
79  "\u004\006\123\001\002\000\006\022\111\052\107\001\002" +
80  "\u000\004\035\064\001\002\000\034\004\ufff5\012\ufff5\014" +
81  "\ufff5\016\ufff5\017\ufff5\020\ufff5\021\ufff5\023\ufff5\024\ufff5" +
82  "\036\ufff5\047\ufff5\050\ufff5\052\ufff5\001\002\000\034\004" +
83  "\ufffc\012\ufffc\014\ufffc\017\ufffc\020\ufffc\021\ufffc\023\ufffc" +
84  "\u024\ufffc\036\ufffc\047\ufffc\050\ufffc\052\ufffc\055\030\001" +
85  "\u002\000\004\002\ufffe\001\002\000\004\004\073\001\002" +
86  "\u000\004\033\040\001\002\000\034\004\ufff8\012\ufff8\014" +
87  "\ufff8\016\ufff8\017\ufff8\020\ufff8\021\ufff8\023\ufff8\024\ufff8" +
88  "\036\ufff8\047\ufff8\050\ufff8\052\ufff8\001\002\000\006\004" +
89  "\u027\052\035\001\002\000\014\004\ufffd\005\ufffd\026\uffdd" +
90  "\u033\ufedd\051\ufedd\001\002\000\014\004\uffde\005\uffde\026" +
91  "\uffde\033\uffde\051\uffde\001\002\000\014\004\045\005\051" +
92  "\u026\050\033\046\051\044\001\002\000\020\027\uffdb\030" +
93  "\uffdb\032\uffdb\034\uffdb\041\uffdb\042\uffdb\043\uffdb\001\002" +
94  "\u000\020\027\uffda\030\uffda\032\uffda\034\uffda\041\uffda\042" +
95  "\uffda\043\uffda\001\002\000\014\004\045\005\051\026\050" +
96  "\u033\046\051\044\001\002\000\012\027\054\030\053\032" +
97  "\u055\041\052\001\002\000\020\027\uffd8\030\uffd8\032\uffd8" +
98  "\u034\uffd8\041\uffd8\042\uffd8\043\uffd8\001\002\000\020\027" +
99  "\uffd9\030\uffd9\032\uffd9\034\uffd9\041\uffd9\042\uffd9\043\uffd9" +
100 "\u001\002\000\004\004\061\001\002\000\014\004\045\005" +
101 "\u051\026\050\033\046\051\044\001\002\000\014\004\045" +
102 "\u005\051\026\050\033\046\051\044\001\002\000\014\004" +
103 "\u045\005\051\026\050\033\046\051\044\001\002\000\020" +
104 "\u027\uffd5\030\uffd5\032\uffd5\034\uffd5\041\uffd5\042\uffd5\043" +
105 "\uffd5\001\002\000\020\027\uffd7\030\uffd7\032\uffd7\034\uffd7" +
106 "\u041\uffd7\042\uffd7\043\uffd7\001\002\000\020\027\uffd6\030" +
107 "\uffd6\032\uffd6\034\uffd6\041\uffd6\042\uffd6\043\uffd6\001\002" +
108 "\u000\004\051\053\030\001\002\000\004\034\063\031\002\000" +
```



MANUAL DE PRÁCTICAS



The screenshot shows a Java code editor with several tabs at the top: LexerCup.java, sym.java, Sintax.cup, Sintax.java, and Sintax.java (active). The code in the editor consists of a single, extremely long string of characters, likely a regular expression or a large character sequence. The string starts with "\000\004\053\070\001\002\000\004\034\063\001\002\000" and continues for approximately 144 lines, ending with "\001\002\000\042\004\uffd1\012\uffd1\013\uffd1\014\uffd1\016" and "\uffd1\017\uffd1\020\uffd1\021\uffd1\023\uffd1\024\uffd1\025\uffd1". The code is written in a monospaced font, with some characters colored green and others black.

1:1

INS Windows (CRLF)



MANUAL DE PRACTICAS



The screenshot shows a Java code editor with several tabs open at the top: LexerCup.java, sym.java, Syntax.cup, Sintax.java, and Sintax.java. The main pane displays a large block of text, likely a grammar or configuration file, with syntax highlighting. The text consists of numerous strings of characters, many of which begin with backslashes and contain sequences like '\001', '\002', etc. The code is heavily annotated with '+' signs, indicating concatenation. The editor interface includes a toolbar with icons for file operations, a search bar, and a status bar at the bottom right showing '1:1' and 'INS Windows (CRLF)'.

```
"\001\002\000\014\004\045\005\051\026\050\033\046\051" +
"\044\001\002\000\012\027\054\030\053\032\055\041\025" +
"\001\002\000\044\004\uffee\005\uffee\012\uffee\014\uffee\016" +
"\uffee\017\uffee\020\uffee\021\uffee\023\uffee\024\uffee\026\uffee" +
"\033\uffee\036\uffee\047\uffee\050\uffee\051\uffee\052\uffee\001" +
"\002\000\014\004\045\005\051\026\050\033\046\051\044" +
"\001\002\000\004\034\l33\001\002\000\014\027\054\030" +
"\053\032\055\034\ufef3\042\131\001\002\000\014\004\045" +
"\005\051\026\050\033\046\051\044\001\002\000\004\034" +
"\ufff2\001\002\000\004\041\134\001\002\000\034\004\ufff2" +
"\012\ufff2\014\ufff2\016\ufff2\017\ufff2\020\ufff2\021\ufff2\023" +
"\ufff2\024\ufff2\036\ufff2\047\ufff2\050\ufff2\052\ufff2\001\002" +
"\000\014\004\045\005\051\026\050\033\046\051\044\001" +
"\002\000\012\027\054\030\053\032\055\034\137\001\002" +
"\000\004\035\064\001\002\000\040\004\uffeb\012\uffeb\013" +
"\141\014\uffeb\016\uffeb\017\uffeb\020\uffeb\021\uffeb\023\uffeb" +
"\024\uffeb\025\142\036\uffeb\047\uffeb\050\uffeb\052\uffeb\001" +
"\002\000\004\035\064\001\002\000\004\033\146\001\002" +
"\000\004\013\144\001\002\000\004\035\064\001\002\000" +
"\034\004\uffe9\012\uffe9\014\uffe9\016\uffe9\017\uffe9\020\uffe9" +
"\021\uffe9\023\uffe9\024\uffe9\036\uffe9\047\uffe9\050\uffe9\052" +
"\uffe9\001\002\000\014\004\045\005\051\026\050\033\046" +
"\051\044\001\002\000\012\027\054\030\053\032\055\034" +
"\150\001\002\000\004\035\064\001\002\000\006\013\uffea" +
"\025\142\001\002\000\004\013\uffe3\001\002\000\034\004" +
"\uffea\012\uffea\014\uffea\016\uffea\017\uffea\020\uffea\021\uffea" +
"\023\uffea\024\uffea\036\uffea\047\uffea\050\uffea\052\uffea\001" +
"\002\000\014\004\045\005\051\026\050\033\046\051\044" +
"\001\002\000\012\027\054\030\053\032\055\034\156\001" +
"\002\000\004\035\157\001\002\000\006\015\162\054\160" +
"\001\002\000\004\043\l73\001\002\000\004\036\172\001" +
"\002\000\014\004\045\005\051\026\050\033\046\051\044" +
"\001\002\000\010\015\162\036\uffe2\054\160\001\002\000" +
"\004\036\uffe1\001\002\000\012\027\054\030\053\032\055" +
"\043\166\001\002\000\030\004\027\012\024\014\023\017" +
"\020\020\015\021\031\023\025\024\036\047\037\050\026" +
```



MANUAL DE PRACTICAS



The screenshot shows a Java code editor with the following details:

- File Tabs:** LexerCup.java X, sym.java X, Sintax.cup X, Sintax.java X
- Toolbar:** Source, History, and various icons for file operations like Open, Save, Print, and Find.
- Code Content:** The code is annotated with several yellow and orange markers:
 - Annotations on lines 180-198 highlight a large block of string concatenations.
 - An annotation on line 200 highlights the start of a block comment.
 - An annotation on line 201 highlights the `action_table()` method.
 - An annotation on line 203 highlights the start of a block comment.
 - An annotation on line 204 highlights the `_reduce_table` field.
 - An annotation on line 205 highlights the `unpackFromStrings` method call.
 - An annotation on line 207 highlights the first part of a long string concatenation.
 - An annotation on line 208 highlights the second part of the same string concatenation.
 - An annotation on line 209 highlights the third part of the same string concatenation.
 - An annotation on line 210 highlights the fourth part of the same string concatenation.
 - An annotation on line 211 highlights the fifth part of the same string concatenation.
 - An annotation on line 212 highlights the sixth part of the same string concatenation.
 - An annotation on line 213 highlights the seventh part of the same string concatenation.
 - An annotation on line 214 highlights the eighth part of the same string concatenation.
 - An annotation on line 215 highlights the ninth part of the same string concatenation.



MANUAL DE PRÁCTICAS



The screenshot shows a Java code editor window with the following details:

- Title Bar:** LexerCup.java X, sym.java X, Sintax.cup X, Sintax.java X
- Toolbar:** Source, History, and various icons for file operations.
- Code Area:** A large block of code in Sintax.java, starting with line 215 and ending at line 250. The code consists of many lines of string literals separated by plus signs (+), representing regular expressions or tokens.
- Status Bar:** 1:1, INS Windows (CRLF)

```
215 " \"002\\001\\001\\000\\002\\001\\001\\000\\004\\017\\107\\001\\001" +
216 " \"000\\004\\014\\100\\001\\001\\000\\002\\001\\001\\000\\006\\015" +
217 " \"032\\016\\077\\001\\001\\000\\002\\001\\001\\000\\002\\001\\001" +
218 " \"000\\002\\001\\001\\000\\002\\001\\001\\000\\010\\010\\040\\011" +
219 " \"041\\022\\042\\001\\001\\000\\002\\001\\001\\000\\002\\001\\001" +
220 " \"000\\004\\013\\046\\001\\001\\000\\002\\001\\001\\000\\002\\001" +
221 " \"001\\000\\004\\013\\070\\001\\001\\000\\002\\001\\000\\002" +
222 " \"001\\001\\000\\002\\001\\001\\000\\004\\023\\061\\001\\001\\000" +
223 " \"004\\013\\057\\001\\001\\000\\004\\013\\056\\001\\001\\000\\004" +
224 " \"013\\055\\001\\001\\000\\002\\001\\001\\000\\002\\001\\001\\000" +
225 " \"002\\001\\001\\000\\002\\001\\001\\000\\002\\001\\001\\000\\004" +
226 " \"014\\064\\001\\001\\000\\014\\004\\065\\007\\015\\010\\016\\011" +
227 " \"031\\012\\021\\001\\001\\000\\002\\001\\001\\000\\002\\001\\001" +
228 " \"000\\002\\001\\001\\000\\002\\001\\001\\000\\002\\001\\001\\000" +
229 " \"002\\001\\001\\000\\002\\001\\001\\000\\004\\013\\075\\001\\001" +
230 " \"000\\002\\001\\001\\000\\002\\001\\001\\000\\002\\001\\001\\000" +
231 " \"002\\001\\001\\000\\002\\001\\001\\000\\002\\001\\001\\000\\004" +
232 " \"013\\103\\001\\001\\000\\002\\001\\001\\000\\002\\001\\001\\000" +
233 " \"002\\001\\001\\000\\002\\001\\001\\000\\002\\001\\001\\000\\002" +
234 " \"001\\001\\000\\002\\002\\001\\001\\000\\006\\017\\114\\020\\113\\001" +
235 " \"001\\000\\002\\001\\001\\000\\002\\001\\001\\000\\002\\001\\001" +
236 " \"000\\006\\017\\114\\020\\117\\001\\001\\000\\002\\001\\001\\000" +
237 " \"004\\014\\121\\001\\001\\000\\002\\001\\001\\000\\004\\013\\123" +
238 " \"001\\001\\000\\002\\002\\001\\001\\000\\002\\001\\001\\000\\006\\013" +
239 " \"127\\021\\126\\001\\001\\000\\002\\001\\001\\000\\002\\001\\001" +
240 " \"000\\006\\013\\127\\021\\131\\001\\001\\000\\002\\001\\001\\000" +
241 " \"002\\001\\001\\000\\002\\001\\001\\000\\004\\013\\135\\001\\001" +
242 " \"000\\002\\001\\001\\000\\004\\014\\137\\001\\001\\000\\004\\003" +
243 " \"142\\001\\001\\000\\004\\014\\152\\001\\001\\000\\002\\001\\001" +
244 " \"000\\002\\001\\001\\000\\004\\014\\144\\001\\001\\000\\002\\001" +
245 " \"001\\000\\004\\013\\146\\001\\001\\000\\002\\001\\001\\000\\004" +
246 " \"014\\150\\001\\001\\000\\004\\003\\151\\001\\001\\000\\002\\001" +
247 " \"001\\000\\002\\001\\001\\000\\004\\013\\154\\001\\001\\000\\002" +
248 " \"001\\001\\000\\002\\001\\001\\000\\006\\005\\160\\006\\162\\001" +
249 " \"001\\000\\002\\001\\001\\000\\002\\001\\001\\000\\004\\013\\164" +
250 " \"001\\001\\000\\006\\005\\163\\006\\162\\001\\001\\000\\002\\001" +
```



MANUAL DE PRÁCTICAS



The screenshot shows a Java code editor with the file `Sintax.java` open. The code is a generated parser class for a grammar defined in `Sintax.cup`. The editor interface includes tabs for other files like `LexerCup.java` and `sym.java`, and various toolbars and status bars.

```
251 "001\000\002\001\001\000\014\004\166\007\015\010\016" +
252 "\011\031\012\021\001\001\000\002\001\001\000\002\001" +
253 "\001\000\002\001\001\000\002\001\001\000\014\004\173" +
254 "\007\015\010\016\011\031\012\021\001\001\000\002\001" +
255 "\001\000\002\001\001\000\002\001\001\000\002\001\001" +
256 "\000\002\001\001\000\002\001\001\000\002\001\001\000" +
257 "\004\013\203\001\001\000\002\001\001\000\004\014\205" +
258 "\001\001\000\002\001\001\000\002\001\001\000\002\001" +
259 "\001\000\002\001\001" );
260
261 /**
262  * Access to <code>reduce_goto</code> table. */
263 public short[][] reduce_table() {return _reduce_table;}
264
265 /**
266  * Instance of action encapsulation class. */
267 protected CUP$Sintax$Actions action_obj;
268
269 /**
270  * Action encapsulation object initializer. */
271 protected void init_actions()
272 {
273     action_obj = new CUP$Sintax$Actions(this);
274 }
275
276 /**
277  * Invoke a user supplied parse action. */
278 public java_cup.runtime.Symbol do_action(
279     int act_num,
280     java_cup.runtime.lr_parser parser,
281     java.util.Stack stack,
282     int top)
283     throws java.lang.Exception
284 {
285     /* call code in generated class */
286     return action_obj.CUP$Sintax$do_action(act_num, parser, stack, top);
287 }
288
289 /**
290  * Indicates start state. */
291 public int start_state() {return 0;}
```



```
LexerCup.java X sym.java X Sintax.cup X Sintax.java X
Source History [File] [Edit] [Search] [Run] [Build] [Help]
287 |  /** Indicates start production. */
288 |  public int start_production() {return 0;}
289 |
290 |  /** <code>EOF</code> Symbol index. */
291 |  public int EOF_sym() {return 0;}
292 |
293 |  /** <code>error</code> Symbol index. */
294 |  public int error_sym() {return 1;}
295 |
296 |
297 |
298 |  private Symbol s;
299 |
300 |  public void syntax_error(Symbol s) {
301 |      this.s = s;
302 |      System.err.println("Syntax error at line: " + s.left + ", column: " + s.right + ", Text: \\" + s.value + "\\");
303 |  }
304 |
305 |  public Symbol gets() {
306 |      return this.s;
307 |  }
308 |
309 |
310 |  /** Cup generated class to encapsulate user supplied action code.*/
311 |  @SuppressWarnings({"rawtypes", "unchecked", "unused"})
312 |  class CUP$Sintax$Actions {
313 |      private final Sintax parser;
314 |
315 |      /** Constructor */
316 |      CUP$Sintax$Actions(Sintax parser) {
317 |          this.parser = parser;
318 |      }
319 |
320 |      /** Method 0 with the actual generated action code for actions 0 to 300. */
321 |      public final java_cup.runtime.Symbol CUP$Sintax$do_action_part00000000(
322 |          int CUP$Sintax$act_num,
323 |          java_cup.runtime.lr_parser.CUP$Sintax$parser
```



MANUAL DE PRÁCTICAS



```
323     java_cup.runtime.lr_parser CUP$Sintax$parser,
324     java.util.Stack          CUP$Sintax$stack,
325     int                     CUP$Sintax$stop)
326     throws java.lang.Exception
327   {
328     /* Symbol object for return from actions */
329     java_cup.runtime.Symbol CUP$Sintax$result;
330
331     /* select the action based on the action number */
332     switch (CUP$Sintax$act_num)
333     {
334       /*. . . . . */
335       case 0: // $START ::= INICIO EOF
336       {
337         Object RESULT =null;
338         int start_valleft = ((java_cup.runtime.Symbol)CUP$Sintax$stack.elementAt(CUP$Sintax$stop-1)).left;
339         int start_valright = ((java_cup.runtime.Symbol)CUP$Sintax$stack.elementAt(CUP$Sintax$stop-1)).right;
340         Object start_val = (Object)((java_cup.runtime.Symbol) CUP$Sintax$stack.elementAt(CUP$Sintax$stop-1)).value;
341         RESULT = start_val;
342         CUP$Sintax$result = parser.getSymbolFactory().newSymbol("$START",0, ((java_cup.runtime.Symbol)CUP$Sintax$stack.elementAt(0)).value);
343       }
344       /* ACCEPT */
345       CUP$Sintax$parser.done_parsing();
346       return CUP$Sintax$result;
347
348       /*. . . . . */
349       case 1: // INICIO ::= TDATO CLASS ID PARENTESISIZQ PARENTESISDER LLAVEIZQ METODO_LISTA SENTENCIAS LLAVEDER
350       {
351         Object RESULT =null;
352
353         CUP$Sintax$result = parser.getSymbolFactory().newSymbol("INICIO",0, ((java_cup.runtime.Symbol)CUP$Sintax$stack.elementAt(0)).value);
354       }
355       return CUP$Sintax$result;
356
357       /*. . . . . */
358       case 2: // INICIO ::= TDATO CLASS ID PARENTESISIZQ PARENTESISDER LLAVEIZQ SENTENCIAS LLAVEDER
```

1:1

INS Windows (CRLF)



MANUAL DE PRÁCTICAS



```
LexerCup.java x sym.java x Sintax.cup x Sintax.java x
Source History |< > □
358 case 2: // INICIO ::= TDATO CLASS ID PARENTESISIZQ PARENTESISIDER LLAVEIZQ SENTENCIAS LLAVEDER
359 {
360     Object RESULT =null;
361
362     CUP$Syntax$result = parser.getSymbolFactory().newSymbol("INICIO",0, ((java_cup.runtime.Symbol)CUP$Syntax$stack.elementAt(0)));
363 }
364 return CUP$Syntax$result;
365
366 /**
367 case 3: // INICIO ::= TDATO CLASS ID PARENTESISIZQ PARENTESISIDER LLAVEIZQ LLAVEDER
368 {
369     Object RESULT =null;
370
371     CUP$Syntax$result = parser.getSymbolFactory().newSymbol("INICIO",0, ((java_cup.runtime.Symbol)CUP$Syntax$stack.elementAt(0)));
372 }
373 return CUP$Syntax$result;
374
375 /**
376 case 4: // INICIO ::= TDATO CLASS ID PARENTESISIZQ PARENTESISIDER LLAVEIZQ METODO_LISTA LLAVEDER
377 {
378     Object RESULT =null;
379
380     CUP$Syntax$result = parser.getSymbolFactory().newSymbol("INICIO",0, ((java_cup.runtime.Symbol)CUP$Syntax$stack.elementAt(0)));
381 }
382 return CUP$Syntax$result;
383
384 /**
385 case 5: // METODO_LISTA ::= METODO
386 {
387     Object RESULT =null;
388
389     CUP$Syntax$result = parser.getSymbolFactory().newSymbol("METODO_LISTA",12, ((java_cup.runtime.Symbol)CUP$Syntax$stack.elementAt(0)));
390 }
391 return CUP$Syntax$result;
392
393 /**

```



MANUAL DE PRÁCTICAS



The screenshot shows a Java code editor with the file `Sintax.java` open. The code is part of a parser implementation using CUP (Java CUP). It contains several switch statements (cases) corresponding to different grammar rules:

- Case 6: `METODO_LISTA ::= METODO METODO_LISTA`
- Case 7: `SENTENCIAS ::= SENTENCIA`
- Case 8: `SENTENCIAS ::= SENTENCIA SENTENCIAS`
- Case 9: `SENTENCIA ::= COMENTARIOLINEA`

Each case block initializes a result object and creates a new symbol using `parser.getSymbolFactory().newSymbol`. The code uses annotations like `/* */` to mark sections of the code.



MANUAL DE PRACTICAS



```
429     /* . . . . . */
430     case 10: // SENTENCIA ::= COMENTARIOBLOQUE
431     {
432         Object RESULT =null;
433
434         CUP$Syntax$result = parser.getSymbolFactory().newSymbol("SENTENCIA",5, ((java_cup.runtime.Symbol)CUP$Syntax$stack.peek(
435             )
436         return CUP$Syntax$result;
437
438     /* . . . . . */
439     case 11: // SENTENCIA ::= DECLARACION
440     {
441         Object RESULT =null;
442
443         CUP$Syntax$result = parser.getSymbolFactory().newSymbol("SENTENCIA",5, ((java_cup.runtime.Symbol)CUP$Syntax$stack.peek(
444             )
445         return CUP$Syntax$result;
446
447     /* . . . . . */
448     case 12: // SENTENCIA ::= ASIGNACION
449     {
450         Object RESULT =null;
451
452         CUP$Syntax$result = parser.getSymbolFactory().newSymbol("SENTENCIA",5, ((java_cup.runtime.Symbol)CUP$Syntax$stack.peek(
453             )
454         return CUP$Syntax$result;
455
456     /* . . . . . */
457     case 13: // SENTENCIA ::= CONTROL_FLUJO
458     {
459         Object RESULT =null;
460
461         CUP$Syntax$result = parser.getSymbolFactory().newSymbol("SENTENCIA",5, ((java_cup.runtime.Symbol)CUP$Syntax$stack.peek(
462             )
463         return CUP$Syntax$result;
464
```



MANUAL DE PRACTICAS



```
465     /* . . . . . */
466     case 14: // SENTENCIA ::= RETURN EXPRESION PUNTOYCOMA
467     {
468         Object RESULT =null;
469
470         CUP$Syntax$result = parser.getSymbolFactory().newSymbol("SENTENCIA",5, ((java_cup.runtime.Symbol)CUP$Syntax$stack.elementAt(0)));
471     }
472     return CUP$Syntax$result;
473
474     /* . . . . . */
475     case 15: // SENTENCIA ::= PRINTF PARENTESISIZQ LISTA_EXPRESIONES PARENTESISDER PUNTOYCOMA
476     {
477         Object RESULT =null;
478
479         CUP$Syntax$result = parser.getSymbolFactory().newSymbol("SENTENCIA",5, ((java_cup.runtime.Symbol)CUP$Syntax$stack.elementAt(0)));
480     }
481     return CUP$Syntax$result;
482
483     /* . . . . . */
484     case 16: // DECLARACION ::= TDATO ID PUNTOYCOMA
485     {
486         Object RESULT =null;
487
488         CUP$Syntax$result = parser.getSymbolFactory().newSymbol("DECLARACION",6, ((java_cup.runtime.Symbol)CUP$Syntax$stack.elementAt(0)));
489     }
490     return CUP$Syntax$result;
491
492     /* . . . . . */
493     case 17: // DECLARACION ::= TDATO ID OPASIGNACION EXPRESION PUNTOYCOMA
494     {
495         Object RESULT =null;
496
497         CUP$Syntax$result = parser.getSymbolFactory().newSymbol("DECLARACION",6, ((java_cup.runtime.Symbol)CUP$Syntax$stack.elementAt(0)));
498     }
499     return CUP$Syntax$result;
500 }
```



MANUAL DE PRACTICAS



```
502     case 18: // METODO ::= GATO TIPO_RETORNO ID PARENTESISIZQ PARAMETROS_METODO PARENTESISDER BLOQUE
503     {
504         Object RESULT =null;
505
506         CUP$Syntax$result = parser.getSymbolFactory().newSymbol("METODO",11, ((java_cup.runtime.Symbol)CUP$Syntax$stack.elementAt(11)));
507     }
508     return CUP$Syntax$result;
509
510     /* . . . . . */
511     case 19: // ASIGNACION ::= ID OPASIGNACION EXPRESION PUNTOYCOMA
512     {
513         Object RESULT =null;
514
515         CUP$Syntax$result = parser.getSymbolFactory().newSymbol("ASIGNACION",7, ((java_cup.runtime.Symbol)CUP$Syntax$stack.elementAt(7)));
516     }
517     return CUP$Syntax$result;
518
519     /* . . . . . */
520     case 20: // TIPO_RETORNO ::= TDATO
521     {
522         Object RESULT =null;
523
524         CUP$Syntax$result = parser.getSymbolFactory().newSymbol("TIPO_RETORNO",13, ((java_cup.runtime.Symbol)CUP$Syntax$stack.elementAt(13)));
525     }
526     return CUP$Syntax$result;
527
528     /* . . . . . */
529     case 21: // TIPO_RETORNO ::= VOID
530     {
531         Object RESULT =null;
532
533         CUP$Syntax$result = parser.getSymbolFactory().newSymbol("TIPO_RETORNO",13, ((java_cup.runtime.Symbol)CUP$Syntax$stack.elementAt(13)));
534     }
535     return CUP$Syntax$result;
536
537     /* . . . . . */
538     case 22: // CONTROL_FILIO ::= IF PARENTESISIZQ EXPRESION PARENTESISDER PT COMPR
539 }
```



MANUAL DE PRÁCTICAS



```
case 22: // CONTROL_FLUJO ::= IF PARENTESISIZQ EXPRESION PARENTESISDER BLOQUE
{
    Object RESULT =null;
    CUP$Sintax$result = parser.getSymbolFactory().newSymbol("CONTROL_FLUJO",8, ((java_cup.runtime.Symbol)CUP$Sintax$stack.elementAt(0)).value);
}
return CUP$Sintax$result;

/*
 */
case 23: // CONTROL_FLUJO ::= IF PARENTESISIZQ EXPRESION PARENTESISDER BLOQUE ELSE BLOQUE
{
    Object RESULT =null;
    CUP$Sintax$result = parser.getSymbolFactory().newSymbol("CONTROL_FLUJO",8, ((java_cup.runtime.Symbol)CUP$Sintax$stack.elementAt(0)).value);
}
return CUP$Sintax$result;

/*
 */
case 24: // CONTROL_FLUJO ::= IF PARENTESISIZQ EXPRESION PARENTESISDER BLOQUE ELSEIF_ANIDADO ELSE BLOQUE
{
    Object RESULT =null;
    CUP$Sintax$result = parser.getSymbolFactory().newSymbol("CONTROL_FLUJO",8, ((java_cup.runtime.Symbol)CUP$Sintax$stack.elementAt(0)).value);
}
return CUP$Sintax$result;

/*
 */
case 25: // CONTROL_FLUJO ::= WHILE PARENTESISIZQ EXPRESION PARENTESISDER BLOQUE
{
    Object RESULT =null;
    CUP$Sintax$result = parser.getSymbolFactory().newSymbol("CONTROL_FLUJO",8, ((java_cup.runtime.Symbol)CUP$Sintax$stack.elementAt(0)).value);
}
return CUP$Sintax$result;

/*
 */
```



MANUAL DE PRÁCTICAS



```
573     /* . . . . . */
574     case 26: // CONTROL_FLUJO ::= FOR PARENTESISIZQ INICIALIZACION_FOR EXPRESION PUNTOYCOMA INCREMENTO_FOR PARENTESISDER BLOQUE
575     {
576         Object RESULT =null;
577
578         CUP$Sintax$result = parser.getSymbolFactory().newSymbol("CONTROL_FLUJO",8, ((java_cup.runtime.Symbol)CUP$Sintax$stack.e
579     }
580     return CUP$Sintax$result;
581
582     /* . . . . . */
583     case 27: // CONTROL_FLUJO ::= SWITCH PARENTESISIZQ EXPRESION PARENTESISDER LLAVEIZQ CASOS LLAVEDER
584     {
585         Object RESULT =null;
586
587         CUP$Sintax$result = parser.getSymbolFactory().newSymbol("CONTROL_FLUJO",8, ((java_cup.runtime.Symbol)CUP$Sintax$stack.e
588     }
589     return CUP$Sintax$result;
590
591     /* . . . . . */
592     case 28: // CONTROL_FLUJO ::= DO BLOQUE WHILE PARENTESISIZQ EXPRESION PARENTESISDER PUNTOYCOMA
593     {
594         Object RESULT =null;
595
596         CUP$Sintax$result = parser.getSymbolFactory().newSymbol("CONTROL_FLUJO",8, ((java_cup.runtime.Symbol)CUP$Sintax$stack.e
597     }
598     return CUP$Sintax$result;
599
600     /* . . . . . */
601     case 29: // ELSEIF_ANIDADO ::= ELSEIF PARENTESISIZQ EXPRESION PARENTESISDER BLOQUE
602     {
603         Object RESULT =null;
604
605         CUP$Sintax$result = parser.getSymbolFactory().newSymbol("ELSEIF_ANIDADO",1, ((java_cup.runtime.Symbol)CUP$Sintax$stack.e
606     }
607     return CUP$Sintax$result;
608 }
```



MANUAL DE PRÁCTICAS



```
609     /* . . . . . */
610     case 30: // ELSEIF_ANIDADO ::= ELSEIF PARENTESISIZQ EXPRESION PARENTESISDER BLOQUE ELSEIF_ANIDADO
611     {
612         Object RESULT =null;
613
614         CUP$Syntax$result = parser.getSymbolFactory().newSymbol("ELSEIF_ANIDADO",1, ((java_cup.runtime.Symbol)CUP$Syntax$stack.
615     }
616     return CUP$Syntax$result;
617
618     /* . . . . . */
619     case 31: // CASOS ::= CASO
620     {
621         Object RESULT =null;
622
623         CUP$Syntax$result = parser.getSymbolFactory().newSymbol("CASOS",3, ((java_cup.runtime.Symbol)CUP$Syntax$stack.peek()),
624     }
625     return CUP$Syntax$result;
626
627     /* . . . . . */
628     case 32: // CASOS ::= CASO CASOS
629     {
630         Object RESULT =null;
631
632         CUP$Syntax$result = parser.getSymbolFactory().newSymbol("CASOS",3, ((java_cup.runtime.Symbol)CUP$Syntax$stack.elementAt
633     }
634     return CUP$Syntax$result;
635
636     /* . . . . . */
637     case 33: // CASOS ::= DEFAULT DOSPUNTOS SENTENCIAS BREAK PUNTOYCOMA
638     {
639         Object RESULT =null;
640
641         CUP$Syntax$result = parser.getSymbolFactory().newSymbol("CASOS",3, ((java_cup.runtime.Symbol)CUP$Syntax$stack.elementAt
642     }
643     return CUP$Syntax$result;
644 }
```



MANUAL DE PRÁCTICAS



```
/* . . . . . */
case 34: // CASO ::= CASE EXPRESION DOSPUNTOS SENTENCIAS BREAK PUNTOYCOMA
{
    Object RESULT =null;

    CUP$Syntax$result = parser.getSymbolFactory().newSymbol("CASO",4, ((java_cup.runtime.Symbol)CUP$Syntax$stack.elementAt(0)).index);
}
return CUP$Syntax$result;

/* . . . . . */
case 35: // INICIALIZACION_FOR ::= ASIGNACION
{
    Object RESULT =null;

    CUP$Syntax$result = parser.getSymbolFactory().newSymbol("INICIALIZACION_FOR",16, ((java_cup.runtime.Symbol)CUP$Syntax$stack.elementAt(0)).index);
}
return CUP$Syntax$result;

/* . . . . . */
case 36: // INICIALIZACION_FOR ::= DECLARACION
{
    Object RESULT =null;

    CUP$Syntax$result = parser.getSymbolFactory().newSymbol("INICIALIZACION_FOR",16, ((java_cup.runtime.Symbol)CUP$Syntax$stack.elementAt(0)).index);
}
return CUP$Syntax$result;

/* . . . . . */
case 37: // INCREMENTO_FOR ::= ID OPINCREMENTO
{
    Object RESULT =null;

    CUP$Syntax$result = parser.getSymbolFactory().newSymbol("INCREMENTO_FOR",17, ((java_cup.runtime.Symbol)CUP$Syntax$stack.elementAt(0)).index);
}
return CUP$Syntax$result;
```



MANUAL DE PRÁCTICAS



The screenshot shows a Java code editor with the file `Sintax.java` open. The code is part of a parser implementation, specifically for handling expressions. It uses a switch statement to handle four cases (38, 39, 40, 41) where the expression is a number, ID, cadena, or booleano respectively. Each case initializes a result object to null, gets a new symbol from the parser's symbol factory, and then returns the result. The code is heavily annotated with comments and includes imports at the top.

```
LexerCup.java X sym.java X Sintax.cup X Sintax.java X
Source History
681     /* . . . . . */
682     case 38: // EXPRESION ::= NUM
683     {
684         Object RESULT =null;
685
686         CUP$Sintax$result = parser.getSymbolFactory().newSymbol("EXPRESION",9, ((java_cup.runtime.Symbol)CUP$Sintax$stack.peek());
687     }
688     return CUP$Sintax$result;
689
690     /* . . . . . */
691     case 39: // EXPRESION ::= ID
692     {
693         Object RESULT =null;
694
695         CUP$Sintax$result = parser.getSymbolFactory().newSymbol("EXPRESION",9, ((java_cup.runtime.Symbol)CUP$Sintax$stack.peek());
696     }
697     return CUP$Sintax$result;
698
699     /* . . . . . */
700     case 40: // EXPRESION ::= CADENA
701     {
702         Object RESULT =null;
703
704         CUP$Sintax$result = parser.getSymbolFactory().newSymbol("EXPRESION",9, ((java_cup.runtime.Symbol)CUP$Sintax$stack.peek());
705     }
706     return CUP$Sintax$result;
707
708     /* . . . . . */
709     case 41: // EXPRESION ::= BOOLEANO
710     {
711         Object RESULT =null;
712
713         CUP$Sintax$result = parser.getSymbolFactory().newSymbol("EXPRESION",9, ((java_cup.runtime.Symbol)CUP$Sintax$stack.peek());
714     }
715     return CUP$Sintax$result;
716
717     /* . . . . . */
```

1:1

INS Windows (CRLF)



MANUAL DE PRÁCTICAS



```
LexerCup.java X sym.java X Sintax.cup X Sintax.java X
Source History [File] [Edit] [View] [Search] [Tools] [Help]
717     /*. . . . . */
718     case 42: // EXPRESION ::= EXPRESION OPARITMETICO EXPRESION
719     {
720         Object RESULT =null;
721
722         CUP$Sintax$result = parser.getSymbolFactory().newSymbol("EXPRESION",9, ((java_cup.runtime.Symbol)CUP$Sintax$stack.elementAt(0)));
723     }
724     return CUP$Sintax$result;
725
726     /*. . . . . */
727     case 43: // EXPRESION ::= EXPRESION OPRELACIONALES EXPRESION
728     {
729         Object RESULT =null;
730
731         CUP$Sintax$result = parser.getSymbolFactory().newSymbol("EXPRESION",9, ((java_cup.runtime.Symbol)CUP$Sintax$stack.elementAt(0)));
732     }
733     return CUP$Sintax$result;
734
735     /*. . . . . */
736     case 44: // EXPRESION ::= EXPRESION OPLOGICOS EXPRESION
737     {
738         Object RESULT =null;
739
740         CUP$Sintax$result = parser.getSymbolFactory().newSymbol("EXPRESION",9, ((java_cup.runtime.Symbol)CUP$Sintax$stack.elementAt(0)));
741     }
742     return CUP$Sintax$result;
743
744     /*. . . . . */
745     case 45: // EXPRESION ::= PARENTESISIZQ EXPRESION PARENTESISDER
746     {
747         Object RESULT =null;
748
749         CUP$Sintax$result = parser.getSymbolFactory().newSymbol("EXPRESION",9, ((java_cup.runtime.Symbol)CUP$Sintax$stack.elementAt(0)));
750     }
751     return CUP$Sintax$result;
752
753     /*. . . . . */
754 }
```

1:1

INS Windows (CRLF)



MANUAL DE PRACTICAS



The screenshot shows a Java code editor with the following tabs at the top: LexerCup.java, sym.java, Sintax.cup, and Sintax.java. The Sintax.cup tab is active. The code in the editor is a CUP grammar file with the following content:

```
753     /* . . . . . */
754     case 46: // LISTA_EXPRESIONES ::= EXPRESION
755     {
756         Object RESULT =null;
757
758         CUP$Syntax$result = parser.getSymbolFactory().newSymbol("LISTA_EXPRESIONES",15, ((java_cup.runtime.Symbol)CUP$Syntax$st
759     }
760     return CUP$Syntax$result;
761
762     /* . . . . . */
763     case 47: // LISTA_EXPRESIONES ::= EXPRESION COMA LISTA_EXPRESIONES
764     {
765         Object RESULT =null;
766
767         CUP$Syntax$result = parser.getSymbolFactory().newSymbol("LISTA_EXPRESIONES",15, ((java_cup.runtime.Symbol)CUP$Syntax$st
768     }
769     return CUP$Syntax$result;
770
771     /* . . . . . */
772     case 48: // BLOQUE ::= LLAVEIZQ SENTENCIAS LLAVEDER
773     {
774         Object RESULT =null;
775
776         CUP$Syntax$result = parser.getSymbolFactory().newSymbol("BLOQUE",10, ((java_cup.runtime.Symbol)CUP$Syntax$stack.element.
777     }
778     return CUP$Syntax$result;
779
780     /* . . . . . */
781     case 49: // PARAMETROS_METODO ::= TIPO_RETORNO ID
782     {
783         Object RESULT =null;
784
785         CUP$Syntax$result = parser.getSymbolFactory().newSymbol("PARAMETROS_METODO",14, ((java_cup.runtime.Symbol)CUP$Syntax$st
786     }
787     return CUP$Syntax$result;
788
789     /* . . . . . */
790 }
```



MANUAL DE PRACTICAS



The screenshot shows a Java code editor interface with the following details:

- File Tabs:** LexerCup.java, sym.java, Sintax.cup, Sintax.java
- Toolbar:** Source, History, and various icons for file operations like Open, Save, Find, and Copy.
- Code Area:** The main area displays the `Sintax.java` file content. The code is heavily annotated with comments and symbols from the `Sintax.cup` grammar file. It includes several switch statements for different action numbers, with specific cases for `PARAMETROS_METODO`. The code uses `CUP$Sintax$result` to store results and `CUP$Sintax$act_num` to store action numbers. A `default` case handles invalid action numbers by throwing an `Exception`.
- Code Block 1 (Lines 815-824):**

```
/** Method splitting the generated action code into several parts. */
public final java_cup.runtime.Symbol CUP$Sintax$do_action(
    int CUP$Sintax$act_num,
    java_cup.runtime.lr_parser CUP$Sintax$parser,
    java.util.Stack CUP$Sintax$stack,
    int CUP$Sintax$stop)
throws java.lang.Exception
{
    return CUP$Sintax$do_action_part00000000(
        CUP$Sintax$act_num,
```
- Code Block 2 (Lines 825-826):**

```
    CUP$Sintax$act_num,
```

```
throw new Exception(
        "Invalid action number "+CUP$Syntax$act_num+" found in internal parse table");

    }
} /* end of method */

/** Method splitting the generated action code into several parts. */
public final java_cup.runtime.Symbol CUP$Syntax$do_action(
    int CUP$Syntax$act_num,
    java_cup.runtime.lr_parser CUP$Syntax$parser,
    java.util.Stack CUP$Syntax$stack,
    int CUP$Syntax$stop)
throws java.lang.Exception
{
    return CUP$Syntax$do_action_part00000000(
        CUP$Syntax$act_num,
        CUP$Syntax$parser,
        CUP$Syntax$stack,
        CUP$Syntax$stop);
}
}
```



TermSintac.java

Este código implementa una aplicación gráfica en Java Swing llamada TermSintac, que sirve como una interfaz para un analizador sintáctico. La aplicación permite cargar archivos de texto, visualizar el contenido con números de línea, realizar análisis sintácticos y mostrar errores o resultados del análisis. Utiliza un diseño personalizado con el tema Nimbus y componentes gráficos como JTextArea, JPanel, y JScrollPane. Destaca la funcionalidad de sincronización entre un área de texto principal y un área auxiliar que muestra números de línea, actualizándose dinámicamente conforme se edita el contenido. También incluye botones para cargar archivos, limpiar el texto, realizar el análisis y acceder a otra ventana relacionada con el análisis léxico. El análisis sintáctico se realiza utilizando un objeto Sintax que procesa el texto cargado, detecta errores de sintaxis e informa detalles como la línea, columna y contenido del error. La interfaz está optimizada para facilitar la interacción del usuario con el análisis léxico y sintáctico.

The screenshot shows a Java IDE interface with the following details:

- Project Bar:** Contains tabs for "LexerCup.java X", "sym.java X", "Syntax.cup X", "Sintaxis.java X", and "TermSintac.java X".
- Toolbar:** Includes icons for Source, Design, History, and various file operations like Open, Save, Print, and Find.
- Code Editor:** Displays the source code for "TermSintac.java". The code initializes a window, sets up components for displaying results and line numbers, and configures the text area to wrap lines.

```
1 package codigo;
2
3 import javax.swing.*;
4 import java.awt.*;
5 import java.awt.event.*;
6 import java.io.*;
7 import java.nio.file.Files;
8 import java.util.logging.*;
9 import java_cup.runtime.Symbol;
10 import javax.swing.plaf.nimbus.NimbusLookAndFeel;
11 import javax.swing.event.*;
12
13 public class TermSintac extends javax.swing.JFrame {
14
15     private JTextArea txtResultadoConLineas;
16     private JTextArea lineNumbersArea;
17
18     public TermSintac() {
19         initComponents();
20         this.setLocationRelativeTo(null);
21         setNimbusLookAndFeel();
22         setSize(1100, 700); // Aumenta el tamaño de la ventana
23         setResizable(false);
24
25         // Crear y configurar el área de texto con los números de línea
26         txtResultadoConLineas = new JTextArea();
27         txtResultadoConLineas.setFont(new Font("Courier New", Font.PLAIN, 18));
28         txtResultadoConLineas.setBackground(Color.BLACK);
29         txtResultadoConLineas.setForeground(Color.WHITE);
30         txtResultadoConLineas.setCaretColor(Color.WHITE);
31         txtResultadoConLineas.setLineWrap(true);
32         txtResultadoConLineas.setWrapStyleWord(true);
33
34         // Crear el área para los números de línea
35         lineNumbersArea = new JTextArea();
36         lineNumbersArea.setEditable(false);
```

- Status Bar:** Shows the time "273:44" and the text "INS Unix (LF)".



MANUAL DE PRÁCTICAS



```
LexerCup.java X sym.java X Sintax.cup X Sintax.java X TermSintac.java X
Source Design History
37     lineNumbersArea.setFont(new Font("Courier New", Font.PLAIN, 18));
38     lineNumbersArea.setBackground(Color.BLACK);
39     lineNumbersArea.setForeground(Color.GRAY);
40     lineNumbersArea.setCaretColor(Color.WHITE);
41
42     // Crear un panel para organizar los componentes
43     JPanel panel = new JPanel();
44     panel.setLayout(new BorderLayout());
45     panel.setBackground(Color.BLACK);
46
47     // Crear un JScrollPane para contener los dos JTextAreas
48     JScrollPane scrollPane = new JScrollPane(txtResultadoConLineas);
49     scrollPane.setRowHeaderView(lineNumbersArea); // Agregar el área de números de línea al lado del JTextArea de código
50
51     // Añadir el JScrollPane al panel
52     panel.add(scrollPane, BorderLayout.CENTER);
53
54     // Agregar el panel a la interfaz de usuario
55     jScrollPane.setViewportView(panel);
56
57     // Escuchar el cambio de texto y actualizar los números de linea
58     txtResultadoConLineas.getDocument().addDocumentListener(new DocumentListener() {
59         public void insertUpdate(DocumentEvent e) {
60             updateLineNumbers();
61         }
62         public void removeUpdate(DocumentEvent e) {
63             updateLineNumbers();
64         }
65         public void changedUpdate(DocumentEvent e) {
66             updateLineNumbers();
67         }
68     });
69 }
70
71 private void setNimbusLookAndFeel() {
72     try {
```



MANUAL DE PRÁCTICAS



The screenshot shows a Java code editor with the following code:

```
72     try {
73         UIManager.setLookAndFeel(new NimbusLookAndFeel());
74     } catch (UnsupportedLookAndFeelException e) {
75         e.printStackTrace();
76     }
77
78
79     private void updateLineNumbers() {
80         try {
81             StringBuilder lineNumbers = new StringBuilder();
82             int lineCount = txtResultadoConLineas.getLineCount();
83             for (int i = 1; i <= lineCount; i++) {
84                 lineNumbers.append(i).append("\n");
85             }
86             lineNumbersArea.setText(lineNumbers.toString());
87         } catch (Exception e) {
88             e.printStackTrace();
89         }
90     }
91
92     private void initComponents() {
93         jPanell = new javax.swing.JPanel();
94         jPanell2 = new javax.swing.JPanel();
95         jScrollPane3 = new javax.swing.JScrollPane();
96         txtAnalizarSin = new javax.swing.JTextArea();
97         btnAnalizarSin = new javax.swing.JButton();
98         btnLimpiaSin = new javax.swing.JButton();
99         jScrollPane1 = new javax.swing.JScrollPane();
100        btnArchivo = new javax.swing.JButton();
101        btnTerminalLexico = new javax.swing.JButton(); // Botón Terminal Léxico
102
103        setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
104        setBackground(new java.awt.Color(255, 255, 255));
105
106        jPanell.setBackground(new java.awt.Color(0, 0, 0));
107        jPanell.setBorder(javax.swing.BorderFactory.createTitledBorder(null, "Analizador Léxico", javax.swing.border.TitledBorder.CEN
```

The code is part of a Java application for a lexical analyzer. It includes methods for initializing components, updating line numbers, and setting up the main window. The code uses Java Swing for the graphical interface.



MANUAL DE PRÁCTICAS



```
108 jPanel12.setBackground(new java.awt.Color(0, 0, 0));
109 jPanel12.setBorder(javax.swing.BorderFactory.createTitledBorder(null, "Analizador Sintactico", javax.swing.border.TitledBorder.DEFAULT_JUSTIFICATION, javax.swing.border.TitledBorder.DEFAULT_POSITION, null, null));
110
111 txtAnalizarSin.setEditable(false);
112 txtAnalizarSin.setBackground(new java.awt.Color(0, 0, 0));
113 txtAnalizarSin.setColumns(20);
114 txtAnalizarSin.setFont(new java.awt.Font("Calibri", 1, 18));
115 txtAnalizarSin.setRows(5);
116 txtAnalizarSin.setCaretColor(new java.awt.Color(255, 255, 255));
117 jScrollPane3.setViewportView(txtAnalizarSin);
118
119
120 btnAnalizarSin.setBackground(new java.awt.Color(0, 51, 51));
121 btnAnalizarSin.setFont(new java.awt.Font("Calibri", 1, 18));
122 btnAnalizarSin.setForeground(new java.awt.Color(255, 255, 255));
123 btnAnalizarSin.setText("ANALIZAR");
124 btnAnalizarSin.addActionListener(evt -> btnAnalizarSinActionPerformed(evt));
125
126 btnLimpiaSin.setBackground(new java.awt.Color(0, 51, 51));
127 btnLimpiaSin.setFont(new java.awt.Font("Calibri", 1, 18));
128 btnLimpiaSin.setForeground(new java.awt.Color(255, 255, 255));
129 btnLimpiaSin.setText("LIMPIAR");
130 btnLimpiaSin.addActionListener(evt -> btnLimpiaSinActionPerformed(evt));
131
132 btnArchivo.setBackground(new java.awt.Color(0, 51, 51));
133 btnArchivo.setFont(new java.awt.Font("Calibri", 1, 18));
134 btnArchivo.setForeground(new java.awt.Color(255, 255, 255));
135 btnArchivo.setText("ABRIR ARCHIVO");
136 btnArchivo.addActionListener(evt -> btnArchivoActionPerformed(evt));
137
138 btnTerminalLexico.setBackground(new java.awt.Color(0, 51, 51));
139 btnTerminalLexico.setFont(new java.awt.Font("Calibri", 1, 18));
140 btnTerminalLexico.setForeground(new java.awt.Color(255, 255, 255));
141 btnTerminalLexico.setText("TERMINAL LEXICO");
142 btnTerminalLexico.addActionListener(evt -> btnTerminalLexicoActionPerformed(evt)); // Acción para abrir TermLexer
143
```

273:44

INS Unix (LF)



MANUAL DE PRACTICAS



```
146     javax.swing.GroupLayout jPanel2Layout = new javax.swing.GroupLayout(jPanel2);
147     jPanel2.setLayout(jPanel2Layout);
148     jPanel2Layout.setHorizontalGroup(
149         jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
150             .addComponent(jPanel1)
151             .addGroup(jPanel2Layout.createSequentialGroup()
152                 .addGap(18, 18, 18)
153                 .addGroup(jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
154                     .addComponent(btnArchivo)
155                     .addGroup(jPanel2Layout.createSequentialGroup()
156                         .addGap(187, 187, 187)
157                         .addComponent(btnTerminalLexico) // Botón añadido
158                         .addGap(187, 187, 187)
159                         .addComponent(btnLimpiarSin)
160                         .addGap(259, 259, 259)
161                         .addComponent(btnAnalizarSin))
162                     .addGap(18, 18, 18)
163                     .addComponent(btnAnalizarSin)))
164             .addGap(12, 12, 12)
165             .addGroup(jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
166                 .addComponent(jPanel2Layout.createSequentialGroup()
167                     .addGap(12, 12, 12)
168                     .addGroup(jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
169                         .addComponent(btnArchivo)
170                         .addComponent(btnTerminalLexico) // Botón añadido
171                         .addGap(187, 187, 187)
172                         .addComponent(btnLimpiarSin)
173                         .addGap(187, 187, 187)
174                         .addComponent(btnAnalizarSin))
175                     .addGap(449, 449, 449)
176                     .addComponent(jPanel1, javax.swing.GroupLayout.PREFERRED_SIZE, 330, javax.swing.GroupLayout.PREFERRED_SIZE)
177                     .addGap(123, 123, 123)
178                     .addComponent(jPanel1, javax.swing.GroupLayout.PREFERRED_SIZE, 123, javax.swing.GroupLayout.PREFERRED_SIZE)
179                     .addGap(449, 449, 449)
180                     .addComponent(jPanel1, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
181             );
182     );
183     jPanel2Layout.setVerticalGroup(
184         jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
185             .addComponent(jPanel1)
186             .addGroup(jPanel2Layout.createSequentialGroup()
187                 .addGap(18, 18, 18)
188                 .addGroup(jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
189                     .addComponent(btnArchivo)
190                     .addGroup(jPanel2Layout.createSequentialGroup()
191                         .addGap(187, 187, 187)
192                         .addComponent(btnTerminalLexico) // Botón añadido
193                         .addGap(187, 187, 187)
194                         .addComponent(btnLimpiarSin)
195                         .addGap(259, 259, 259)
196                         .addComponent(btnAnalizarSin))
197                     .addGap(18, 18, 18)
198                     .addComponent(btnAnalizarSin)))
199             .addGap(12, 12, 12)
200             .addGroup(jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
201                 .addComponent(jPanel2Layout.createSequentialGroup()
202                     .addGap(12, 12, 12)
203                     .addGroup(jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
204                         .addComponent(btnArchivo)
205                         .addComponent(btnTerminalLexico) // Botón añadido
206                         .addGap(187, 187, 187)
207                         .addComponent(btnLimpiarSin)
208                         .addGap(187, 187, 187)
209                         .addComponent(btnAnalizarSin))
210                     .addGap(449, 449, 449)
211                     .addComponent(jPanel1, javax.swing.GroupLayout.PREFERRED_SIZE, 330, javax.swing.GroupLayout.PREFERRED_SIZE)
212                     .addGap(123, 123, 123)
213                     .addComponent(jPanel1, javax.swing.GroupLayout.PREFERRED_SIZE, 123, javax.swing.GroupLayout.PREFERRED_SIZE)
214                     .addGap(449, 449, 449)
215                     .addComponent(jPanel1, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
216             );
217     );
218 }
```



MANUAL DE PRACTICAS



```
181     javax.swing.GroupLayout jPanel1Layout = new javax.swing.GroupLayout(jPanel1);
182     jPanel1.setLayout(jPanel1Layout);
183     jPanel1Layout.setHorizontalGroup(
184         jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
185             .addGroup(jPanel1Layout.createSequentialGroup()
186                 .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
187                     .addComponent(jPanel2, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
188                 );
189             .addGroup(jPanel1Layout.createSequentialGroup()
190                 .addGap(10, 10, 10)
191                 .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
192                     .addComponent(jPanel2, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
193                     .addGap(10, 10, 10)
194                 );
195             )
196             .addComponent(jPanel1, java.awt.BorderLayout.CENTER);
197
198     jPanel1Layout.setVerticalGroup(
199         jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
200             .addGroup(jPanel1Layout.createSequentialGroup()
201                 .addGap(10, 10, 10)
202                 .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
203                     .addComponent(jPanel2, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
204                     .addGap(10, 10, 10)
205                     .addComponent(jPanel1, java.awt.BorderLayout.CENTER));
206             .addGroup(jPanel1Layout.createSequentialGroup()
207                 .addGap(10, 10, 10)
208                 .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
209                     .addComponent(jPanel2, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
210                     .addGap(10, 10, 10)
211                     .addComponent(jPanel1, java.awt.BorderLayout.CENTER));
212             .addGroup(jPanel1Layout.createSequentialGroup()
213                 .addGap(10, 10, 10)
214                 .addComponent(jPanel2, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE));
215         );
216
217     private void btnArchivoActionPerformed(java.awt.event.ActionEvent evt) {
218         JFileChooser chooser = new JFileChooser();
219         chooser.showOpenDialog(null);
220         File archivo = new File(chooser.getSelectedFile().getAbsolutePath());
221
222         try {
223             String ST = new String(Files.readAllBytes(archivo.toPath()));
224             txtResultadoConLineas.setText(ST);
225         } catch (FileNotFoundException ex) {
226             Logger.getLogger(TermSintac.class.getName()).log(Level.SEVERE, null, ex);
227         } catch (IOException ex) {
228             Logger.getLogger(TermSintac.class.getName()).log(Level.SEVERE, null, ex);
229         }
230     }
231
232     private void btnLimpiarSinActionPerformed(java.awt.event.ActionEvent evt) {
```



MANUAL DE PRÁCTICAS



```
215     private void btnLimpiarSinActionPerformed(java.awt.event.ActionEvent evt) {
216         txtResultadoConLineas.setText(null);
217     }
218
219     private void btnAnalizarSinActionPerformed(java.awt.event.ActionEvent evt) {
220         String ST = txtResultadoConLineas.getText();
221         Sintax s = new Sintax(new codigo.LexerCup(new StringReader(ST)));
222
223         StringBuilder errores = new StringBuilder();
224         boolean errorEncontrado = false;
225
226         // Variable que marca si aún estamos en ejecución
227         boolean continuarAnalisis = true;
228
229         try {
230             // Aquí es donde se empieza el análisis
231             s.parse();
232             txtAnalizarSin.setText("Análisis realizado correctamente");
233             txtAnalizarSin.setForeground(new Color(25, 111, 61));
234         } catch (Exception ex) {
235             // Obtén el error actual
236             Symbol sym = s.gets(); // La posición del error actual
237             if (sym != null) {
238                 errores.append("Error de sintaxis. Línea: ").append(sym.right + 1)
239                     .append(" Columna: ").append(sym.left + 1)
240                     .append(", Texto: \"").append(sym.value).append("\">\n");
241                 errorEncontrado = true;
242             }
243
244             // En este punto puedes intentar capturar más errores, si es necesario.
245             if (errorEncontrado) {
246                 txtAnalizarSin.setText(errores.toString());
247                 txtAnalizarSin.setForeground(Color.red);
248             } else {
249                 txtAnalizarSin.setText("No se detectaron errores en la sintaxis.");
250                 txtAnalizarSin.setForeground(Color.green);
251             }
252         }
253     }
```

273:44

INS Unix (LF)



The screenshot shows a Java IDE interface with multiple tabs at the top: LexerCup.java X, sym.java X, Sintax.cup X, Sintax.java X, and TermSintac.java X. The main window displays the content of the TermSintac.java file. The code is written in Java and includes imports for java.awt.event.ActionEvent, java.awt.EventQueue, and javax.swing.JButton. It contains several methods: a constructor for a JPanel, an actionPerformed method for a button, and a static main method. The static main method uses invokeLater to run a Runnable task which sets the visibility of a TermLexer window to true. The code also declares various swing components like JButton, JPanel, JScrollPane, and JTextArea, and initializes them with specific colors and actions.

```
251     txtAnalizarSin.setForeground(Color.green);
252 }
253 }
254 }
255
256
257 // Acción del nuevo botón "Terminal Léxico"
258 private void btnTerminalLexicoActionPerformed(java.awt.event.ActionEvent evt) {
259     new TermLexer().setVisible(true); // Mostrar la ventana de TermLexer
260     dispose();
261 }
262
263 public static void main(String args[]) {
264     java.awt.EventQueue.invokeLater(new Runnable() {
265         public void run() {
266             new TermSintac().setVisible(true);
267         }
268     });
269 }
270
271 // Variables declaration
272 private javax.swing.JButton btnAnalizarSin;
273 private javax.swing.JButton btnArchivo;
274 private javax.swing.JButton btnLimpiarSin;
275 private javax.swing.JButton btnTerminalLexico; // Declaración del botón
276 private javax.swing.JPanel jPanel11;
277 private javax.swing.JPanel jPanel12;
278 private javax.swing.JScrollPane jScrollPane1;
279 private javax.swing.JScrollPane jScrollPane3;
280 private javax.swing.JTextArea txtAnalizarSin;
281 }
282 }
```

PANTALLAS RESULTANTES CON PRUEBAS DEL ANALIZADOR SINTACTICO

V. Conclusiones:

EXPLICAR EN UNA CUARTILLA, EL OBJETIVO DEL DESARROLLO DEL PROYECTO, SOFTWARE Y LIBRERIAS QUE UTILIZARON Y SU FUNCION, PROBLEMATICAS PERSONALES QUE ENFRENTARON EN EL ANALISIS LEX, RESULTADO QUE ALCANZARON, COMO SE VINCULAN LOS APRENDIZAJES DE LAS OTRAS UNIDADES CON EL ANALISIS LEXICO Y ANALISIS SINTACTICO DEL COMPILADOR. PORQUE ES UTIL LA MATERIA PARA SU CARRERA.

El proyecto se centra en el desarrollo de un analizador léxico y sintáctico para un lenguaje de programación personalizado, proporcionando una herramienta educativa robusta que facilita la comprensión de las primeras etapas del procesamiento de código fuente en un compilador. En la fase léxica, se logró descomponer el texto de entrada en tokens fundamentales, como palabras reservadas, identificadores, operadores y otros símbolos, utilizando JFlex, una herramienta basada en expresiones regulares para la definición de reglas léxicas. Estas reglas permiten identificar con precisión los elementos del lenguaje y convertirlos en una secuencia estructurada que sirve como entrada para el análisis sintáctico. Adicionalmente, se implementó un sistema para manejar errores léxicos, asegurando retroalimentación clara y detallada a través de una interfaz gráfica que facilita la interacción del usuario.

Por su parte, la interfaz gráfica, desarrollada con Java Swing, ofrece una implementación eficiente y bien estructurada que complementa la funcionalidad del analizador. Incorpora herramientas visuales intuitivas, como un área de texto con números de línea sincronizados dinámicamente y un diseño atractivo utilizando el tema Nimbus. Además, permite cargar archivos, realizar análisis sintácticos, reportar errores en tiempo real y acceder a un terminal léxico a través de botones específicos, lo que añade extensibilidad y mejora la experiencia del usuario. Este diseño no solo resuelve problemas técnicos asociados con la integración del analizador, sino que también convierte la herramienta en un entorno práctico para la depuración y evaluación de código.

Durante el desarrollo del proyecto, se enfrentaron desafíos significativos, como diseñar patrones léxicos suficientemente específicos para evitar ambigüedades entre tokens similares, integrar el analizador con una consola interactiva y garantizar la claridad de los resultados en tiempo real. A pesar de estos retos, el proyecto cumple exitosamente con su propósito principal: servir como una herramienta que permite a los usuarios comprender cómo un compilador procesa el texto fuente, desde la descomposición léxica hasta el análisis sintáctico. Esto no solo contribuye al aprendizaje práctico sobre los fundamentos de los compiladores, sino que también establece una base sólida para proyectos futuros en el desarrollo de lenguajes de programación y herramientas de procesamiento de código, se vincula con las unidades pasadas porque habla sobre los tokens terminales y no terminales, también como todo lo teórico de un inicio se concirtio



GOBIERNO DEL
ESTADO DE MÉXICO

MANUAL DE PRÁCTICAS



en un compilador.