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 LEXICO

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	"*"

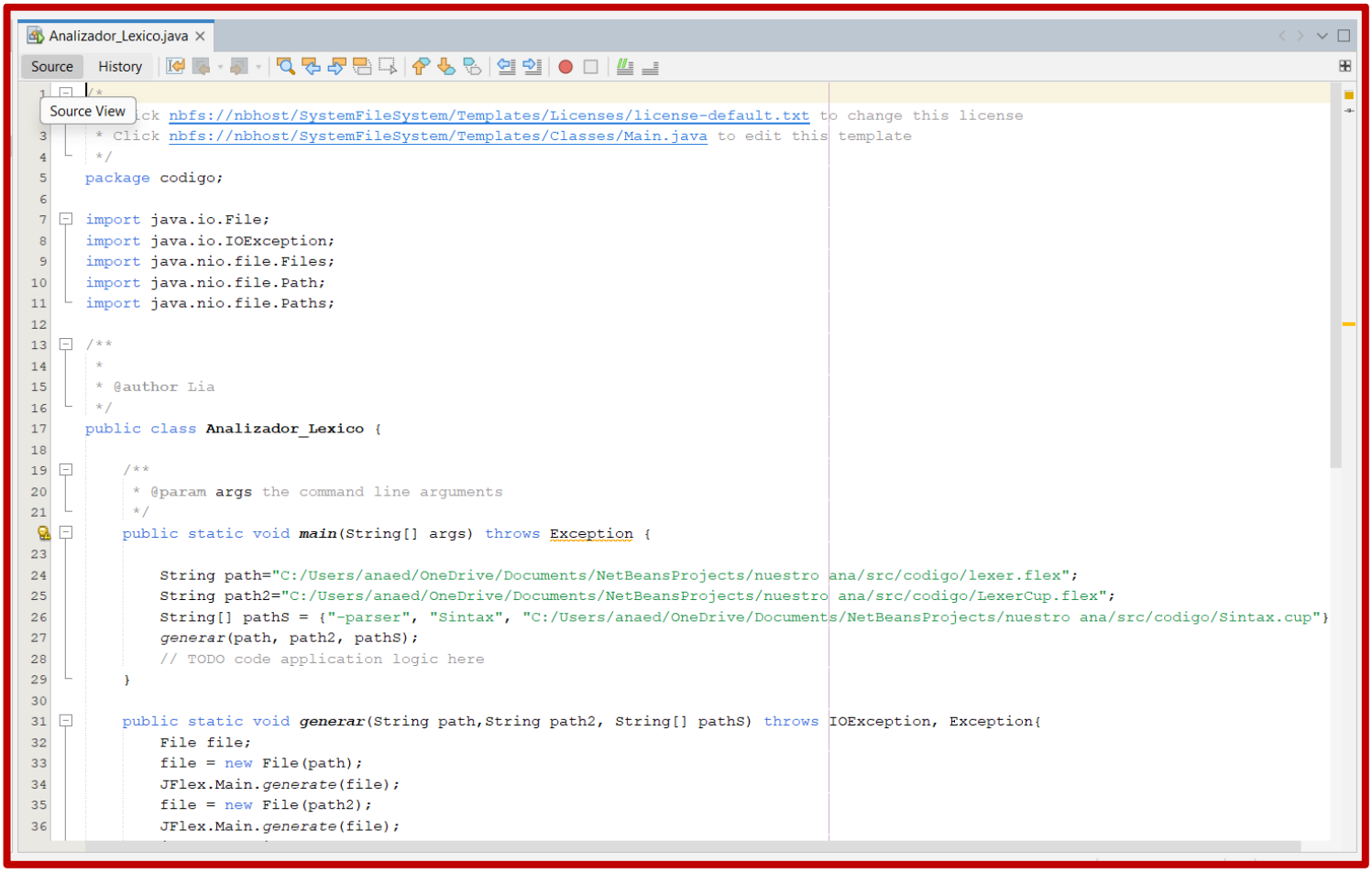
opAritmetico	24	"/"
opAritmetico	25	"%"
opAsignacion	26	"="
opAsignacion	27	"+="
opAsignacion	28	"-="
opAsignacion	29	"*="
opAsignacion	30	"/="
opAsignacion	31	"%="
opRelacionales	32	"=="
opRelacionales	33	"!="
opRelacionales	34	">"
opRelacionales	35	"<"
opRelacionales	36	">="
opRelacionales	37	"<="
opLogicos	38	"&&"
opLogicos	39	"||"
opIncremeneto	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"

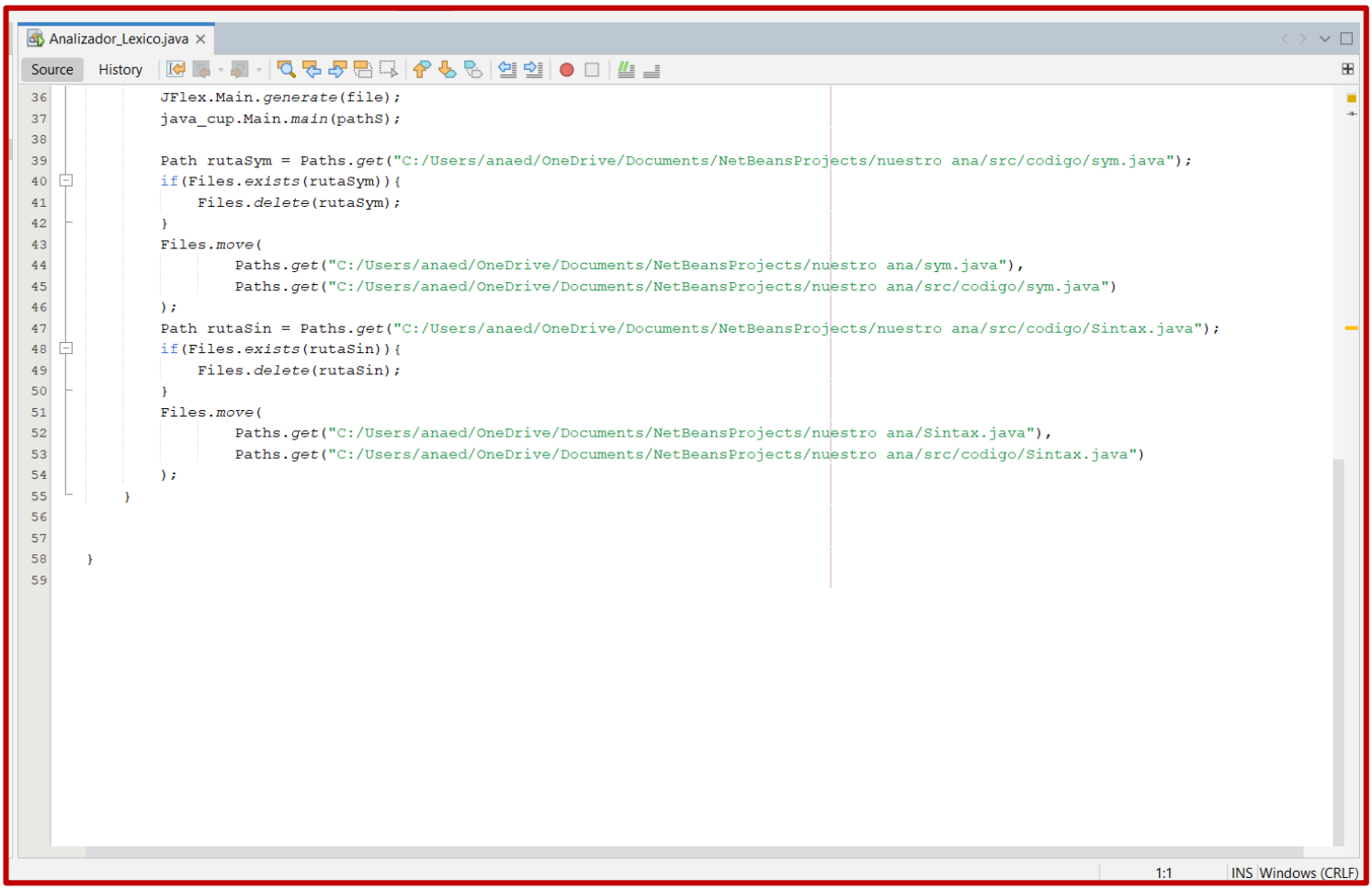
DESCRIPCION DE CADA UNO DE LOS ARCHIVOS GENERADOS EN EL PROYECTO – (1 A 2 PARRAFOS) CONTINUAR COLOCANDO EL CODIGO, COLOCAR UNA LINEA DE SEPARACION ENTRE CADA ARCHIVO A EXPLICAR

(NOTA: COLOCAR BORDES AL CODIGO, Y SE DEBE OBSERVAR LAS LINEAS DE CODIGO, TODO EL CODIGO Y LETRA VISIBLE)

Analizador_Lexico

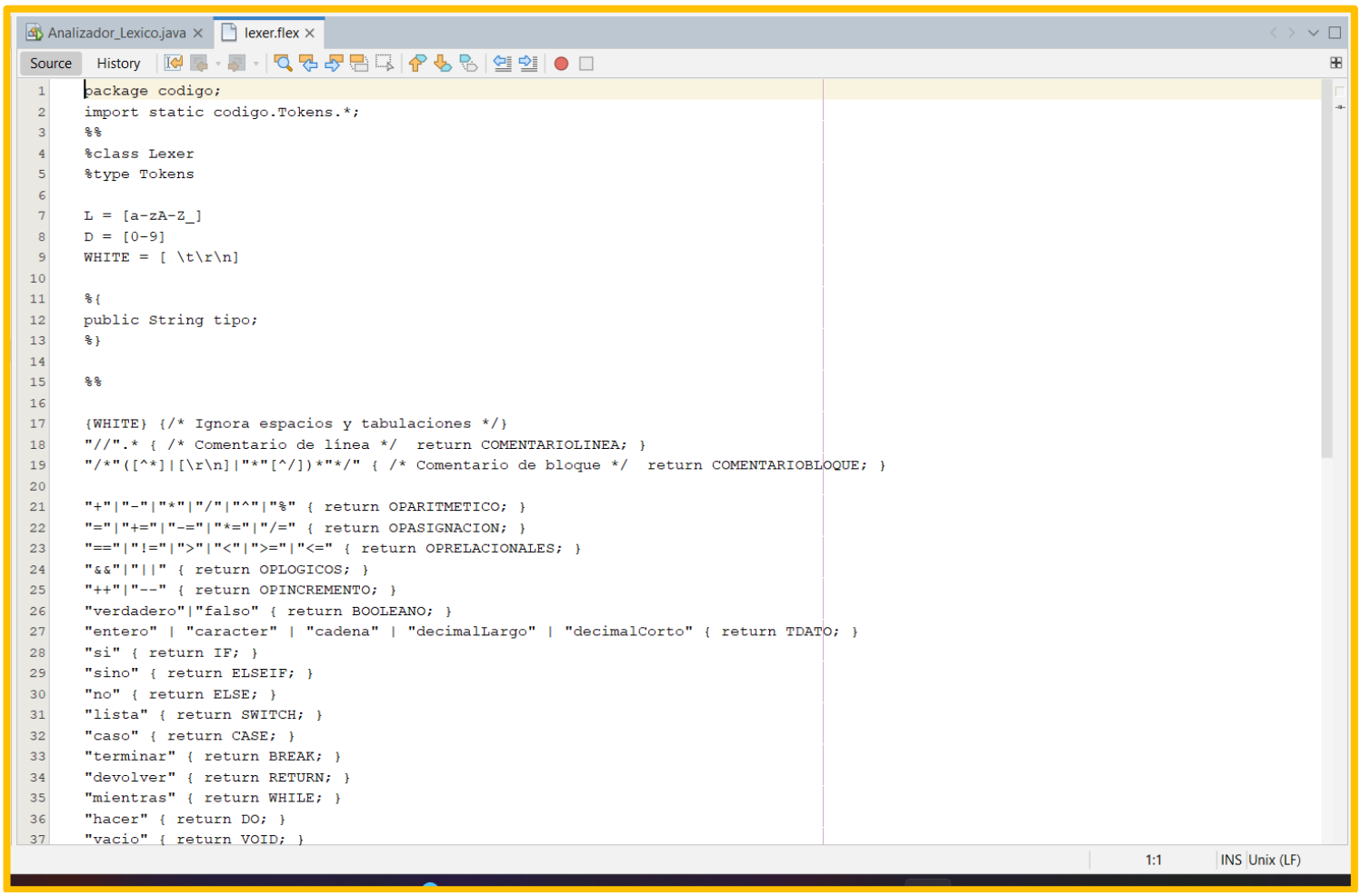
El código en Java implementa un analizador léxico y sintáctico para un proyecto utilizando herramientas como JFlex y CUP. La clase principal Analizador_Lexico contiene el método main, donde se definen las rutas de dos archivos .flex utilizados por JFlex para generar analizadores léxicos y un archivo .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/codigo), previa eliminación de las versiones existentes para evitar conflictos. Esto permite mantener el código fuente del proyecto organizado y actualizado con los cambios en la gramática o las reglas léxicas definidas. El programa también utiliza las clases Path y Files para manipular archivos y garantizar que las rutas de destino estén correctamente estructuradas.



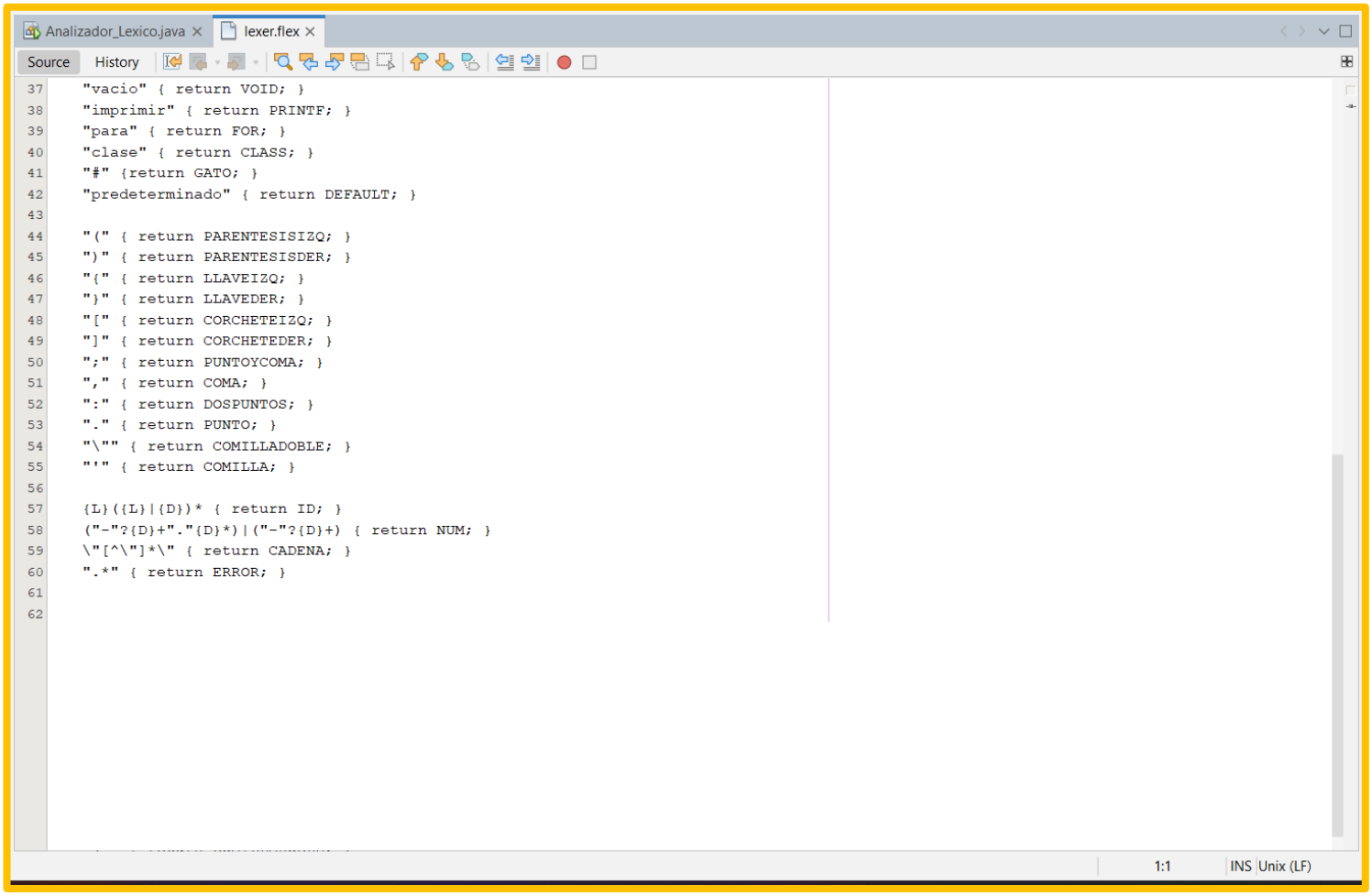


lexer.flex

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

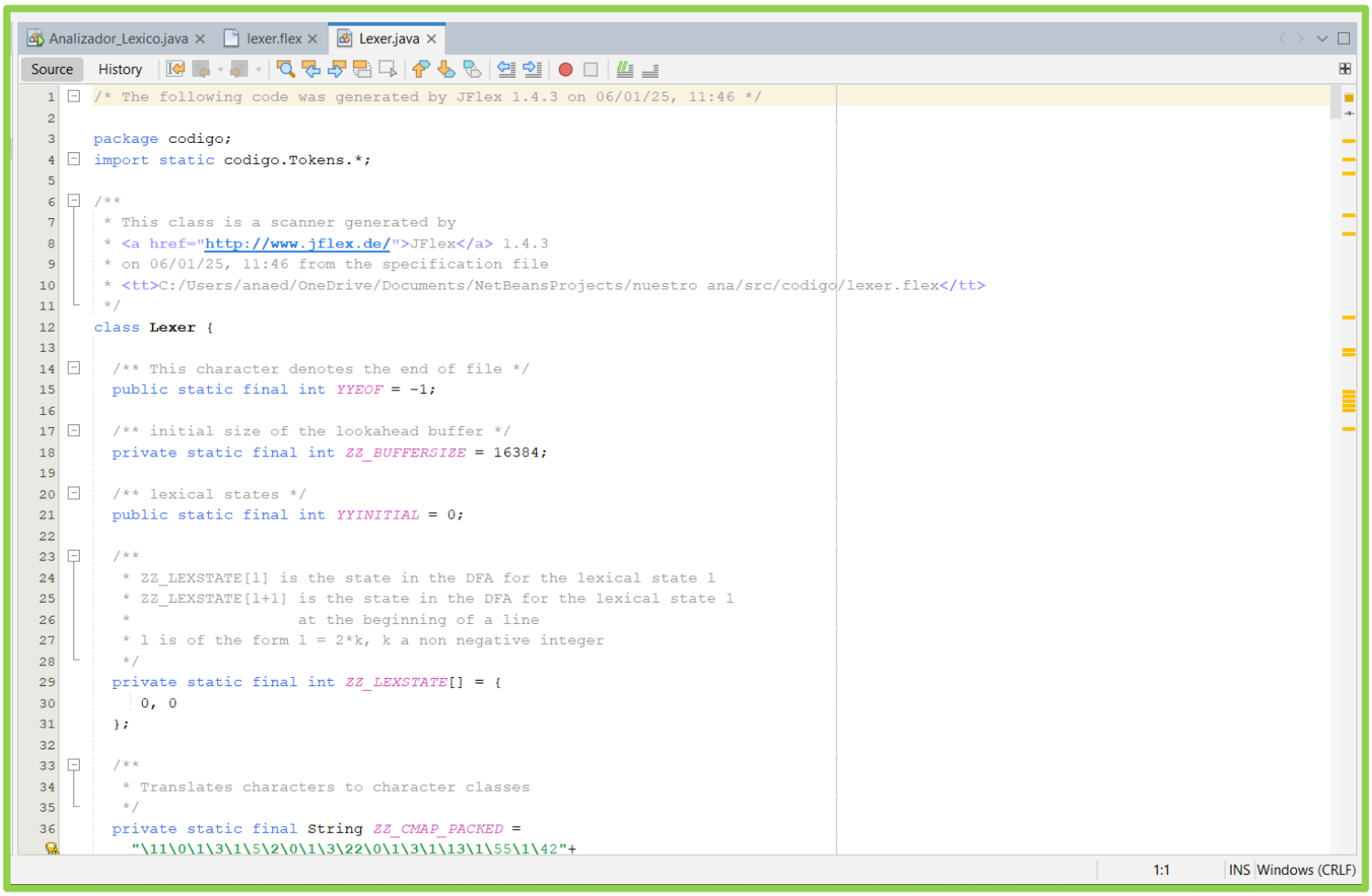



```
package codigo;
import static codigo.Tokens.*;
%%
%class Lexer
%type Tokens

L = [a-zA-Z_]
D = [0-9]
WHITE = [ \t\r\n]

%{
public String tipo;
%}

%%

{WHITE} {/* Ignora espacios y tabulaciones */}
"//".* { /* Comentario de línea */  return COMENTARIOLINEA; }
"/*"([^*]|[\r\n]|"*"[^/])*"*/" { /* Comentario de bloque */  return COMENTARIOBLOQUE; }

"+"|"-"|"*"|"/"|"^"|"%" { return OPARITMETICO; }
"="|"+="|"-="|"*="|"/=" { return OPASIGNACION; }
"=="|"!="|">"|"<"|">="|"<=" { return OPRELACIONALES; }
"&&"|"||" { return OPLOGICOS; }
"++"|"--" { return OPINCREMENTO; }
"verdadero"|"falso" { return BOOLEANO; }
"entero" | "caracter" | "cadena" | "decimalLargo" | "decimalCorto" { return TDATO; }
"si" { return IF; }
"sino" { return ELSEIF; }
"no" { return ELSE; }
"lista" { return SWITCH; }
"caso" { return CASE; }
"terminar" { return BREAK; }
"devolver" { return RETURN; }
"mientras" { return WHILE; }
"hacer" { return DO; }
"vacio" { return VOID; }
```

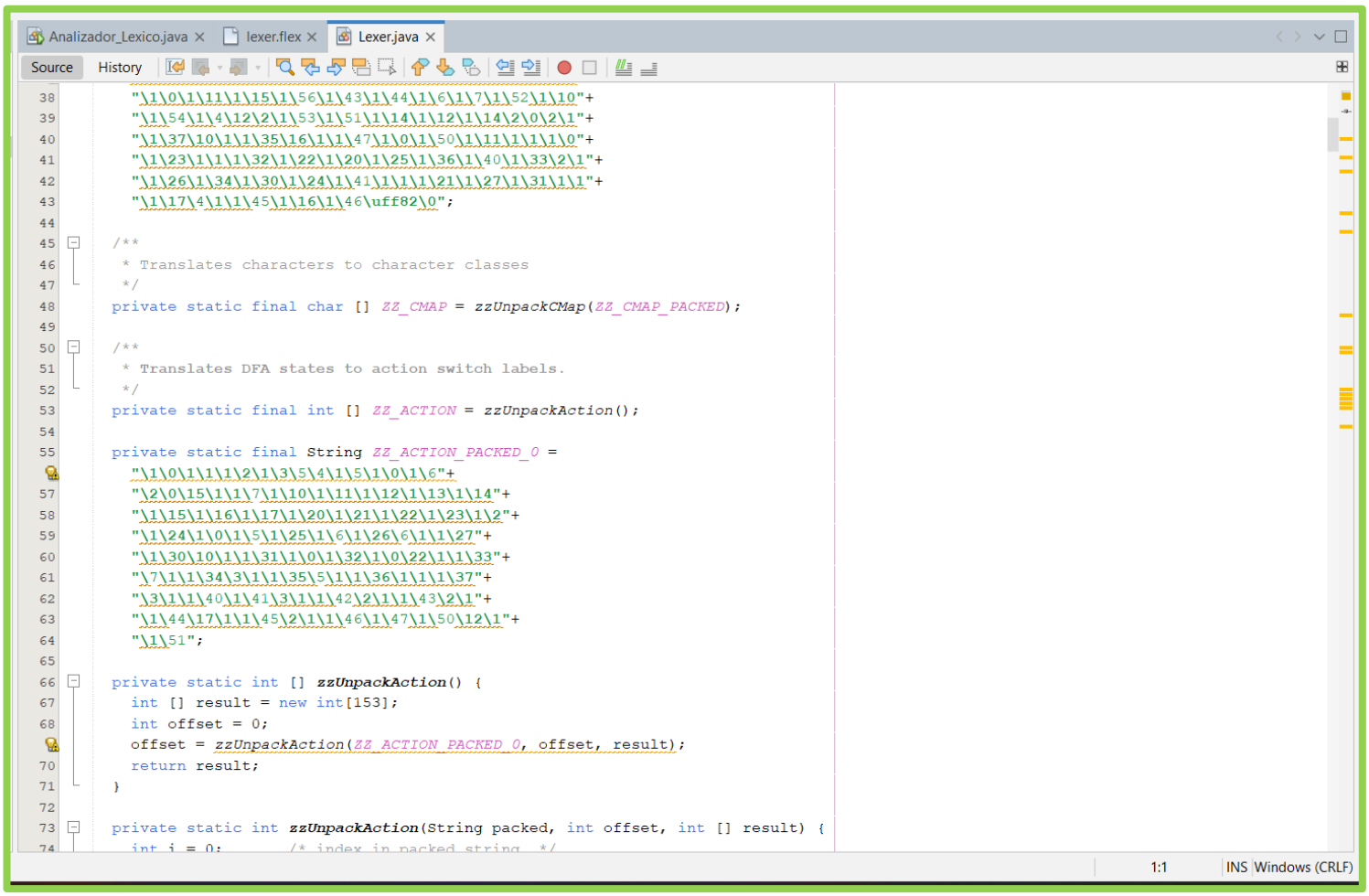


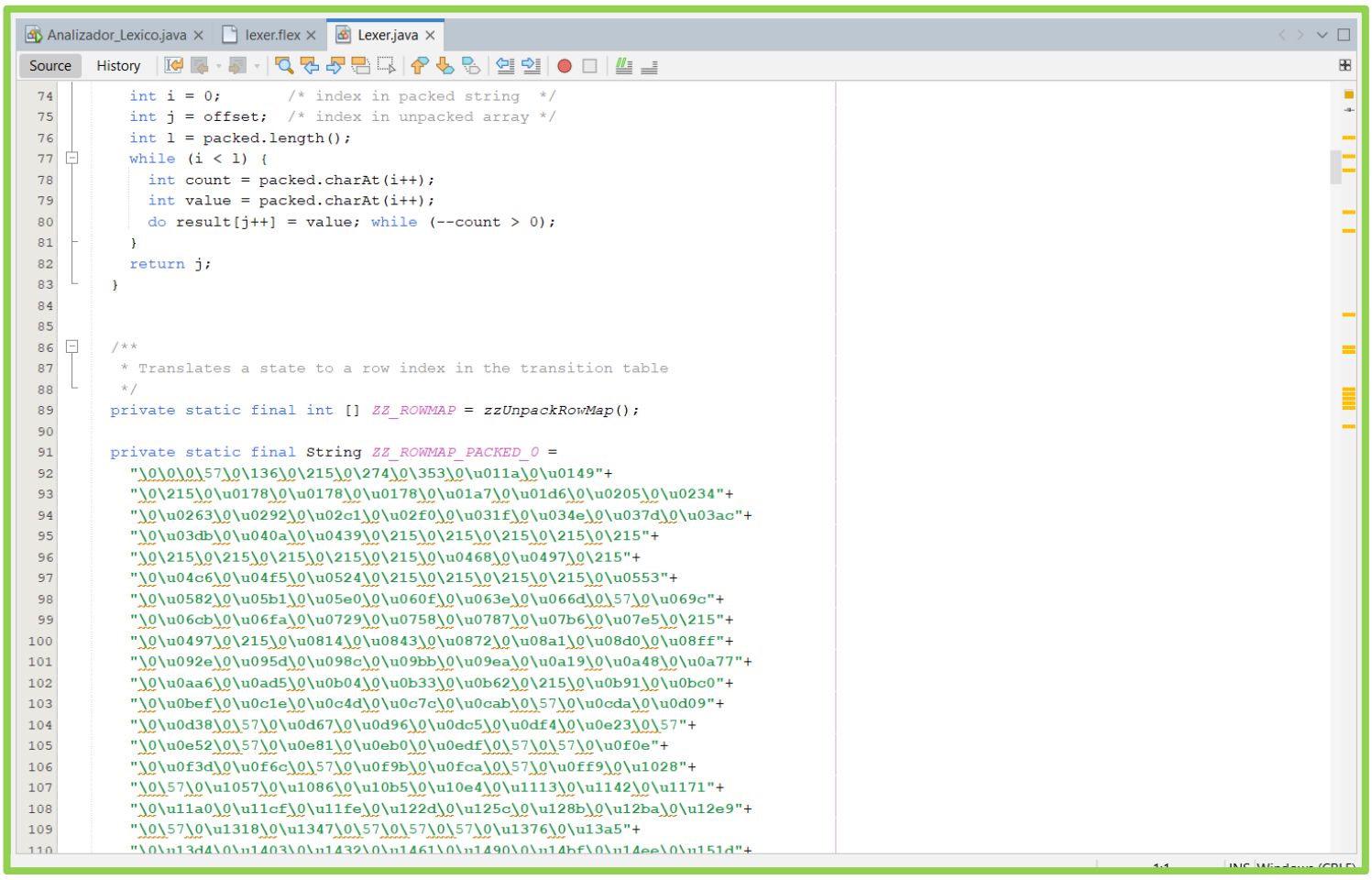
Lexer.java

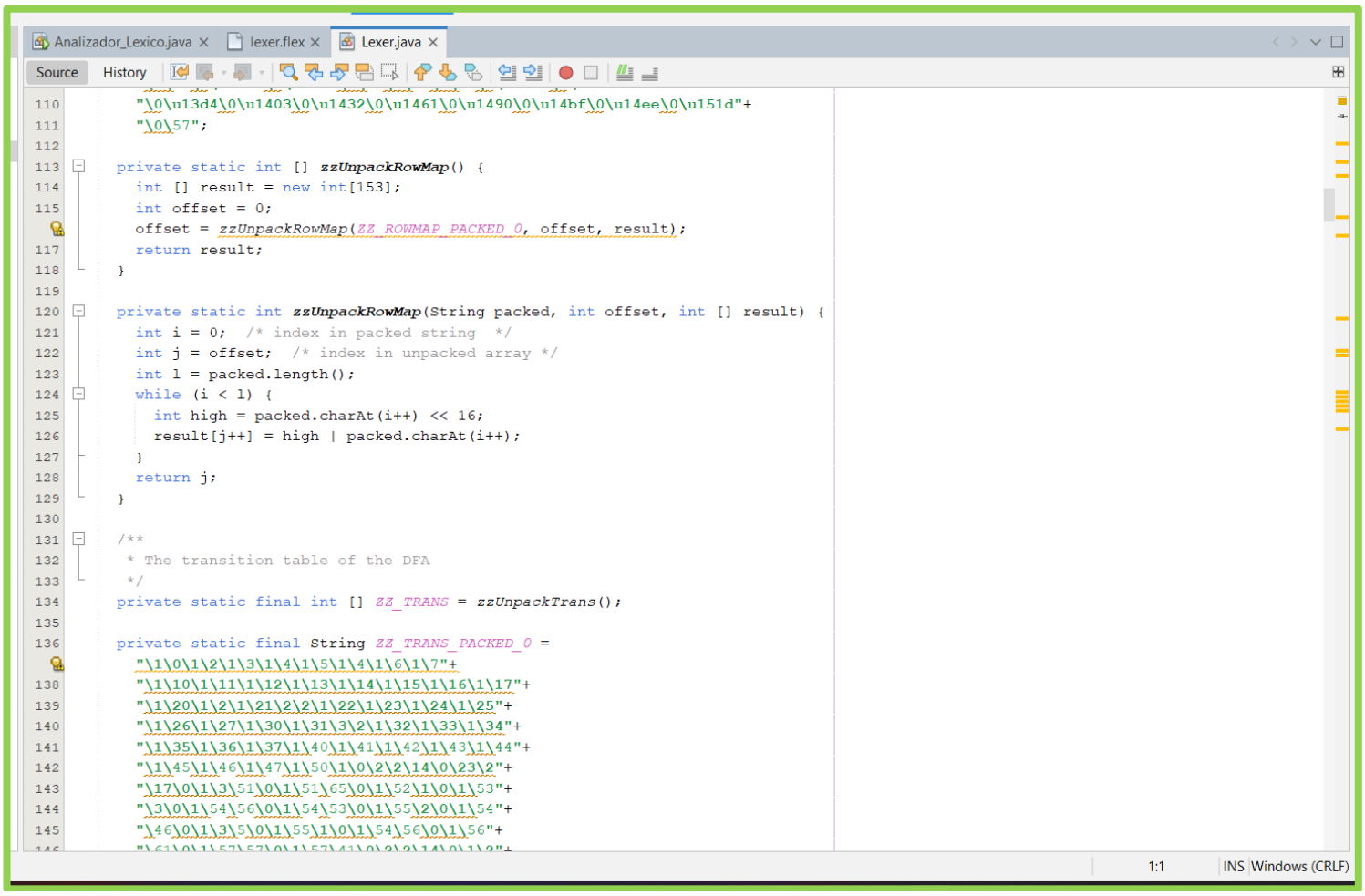
El Lexer.java se genera mediante el lexer.flex lo que este genera es un autómata finito, con las tablas de transiciones para que pueda reconocer los tokens y los convierta en un lexema y lo pueda reconocer el analizador.

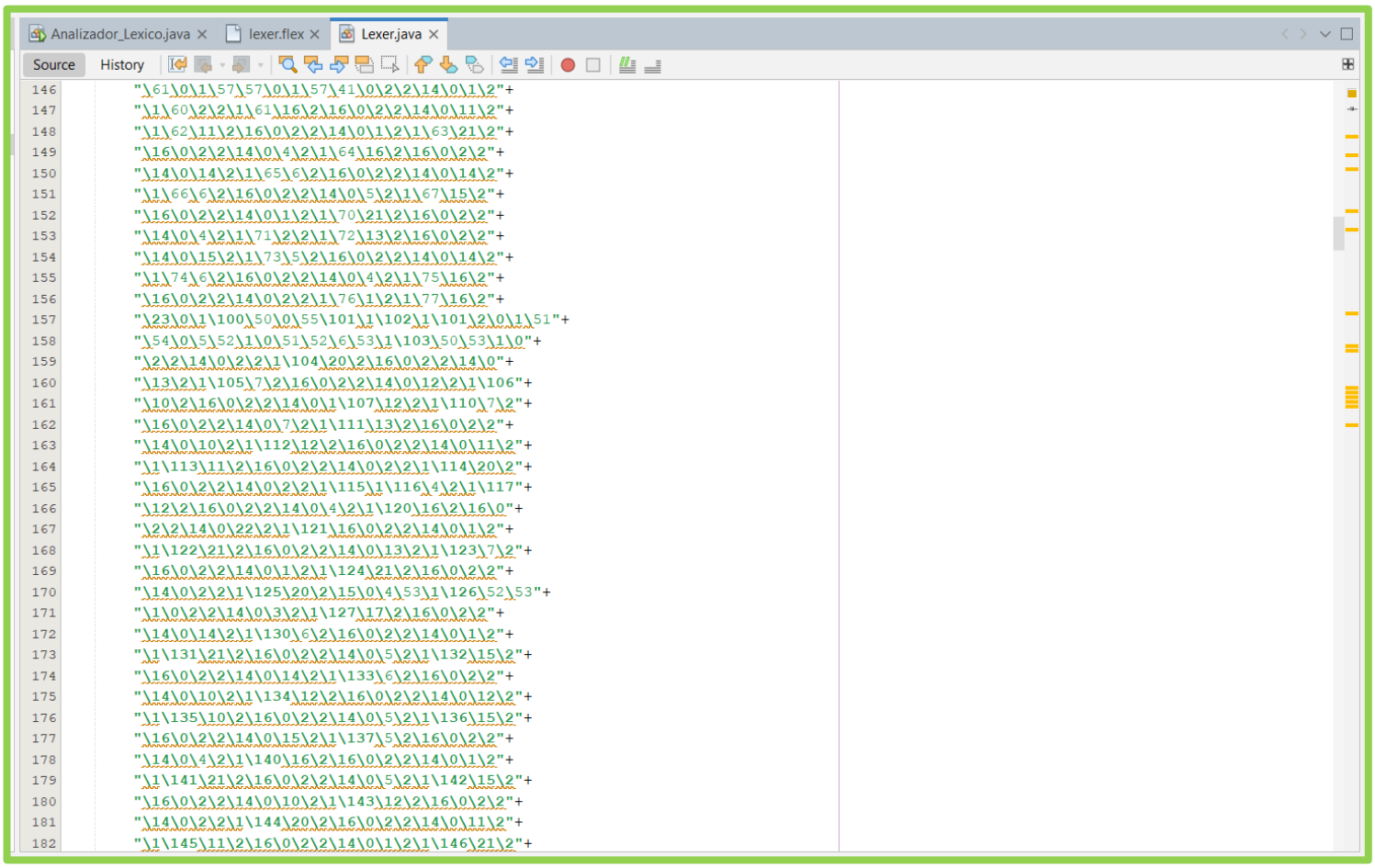



```
/* The following code was generated by JFlex 1.4.3 on 06/01/25, 11:46 */

package codigo;
import static codigo.Tokens.*;

/**
 * This class is a scanner generated by
 * <a href="http://www.jflex.de/">JFlex</a> 1.4.3
 * on 06/01/25, 11:46 from the specification file
 * <tt>C:/Users/anaed/OneDrive/Documents/NetBeansProjects/nuestro ana/src/codigo/lexer.flex</tt>
 */
class Lexer {

  /** This character denotes the end of file */
  public static final int YYEOF = -1;

  /** initial size of the lookahead buffer */
  private static final int ZZ_BUFFERSIZE = 16384;

  /** lexical states */
  public static final int YYINITIAL = 0;

  /**
   * ZZ_LEXSTATE[l] is the state in the DFA for the lexical state l
   * ZZ_LEXSTATE[l+1] is the state in the DFA for the lexical state l
   *                  at the beginning of a line
   * l is of the form l = 2*k, k a non negative integer
   */
  private static final int ZZ_LEXSTATE[] = {
     0, 0
  };

  /**
   * Translates characters to character classes
   */
  private static final String ZZ_CMAP_PACKED =
    "\11\0\1\3\1\5\2\0\1\3\22\0\1\3\1\13\1\55\1\42"+
```

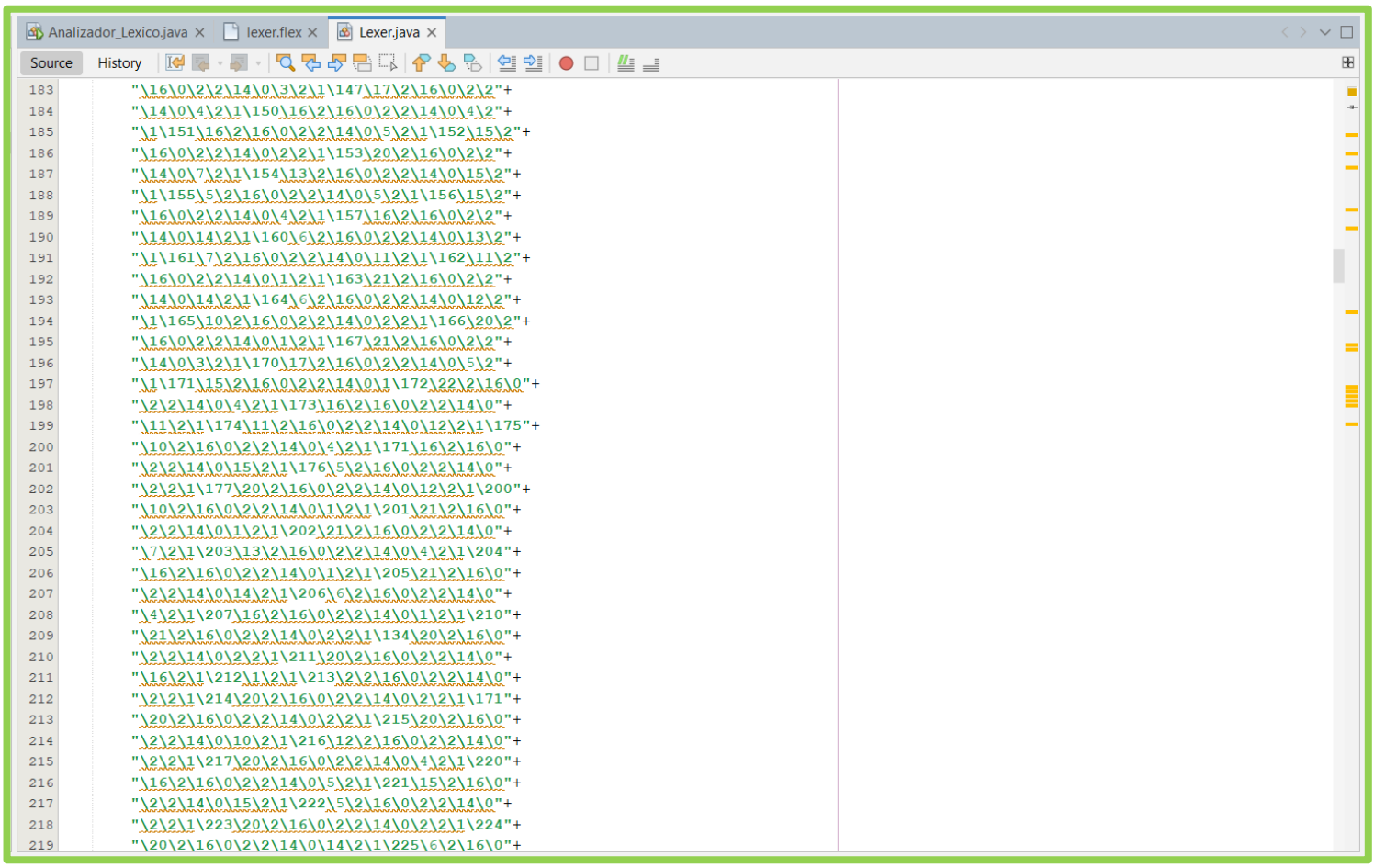




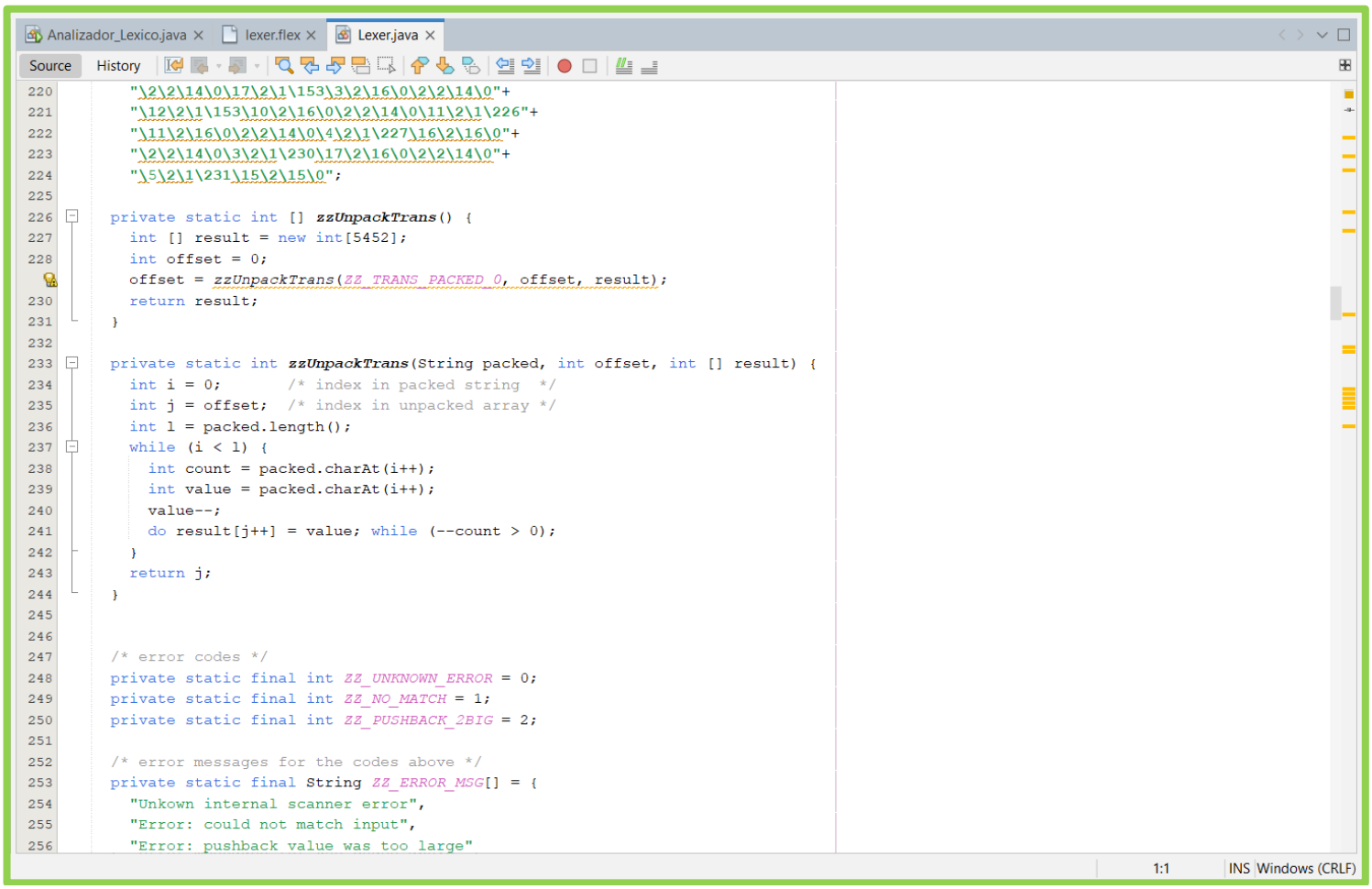


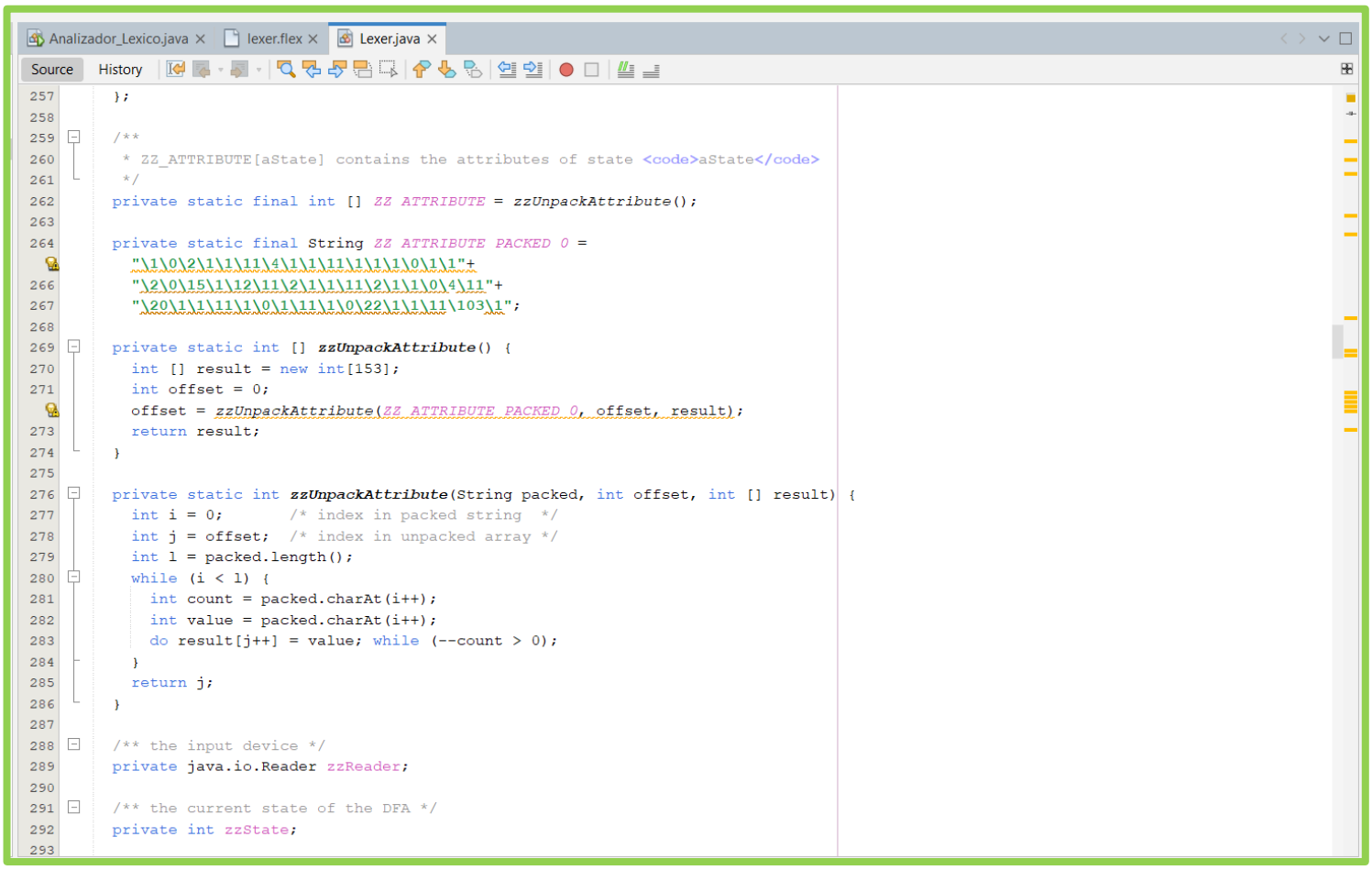


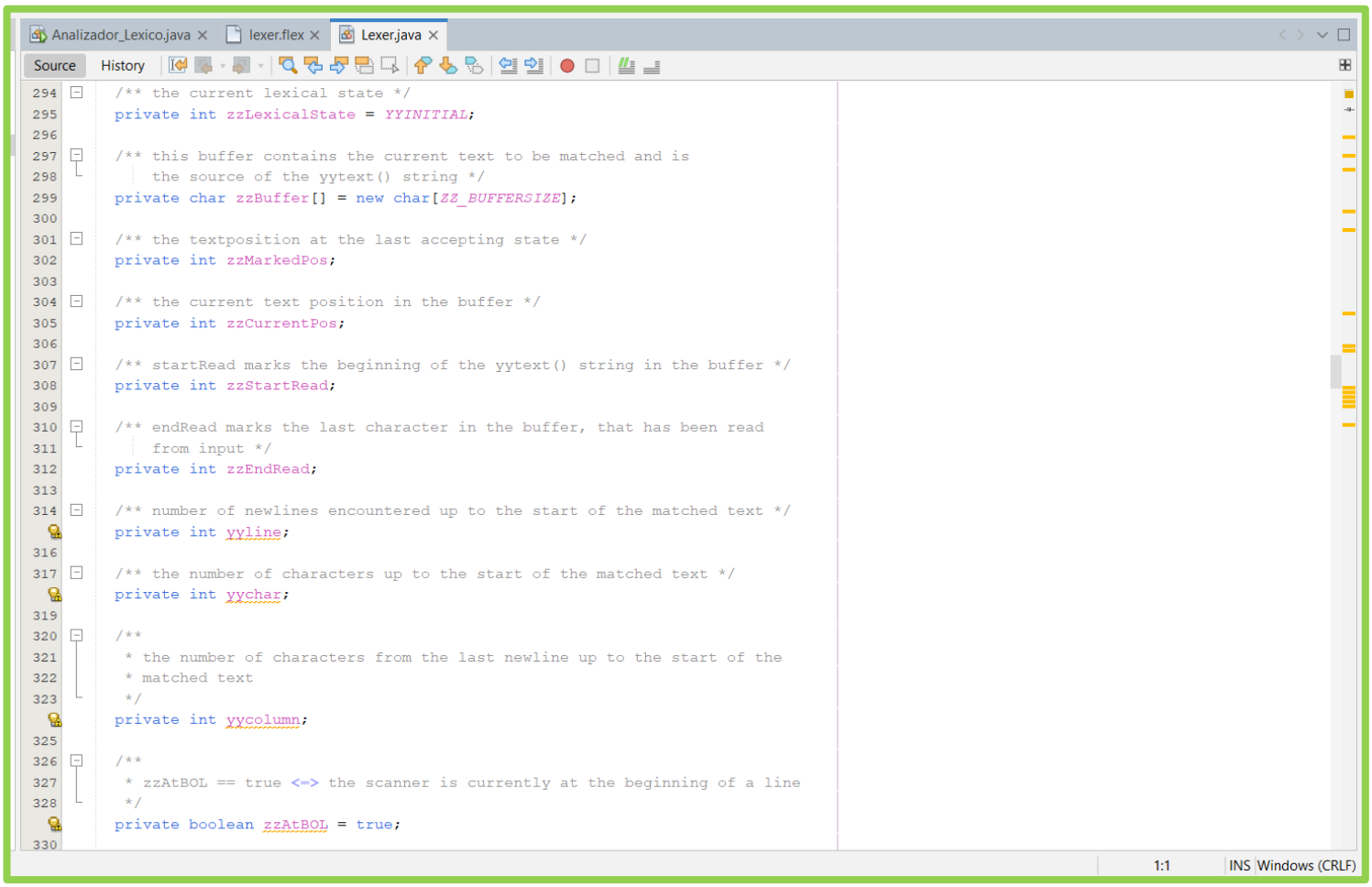

```
        "\61\0\1\57\57\0\1\57\41\0\2\2\14\0\1\2"+
        "\1\60\2\2\1\61\16\2\16\0\2\2\14\0\11\2"+
        "\1\62\11\2\16\0\2\2\14\0\1\2\1\63\21\2"+
        "\16\0\2\2\14\0\4\2\1\64\16\2\16\0\2\2"+
        "\14\0\14\2\1\65\6\2\16\0\2\2\14\0\14\2"+
        "\1\66\6\2\16\0\2\2\14\0\5\2\1\67\15\2"+
        "\16\0\2\2\14\0\1\2\1\70\21\2\16\0\2\2"+
        "\14\0\4\2\1\71\2\2\1\72\13\2\16\0\2\2"+
        "\14\0\15\2\1\73\5\2\16\0\2\2\14\0\14\2"+
        "\1\74\6\2\16\0\2\2\14\0\4\2\1\75\16\2"+
        "\16\0\2\2\14\0\2\2\1\76\1\2\1\77\16\2"+
        "\23\0\1\100\50\0\55\101\1\102\1\101\2\0\1\51"+
        "\54\0\5\52\1\0\51\52\6\53\1\103\50\53\1\0"+
        "\2\2\14\0\2\2\1\104\20\2\16\0\2\2\14\0"+
        "\13\2\1\105\7\2\16\0\2\2\14\0\12\2\1\106"+
        "\10\2\16\0\2\2\14\0\1\107\12\2\1\110\7\2"+
        "\16\0\2\2\14\0\7\2\1\111\13\2\16\0\2\2"+
        "\14\0\10\2\1\112\12\2\16\0\2\2\14\0\11\2"+
        "\1\113\11\2\16\0\2\2\14\0\2\2\1\114\20\2"+
        "\16\0\2\2\14\0\2\2\1\115\1\116\4\2\1\117"+
        "\12\2\16\0\2\2\14\0\4\2\1\120\16\2\16\0"+
        "\2\2\14\0\22\2\1\121\16\0\2\2\14\0\1\2"+
        "\1\122\21\2\16\0\2\2\14\0\13\2\1\123\7\2"+
        "\16\0\2\2\14\0\1\2\1\124\21\2\16\0\2\2"+
        "\14\0\2\2\1\125\20\2\15\0\4\53\1\126\52\53"+
        "\1\0\2\2\14\0\3\2\1\127\17\2\16\0\2\2"+
        "\14\0\14\2\1\130\6\2\16\0\2\2\14\0\1\2"+
        "\1\131\21\2\16\0\2\2\14\0\5\2\1\132\15\2"+
        "\16\0\2\2\14\0\14\2\1\133\6\2\16\0\2\2"+
        "\14\0\10\2\1\134\12\2\16\0\2\2\14\0\12\2"+
        "\1\135\10\2\16\0\2\2\14\0\5\2\1\136\15\2"+
        "\16\0\2\2\14\0\15\2\1\137\5\2\16\0\2\2"+
        "\14\0\4\2\1\140\16\2\16\0\2\2\14\0\1\2"+
        "\1\141\21\2\16\0\2\2\14\0\5\2\1\142\15\2"+
        "\16\0\2\2\14\0\10\2\1\143\12\2\16\0\2\2"+
        "\14\0\2\2\1\144\20\2\16\0\2\2\14\0\11\2"+
        "\1\145\11\2\16\0\2\2\14\0\1\2\1\146\21\2"+
```

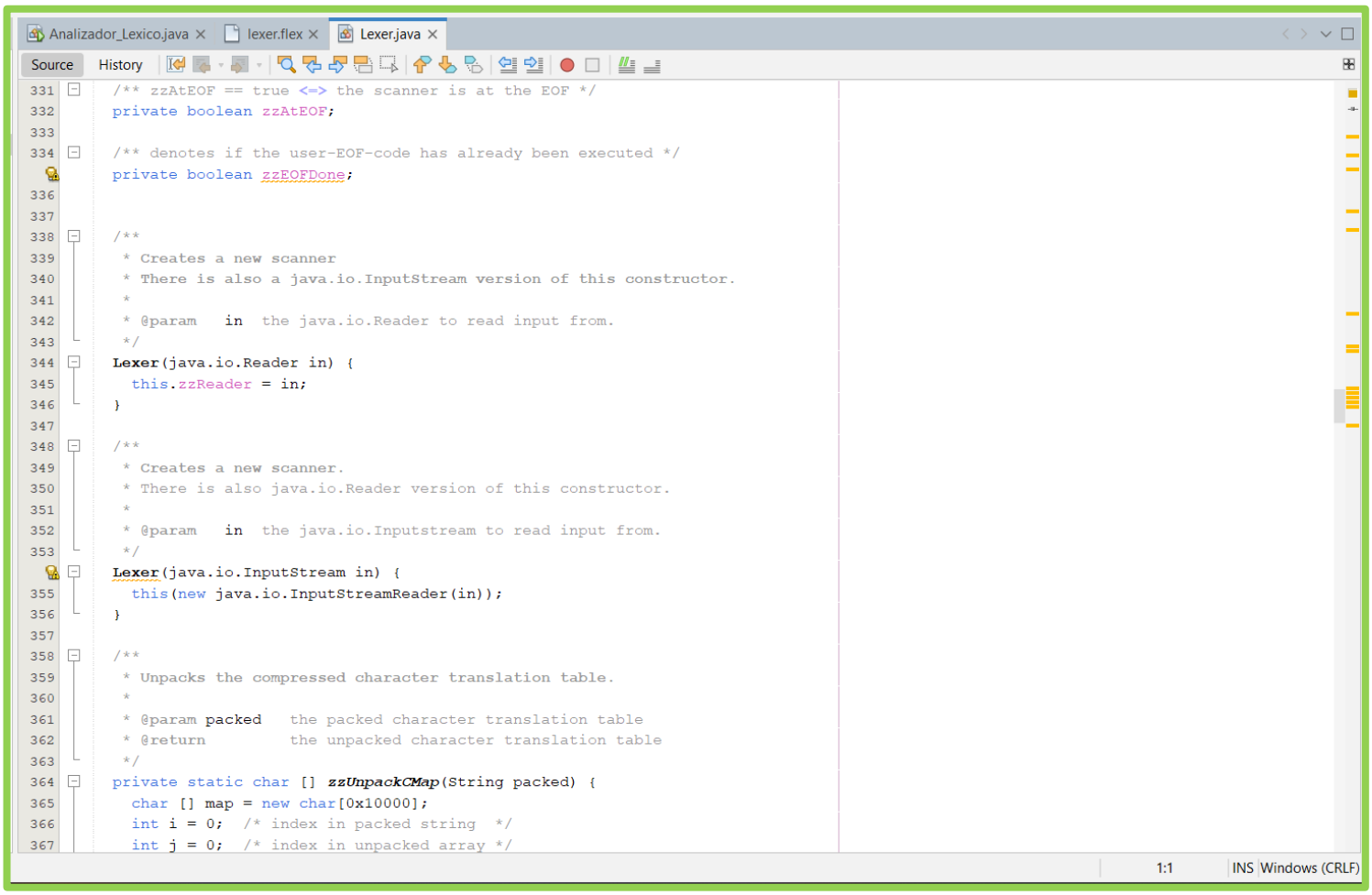


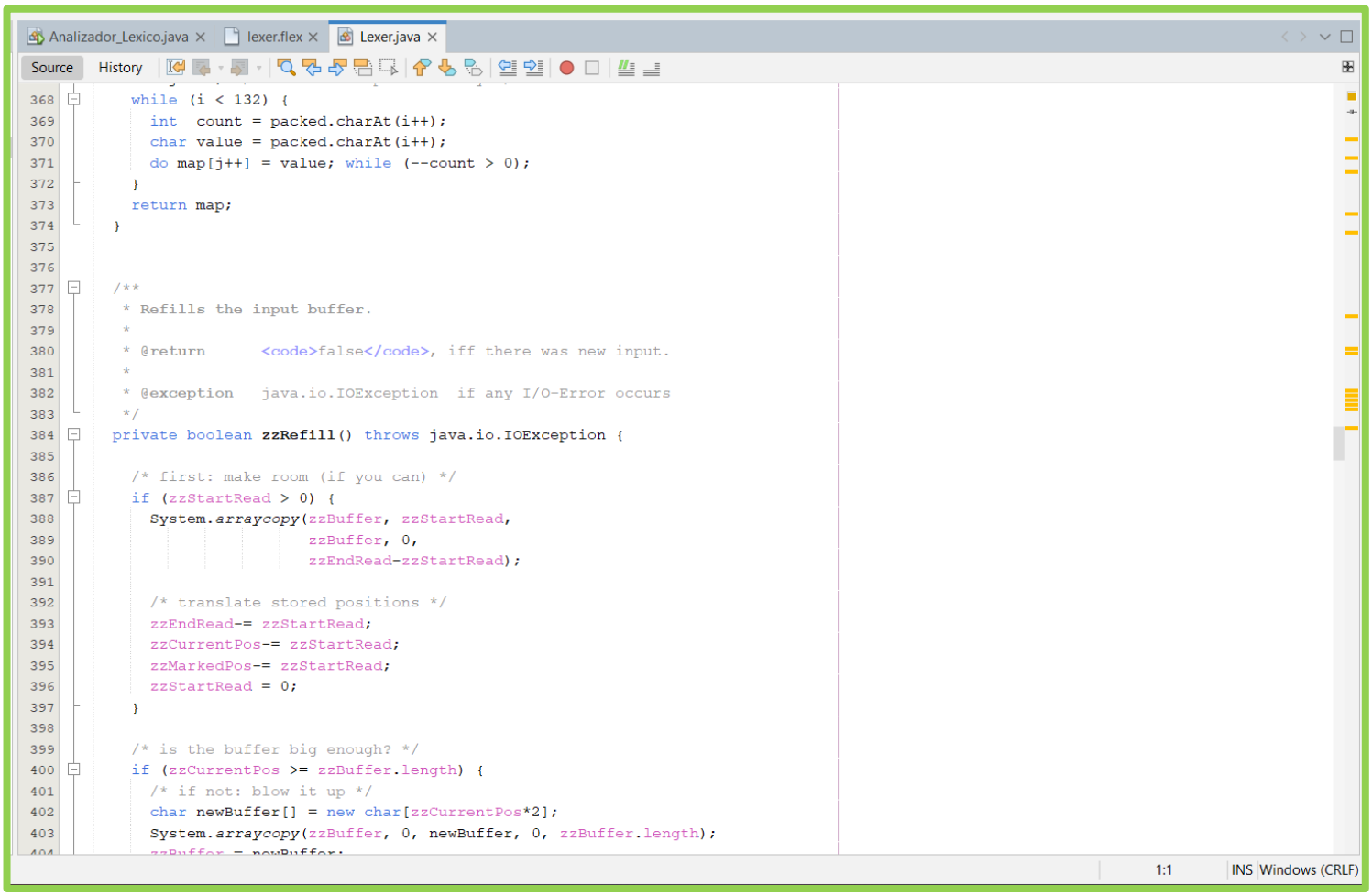

```
        "\16\0\2\2\14\0\3\2\1\147\17\2\16\0\2\2"+
        "\14\0\4\2\1\150\16\2\16\0\2\2\14\0\4\2"+
        "\1\151\16\2\16\0\2\2\14\0\5\2\1\152\15\2"+
        "\16\0\2\2\14\0\2\2\1\153\20\2\16\0\2\2"+
        "\14\0\7\2\1\154\13\2\16\0\2\2\14\0\15\2"+
        "\1\155\5\2\16\0\2\2\14\0\5\2\1\156\15\2"+
        "\16\0\2\2\14\0\4\2\1\157\16\2\16\0\2\2"+
        "\14\0\14\2\1\160\6\2\16\0\2\2\14\0\13\2"+
        "\1\161\7\2\16\0\2\2\14\0\11\2\1\162\11\2"+
        "\16\0\2\2\14\0\1\2\1\163\21\2\16\0\2\2"+
        "\14\0\14\2\1\164\6\2\16\0\2\2\14\0\12\2"+
        "\1\165\10\2\16\0\2\2\14\0\2\2\1\166\20\2"+
        "\16\0\2\2\14\0\1\2\1\167\21\2\16\0\2\2"+
        "\14\0\3\2\1\170\17\2\16\0\2\2\14\0\5\2"+
        "\1\171\15\2\16\0\2\2\14\0\1\172\22\2\16\0"+
        "\2\2\14\0\4\2\1\173\16\2\16\0\2\2\14\0"+
        "\11\2\1\174\11\2\16\0\2\2\14\0\12\2\1\175"+
        "\10\2\16\0\2\2\14\0\4\2\1\171\16\2\16\0"+
        "\2\2\14\0\15\2\1\176\5\2\16\0\2\2\14\0"+
        "\2\2\1\177\20\2\16\0\2\2\14\0\12\2\1\200"+
        "\10\2\16\0\2\2\14\0\1\2\1\201\21\2\16\0"+
        "\2\2\14\0\1\2\1\202\21\2\16\0\2\2\14\0"+
        "\7\2\1\203\13\2\16\0\2\2\14\0\4\2\1\204"+
        "\16\2\16\0\2\2\14\0\1\2\1\205\21\2\16\0"+
        "\2\2\14\0\14\2\1\206\6\2\16\0\2\2\14\0"+
        "\4\2\1\207\16\2\16\0\2\2\14\0\1\2\1\210"+
        "\21\2\16\0\2\2\14\0\2\2\1\134\20\2\16\0"+
        "\2\2\14\0\2\2\1\211\20\2\16\0\2\2\14\0"+
        "\16\2\1\212\1\2\1\213\2\2\16\0\2\2\14\0"+
        "\2\2\1\214\20\2\16\0\2\2\14\0\2\2\1\171"+
        "\20\2\16\0\2\2\14\0\2\2\1\215\20\2\16\0"+
        "\2\2\14\0\10\2\1\216\12\2\16\0\2\2\14\0"+
        "\2\2\1\217\20\2\16\0\2\2\14\0\4\2\1\220"+
        "\16\2\16\0\2\2\14\0\5\2\1\221\15\2\16\0"+
        "\2\2\14\0\15\2\1\222\5\2\16\0\2\2\14\0"+
        "\2\2\1\223\20\2\16\0\2\2\14\0\2\2\1\224"+
        "\20\2\16\0\2\2\14\0\14\2\1\225\6\2\16\0"+
```

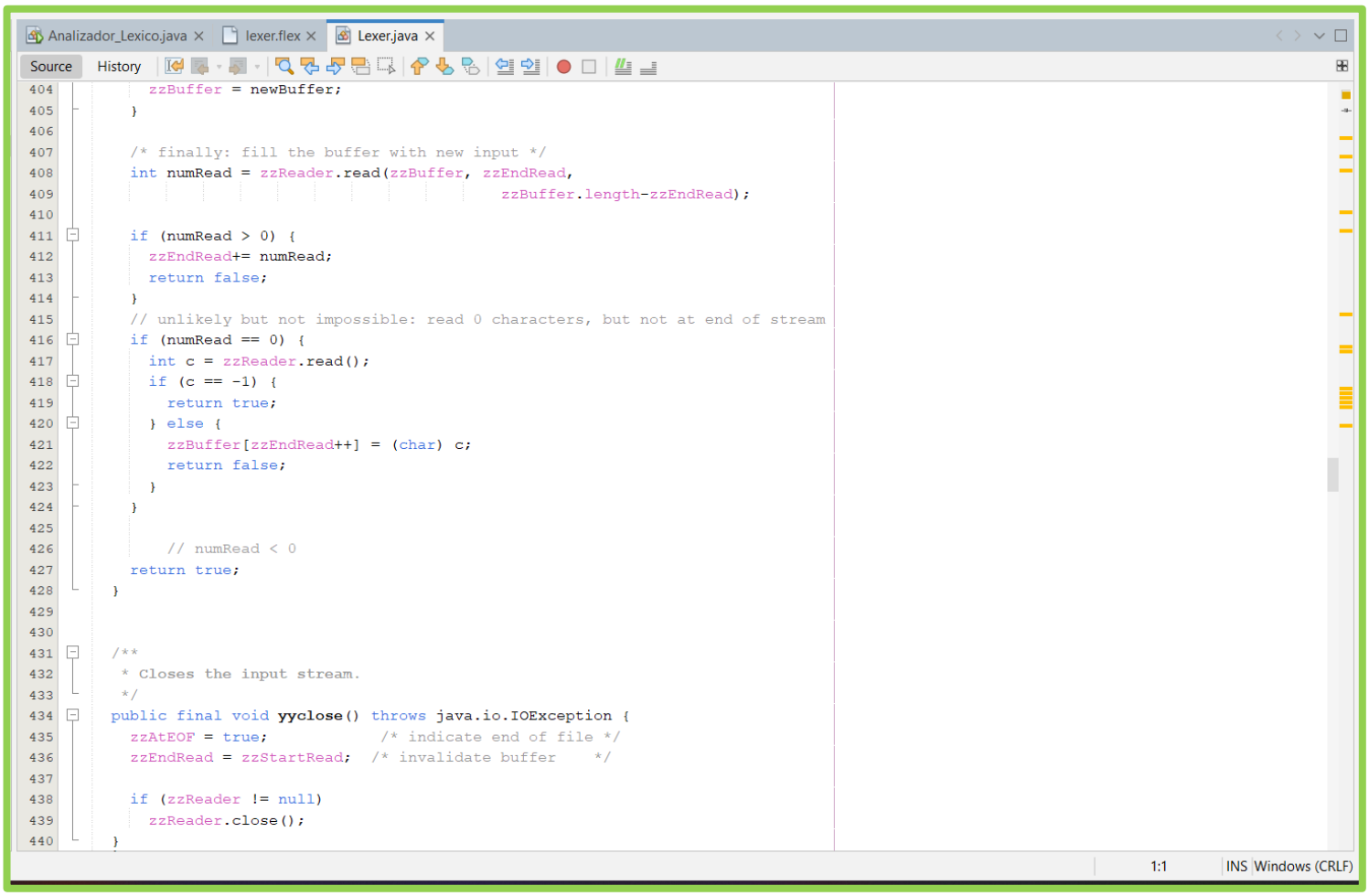


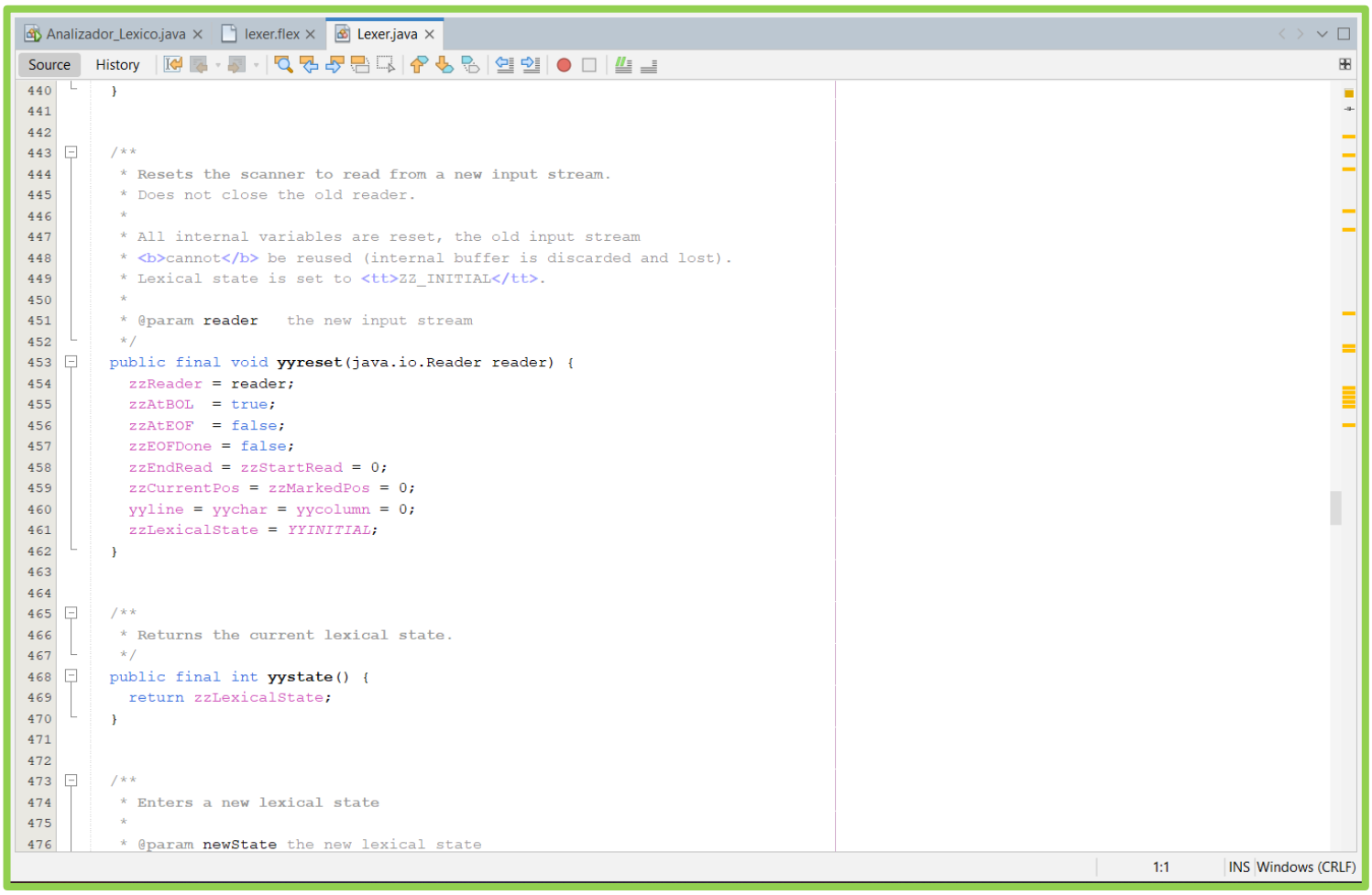

Analizador_Lexico.java
lexer.flex
Lexer.java
Source
History
};
/**
* ZZ_ATTRIBUTE[aState] contains the attributes of state <code>aState</code>
*/
private static final int [] ZZ_ATTRIBUTE = zzUnpackAttribute();
private static final String ZZ_ATTRIBUTE_PACKED_0 =
"\1\0\2\1\1\11\4\1\1\11\1\1\1\0\1\1"+
"\2\0\15\1\12\11\2\1\1\11\2\1\1\0\4\11"+
"\20\1\1\11\1\0\1\11\1\0\22\1\1\11\103\1";
private static int [] zzUnpackAttribute() {
int [] result = new int[153];
int offset = 0;
offset = zzUnpackAttribute(ZZ_ATTRIBUTE_PACKED_0, offset, result);
return result;
}
private static int zzUnpackAttribute(String packed, int offset, int [] result) {
int i = 0; /* index in packed string */
int j = offset; /* index in unpacked array */
int l = packed.length();
while (i < l) {
int count = packed.charAt(i++);
int value = packed.charAt(i++);
do result[j++] = value; while (--count > 0);
}
return j;
}
/** the input device */
private java.io.Reader zzReader;
/** the current state of the DFA */
private int zzState;

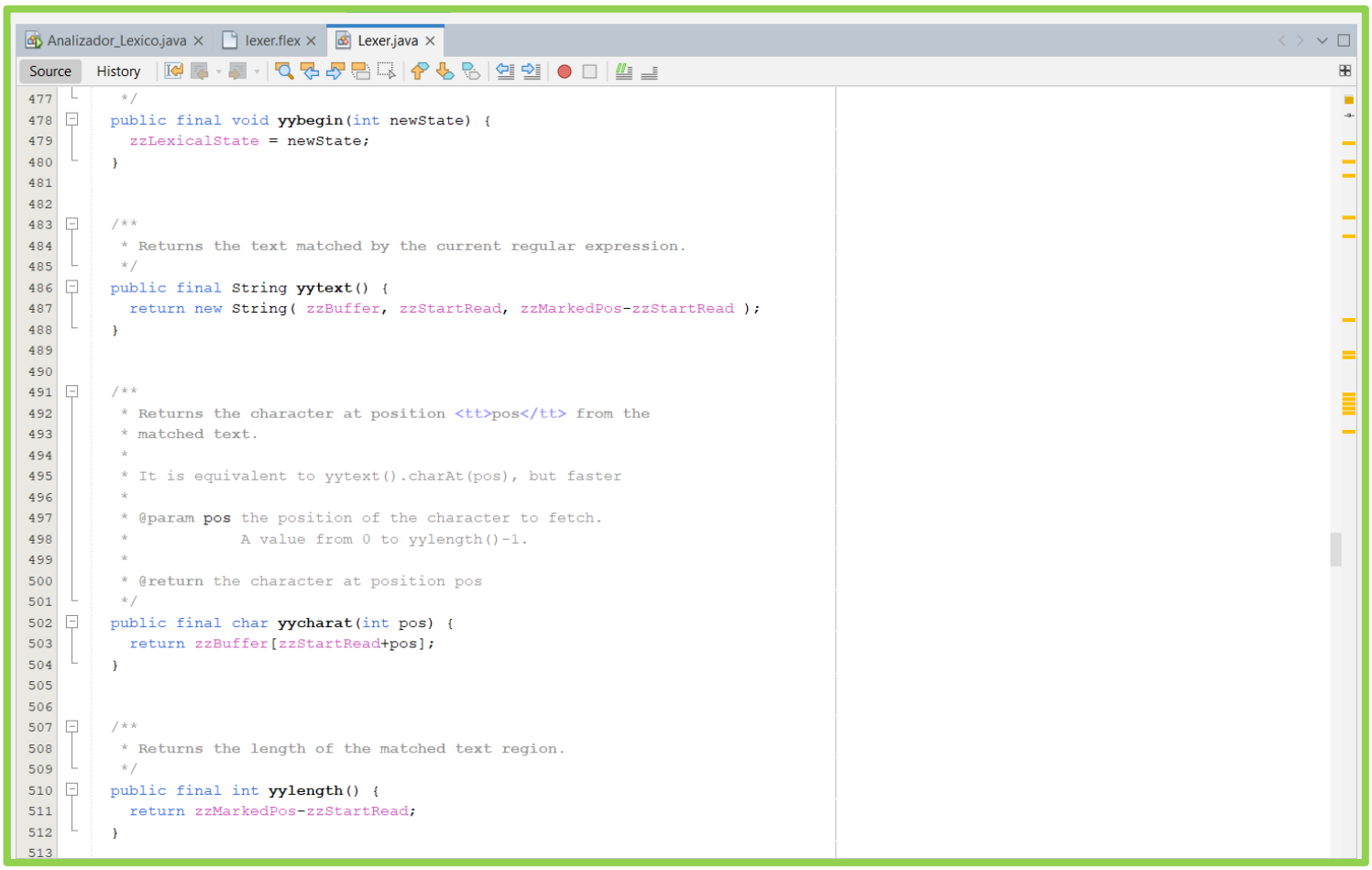

Analizador_Lexico.java
lexer.flex
Lexer.java
Source
History
/** the current lexical state */
private int zzLexicalState = YYINITIAL;
/** this buffer contains the current text to be matched and is
the source of the yytext() string */
private char zzBuffer[] = new char[ZZ_BUFFERSIZE];
/** the textposition at the last accepting state */
private int zzMarkedPos;
/** the current text position in the buffer */
private int zzCurrentPos;
/** startRead marks the beginning of the yytext() string in the buffer */
private int zzStartRead;
/** endRead marks the last character in the buffer, that has been read
from input */
private int zzEndRead;
/** number of newlines encountered up to the start of the matched text */
private int yyline;
/** the number of characters up to the start of the matched text */
private int yychar;
/**
* the number of characters from the last newline up to the start of the
* matched text
*/
private int yycolumn;
/**
* zzAtBOL == true <=> the scanner is currently at the beginning of a line
*/
private boolean zzAtBOL = true;
1:1
INS
Windows (CRLF)

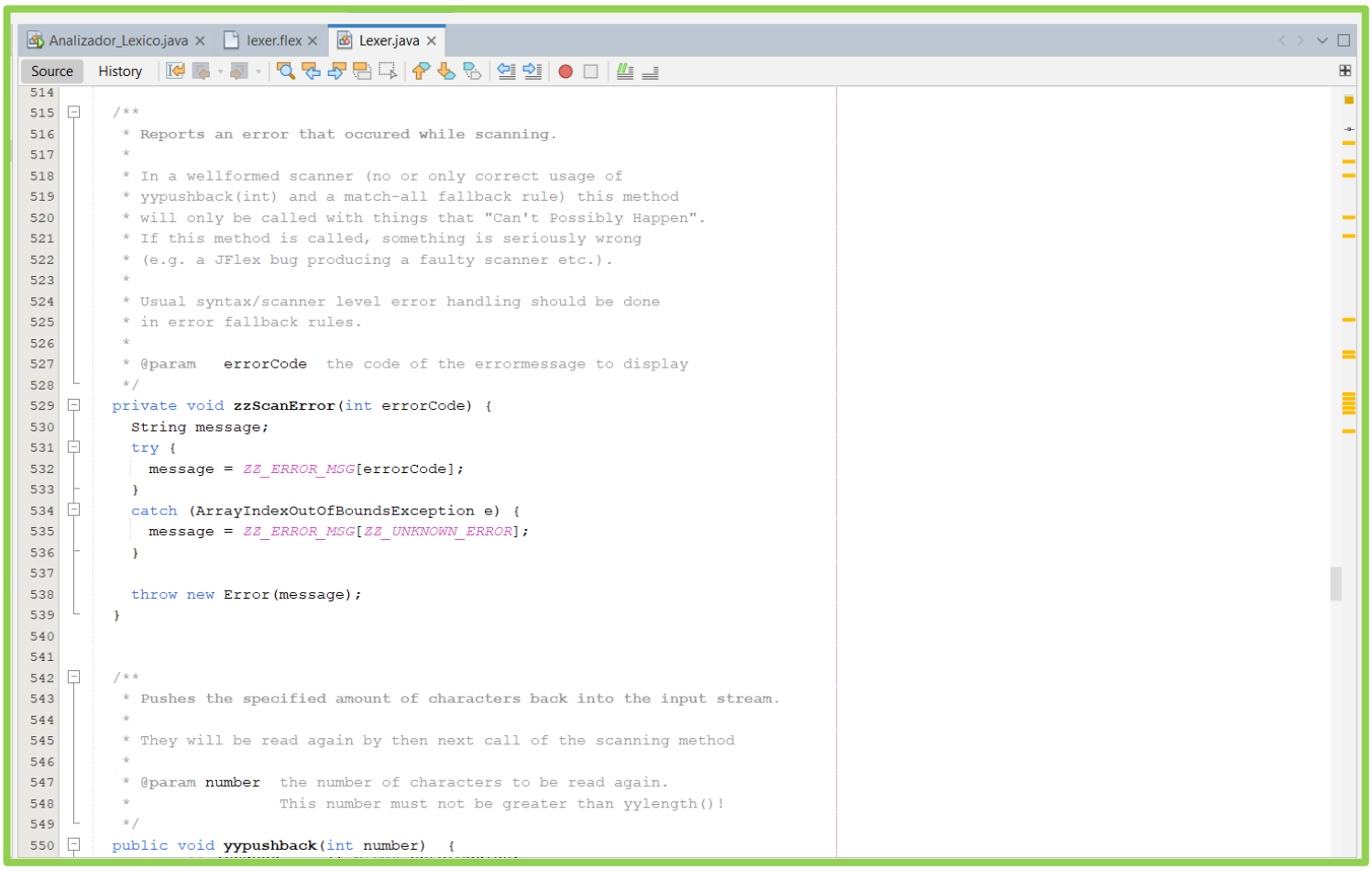


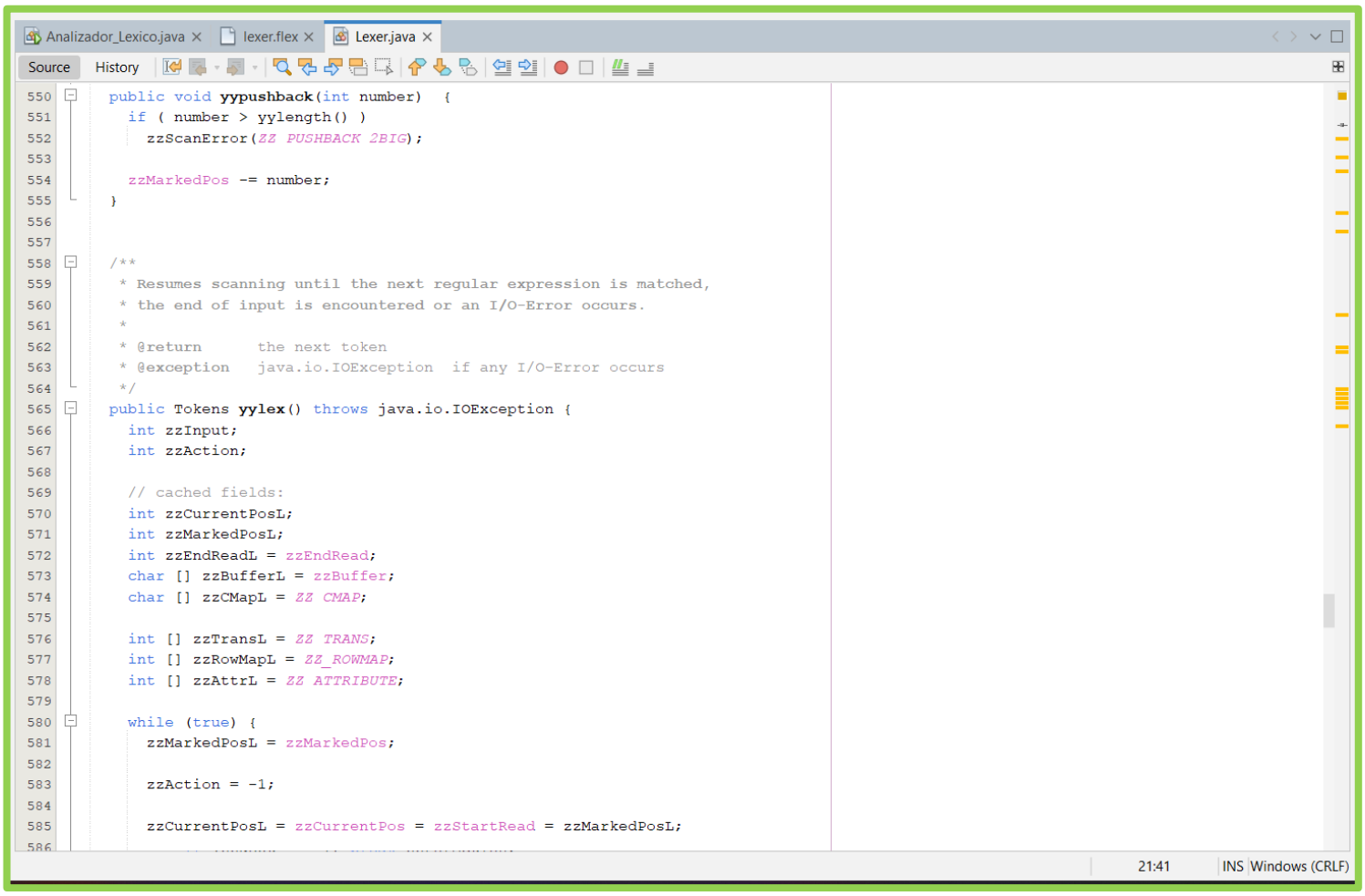


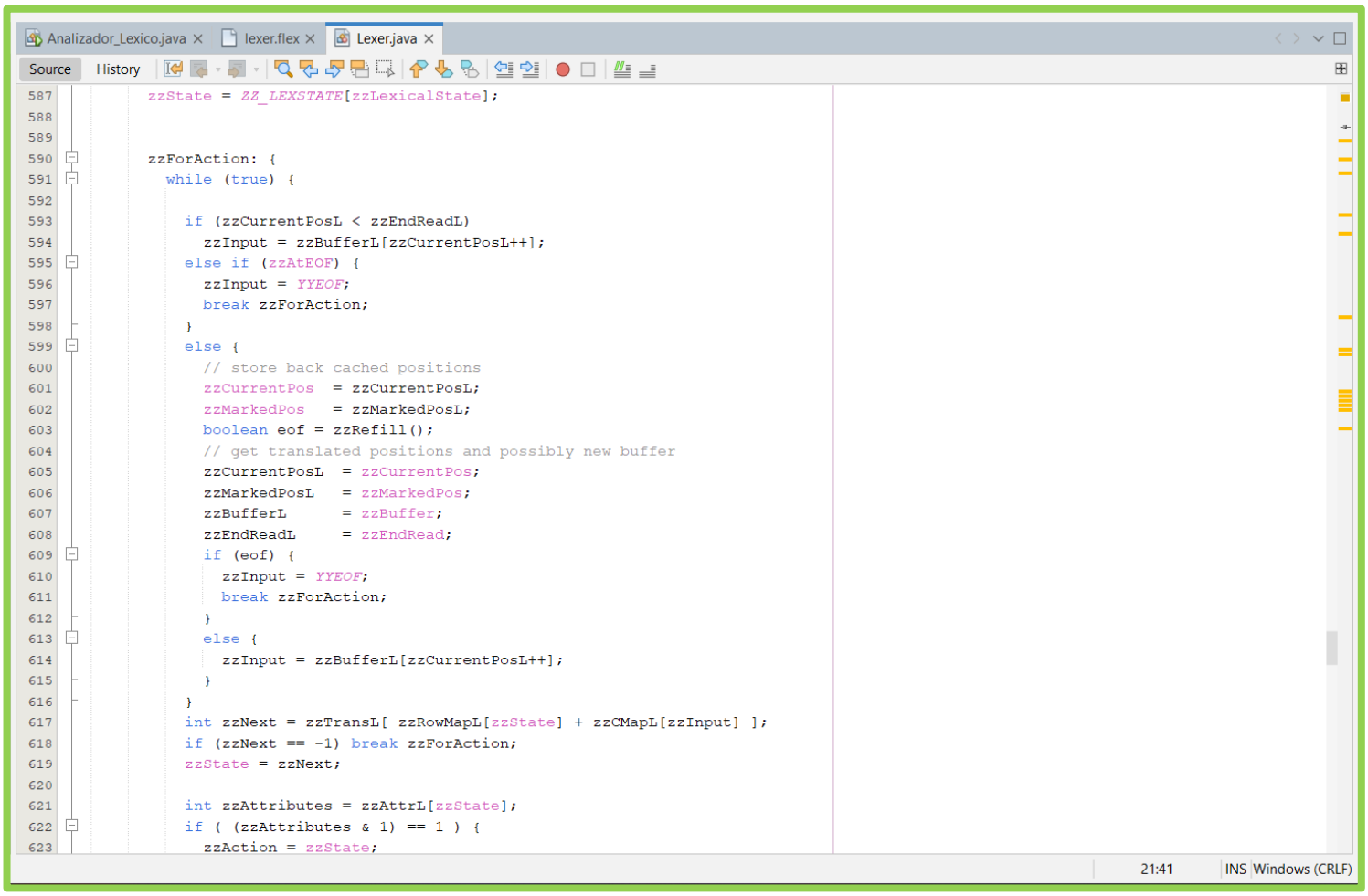


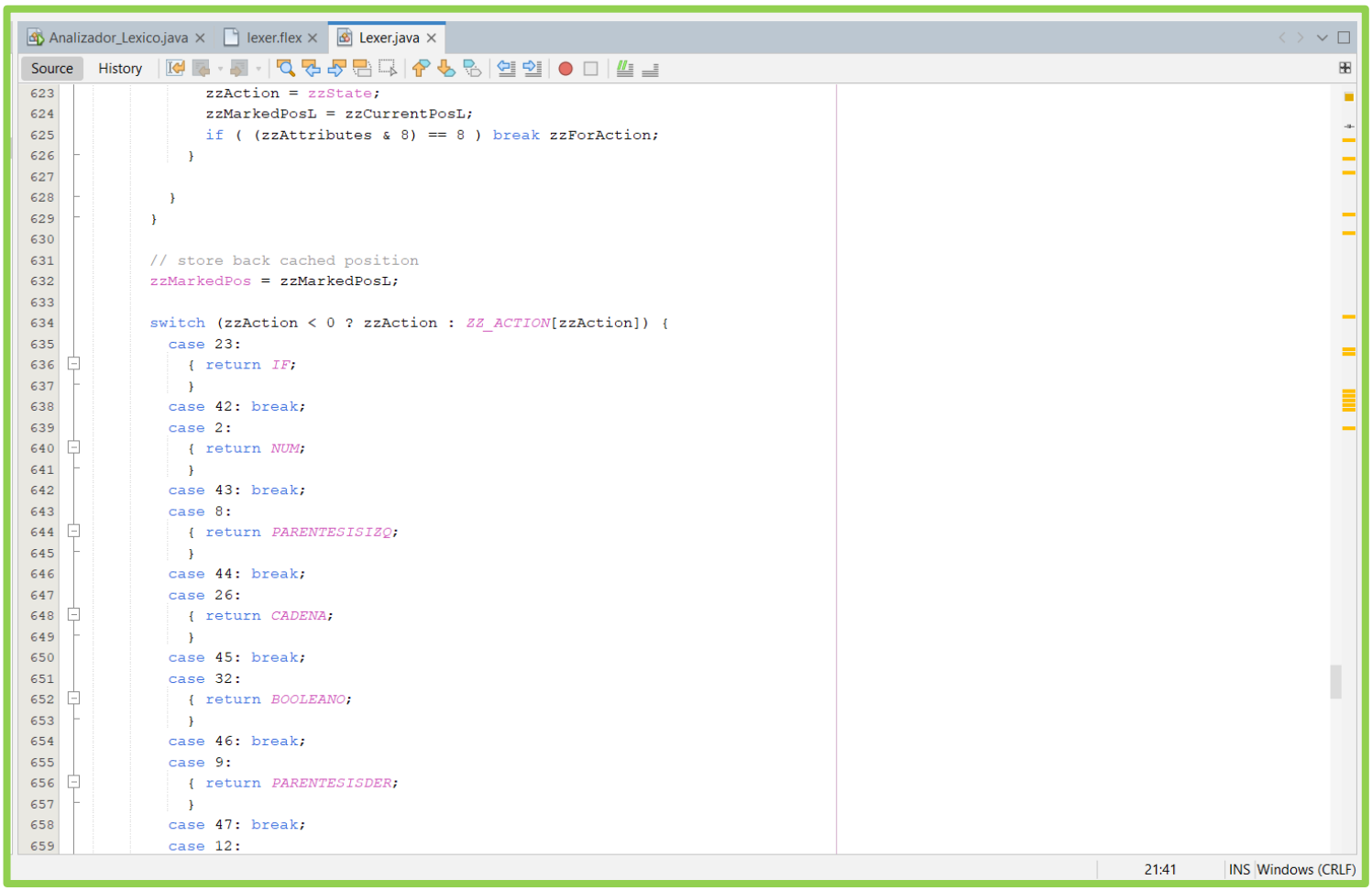


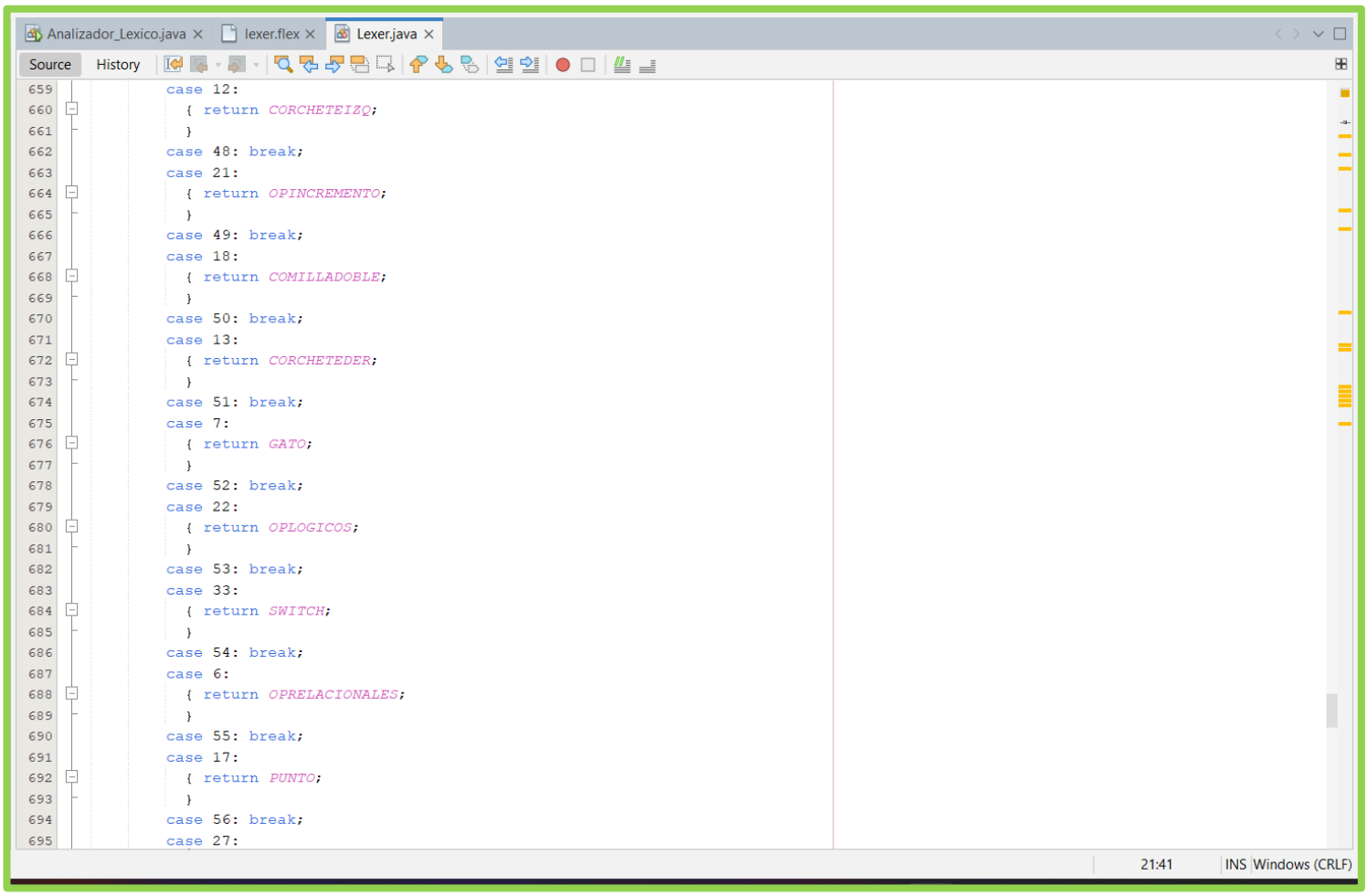


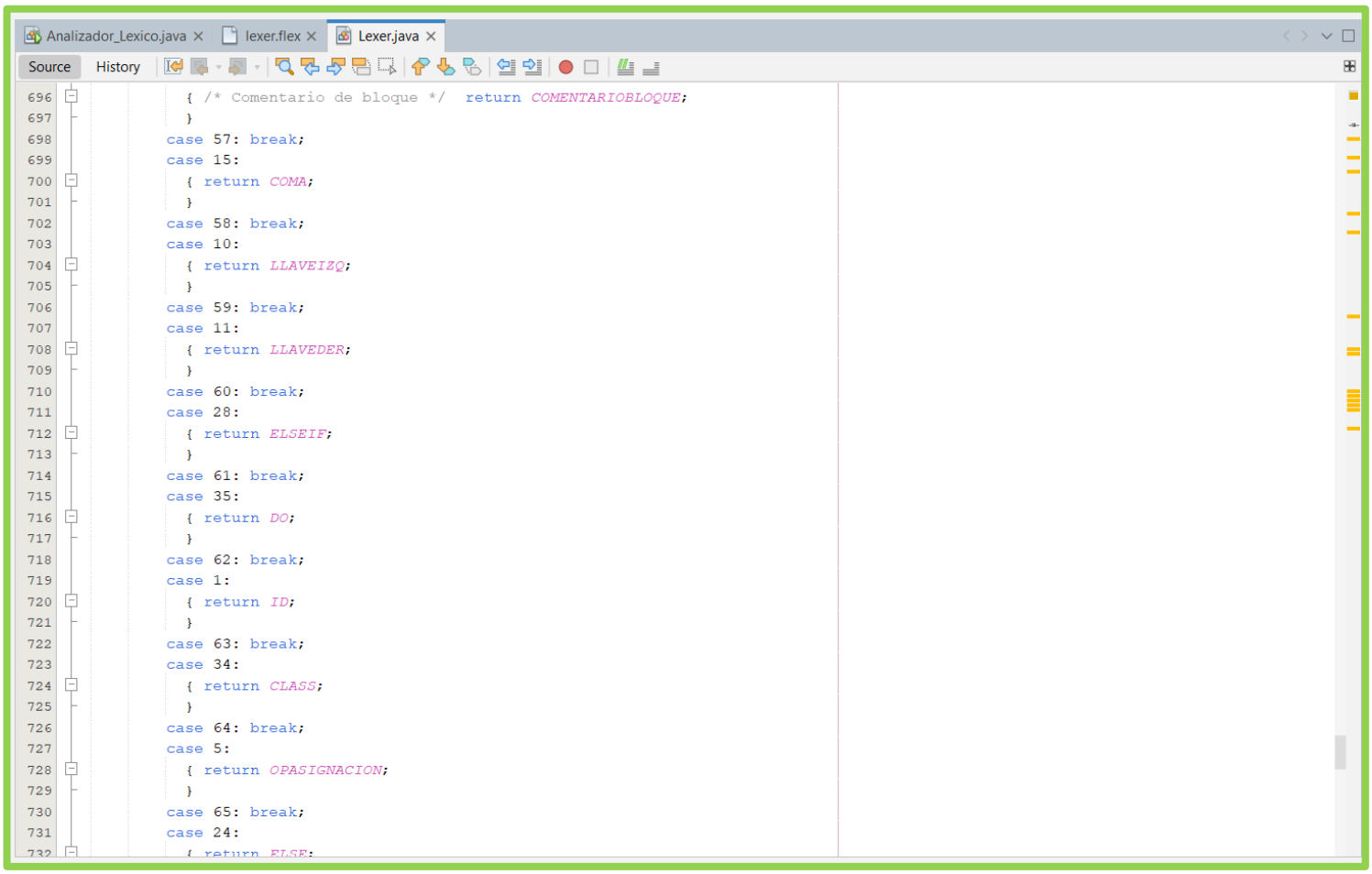


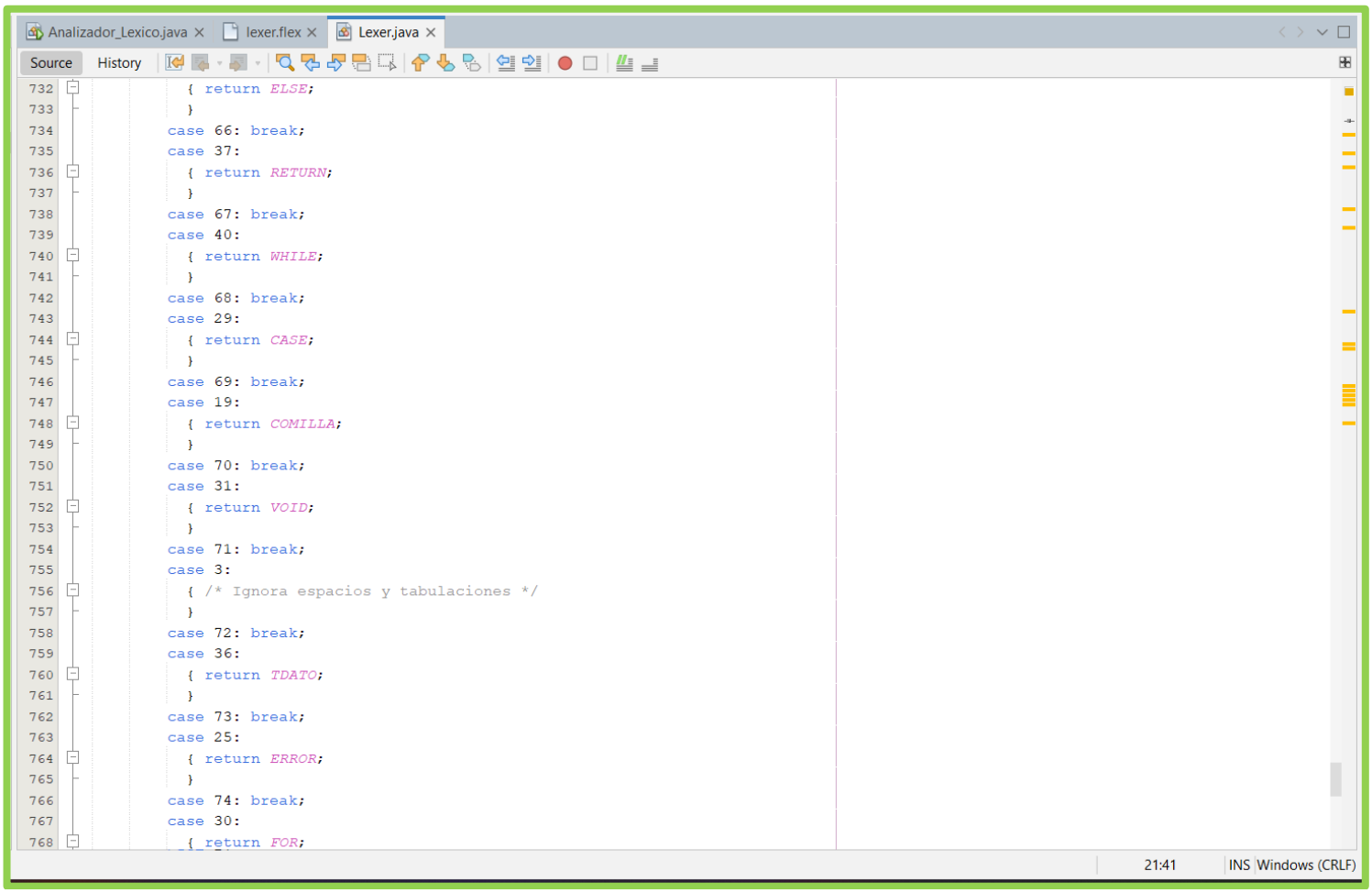


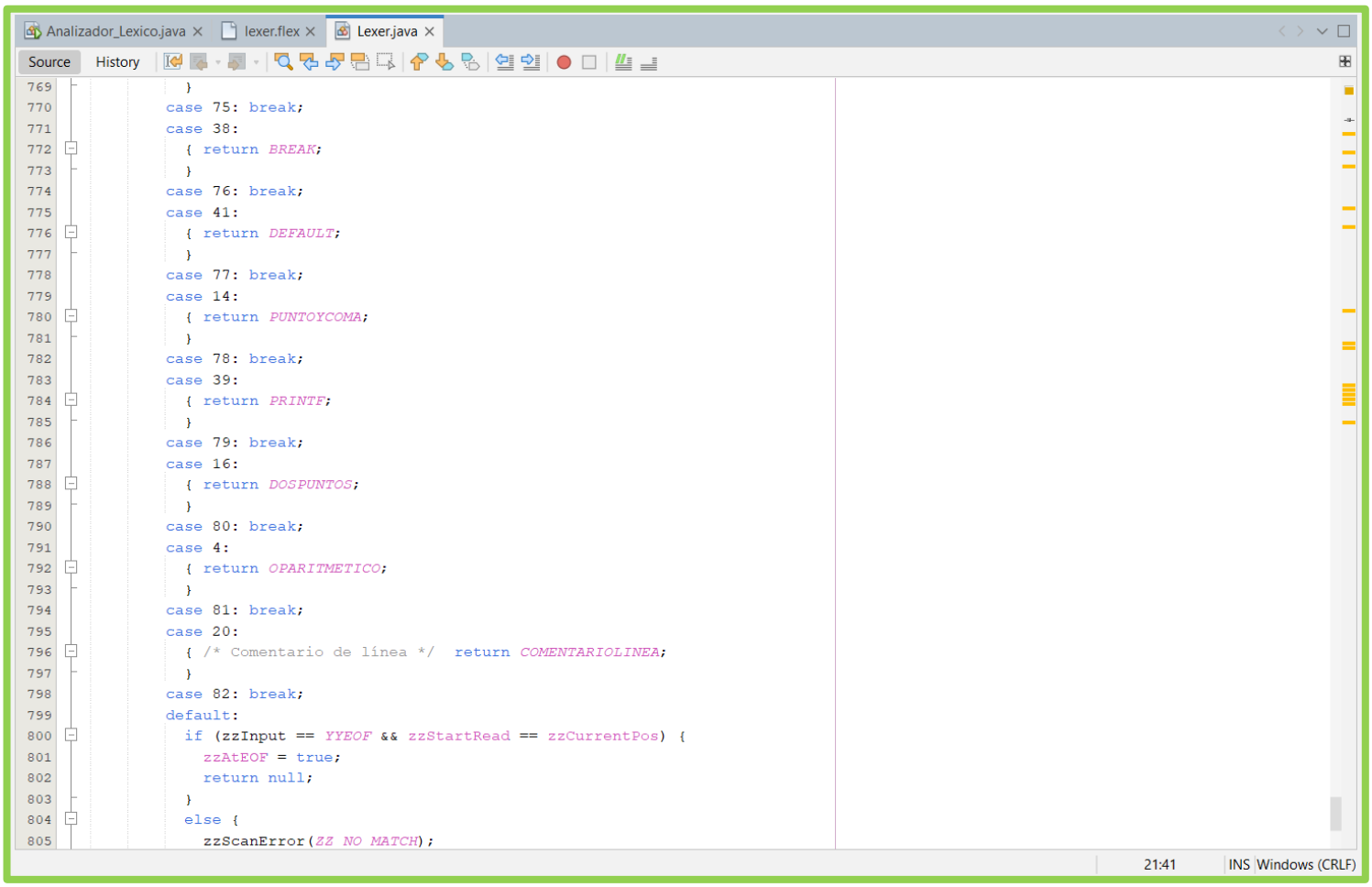


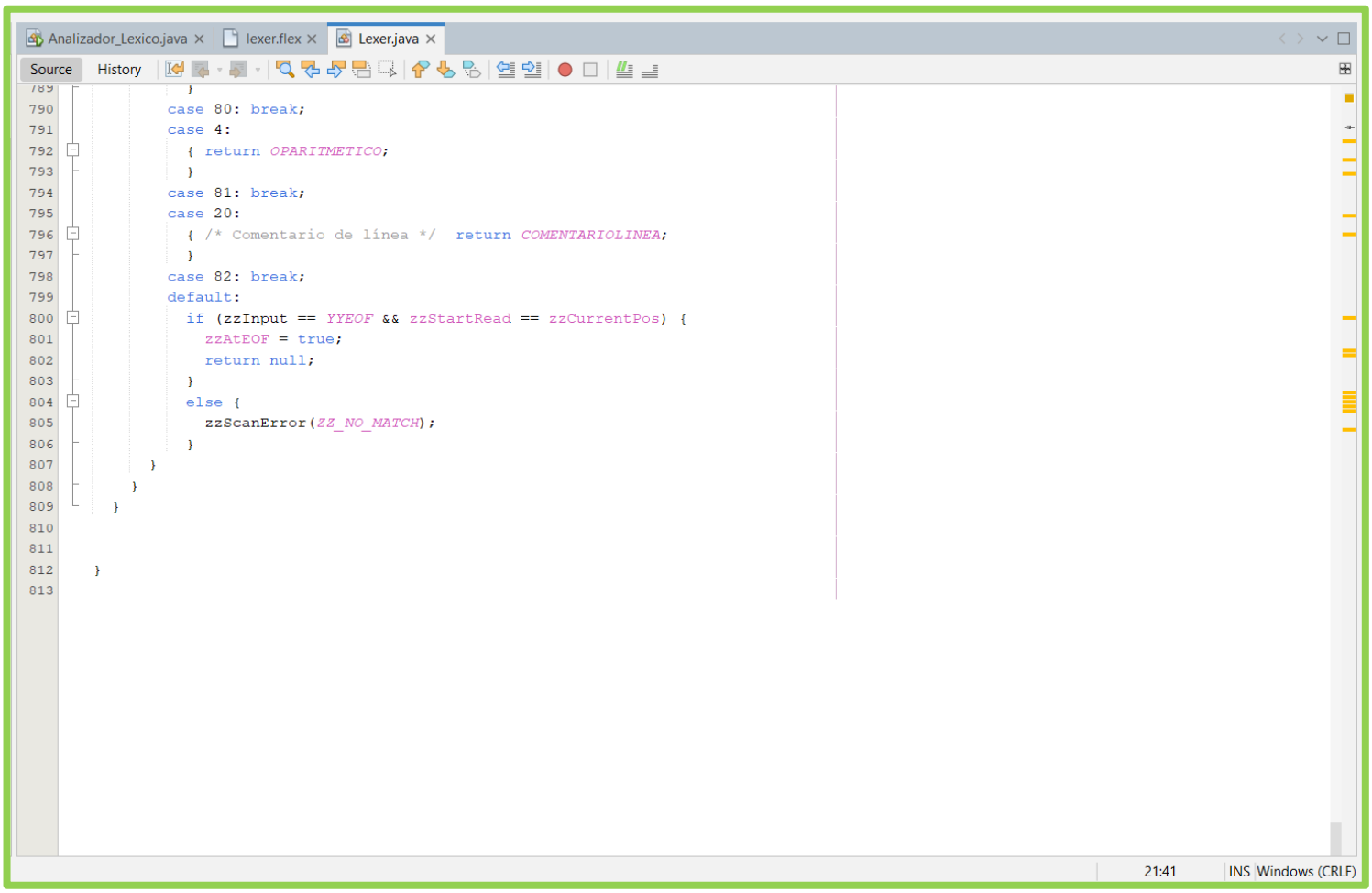






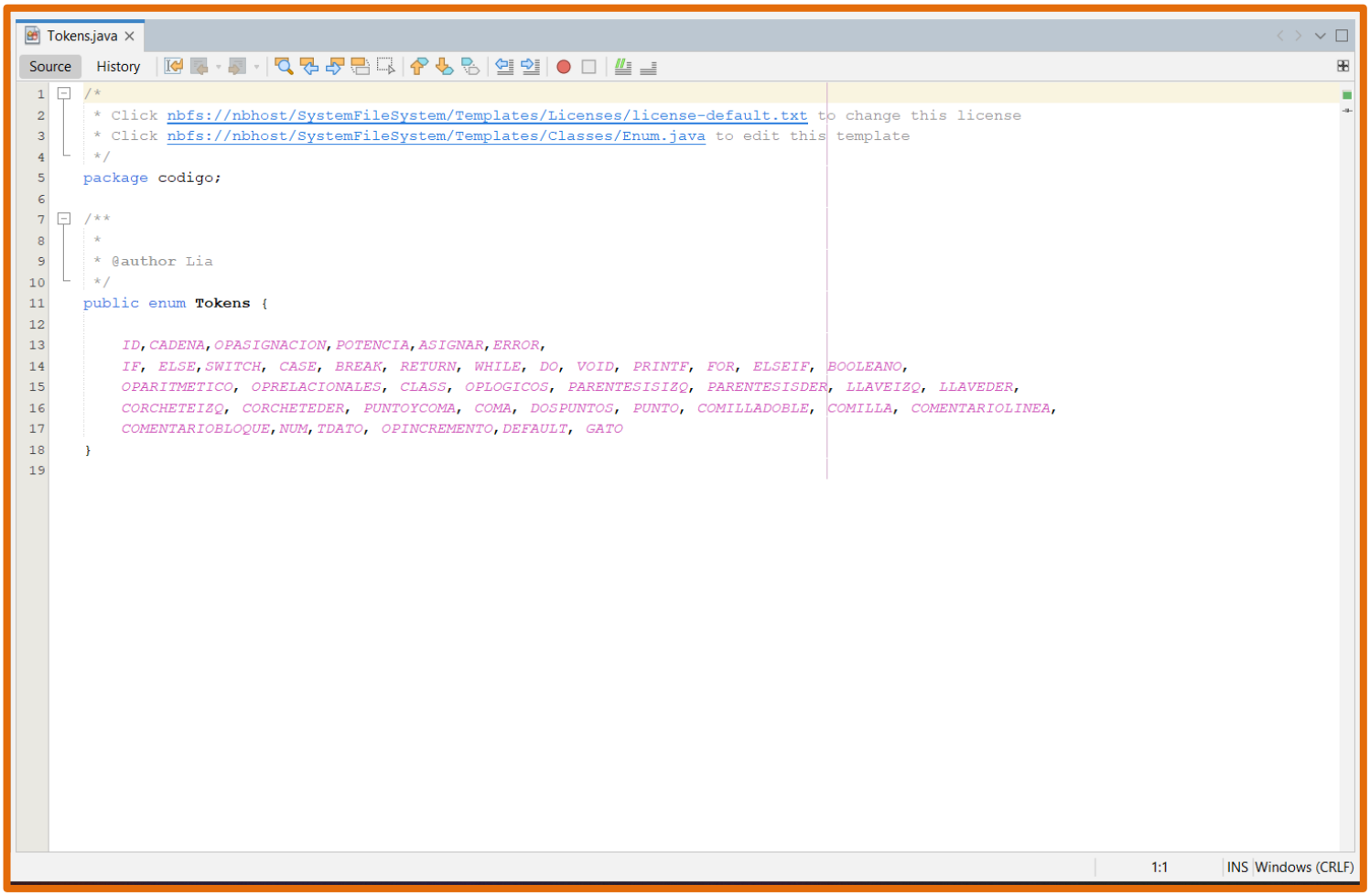

Analizador_Lexico.java
lexer.flex
Lexer.java
Source
History
{ return ELSE;
}
case 66: break;
case 37:
{ return RETURN;
}
case 67: break;
case 40:
{ return WHILE;
}
case 68: break;
case 29:
{ return CASE;
}
case 69: break;
case 19:
{ return COMILLA;
}
case 70: break;
case 31:
{ return VOID;
}
case 71: break;
case 3:
{ /* Ignora espacios y tabulaciones */
}
case 72: break;
case 36:
{ return TDATO;
}
case 73: break;
case 25:
{ return ERROR;
}
case 74: break;
case 30:
{ return FOR;
21:41
INS Windows (CRLF)





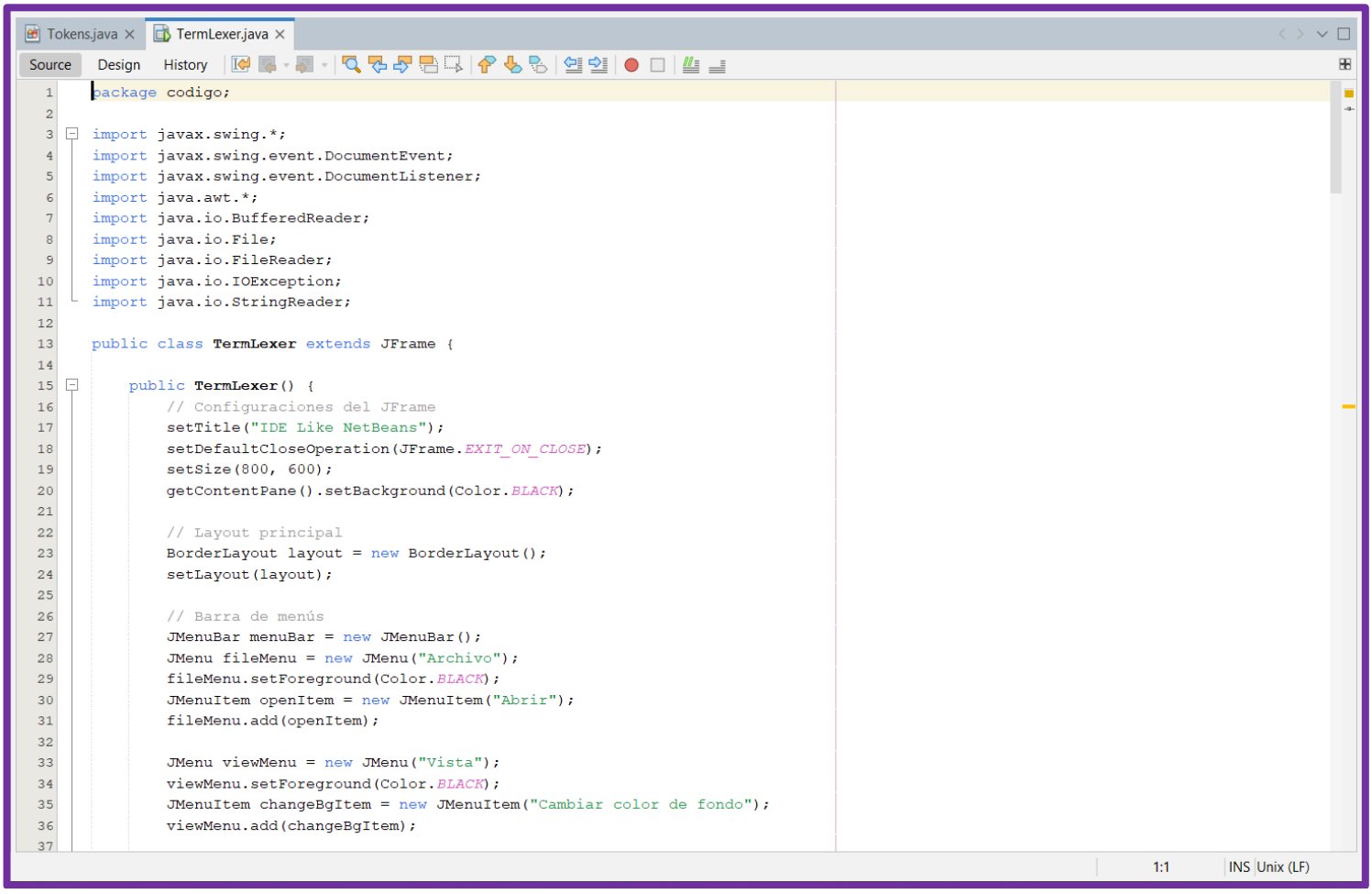
Tokens.java

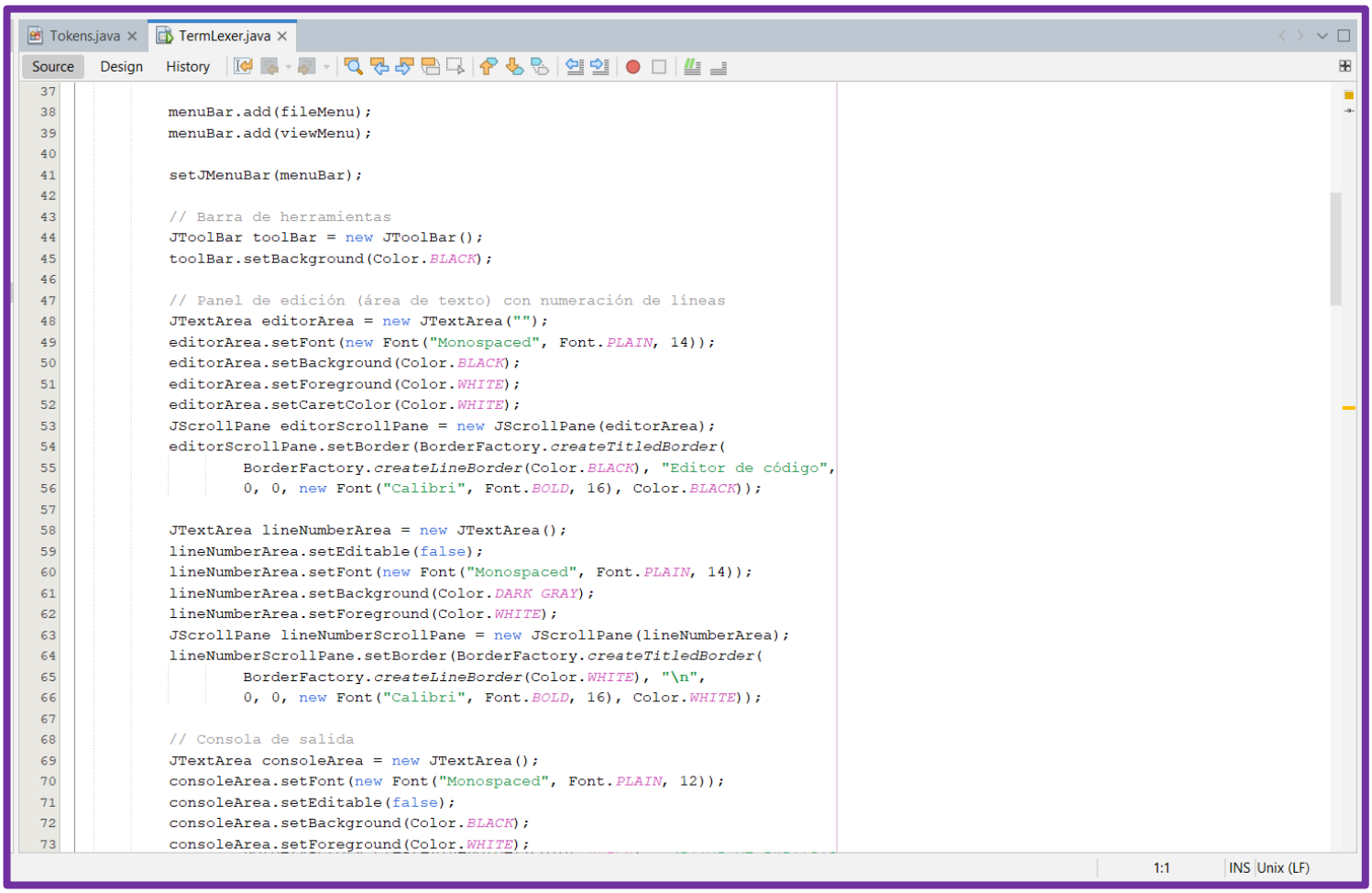
Este código define una enumeración llamada Tokens en Java, que lista los tipos de tokens reconocidos por un analizador léxico. Incluye identificadores (ID), cadenas (CADENA), operadores de asignación, relacionales, aritméticos y lógicos, palabras clave (como IF, WHILE, CLASS), símbolos especiales (paréntesis, 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.

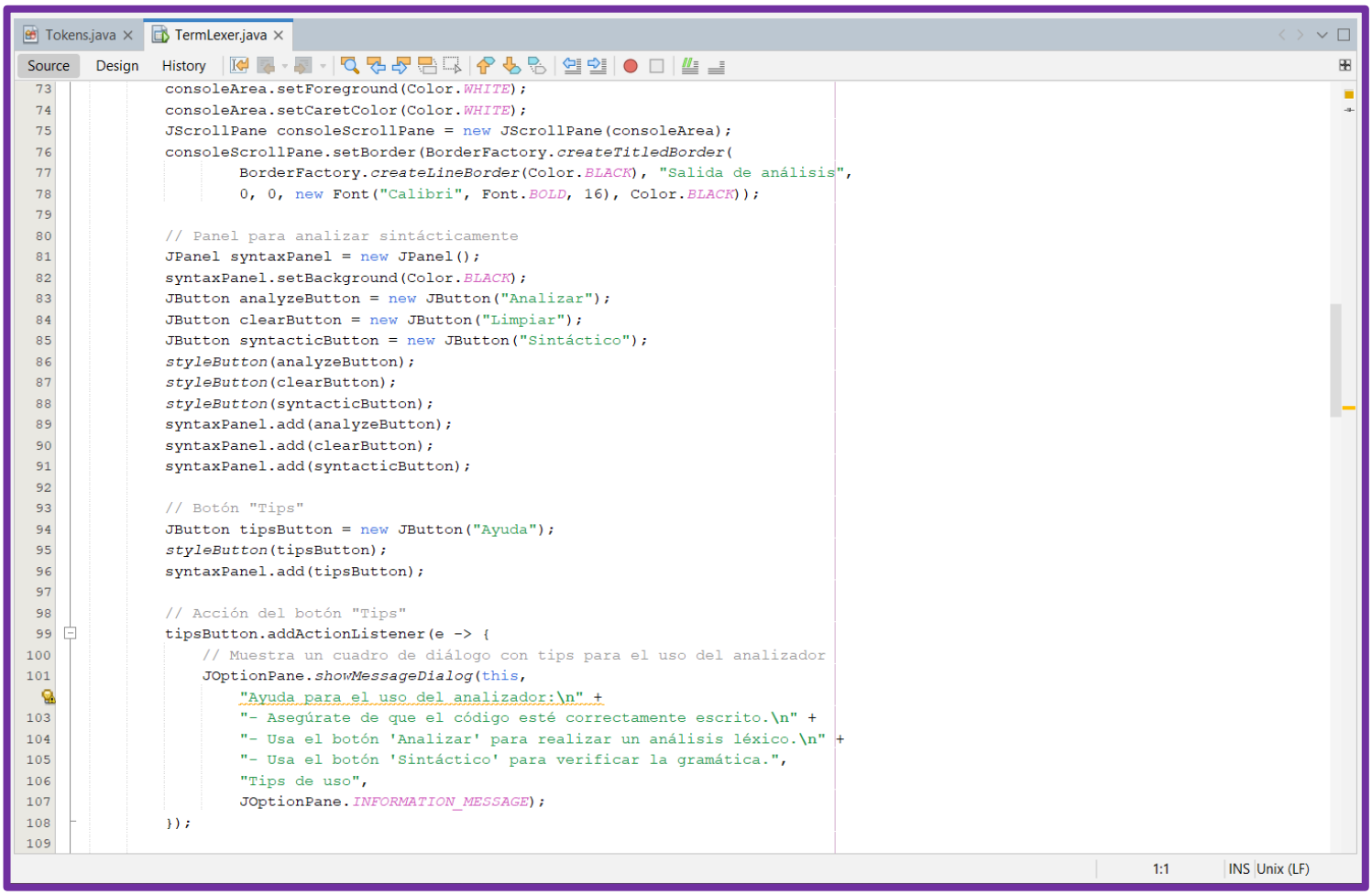


TermLexer.java

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.





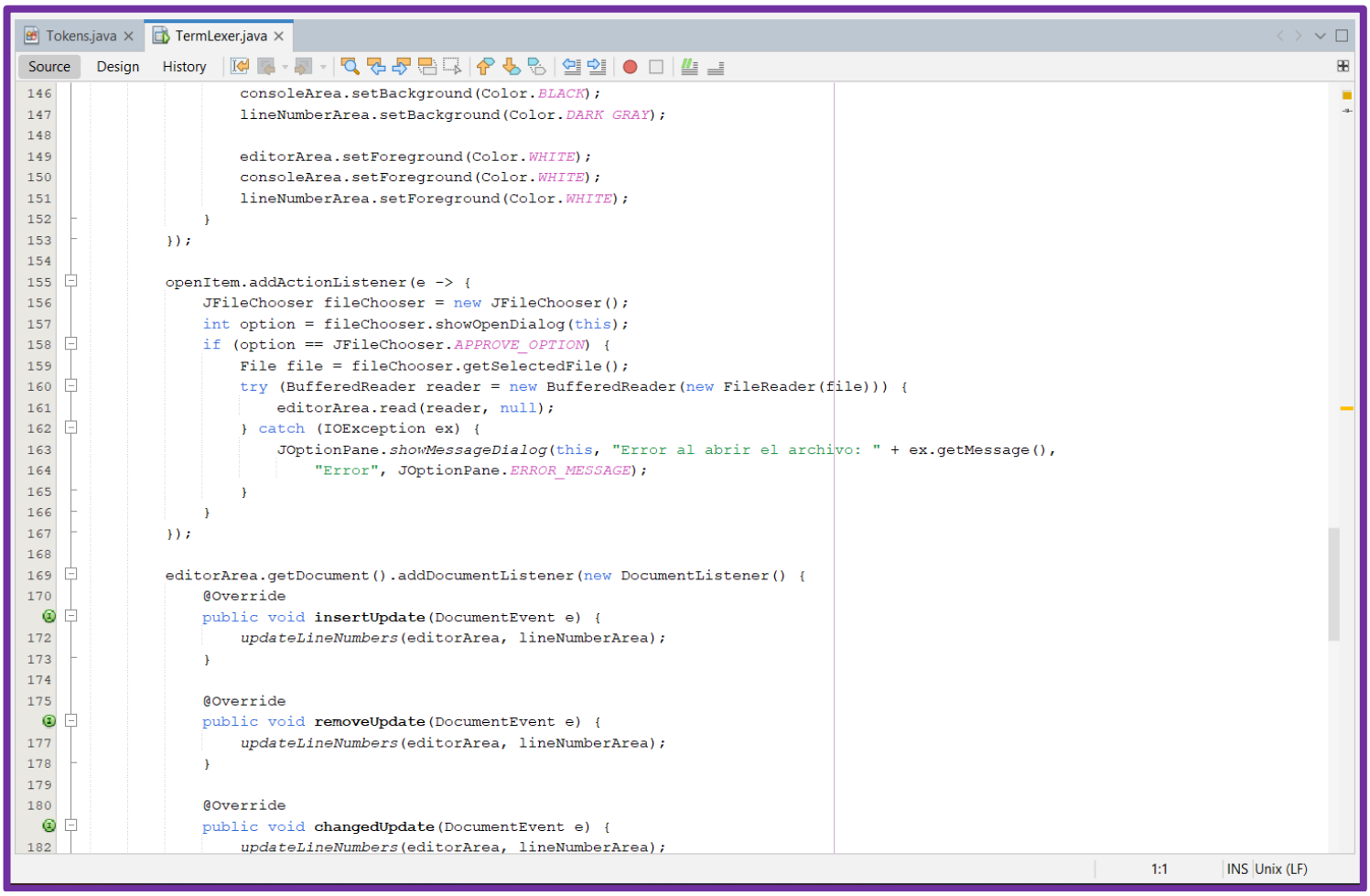


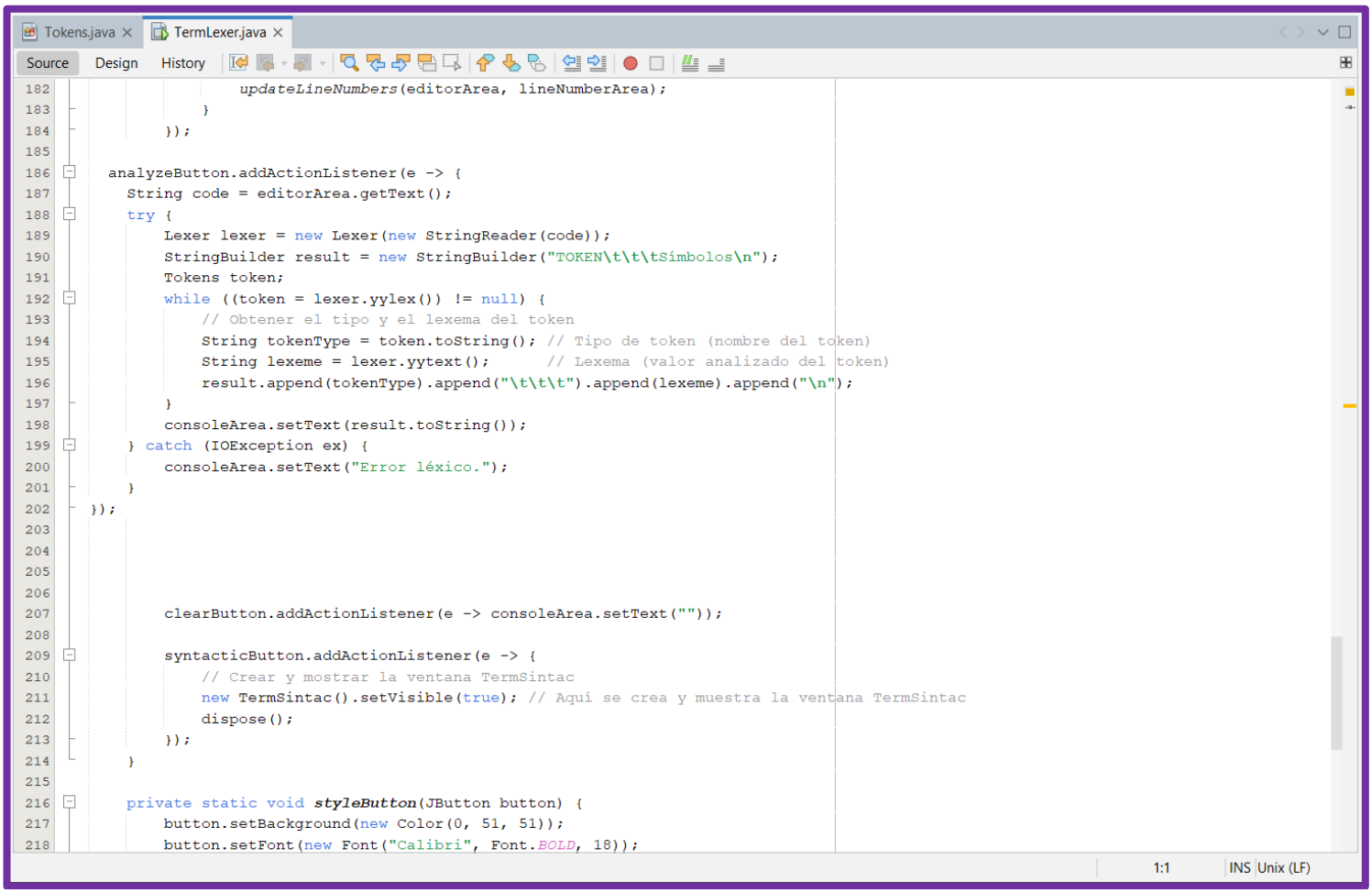
Tokens.java × TermLexer.java ×

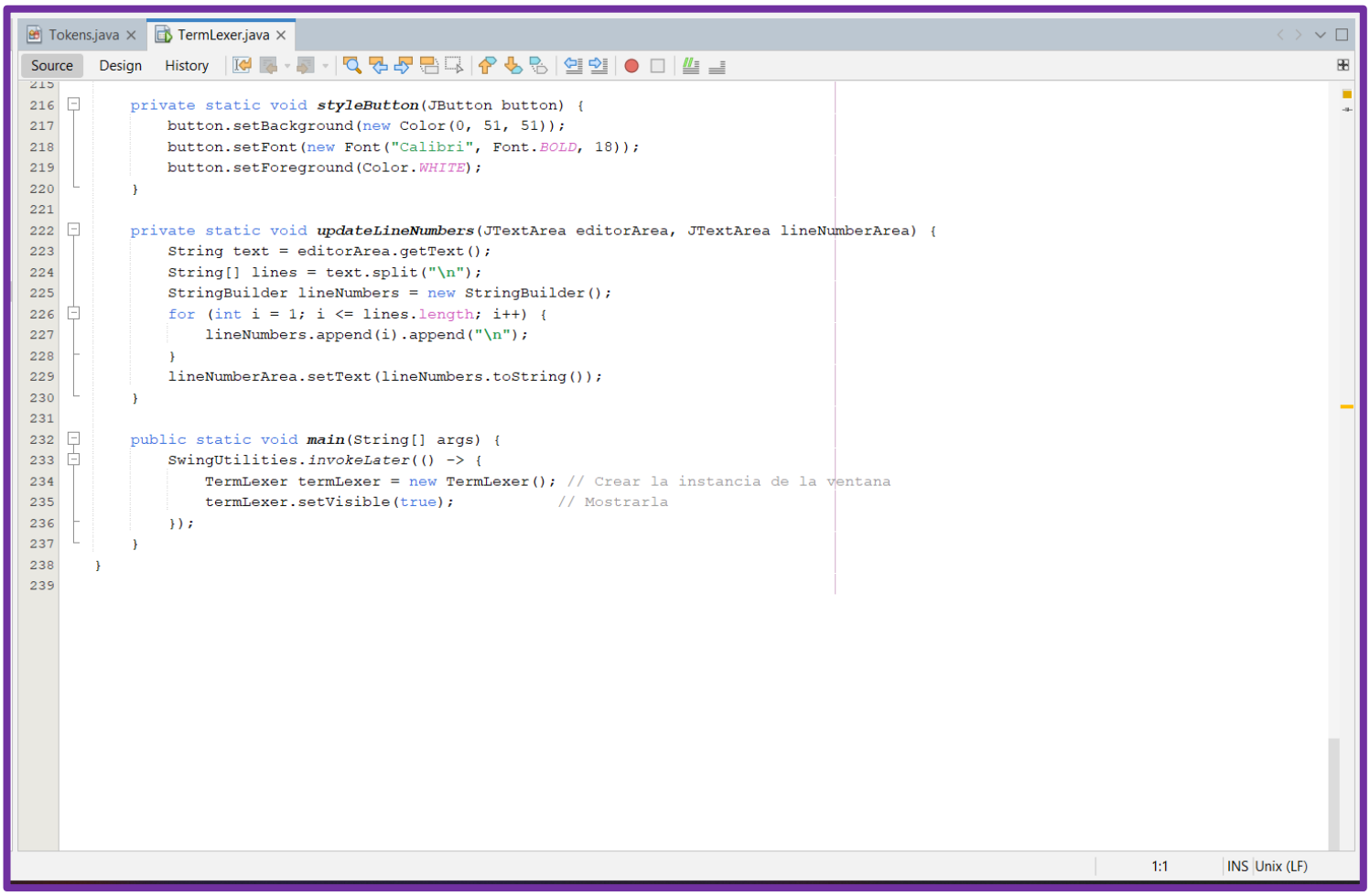
Source Design History

```
110
111         // Crear un panel central para centrar los elementos
112         JPanel centerPanel = new JPanel();
113         centerPanel.setLayout(new BorderLayout());
114
115         // Crear JSplitPane para dividir el editor y la consola
116         JSplitPane splitPane = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT, editorScrollPane, consoleScrollPane);
117         splitPane.setDividerLocation(0.47);  // 80% del espacio para el editor, 20% para la consola
118         splitPane.setResizeWeight(0.47); // El editor toma el 80% del espacio
119
120         // Crear otro JSplitPane que divide la numeración de líneas y el panel con el editor y consola
121         JSplitPane editorSplitPane = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT, lineNumberScrollPane, splitPane);
122         editorSplitPane.setDividerLocation(40);  // Ajustamos para que el lineNumberScrollPane ocupe un 10% del ancho
123         editorSplitPane.setResizeWeight(0.1); // Dejar que el editor se redimensione, pero mantener fija la numeración de líneas
124
125         // Añadir todo al panel central
126         centerPanel.add(editorSplitPane, BorderLayout.CENTER);
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);
```

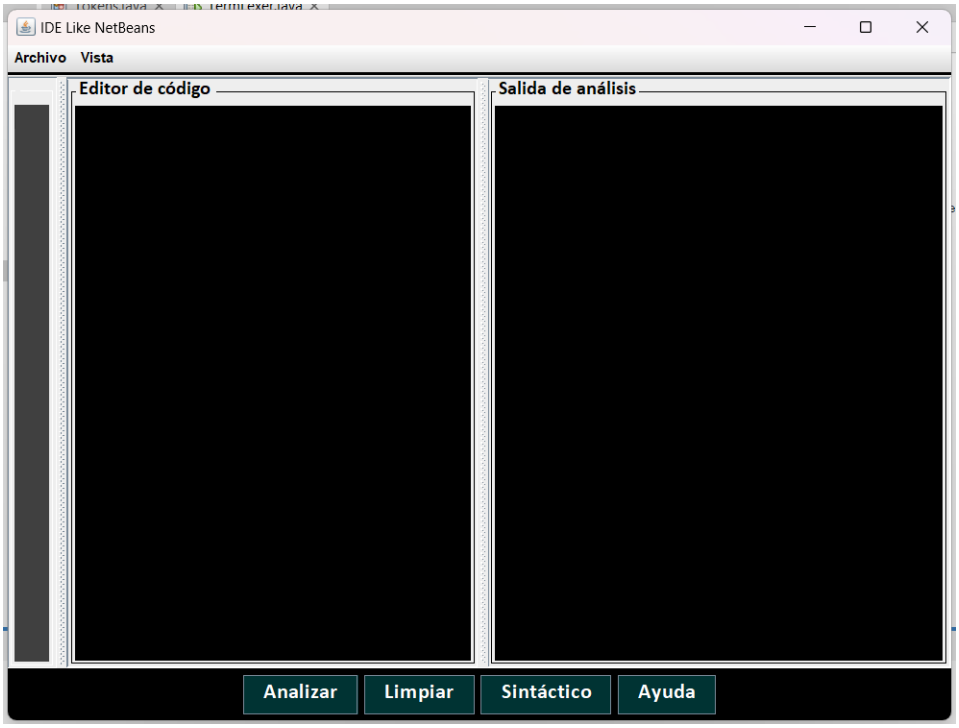
1:1 INS Unix (LF)

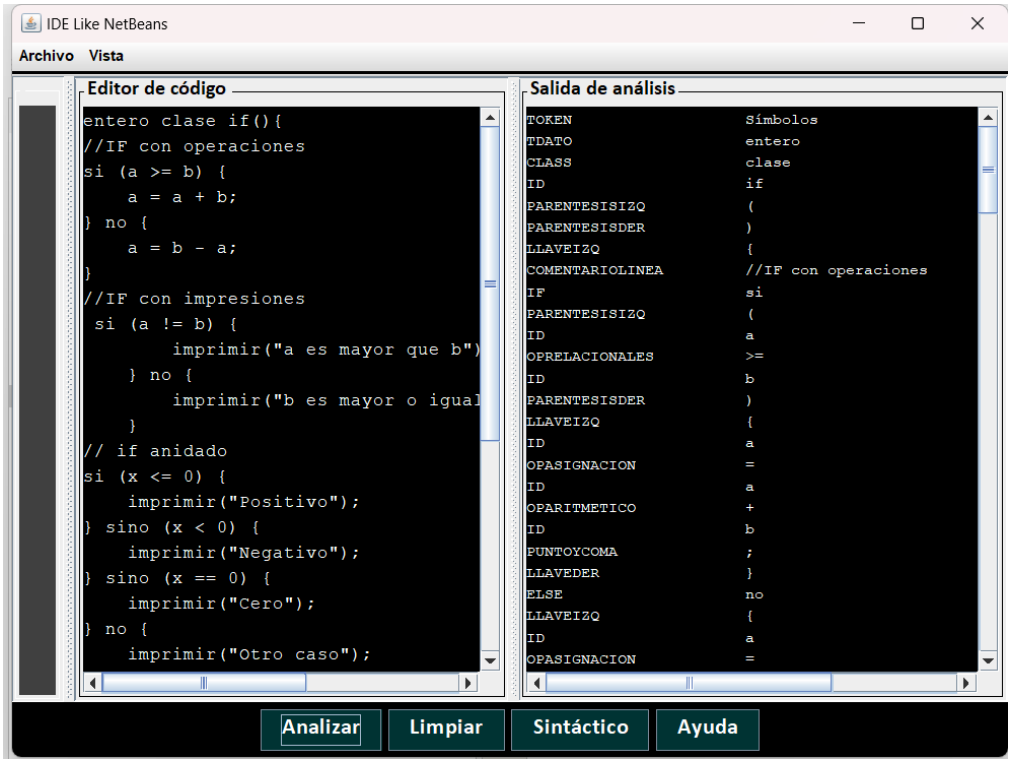

Tokens.java
TermLexer.java
Source
Design
History
consoleArea.setBackground(Color.BLACK);
lineNumberArea.setBackground(Color.DARK GRAY);
editorArea.setForeground(Color.WHITE);
consoleArea.setForeground(Color.WHITE);
lineNumberArea.setForeground(Color.WHITE);
}
});
openItem.addActionListener(e -> {
JFileChooser fileChooser = new JFileChooser();
int option = fileChooser.showOpenDialog(this);
if (option == JFileChooser.APPROVE_OPTION) {
File file = fileChooser.getSelectedFile();
try (BufferedReader reader = new BufferedReader(new FileReader(file))) {
editorArea.read(reader, null);
} catch (IOException ex) {
JOptionPane.showMessageDialog(this, "Error al abrir el archivo: " + ex.getMessage(),
"Error", JOptionPane.ERROR_MESSAGE);
}
}
});
editorArea.getDocument().addDocumentListener(new DocumentListener() {
@Override
public void insertUpdate(DocumentEvent e) {
updateLineNumbers(editorArea, lineNumberArea);
}
@Override
public void removeUpdate(DocumentEvent e) {
updateLineNumbers(editorArea, lineNumberArea);
}
@Override
public void changedUpdate(DocumentEvent e) {
updateLineNumbers(editorArea, lineNumberArea);
1:1
INS Unix (LF)


Tokens.java
TermLexer.java
Source
Design
History
182 updateLineNumbers(editorArea, lineNumberArea);
183 }
184 });
185
186 analyzeButton.addActionListener(e -> {
187 String code = editorArea.getText();
188 try {
189 Lexer lexer = new Lexer(new StringReader(code));
190 StringBuilder result = new StringBuilder("TOKEN\t\t\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));
1:1
INS Unix (LF)


Tokens.java
TermLexer.java
Source
Design
History
private static void styleButton(JButton button) {
button.setBackground(new Color(0, 51, 51));
button.setFont(new Font("Calibri", Font.BOLD, 18));
button.setForeground(Color.WHITE);
}
private static void updateLineNumbers(JTextArea editorArea, JTextArea lineNumberArea) {
String text = editorArea.getText();
String[] lines = text.split("\n");
StringBuilder lineNumbers = new StringBuilder();
for (int i = 1; i <= lines.length; i++) {
lineNumbers.append(i).append("\n");
}
lineNumberArea.setText(lineNumbers.toString());
}
public static void main(String[] args) {
SwingUtilities.invokeLater(() -> {
TermLexer termLexer = new TermLexer(); // Crear la instancia de la ventana
termLexer.setVisible(true); // Mostrarla
});
}
}
1:1
INS
Unix (LF)

PANTALLAS RESULTANTES CON PRUEBAS DEL RECONOCIMIENTO DE LOS TOKENS





ANALIZADOR SINTACTICO

…..

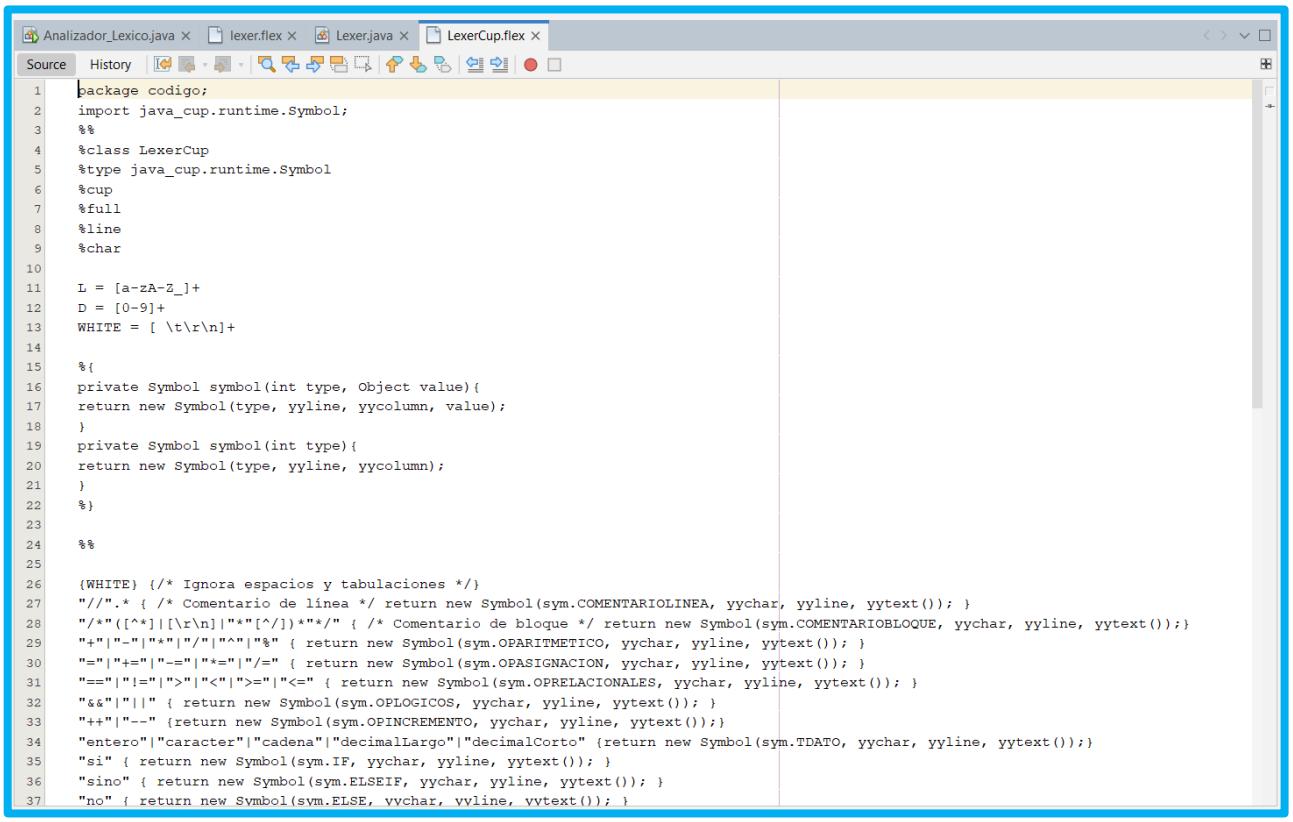
DESCRIPCION DEL PROBLEMA

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

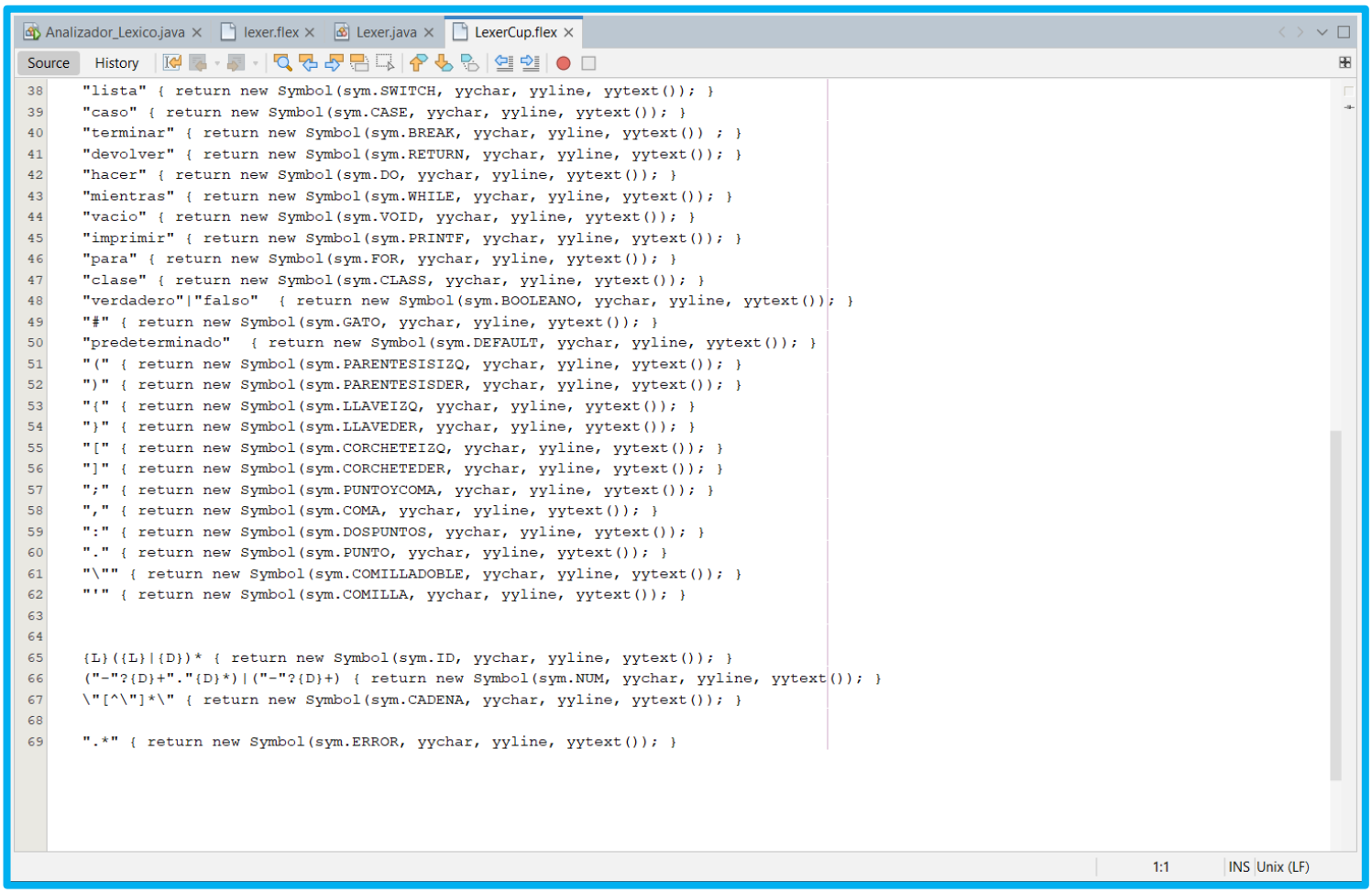
DESCRIPCION DE CADA UNO DE LOS ARCHIVOS GENERADOS EN EL PROYECTO – (1 A 2 PARRAFOS) CONTINUAR COLOCANDO EL CODIGO, COLOCAR UNA LINEA DE SEPARACION ENTRE CADA ARCHIVO A EXPLICAR
(NOTA: COLOCAR BORDES AL CODIGO, Y SE DEBE OBSERVAR LAS LINEAS DE CODIGO, TODO EL CODIGO Y LETRA VISIBLE)

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

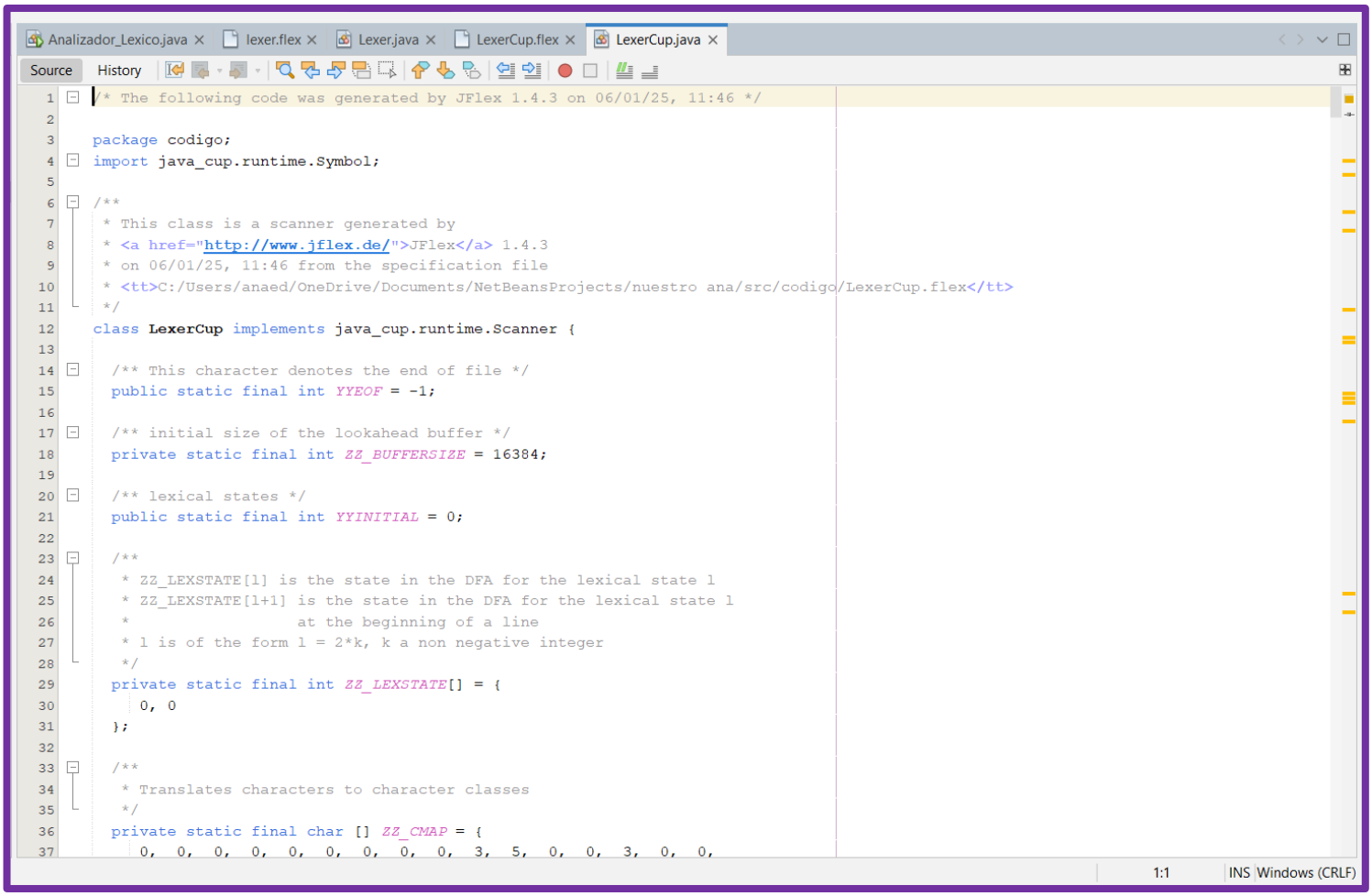



```
package codigo;
import java_cup.runtime.Symbol;
%%
%class LexerCup
%type java_cup.runtime.Symbol
%cup
%full
%line
%char

L = [a-zA-Z_]+
D = [0-9]+
WHITE = [ \t\r\n]+

%{
private Symbol symbol(int type, Object value){
return new Symbol(type, yyline, yycolumn, value);
}
private Symbol symbol(int type){
return new Symbol(type, yyline, yycolumn);
}
%}

%%

{WHITE} {/* Ignora espacios y tabulaciones */}
"//".* { /* Comentario de línea */ return new Symbol(sym.COMENTARIOLINEA, yychar, yyline, yytext()); }
"/*"([^*]|[\r\n]|"*"[^/])*"*/" { /* Comentario de bloque */ return new Symbol(sym.COMENTARIOBLOQUE, yychar, yyline, yytext());}
"+"|"-"|"*"|"/"|"^"|"%" { return new Symbol(sym.OPARITMETICO, yychar, yyline, yytext()); }
"="|"+="|"-="|"*="|"/=" { return new Symbol(sym.OPASIGNACION, yychar, yyline, yytext()); }
"=="|"!="|">"|"<"|">="|"<=" { return new Symbol(sym.OPRELACIONALES, yychar, yyline, yytext()); }
"&&"|"||" { return new Symbol(sym.OPLOGICOS, yychar, yyline, yytext()); }
"++"|"--" {return new Symbol(sym.OPINCREMENTO, yychar, yyline, yytext());}
"entero"|"caracter"|"cadena"|"decimalLargo"|"decimalCorto" {return new Symbol(sym.TDATO, yychar, yyline, yytext());}
"si" { return new Symbol(sym.IF, yychar, yyline, yytext()); }
"sino" { return new Symbol(sym.ELSEIF, yychar, yyline, yytext()); }
"no" { return new Symbol(sym.ELSE, 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.

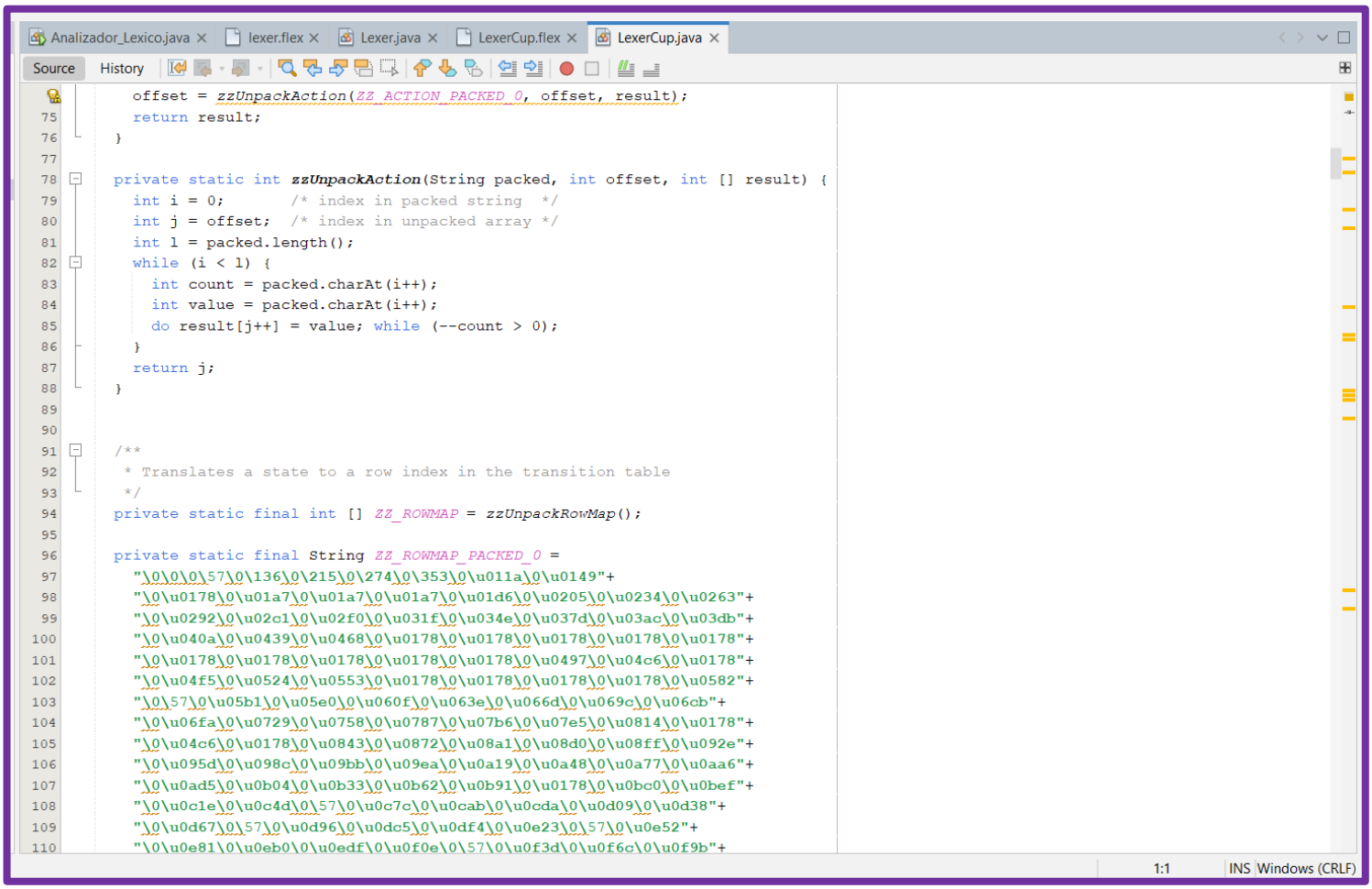


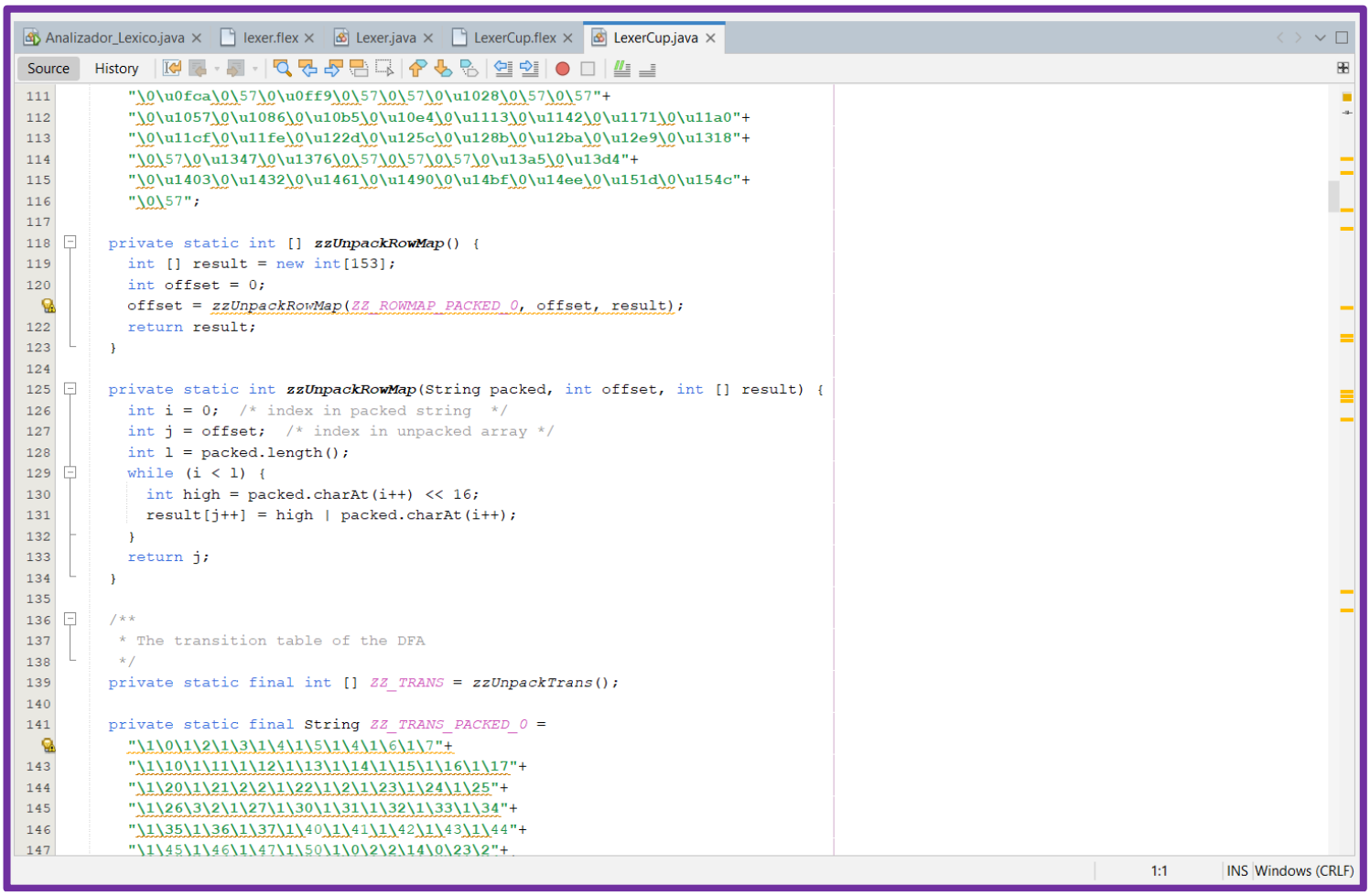
Analizador_Lexico.java | lexer.flex | Lexer.java | LexerCup.flex | LexerCup.java

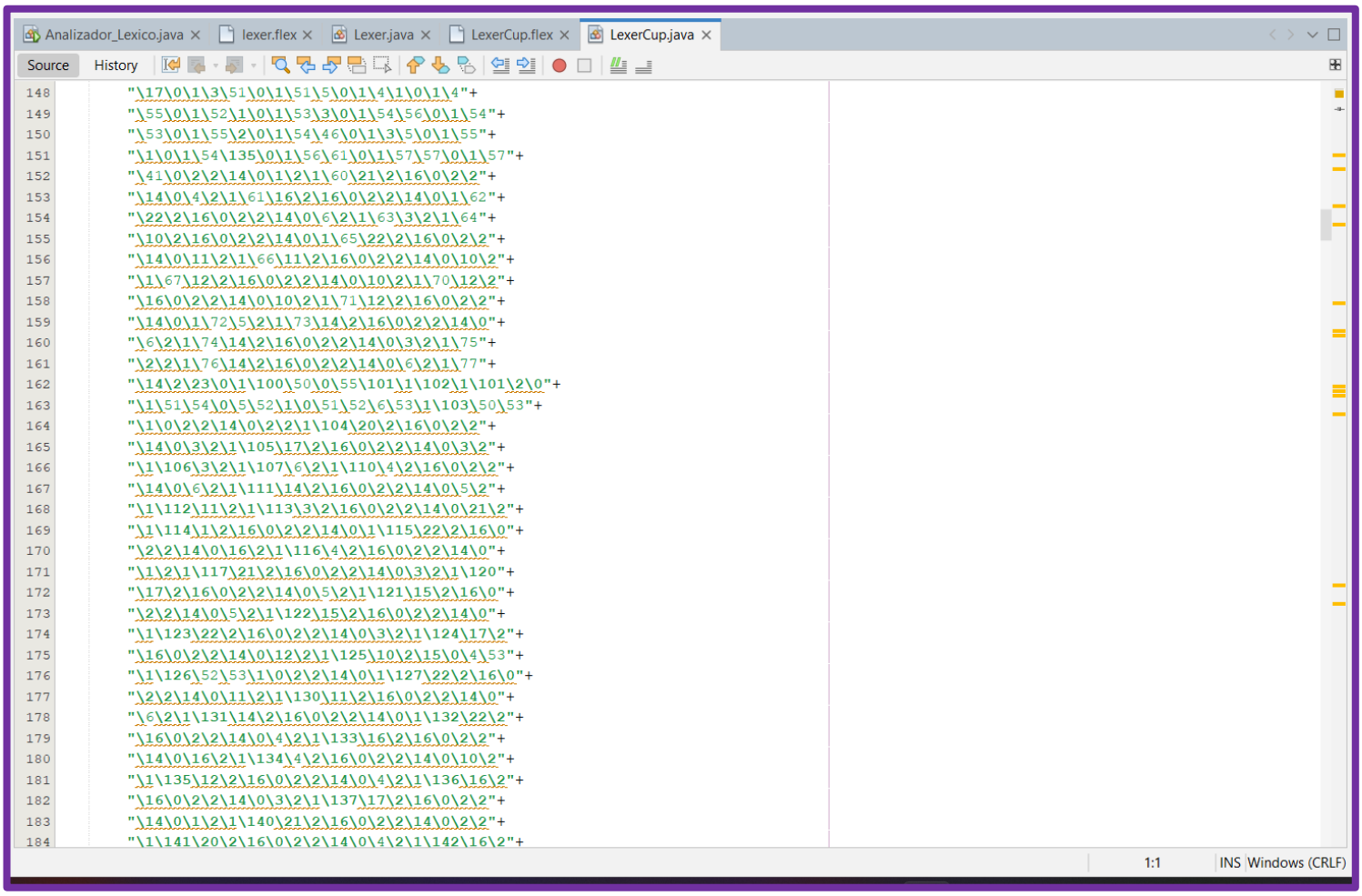
Source | History

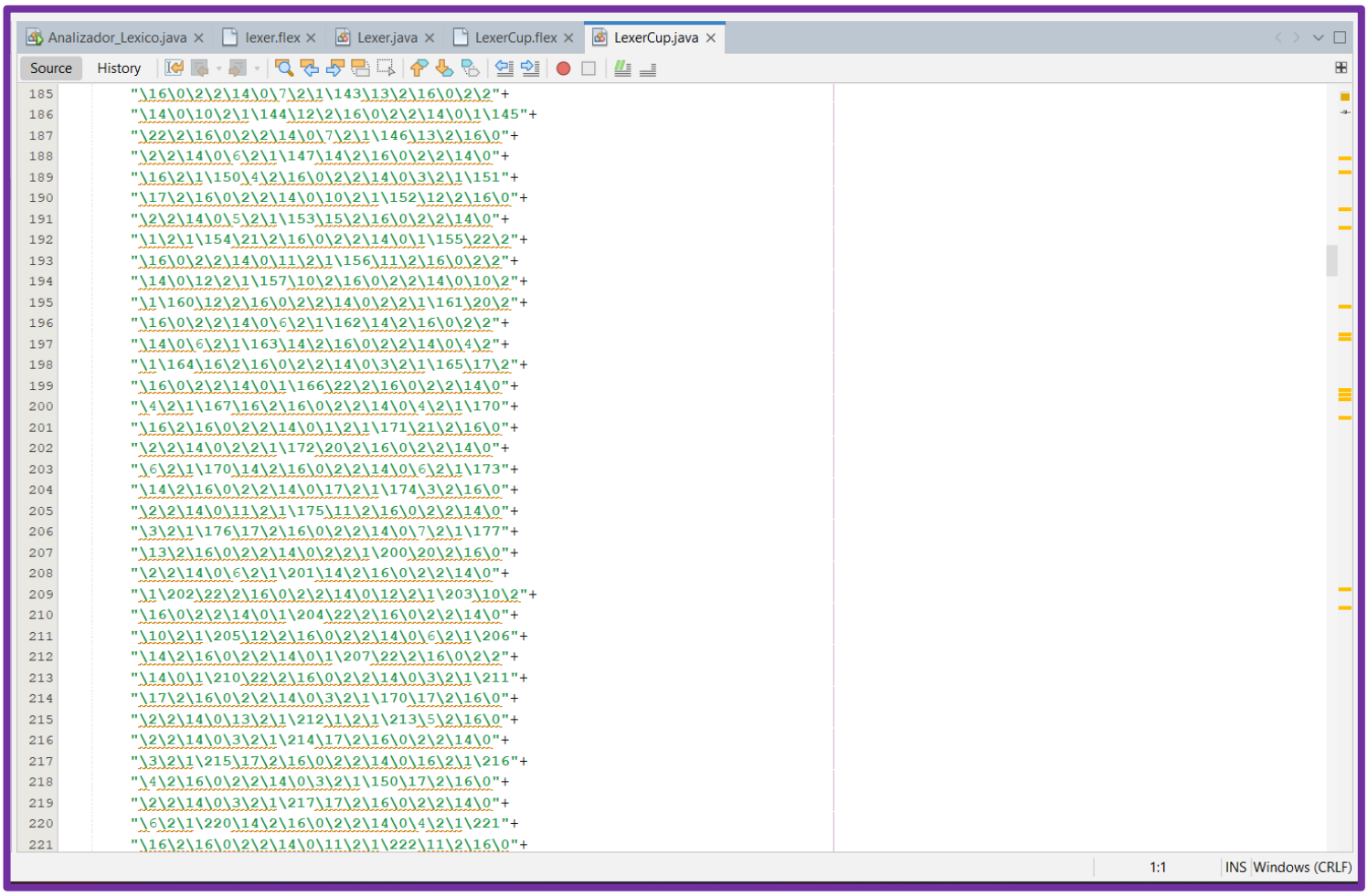
```java
     0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
     3, 11, 45, 34,  0,  9, 13, 46, 35, 36,  6,  7, 42,  8, 44,  4,
     2,  2,  2,  2,  2,  2,  2,  2,  2,  2, 43, 41, 12, 10, 12,  0,
     0,  1,  1, 28,  1,  1,  1,  1,  1,  1,  1,  1, 26,  1,  1,  1,
     1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1, 39,  0, 40,  9,  1,
     0, 21,  1, 20, 22, 15, 33, 27, 31, 23,  1,  1, 25, 24, 16, 19,
    32,  1, 18, 29, 17,  1, 30,  1,  1,  1,  1, 37, 14, 38,  0,  0,
     0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
     0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
     0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
     0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
     0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
     0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
     0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
     0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0
  };

  /**
   * Translates DFA states to action switch labels.
   */
  private static final int [] ZZ_ACTION = zzUnpackAction();

  private static final String ZZ_ACTION_PACKED_0 =
    "\1\0\1\1\1\2\1\3\5\4\1\5\1\0\1\6"+
    "\2\0\15\1\1\7\1\10\1\11\1\12\1\13\1\14"+
    "\1\15\1\16\1\17\1\20\1\21\1\22\1\23\1\2"+
    "\1\24\1\0\1\5\1\25\1\6\1\26\1\1\1\27"+
    "\7\1\1\30\6\1\1\31\1\0\1\32\1\0\22\1"+
    "\1\33\4\1\1\34\6\1\1\35\4\1\1\36\5\1"+
    "\1\37\4\1\1\40\1\1\1\41\1\42\1\1\1\43"+
    "\1\44\20\1\1\45\2\1\1\46\1\47\1\50\12\1"+
    "\1\51";

  private static int [] zzUnpackAction() {
    int [] result = new int[153];
    int offset = 0;
    offset = zzUnpackAction(ZZ_ACTION_PACKED_0, offset, result);
```

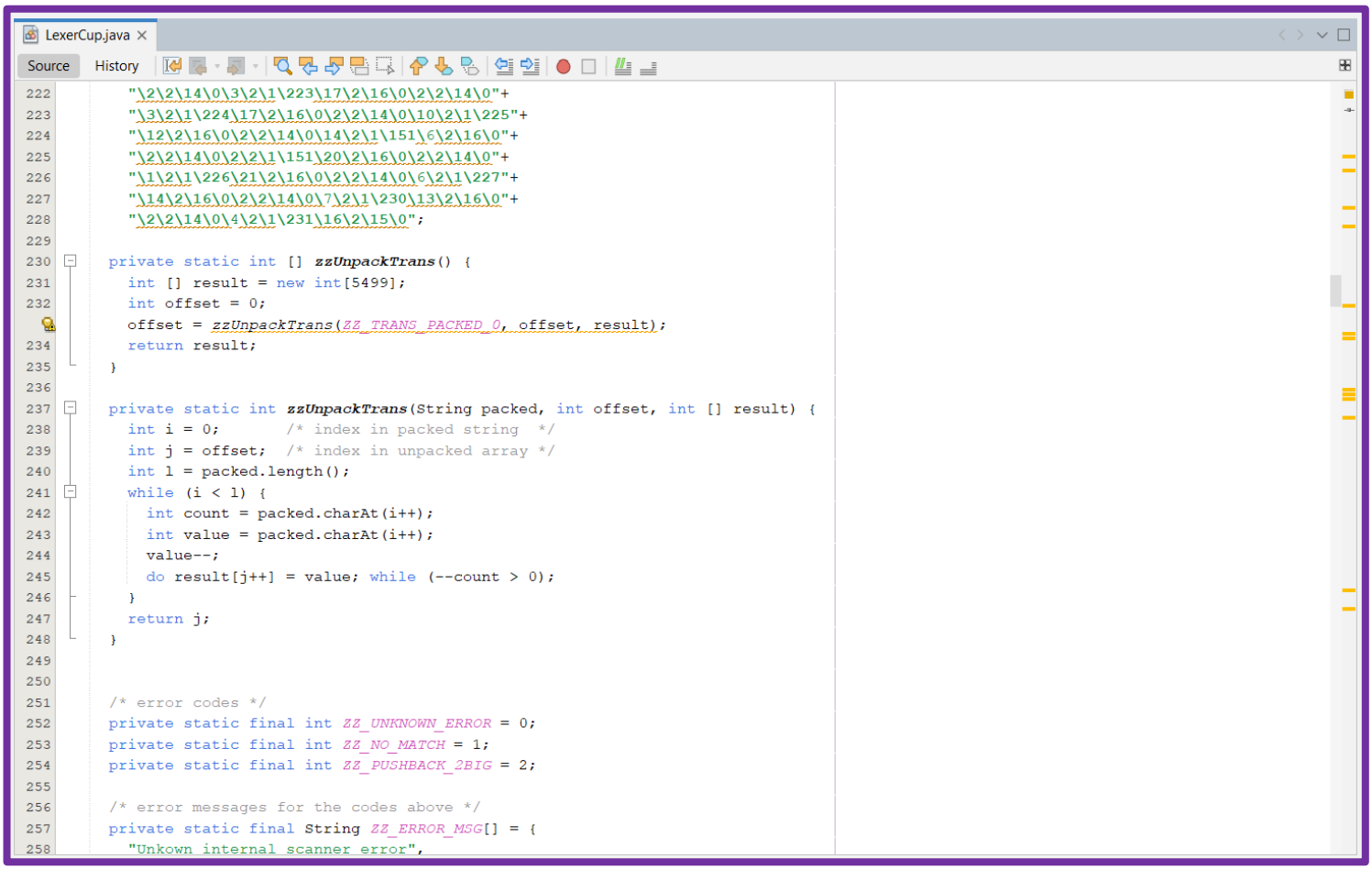
1:1 INS Windows (CRLF)

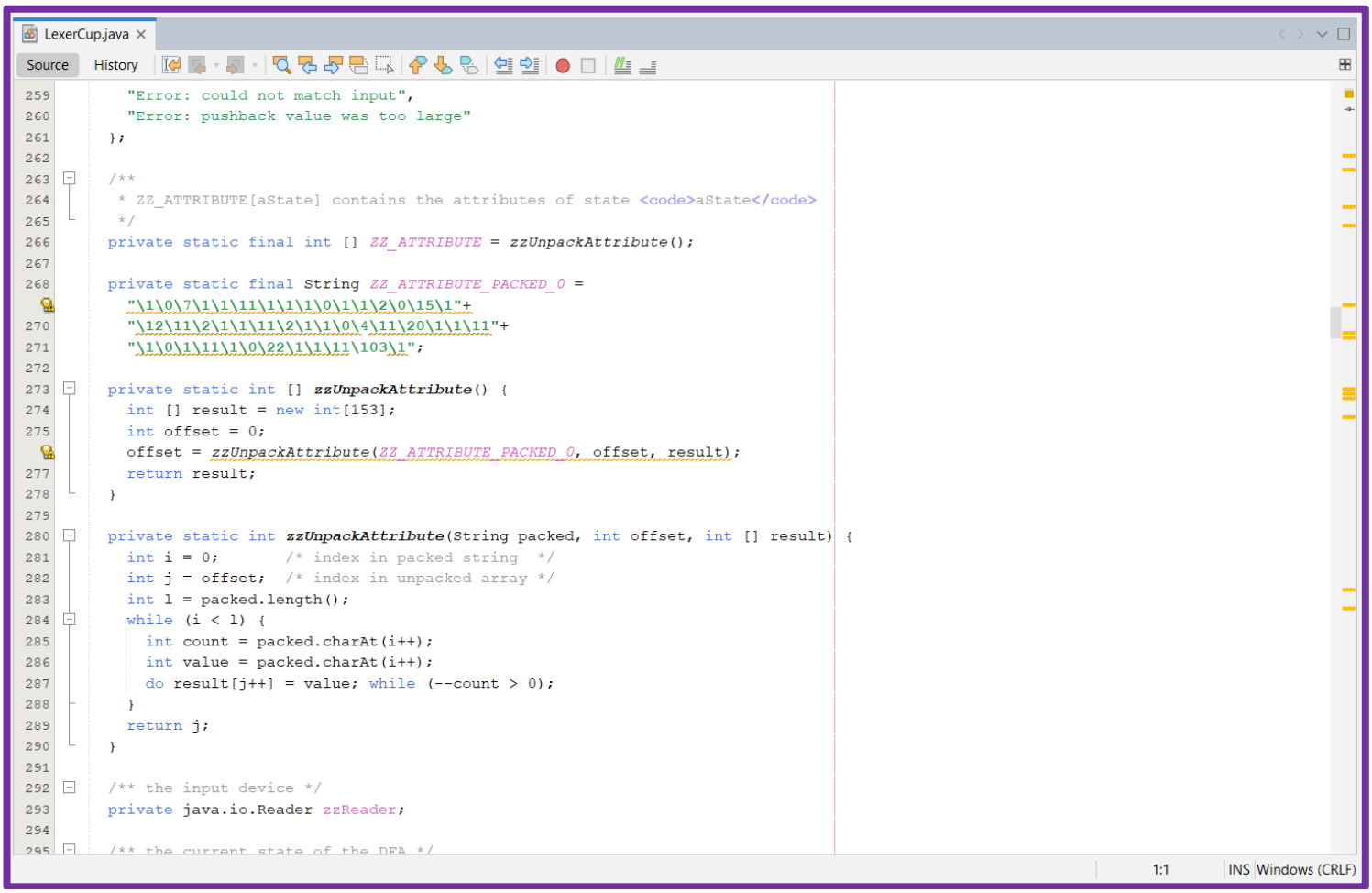


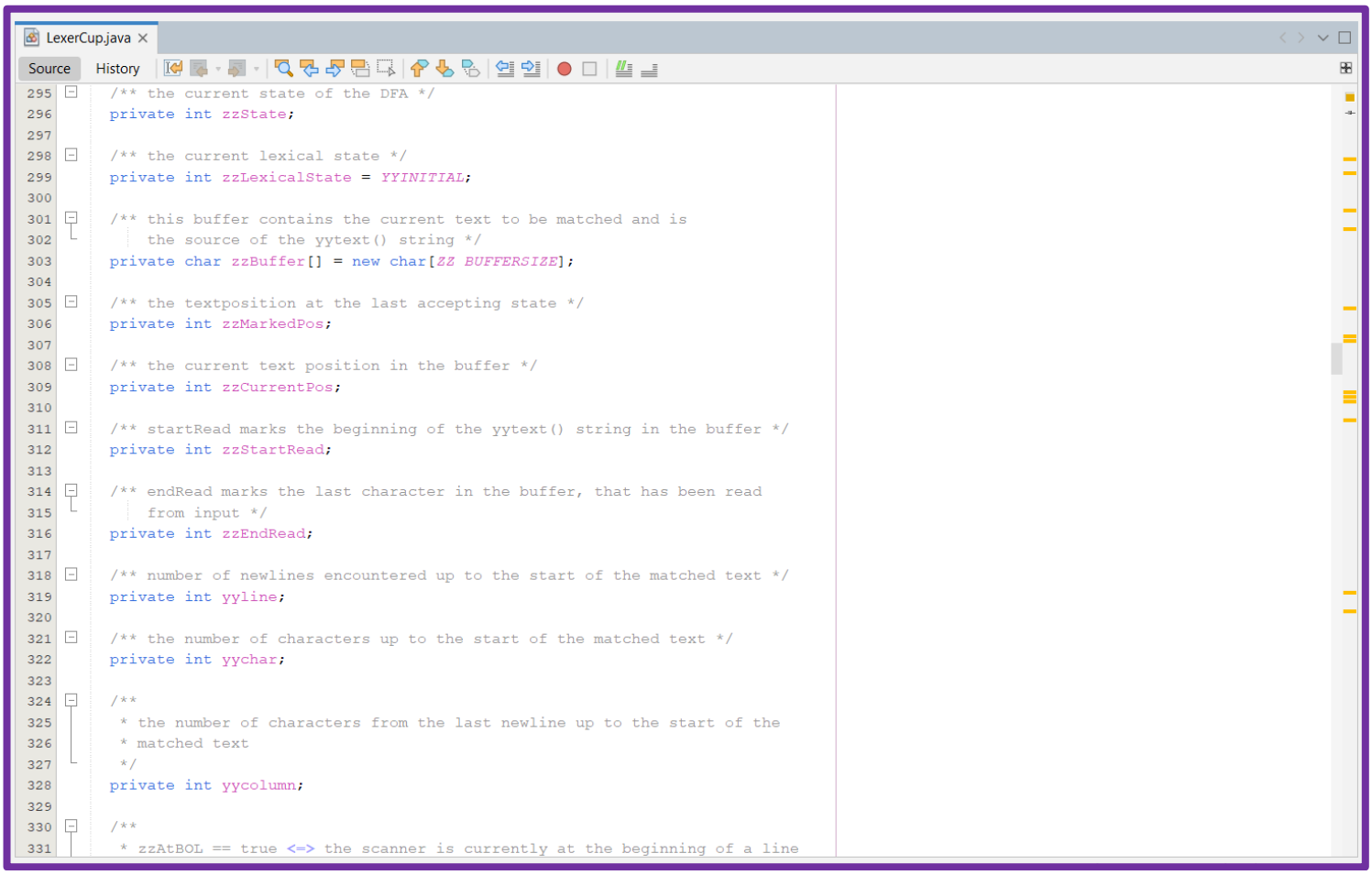








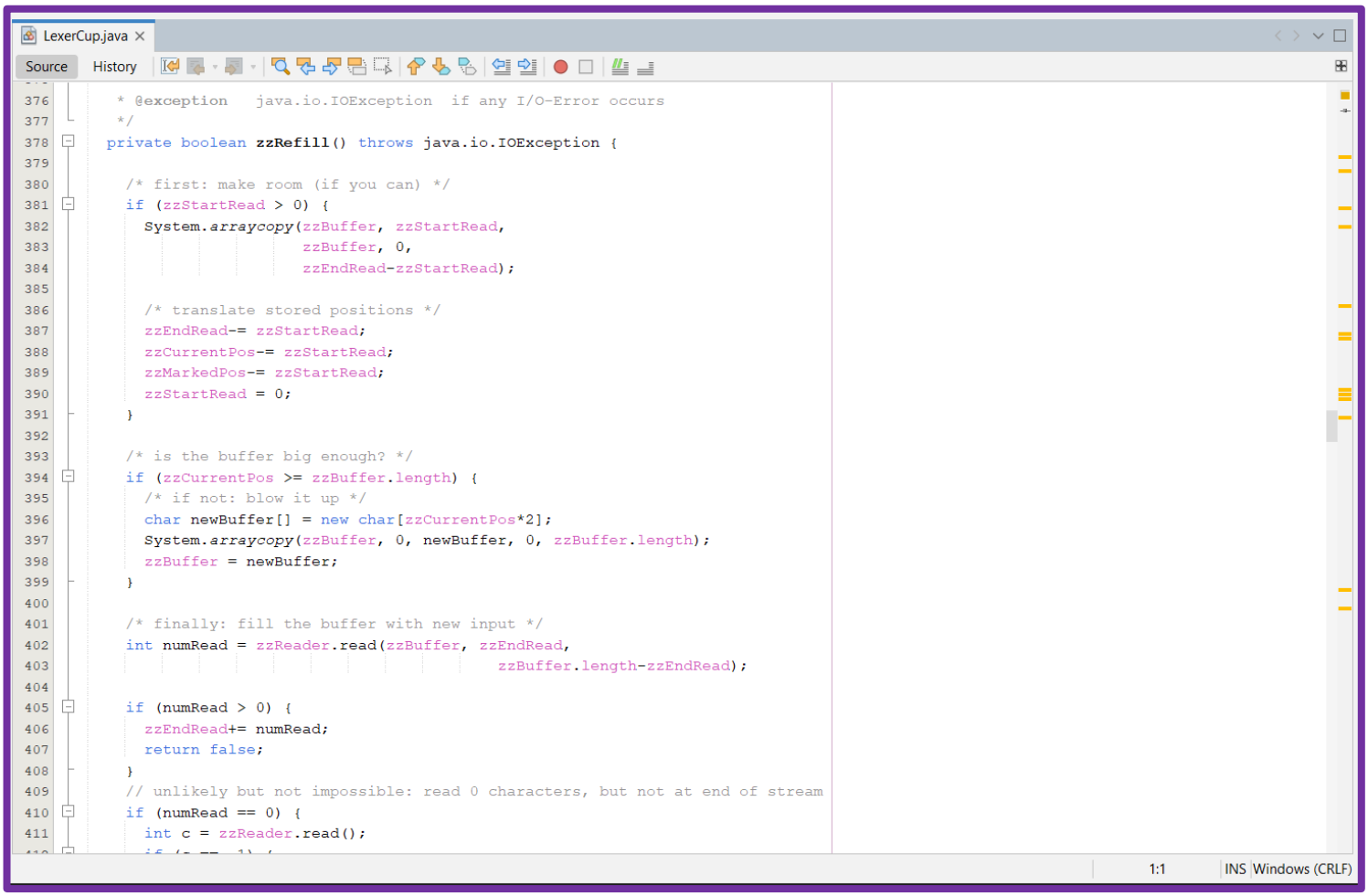


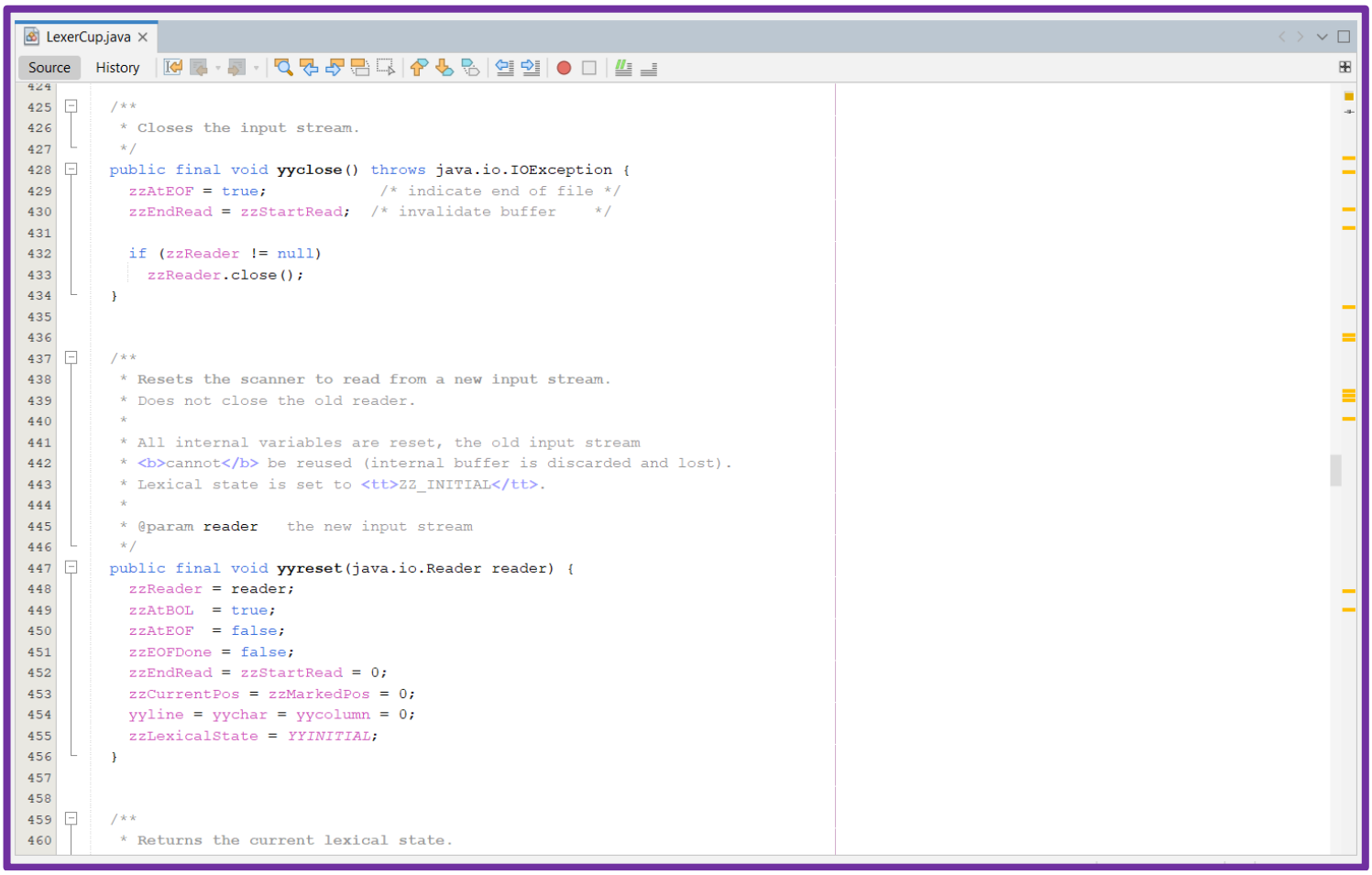


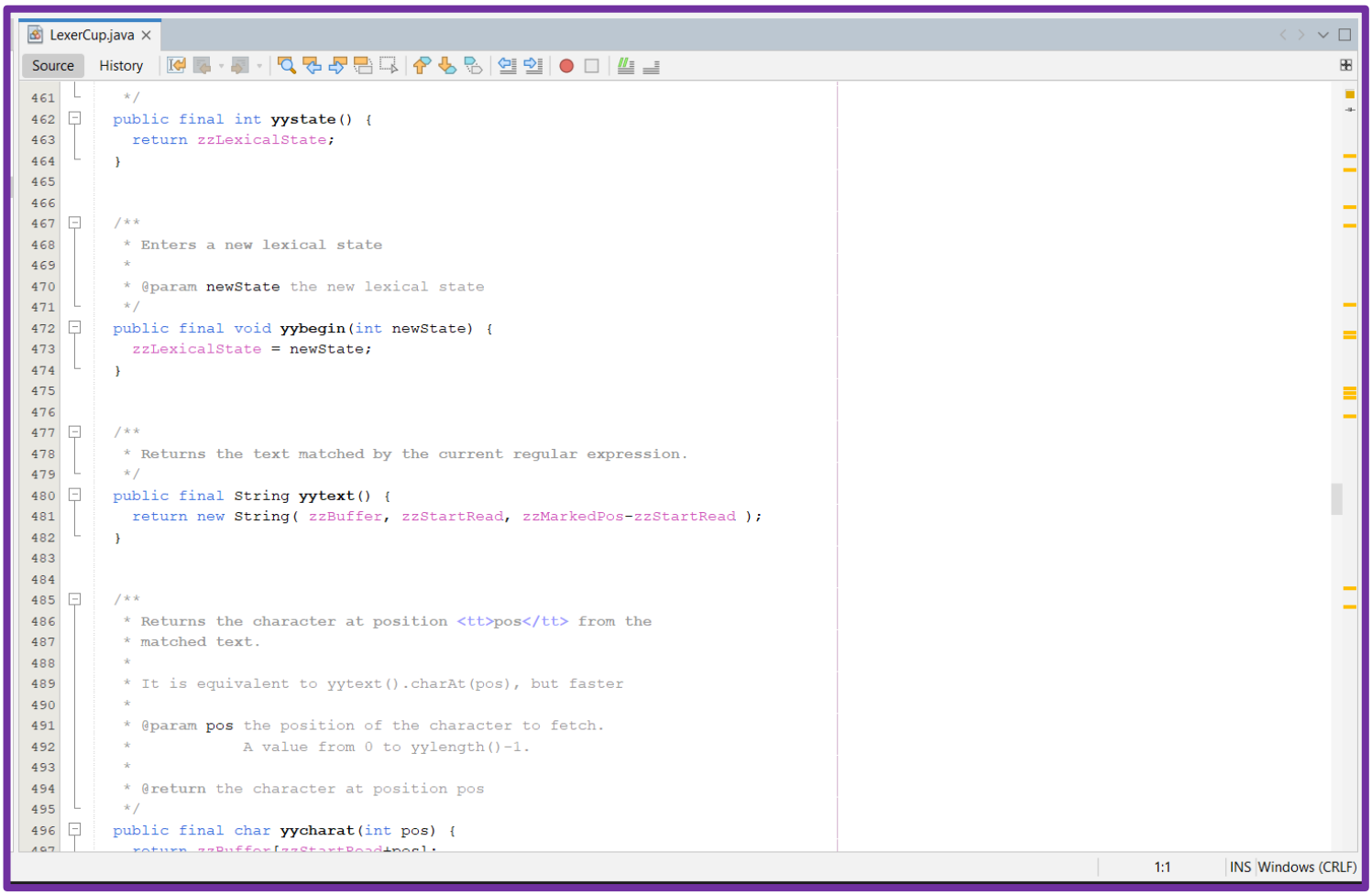
LexerCup.java

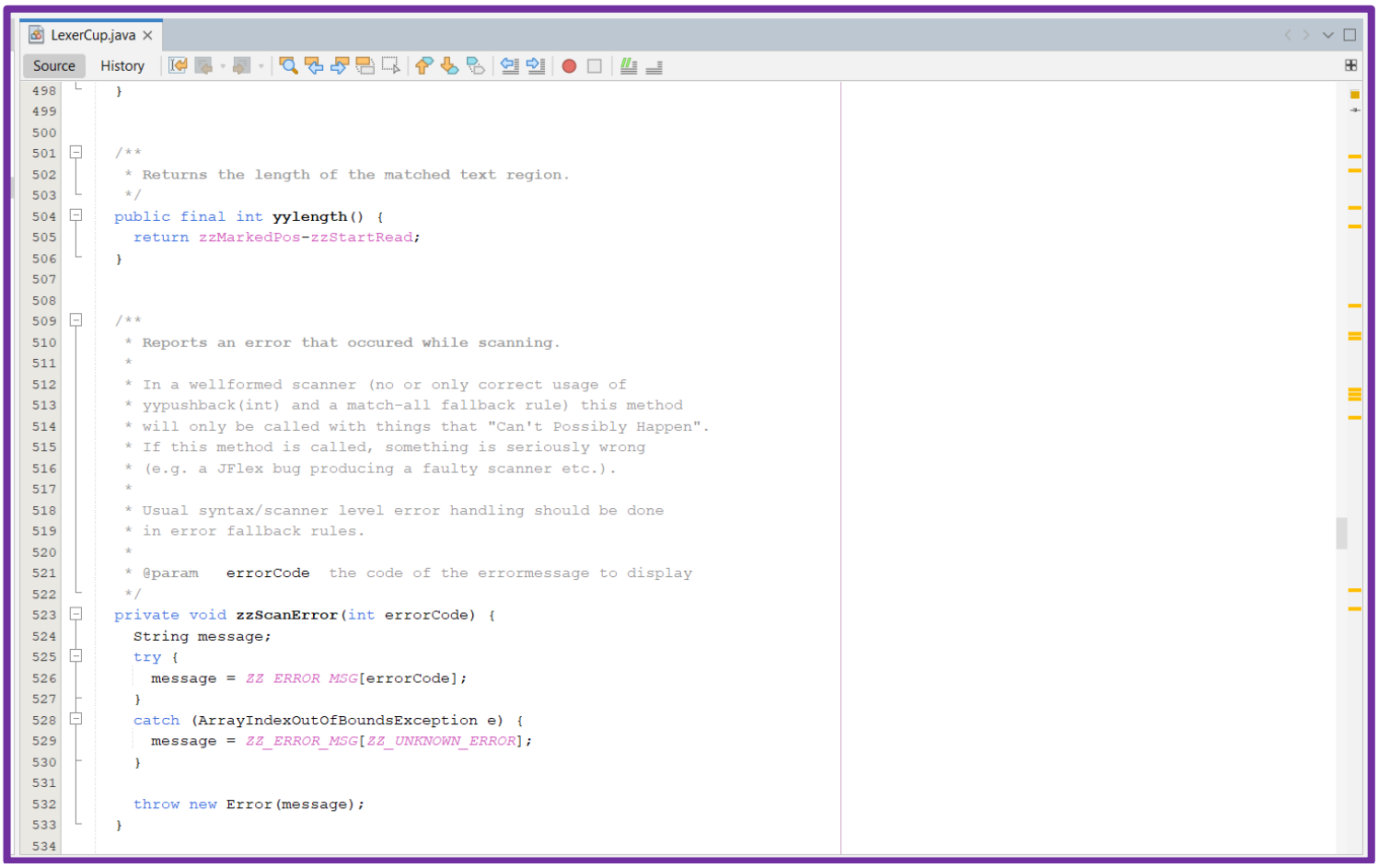
Source History

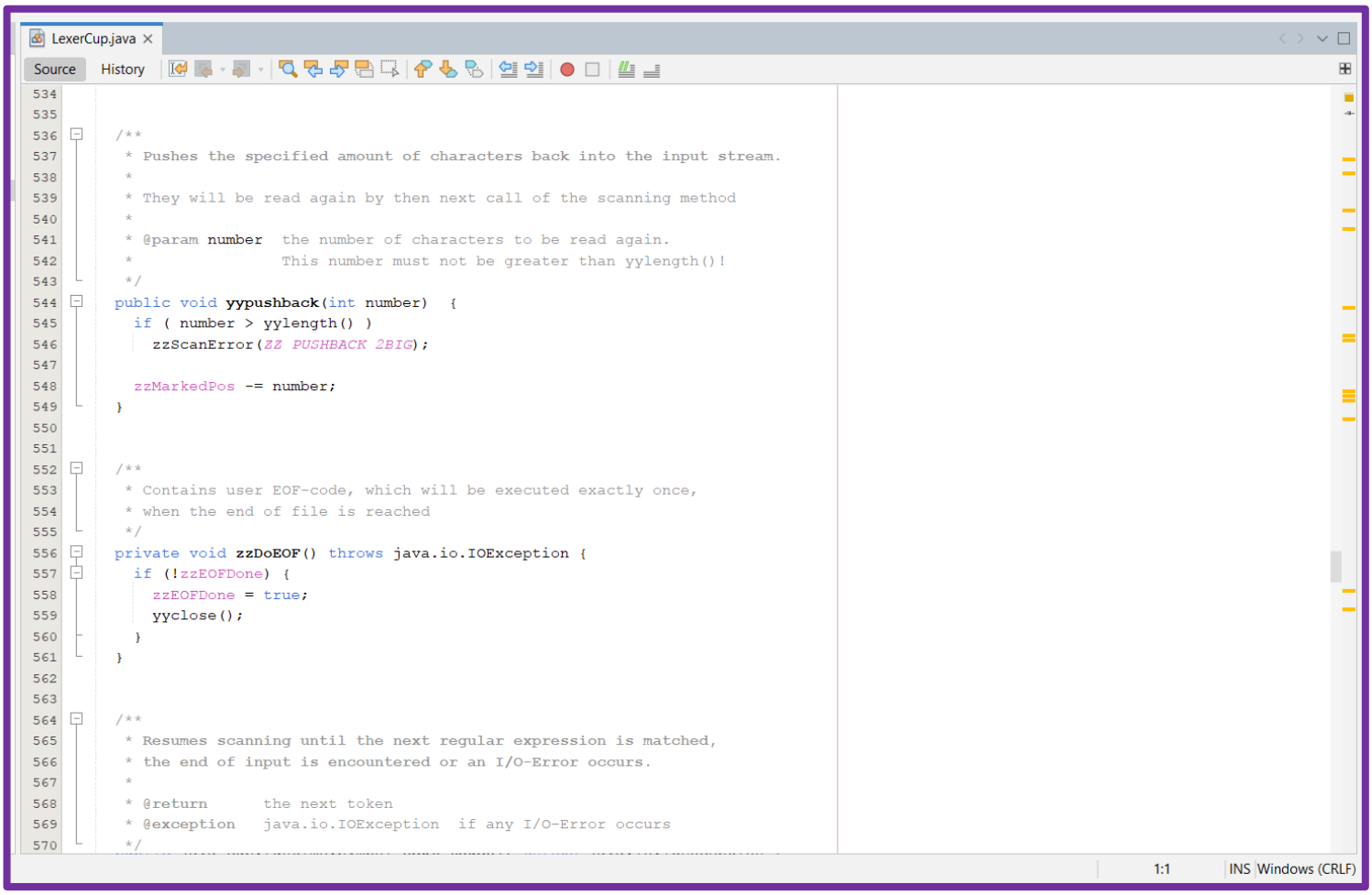
```
  */
  private boolean zzAtBOL = true;

  /** zzAtEOF == true <=> the scanner is at the EOF */
  private boolean zzAtEOF;

  /** denotes if the user-EOF-code has already been executed */
  private boolean zzEOFDone;

  /* user code: */
private Symbol symbol(int type, Object value){
return new Symbol(type, yyline, yycolumn, value);
}
private Symbol symbol(int type){
return new Symbol(type, yyline, yycolumn);
}


  /**
   * Creates a new scanner
   * There is also a java.io.InputStream version of this constructor.
   *
   * @param   in  the java.io.Reader to read input from.
   */
  LexerCup(java.io.Reader in) {
    this.zzReader = in;
  }

  /**
   * Creates a new scanner.
   * There is also java.io.Reader version of this constructor.
   *
   * @param   in  the java.io.Inputstream to read input from.
   */
  LexerCup(java.io.InputStream in) {
    this(new java.io.InputStreamReader(in));
  }
```

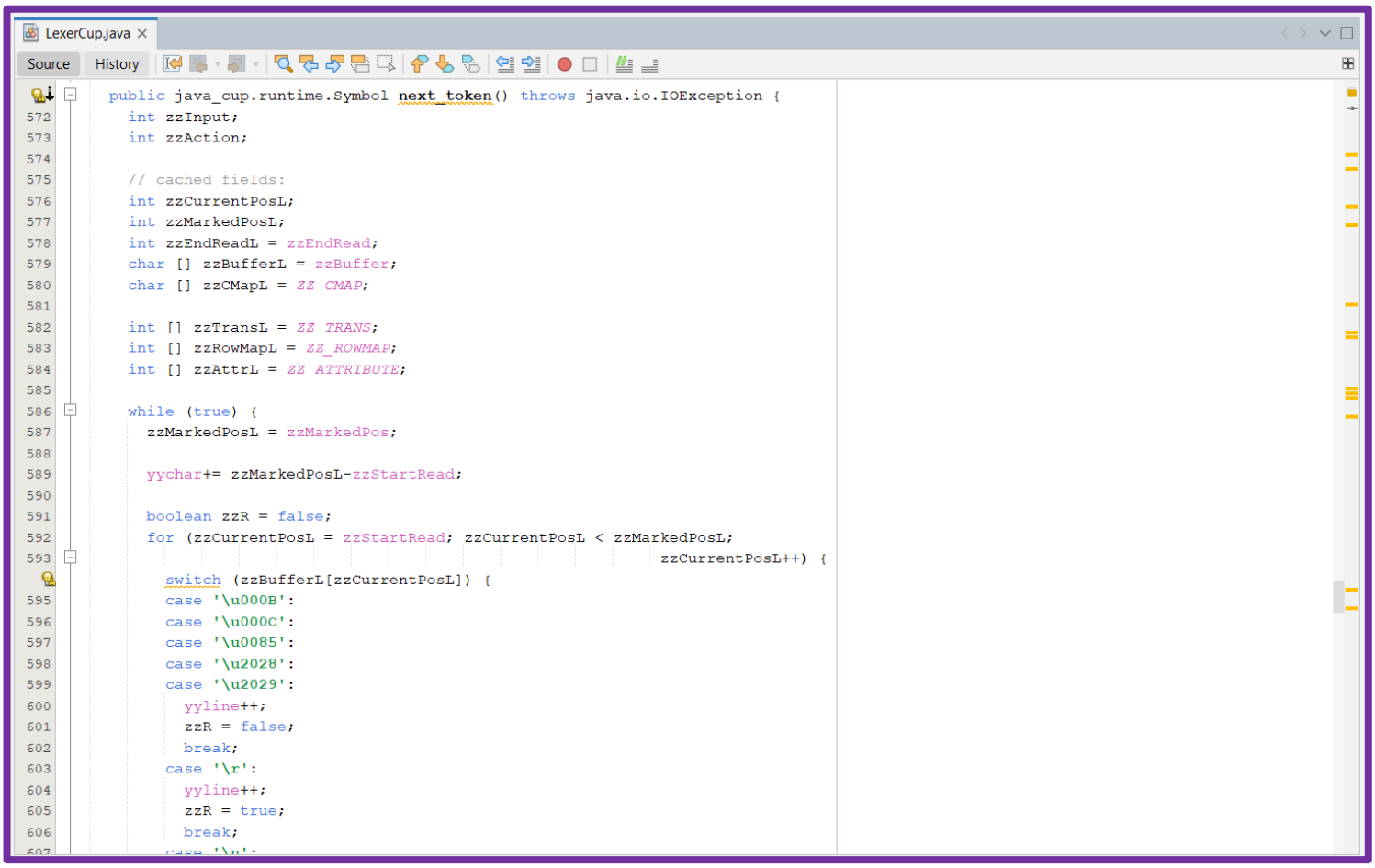


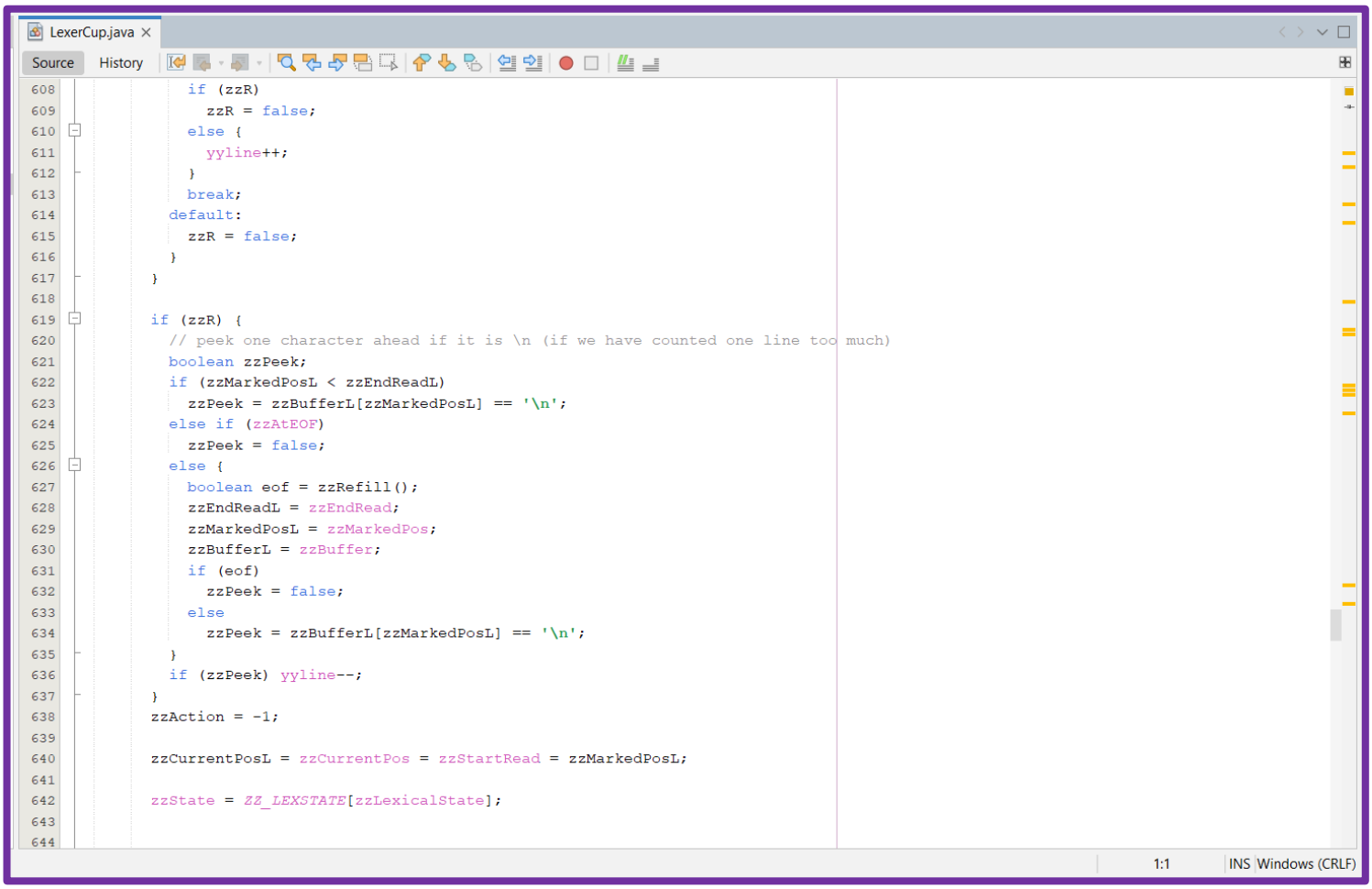


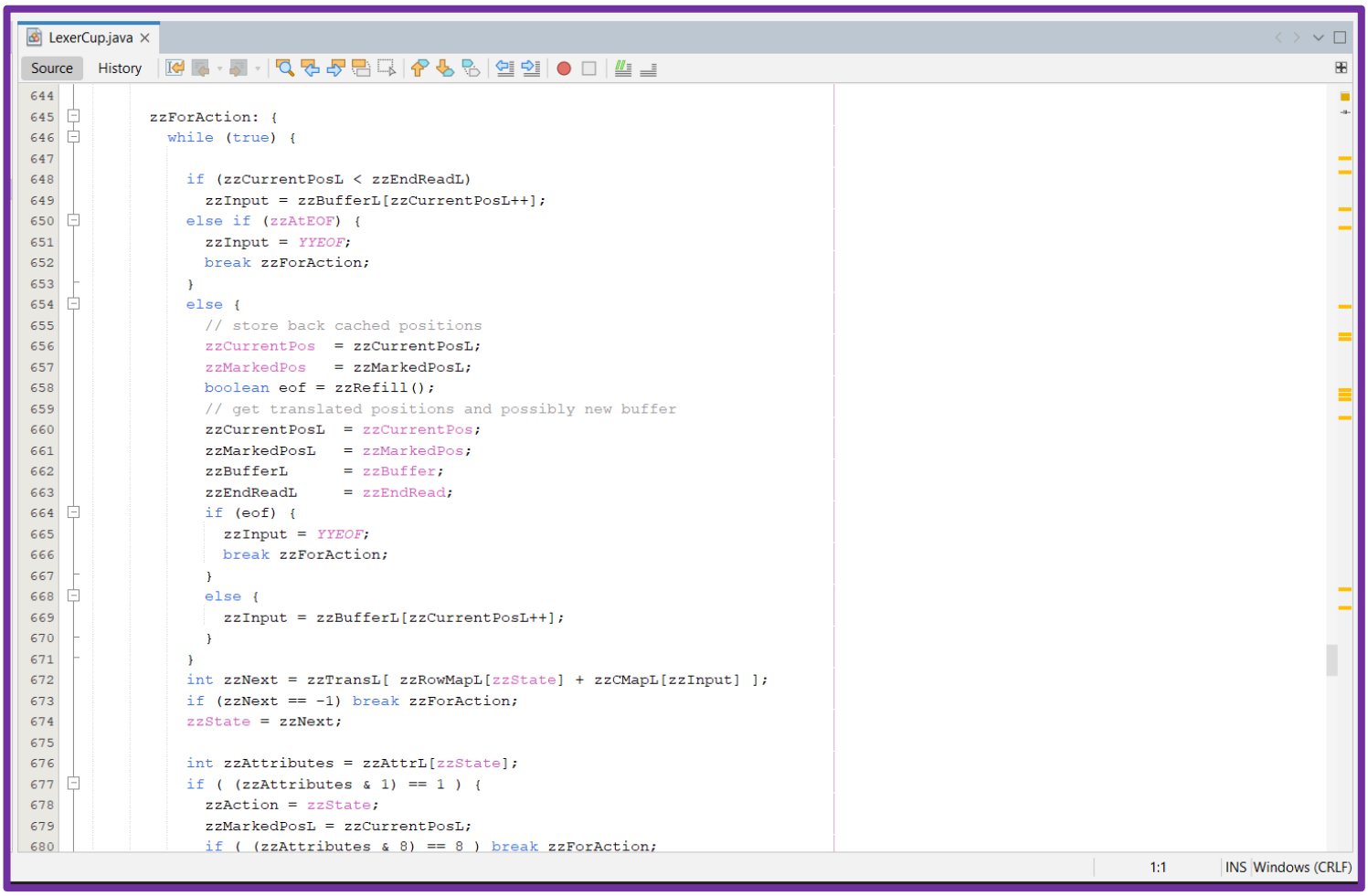


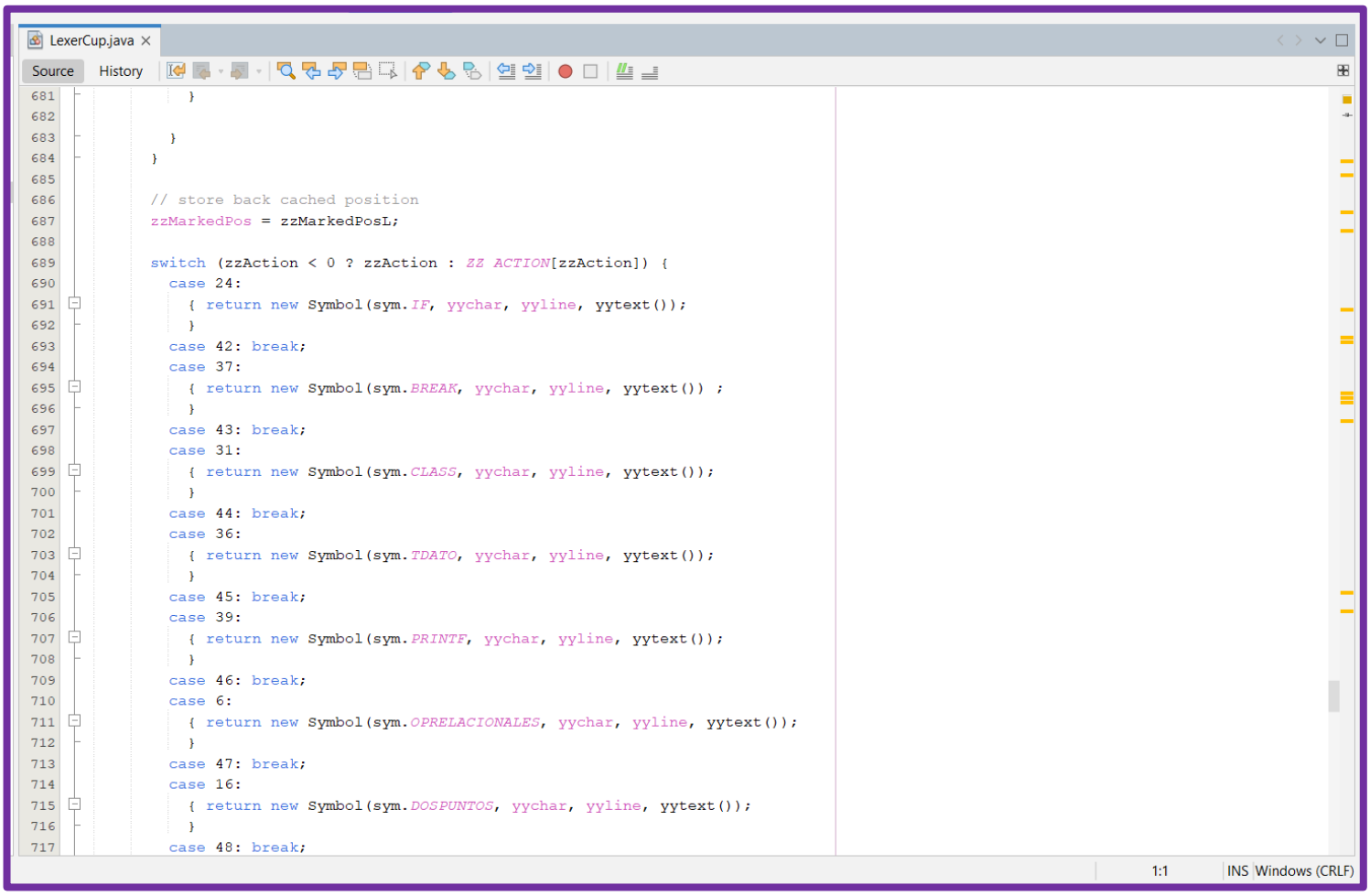


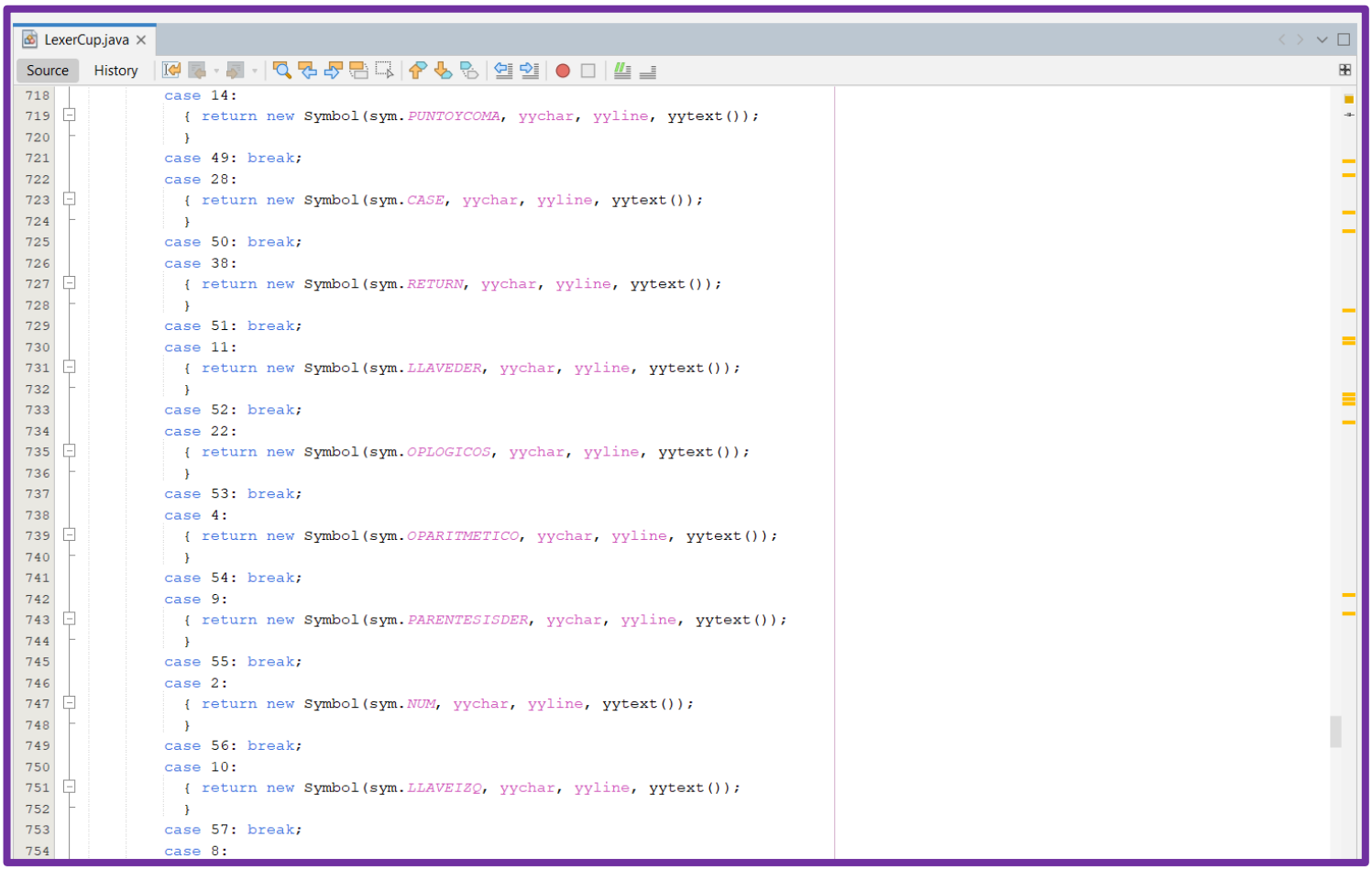


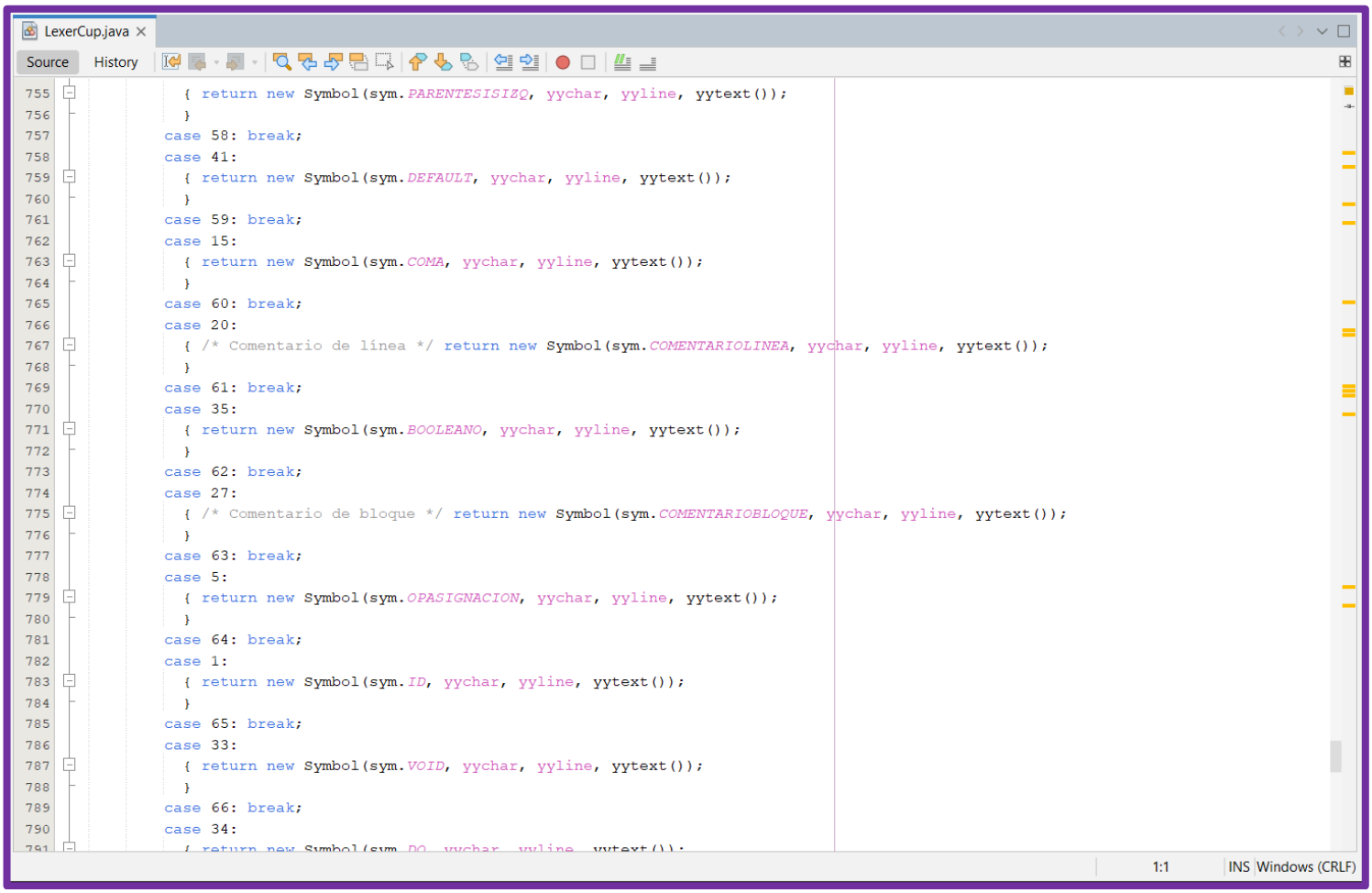

LexerCup.java
Source
History
public java_cup.runtime.Symbol next_token() throws java.io.IOException {
int zzInput;
int zzAction;
// cached fields:
int zzCurrentPosL;
int zzMarkedPosL;
int zzEndReadL = zzEndRead;
char [] zzBufferL = zzBuffer;
char [] zzCMapL = ZZ_CMAP;
int [] zzTransL = ZZ_TRANS;
int [] 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;

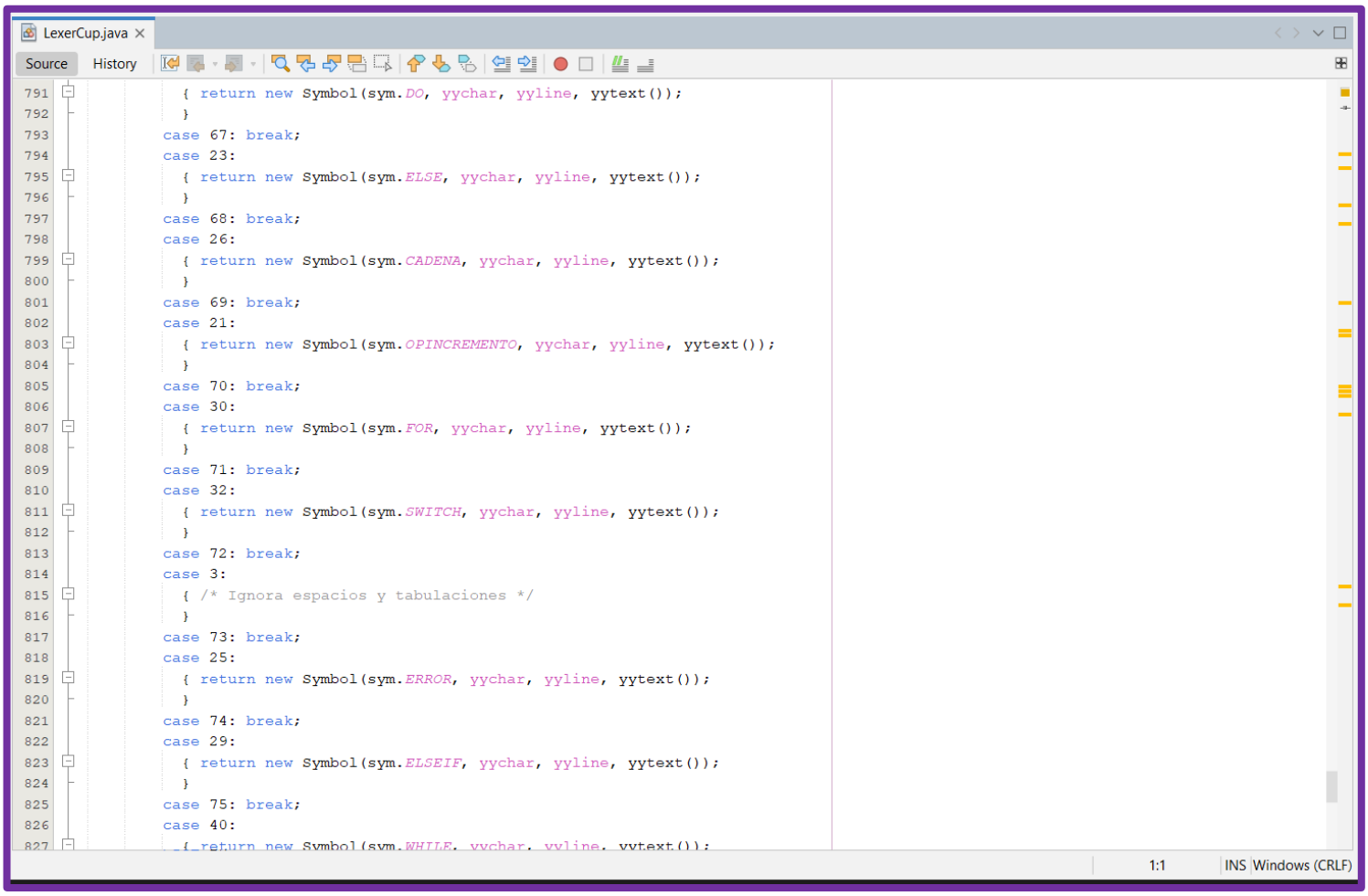


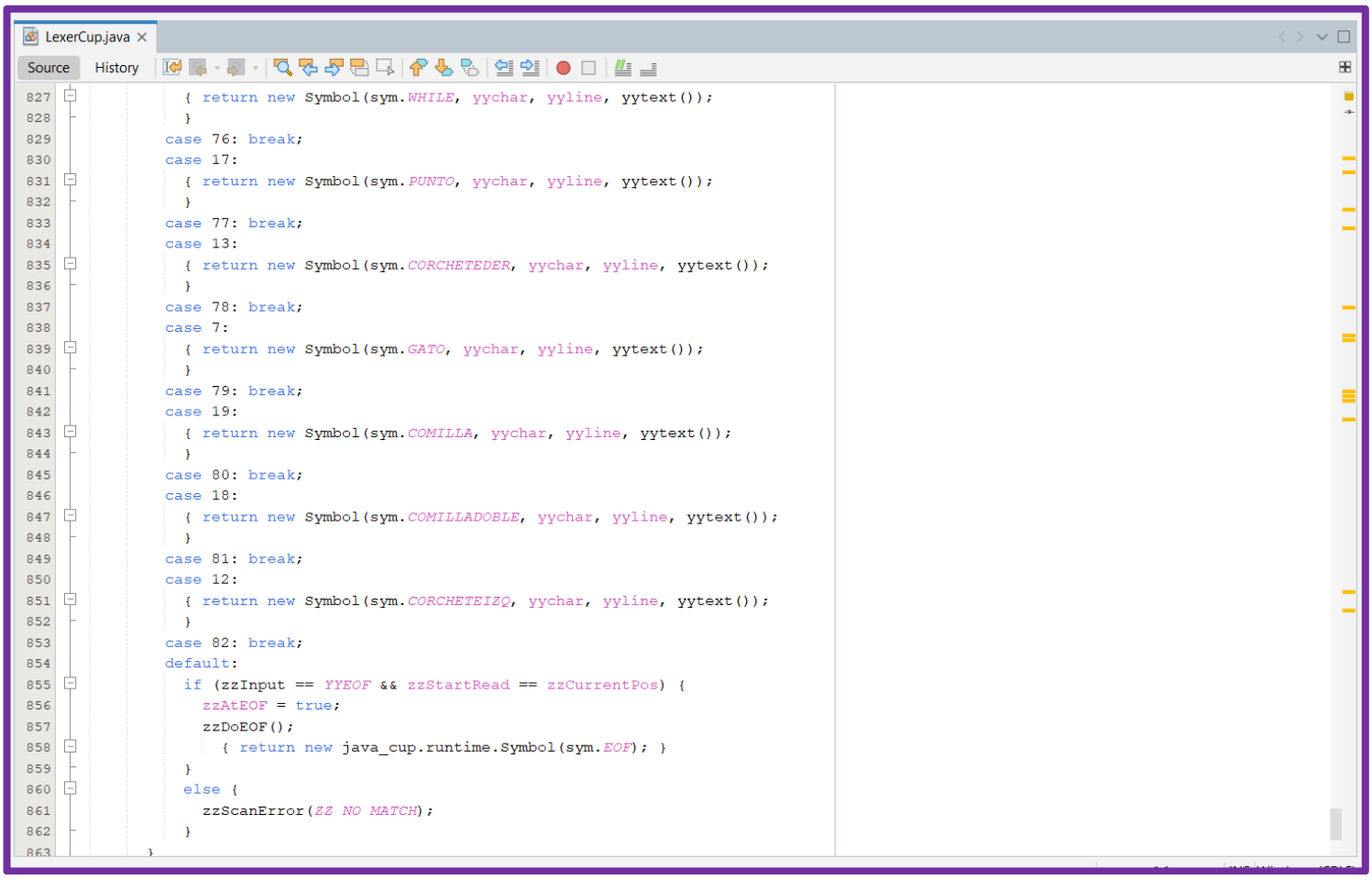








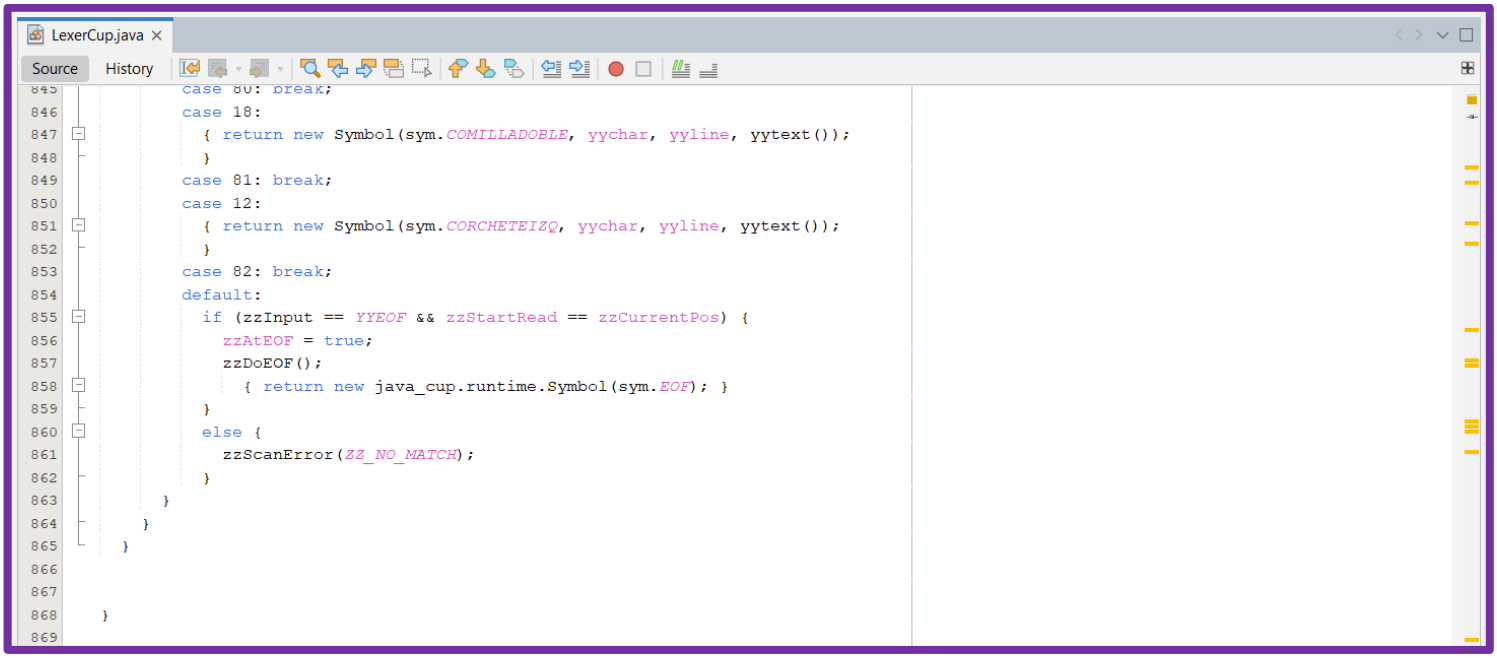

LexerCup.java
Source
History
{ return new Symbol(sym.DO, yychar, yyline, yytext());
}
case 67: break;
case 23:
{ return new Symbol(sym.ELSE, yychar, yyline, yytext());
}
case 68: break;
case 26:
{ return new Symbol(sym.CADENA, yychar, yyline, yytext());
}
case 69: break;
case 21:
{ return new Symbol(sym.OPINCREMENTO, yychar, yyline, yytext());
}
case 70: break;
case 30:
{ return new Symbol(sym.FOR, yychar, yyline, yytext());
}
case 71: break;
case 32:
{ return new Symbol(sym.SWITCH, yychar, yyline, yytext());
}
case 72: break;
case 3:
{ /* Ignora espacios y tabulaciones */
}
case 73: break;
case 25:
{ return new Symbol(sym.ERROR, yychar, yyline, yytext());
}
case 74: break;
case 29:
{ return new Symbol(sym.ELSEIF, yychar, yyline, yytext());
}
case 75: break;
case 40:
1:1
INS Windows (CRLF)



LexerCup.java

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         }
```



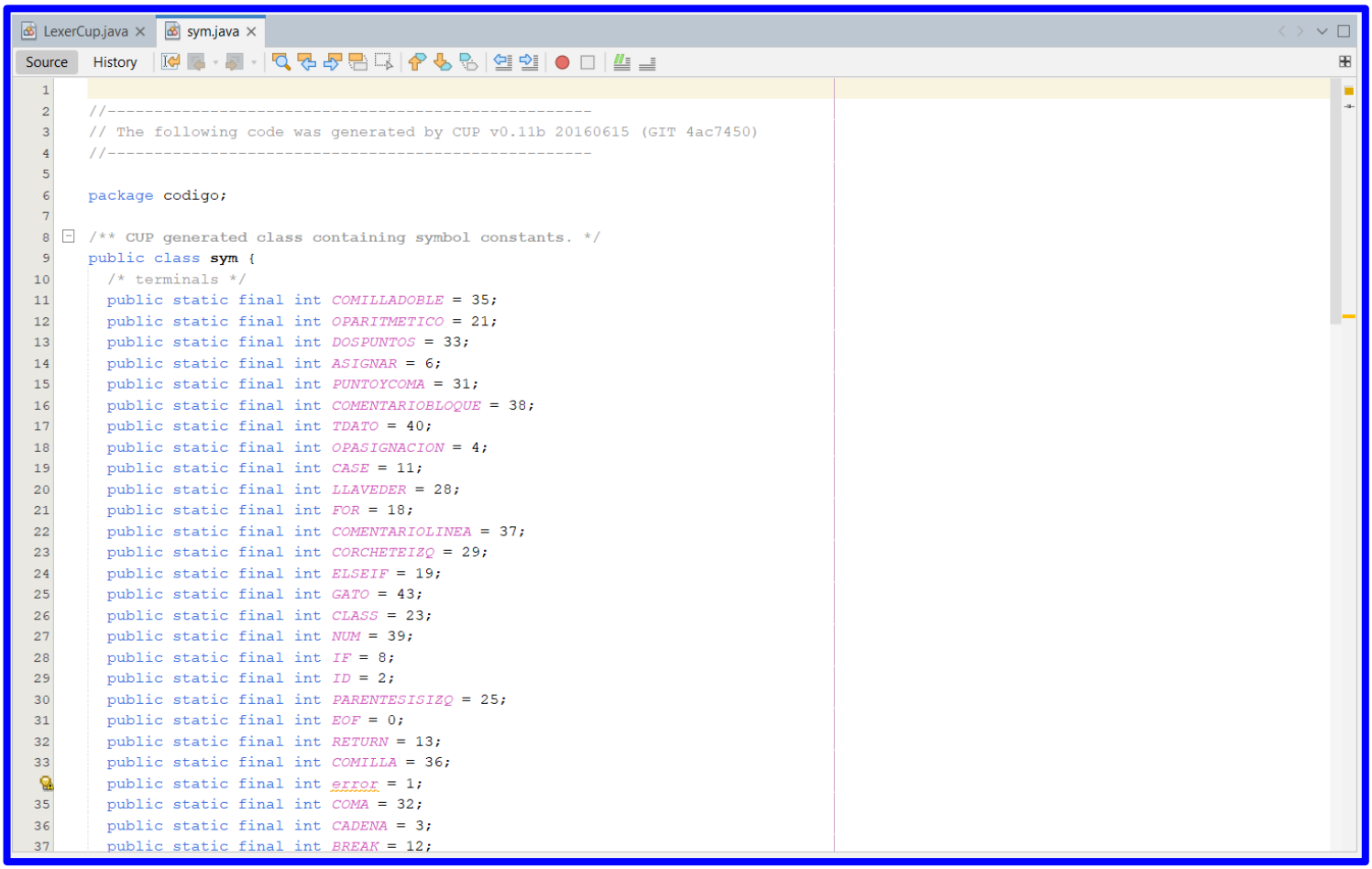
LexerCup.java

Source History

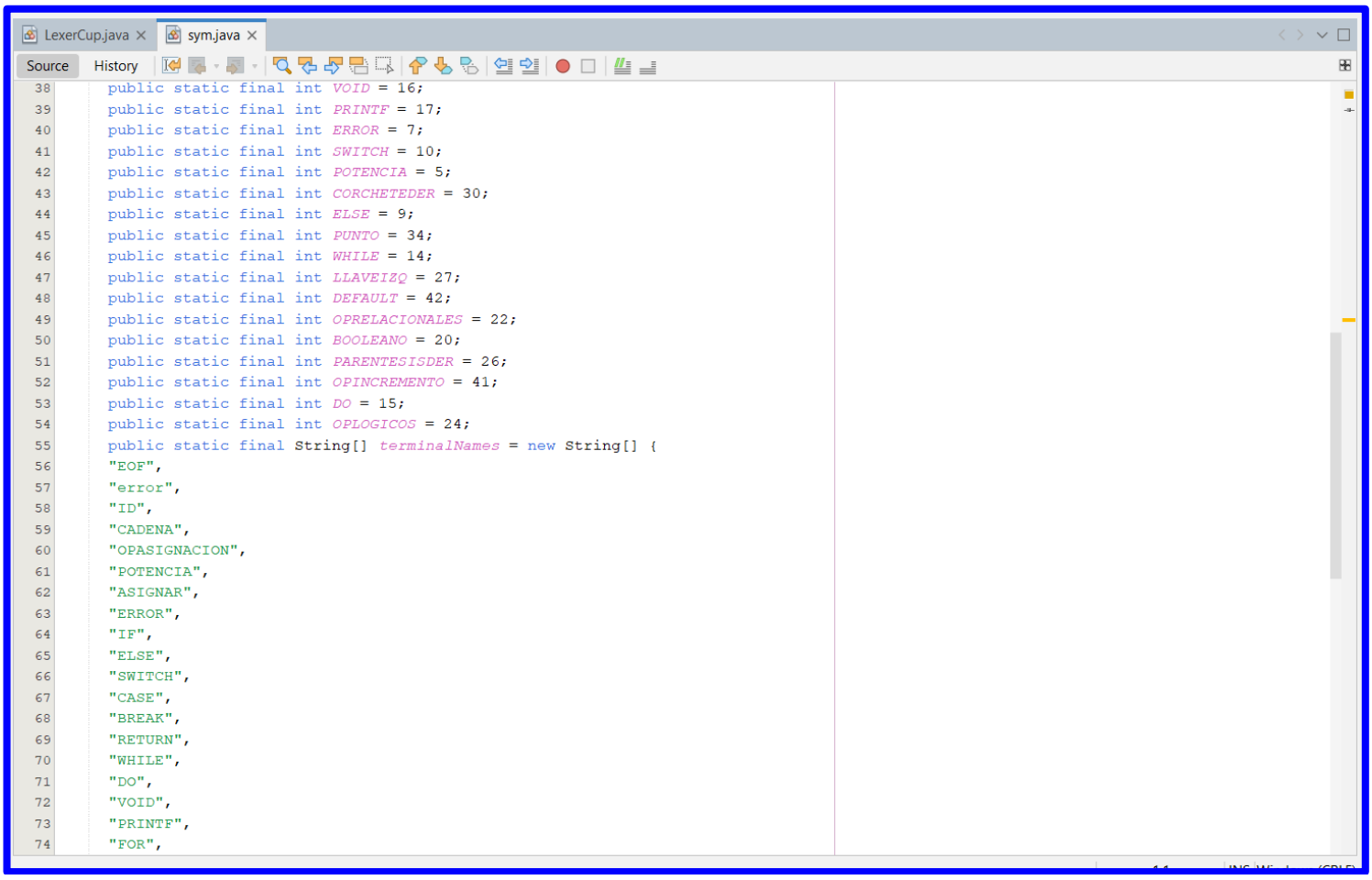
```
845           case 80: break;
846           case 18:
847             { return new Symbol(sym.COMILLADOBLE, yychar, yyline, yytext());
848             }
849           case 81: break;
850           case 12:
851             { return new Symbol(sym.CORCHETEIZQ, yychar, yyline, yytext());
852             }
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     }
866
867
868   }
869
```

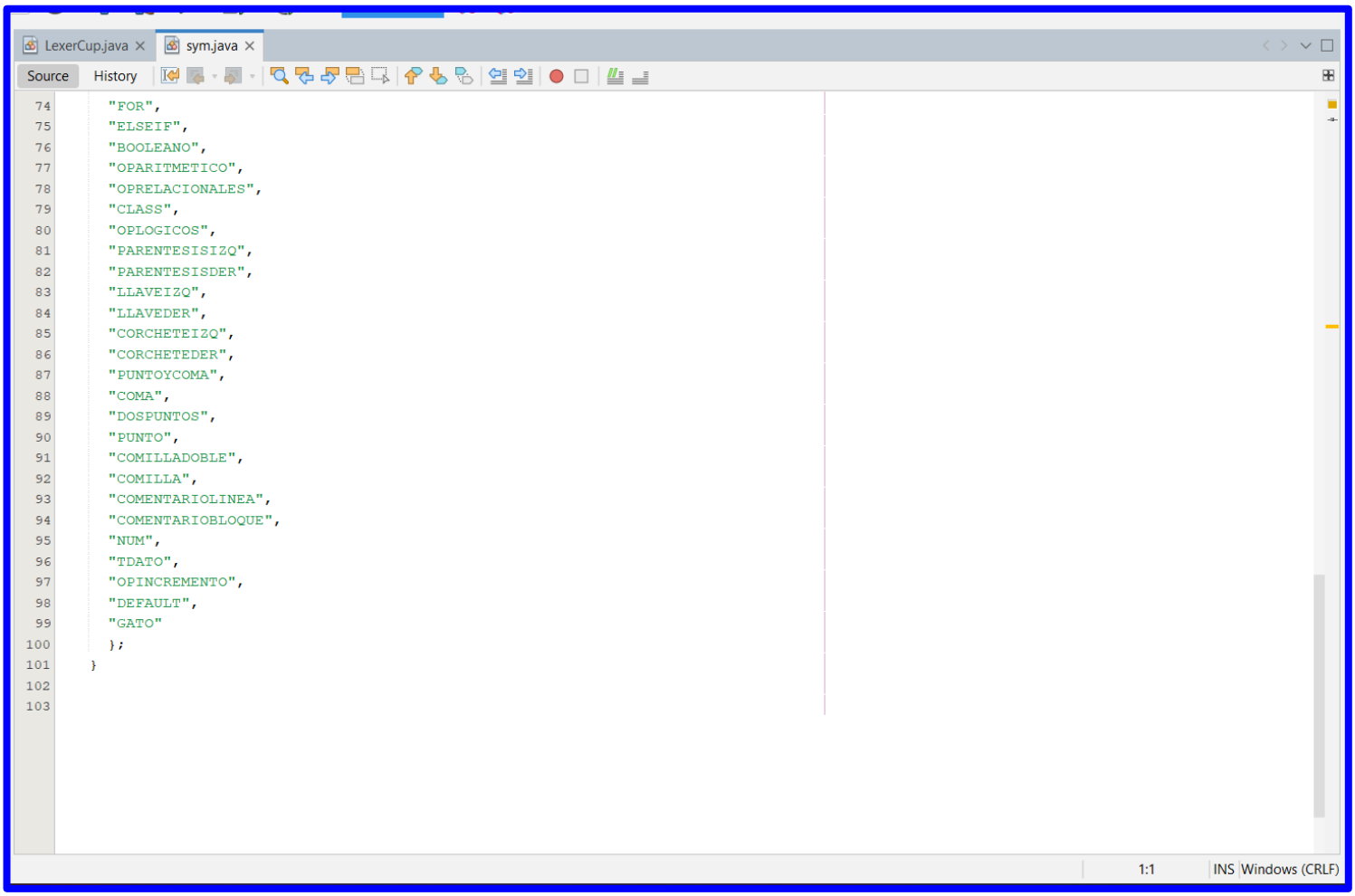
Sym.java

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



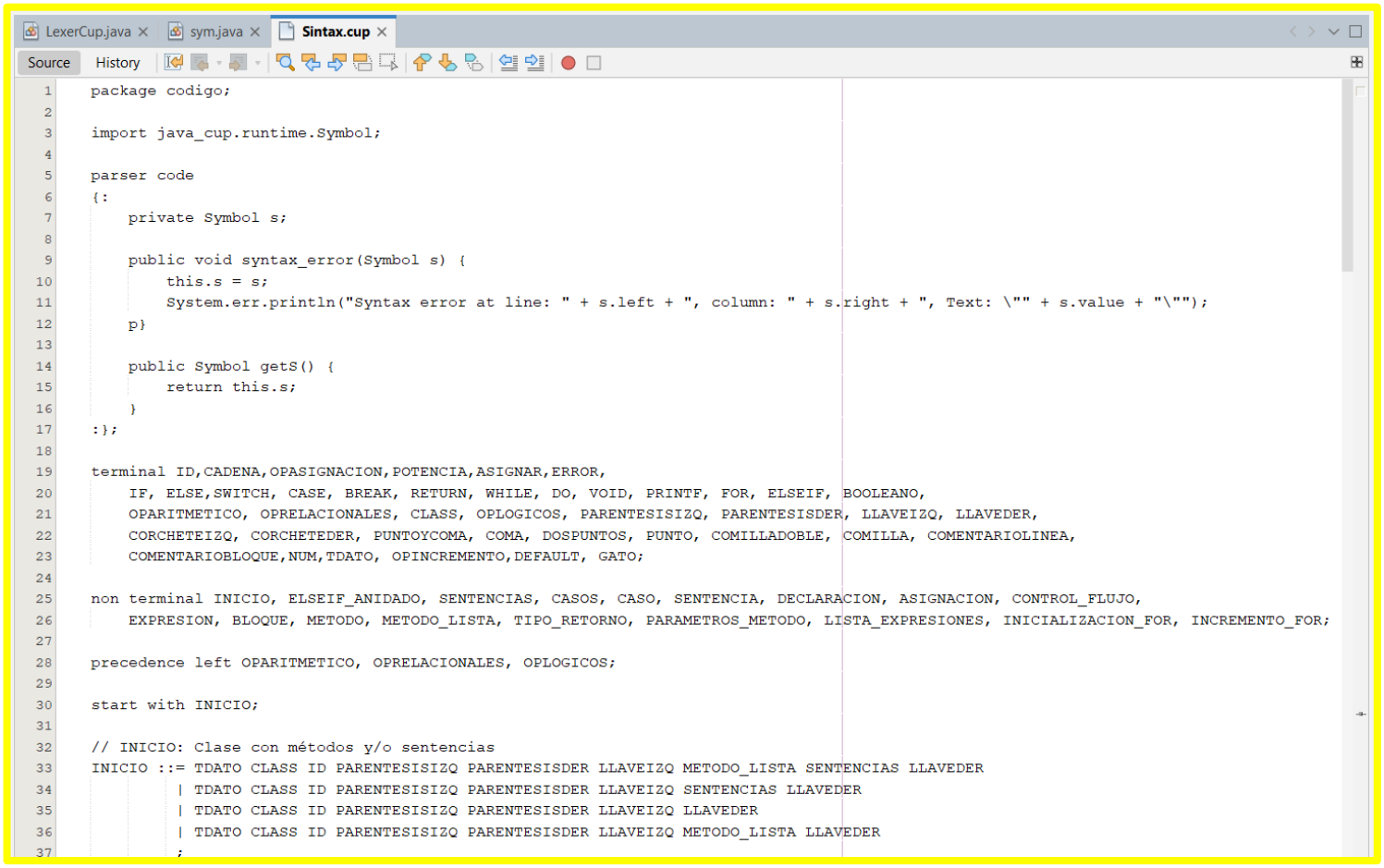

```
1
2   //----------------------------------------------------
3   // The following code was generated by CUP v0.11b 20160615 (GIT 4ac7450)
4   //----------------------------------------------------
5
6   package codigo;
7
8   /** CUP generated class containing symbol constants. */
9   public class 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;
      public static final int error = 1;
35    public static final int COMA = 32;
36    public static final int CADENA = 3;
37    public static final int BREAK = 12;
```





Sintax.cup

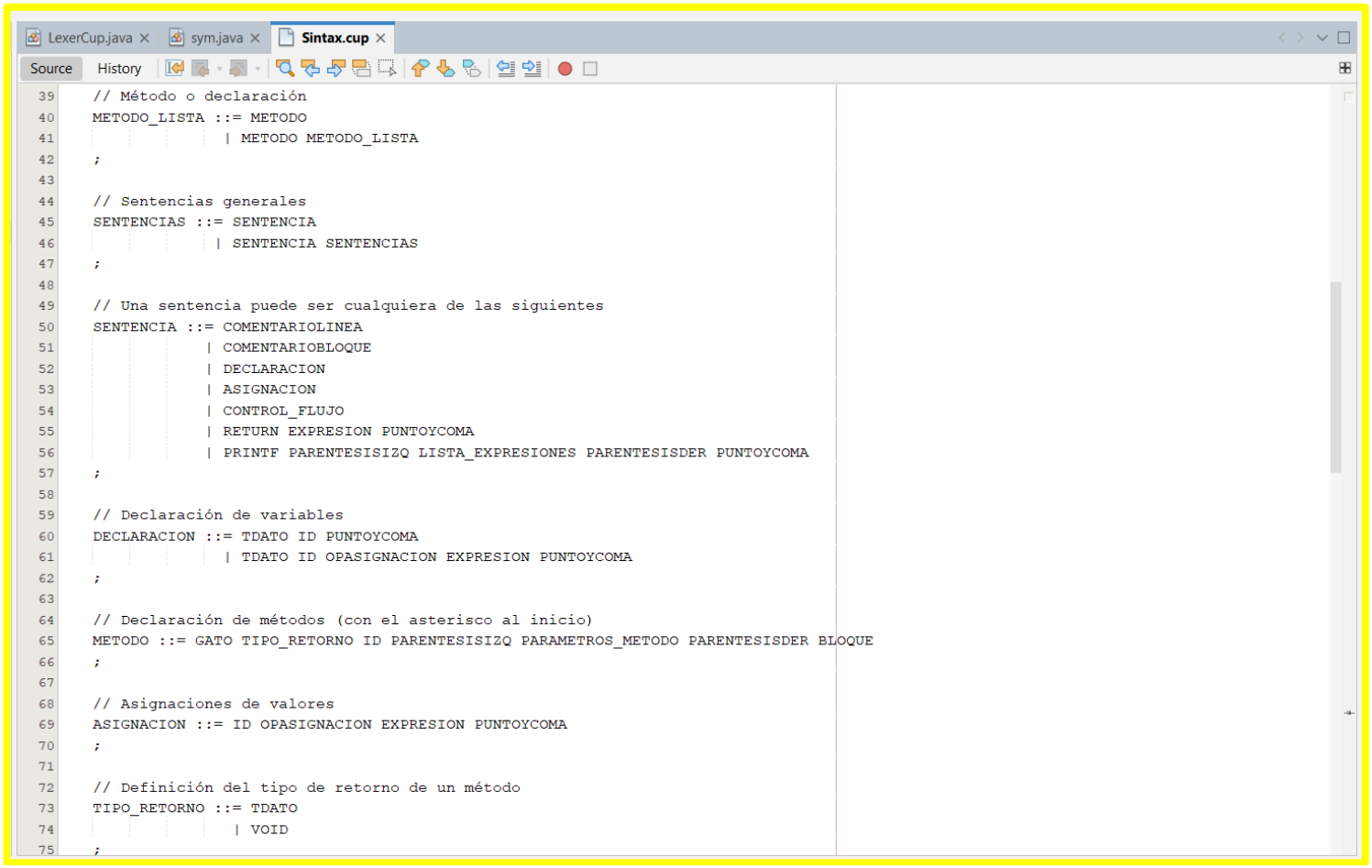
Este código implementa un parser (analizador sintáctico) en Java CUP para un lenguaje de programación personalizado. Define la gramática del lenguaje usando reglas que describen cómo deben estructurarse las clases, métodos, declaraciones, asignaciones, estructuras de control de flujo, y expresiones. 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.

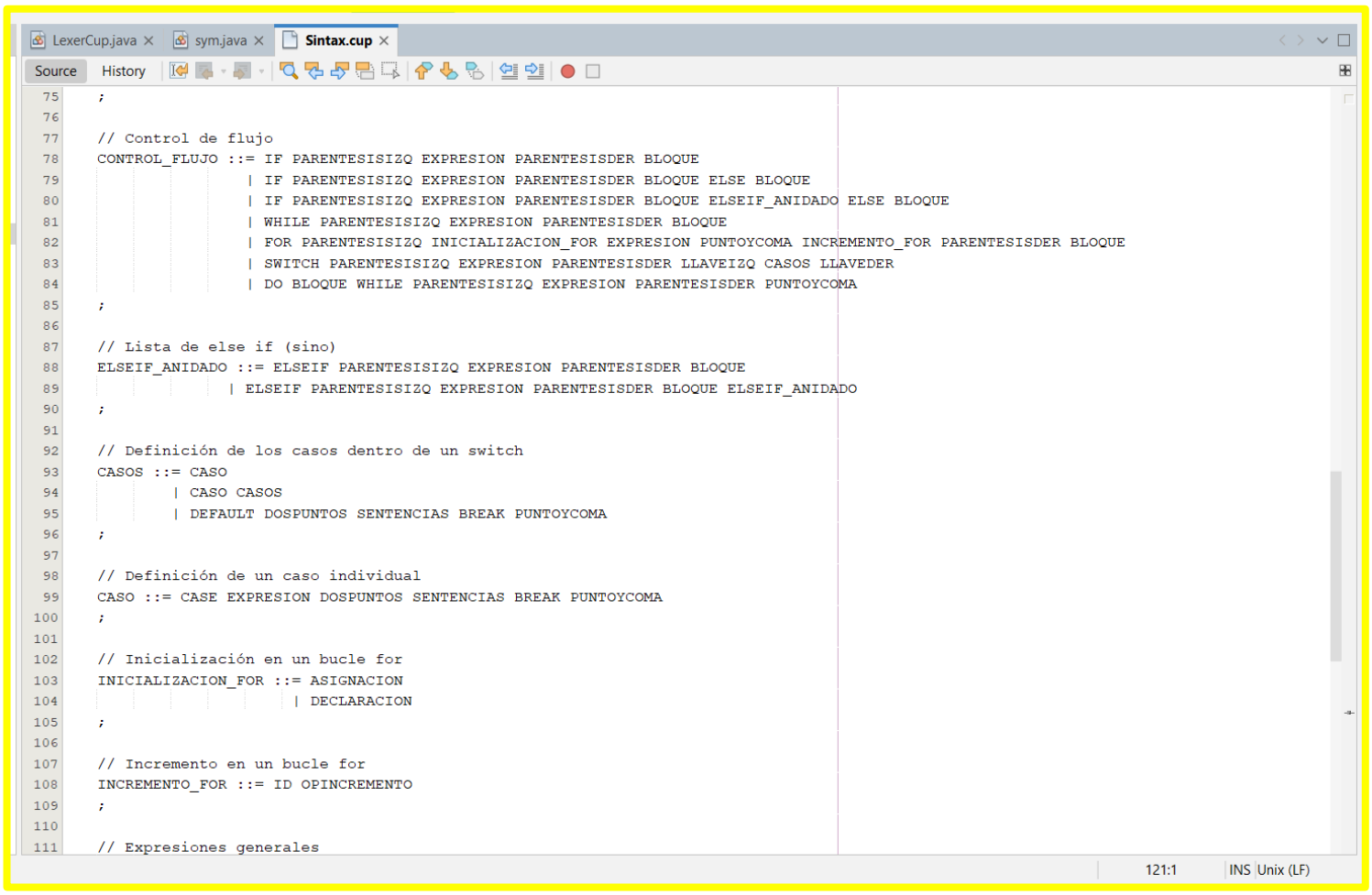


LexerCup.java | sym.java | Sintax.cup

Source | History

```
package codigo;

import java_cup.runtime.Symbol;

parser code
{:
    private Symbol s;

    public void syntax_error(Symbol s) {
        this.s = s;
        System.err.println("Syntax error at line: " + s.left + ", column: " + s.right + ", Text: \"" + s.value + "\"");
    p}

    public Symbol getS() {
        return this.s;
    }
:};

terminal ID,CADENA,OPASIGNACION,POTENCIA,ASIGNAR,ERROR,
    IF, ELSE,SWITCH, CASE, BREAK, RETURN, WHILE, DO, VOID, PRINTF, FOR, ELSEIF, BOOLEANO,
    OPARITMETICO, OPRELACIONALES, CLASS, OPLOGICOS, PARENTESISIZQ, PARENTESISDER, LLAVEIZQ, LLAVEDER,
    CORCHETEIZQ, CORCHETEDER, PUNTOYCOMA, COMA, DOSPUNTOS, PUNTO, COMILLADOBLE, COMILLA, COMENTARIOLINEA,
    COMENTARIOBLOQUE,NUM,TDATO, OPINCREMENTO,DEFAULT, GATO;

non terminal INICIO, ELSEIF_ANIDADO, SENTENCIAS, CASOS, CASO, SENTENCIA, DECLARACION, ASIGNACION, CONTROL_FLUJO,
    EXPRESION, BLOQUE, METODO, METODO_LISTA, TIPO_RETORNO, PARAMETROS_METODO, LISTA_EXPRESIONES, INICIALIZACION_FOR, INCREMENTO_FOR;

precedence left OPARITMETICO, OPRELACIONALES, OPLOGICOS;

start with INICIO;

// INICIO: Clase con métodos y/o sentencias
INICIO ::= TDATO CLASS ID PARENTESISIZQ PARENTESISDER LLAVEIZQ METODO_LISTA SENTENCIAS LLAVEDER
         | TDATO CLASS ID PARENTESISIZQ PARENTESISDER LLAVEIZQ SENTENCIAS LLAVEDER
         | TDATO CLASS ID PARENTESISIZQ PARENTESISDER LLAVEIZQ LLAVEDER
         | TDATO CLASS ID PARENTESISIZQ PARENTESISDER LLAVEIZQ METODO_LISTA LLAVEDER
         ;
```


LexerCup.java
sym.java
Sintax.cup
Source
History
// 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
;

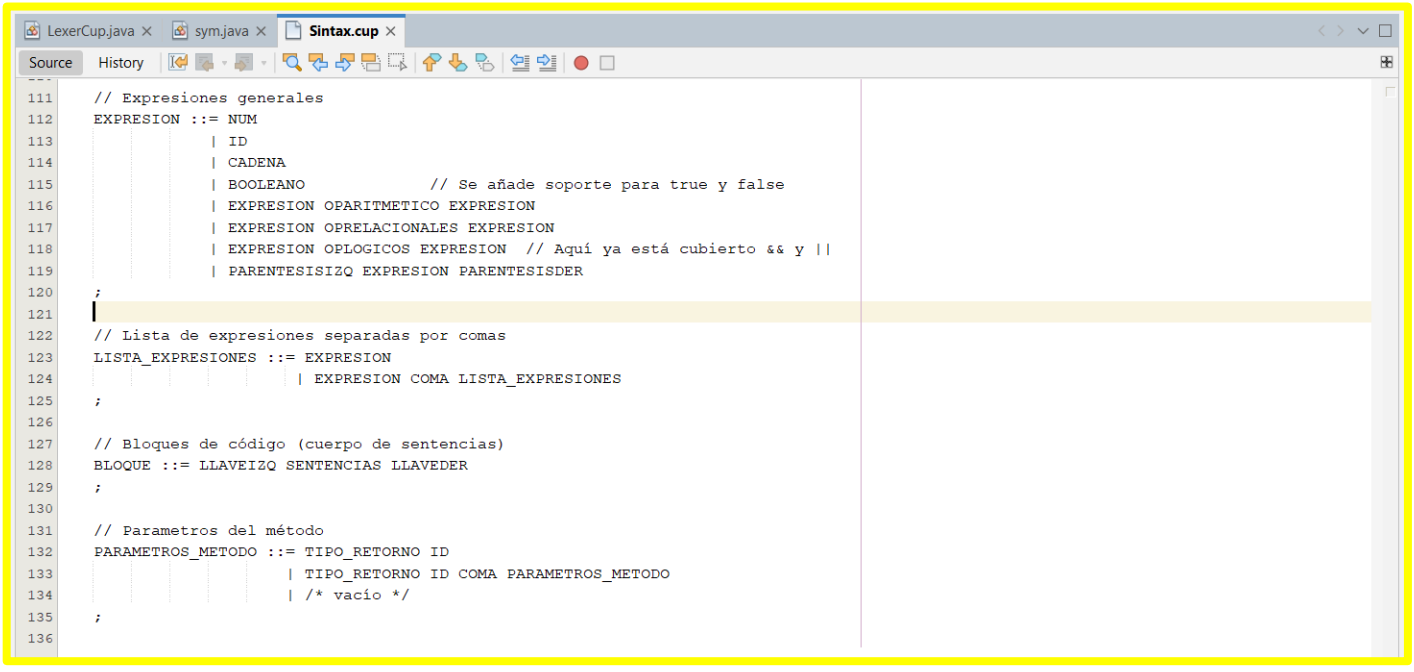


LexerCup.java | sym.java | Sintax.cup

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
```

121:1 INS Unix (LF)



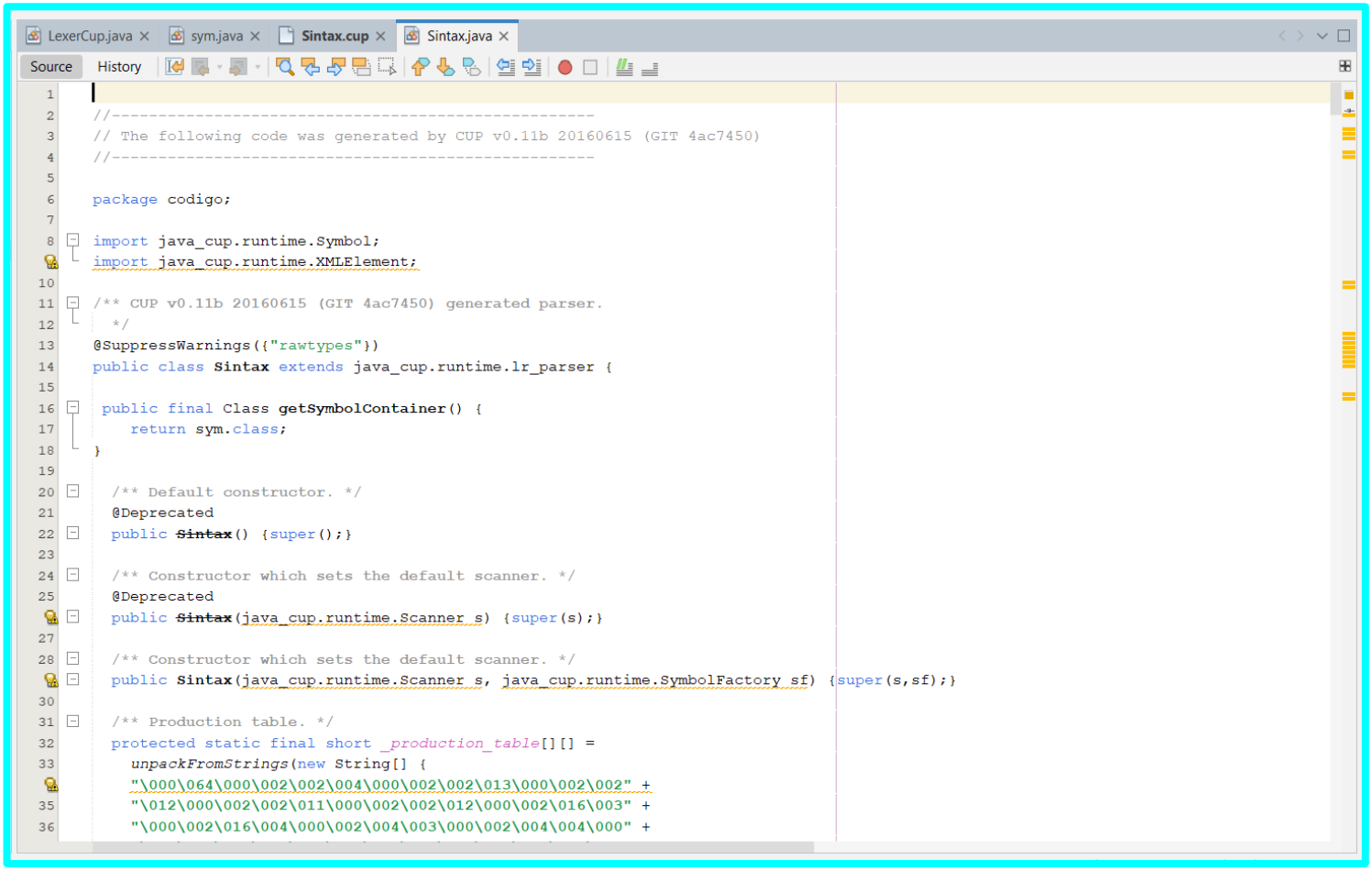
LexerCup.java | sym.java | Sintax.cup

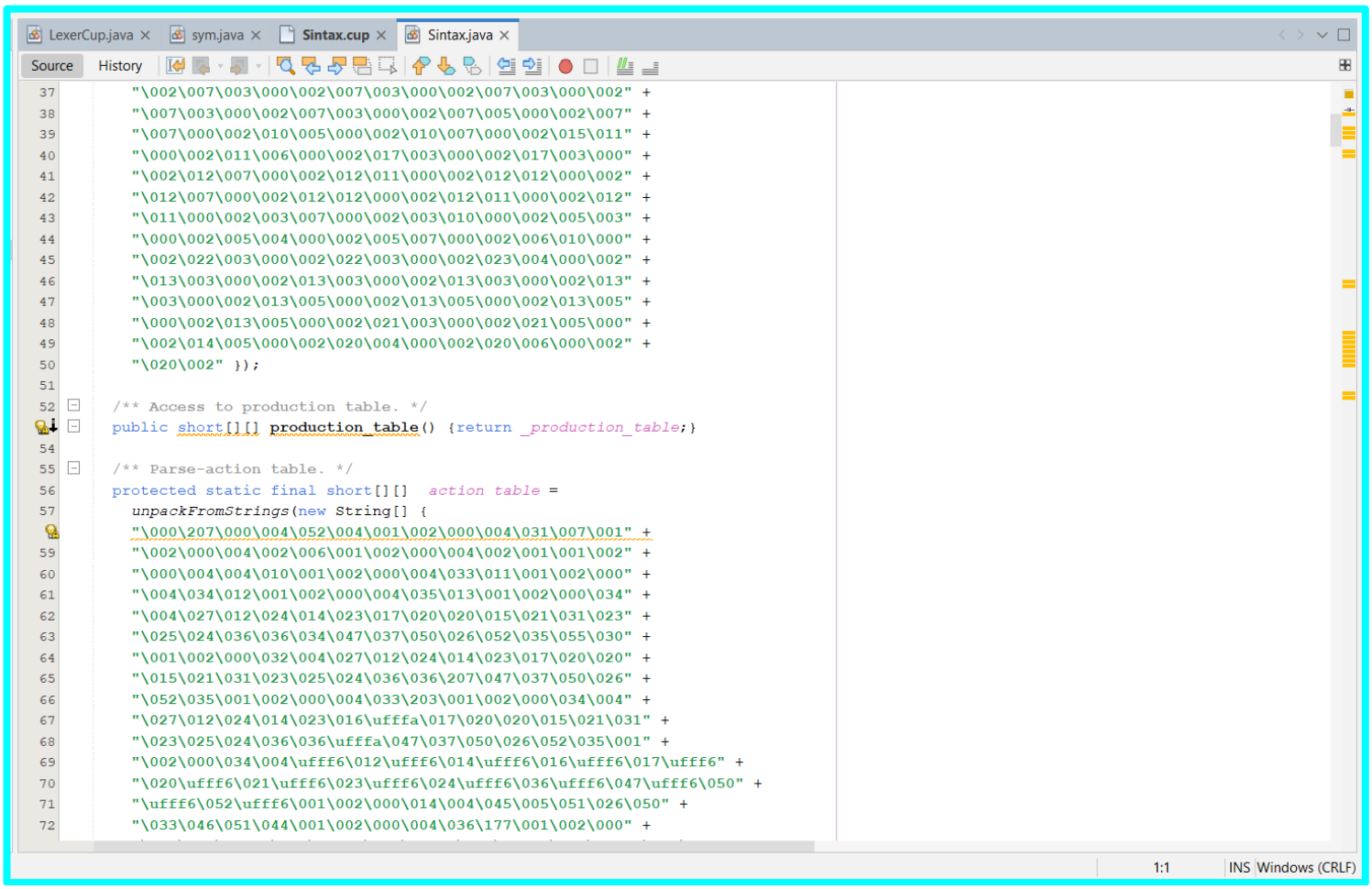
Source | History

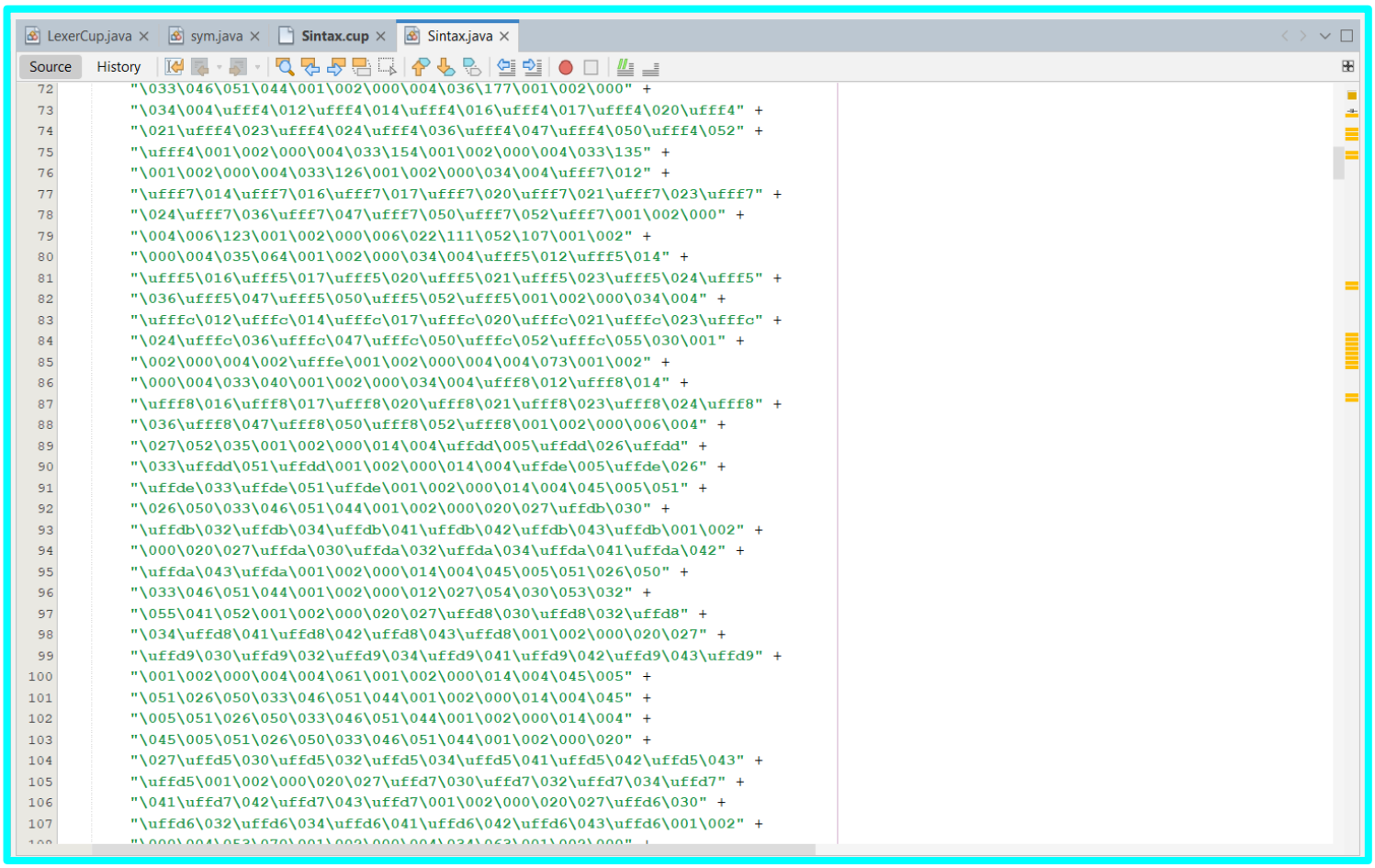
```
111 // Expresiones generales
112 EXPRESION ::= NUM
113              | ID
114              | CADENA
115              | BOOLEANO            // Se añade soporte para true y false
116              | EXPRESION OPARITMETICO EXPRESION
117              | EXPRESION OPRELACIONALES EXPRESION
118              | EXPRESION OPLOGICOS EXPRESION  // Aquí ya está cubierto && y ||
119              | PARENTESISIZQ EXPRESION PARENTESISDER
120 ;
121
122 // Lista de expresiones separadas por comas
123 LISTA_EXPRESIONES ::= EXPRESION
124                     | EXPRESION COMA LISTA_EXPRESIONES
125 ;
126
127 // Bloques de código (cuerpo de sentencias)
128 BLOQUE ::= LLAVEIZQ SENTENCIAS LLAVEDER
129 ;
130
131 // Parametros del método
132 PARAMETROS_METODO ::= TIPO_RETORNO ID
133                     | TIPO_RETORNO ID COMA PARAMETROS_METODO
134                     | /* vacío */
135 ;
136
```

Sintax.java

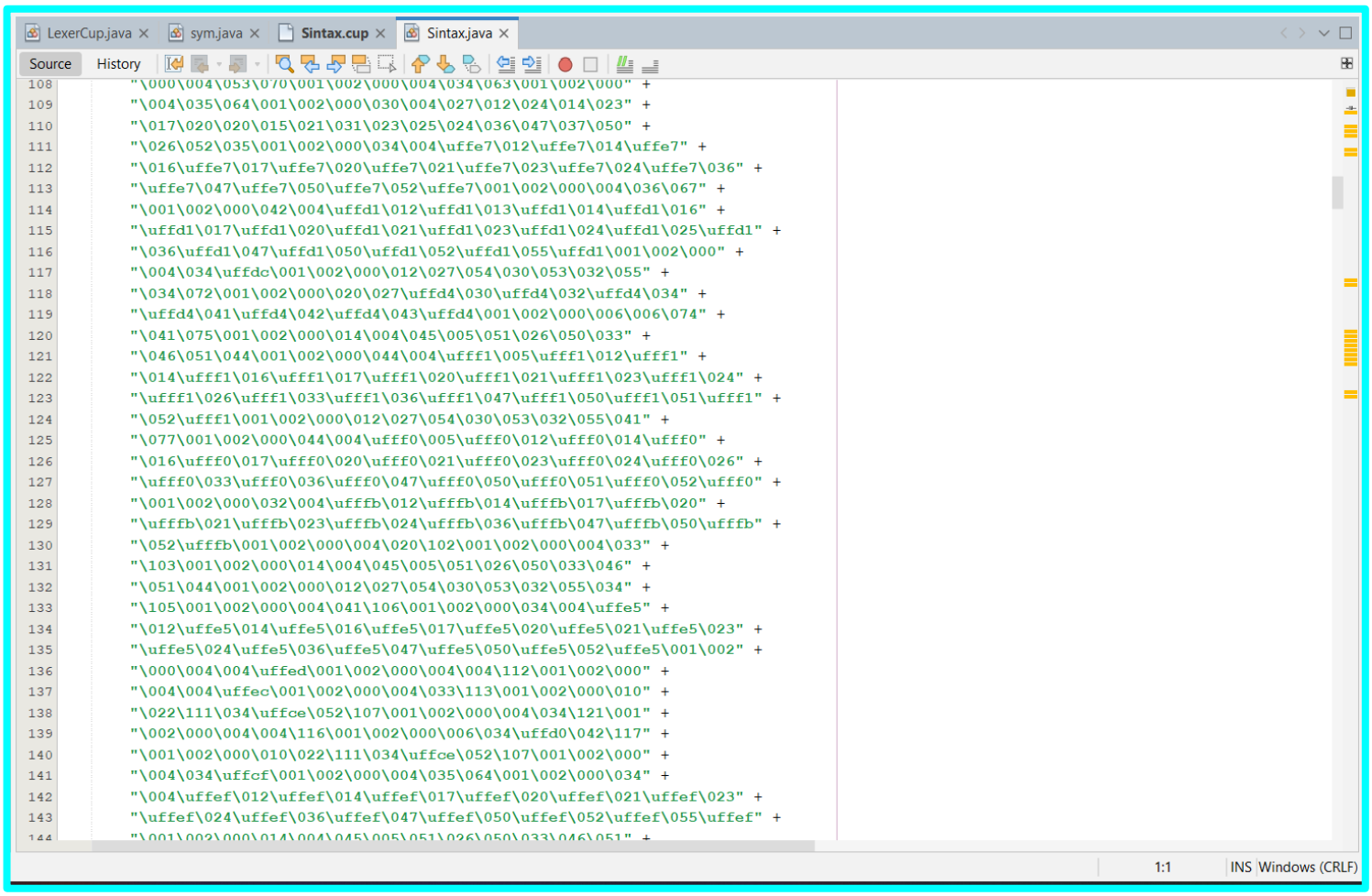
Este se genera mediante el Sintax.cup lo que este hace es reconocer las sentencias para que a la hora de colocar un código reconozca esa sintaxis o marque un error.

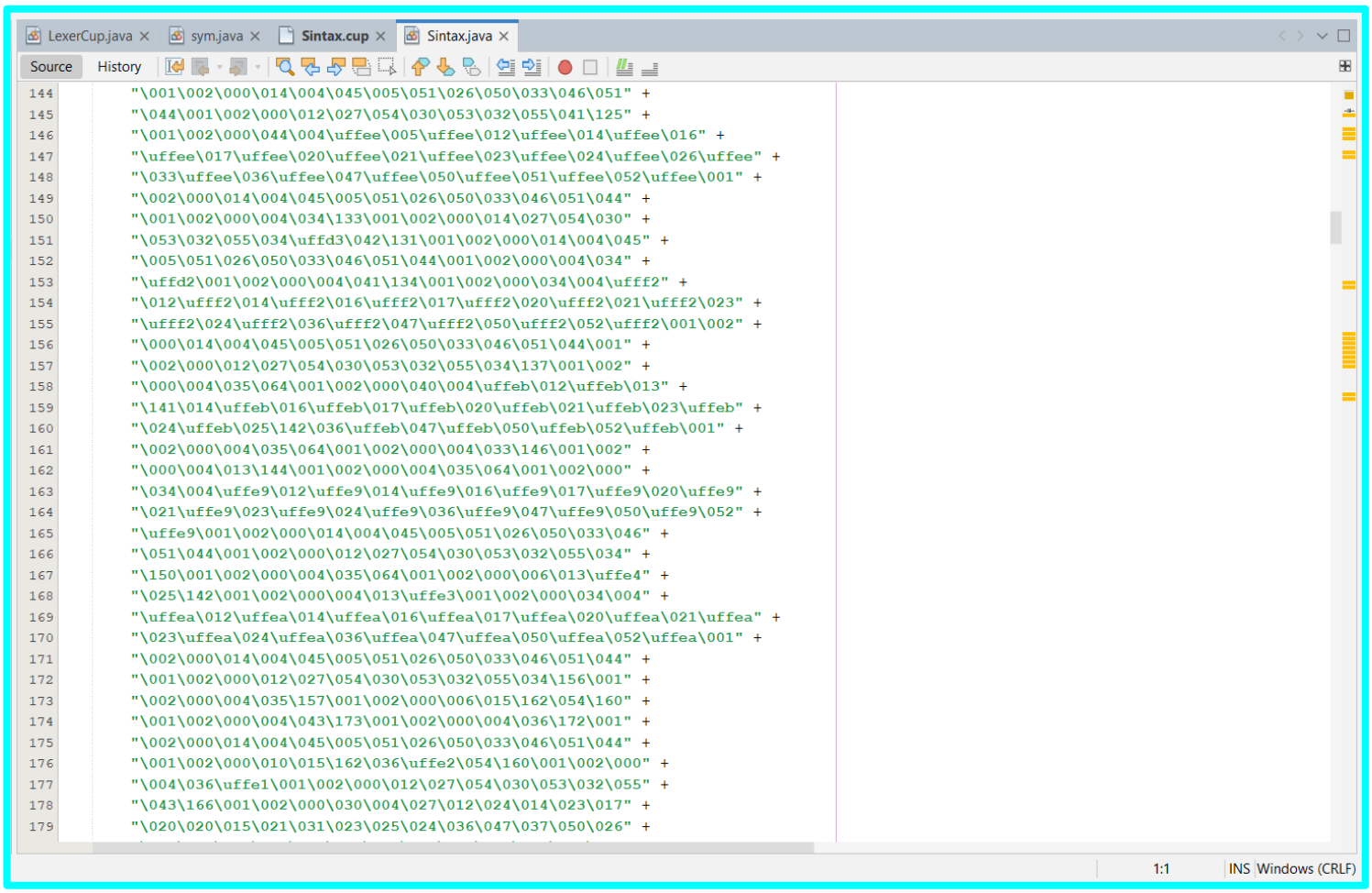


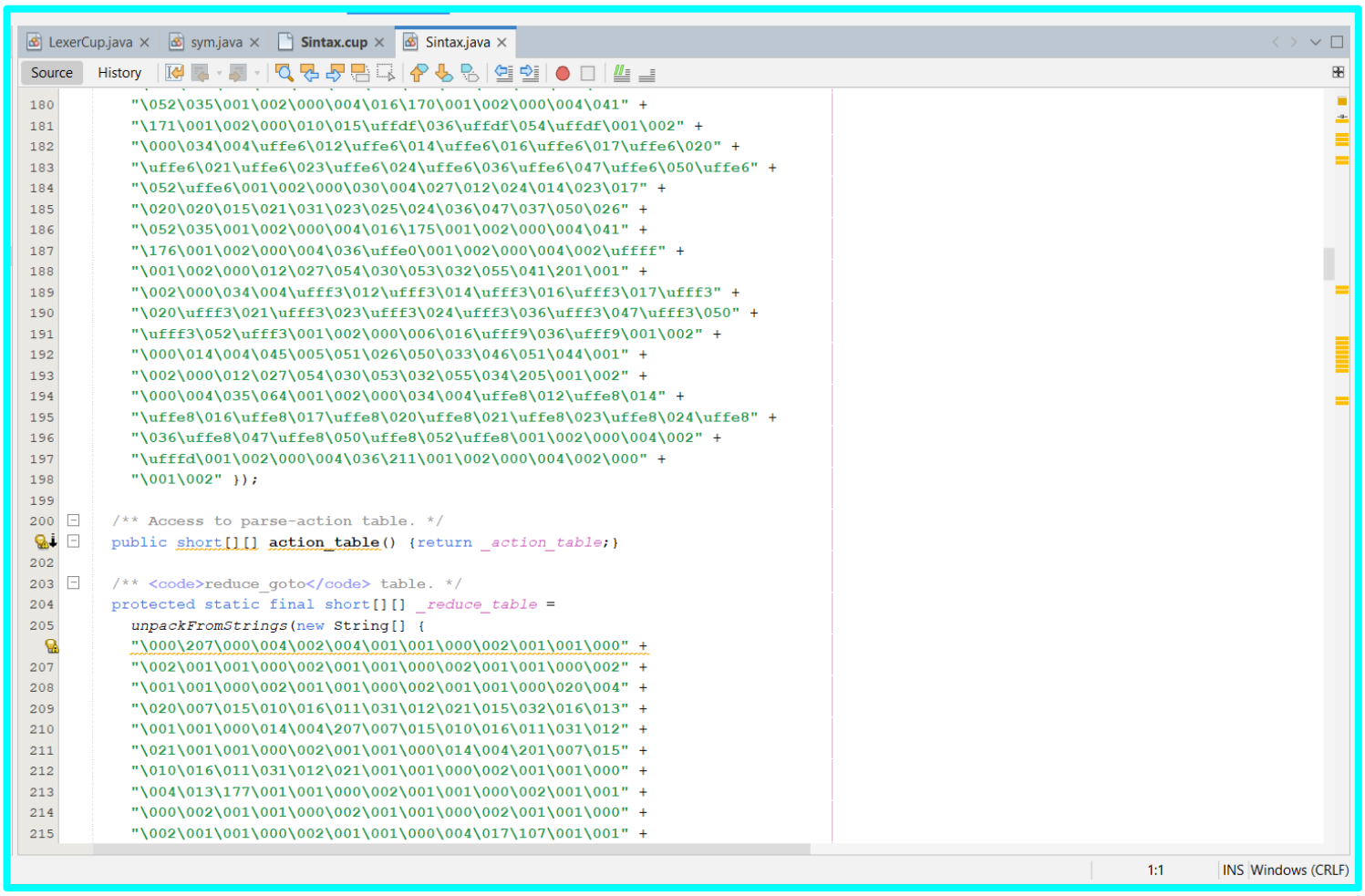


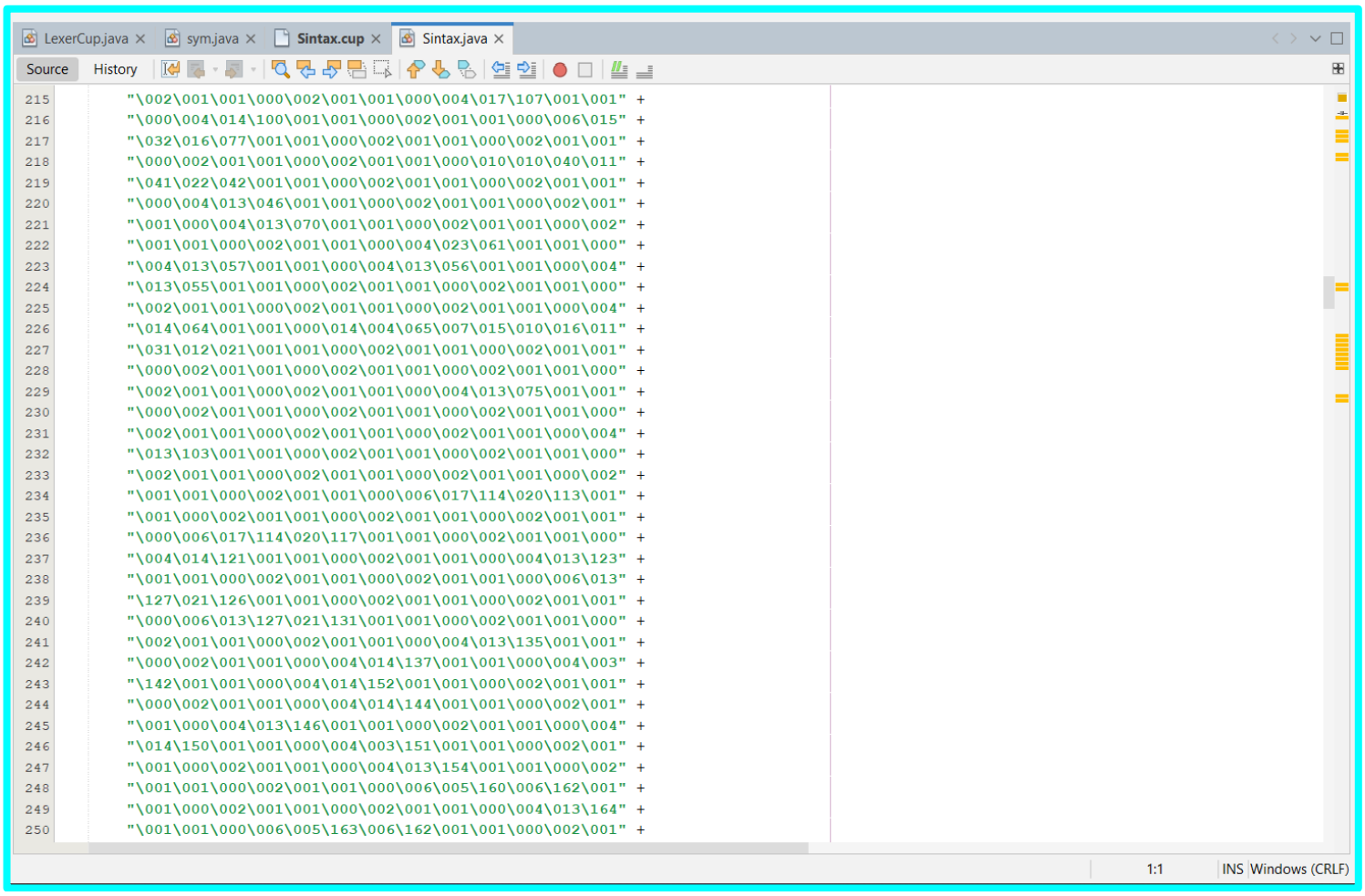


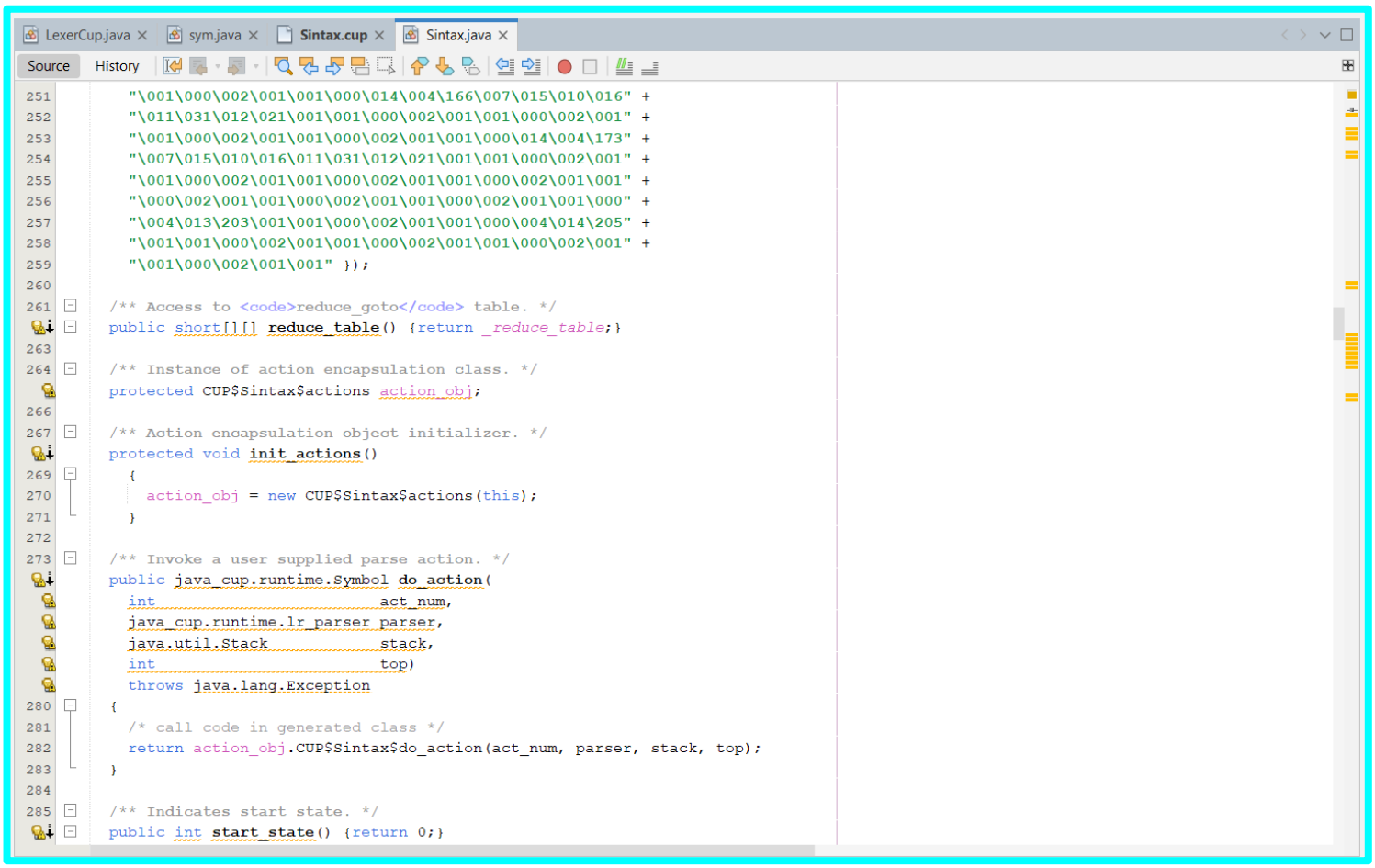

```
        "\033\046\051\044\001\002\000\004\036\177\001\002\000" +
        "\034\004\ufff4\012\ufff4\014\ufff4\016\ufff4\017\ufff4\020\ufff4" +
        "\021\ufff4\023\ufff4\024\ufff4\036\ufff4\047\ufff4\050\ufff4\052" +
        "\ufff4\001\002\000\004\033\154\001\002\000\004\033\135" +
        "\001\002\000\004\033\126\001\002\000\034\004\ufff7\012" +
        "\ufff7\014\ufff7\016\ufff7\017\ufff7\020\ufff7\021\ufff7\023\ufff7" +
        "\024\ufff7\036\ufff7\047\ufff7\050\ufff7\052\ufff7\001\002\000" +
        "\004\006\123\001\002\000\006\022\111\052\107\001\002" +
        "\000\004\035\064\001\002\000\034\004\ufff5\012\ufff5\014" +
        "\ufff5\016\ufff5\017\ufff5\020\ufff5\021\ufff5\023\ufff5\024\ufff5" +
        "\036\ufff5\047\ufff5\050\ufff5\052\ufff5\001\002\000\034\004" +
        "\ufffc\012\ufffc\014\ufffc\017\ufffc\020\ufffc\021\ufffc\023\ufffc" +
        "\024\ufffc\036\ufffc\047\ufffc\050\ufffc\052\ufffc\055\030\001" +
        "\002\000\004\002\ufffe\001\002\000\004\004\073\001\002" +
        "\000\004\033\040\001\002\000\034\004\ufff8\012\ufff8\014" +
        "\ufff8\016\ufff8\017\ufff8\020\ufff8\021\ufff8\023\ufff8\024\ufff8" +
        "\036\ufff8\047\ufff8\050\ufff8\052\ufff8\001\002\000\006\004" +
        "\027\052\035\001\002\000\014\004\uffdd\005\uffdd\026\uffdd" +
        "\033\uffdd\051\uffdd\001\002\000\014\004\uffde\005\uffde\026" +
        "\uffde\033\uffde\051\uffde\001\002\000\014\004\045\005\051" +
        "\026\050\033\046\051\044\001\002\000\020\027\uffdb\030" +
        "\uffdb\032\uffdb\034\uffdb\041\uffdb\042\uffdb\043\uffdb\001\002" +
        "\000\020\027\uffda\030\uffda\032\uffda\034\uffda\041\uffda\042" +
        "\uffda\043\uffda\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\052\001\002\000\020\027\uffd8\030\uffd8\032\uffd8" +
        "\034\uffd8\041\uffd8\042\uffd8\043\uffd8\001\002\000\020\027" +
        "\uffd9\030\uffd9\032\uffd9\034\uffd9\041\uffd9\042\uffd9\043\uffd9" +
        "\001\002\000\004\004\061\001\002\000\014\004\045\005" +
        "\051\026\050\033\046\051\044\001\002\000\014\004\045" +
        "\005\051\026\050\033\046\051\044\001\002\000\014\004" +
        "\045\005\051\026\050\033\046\051\044\001\002\000\020" +
        "\027\uffd5\030\uffd5\032\uffd5\034\uffd5\041\uffd5\042\uffd5\043" +
        "\uffd5\001\002\000\020\027\uffd7\030\uffd7\032\uffd7\034\uffd7" +
        "\041\uffd7\042\uffd7\043\uffd7\001\002\000\020\027\uffd6\030" +
        "\uffd6\032\uffd6\034\uffd6\041\uffd6\042\uffd6\043\uffd6\001\002" +
```

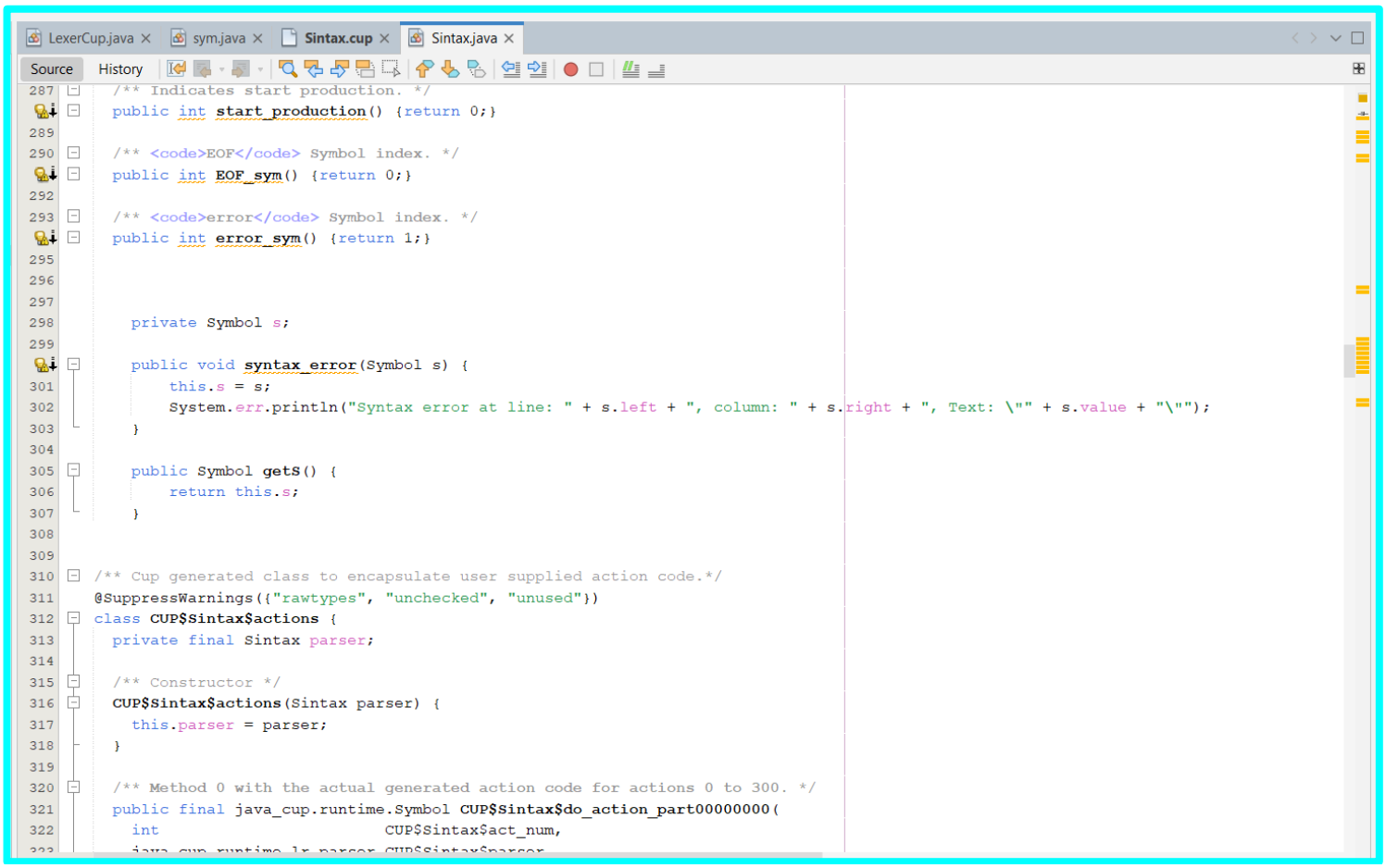


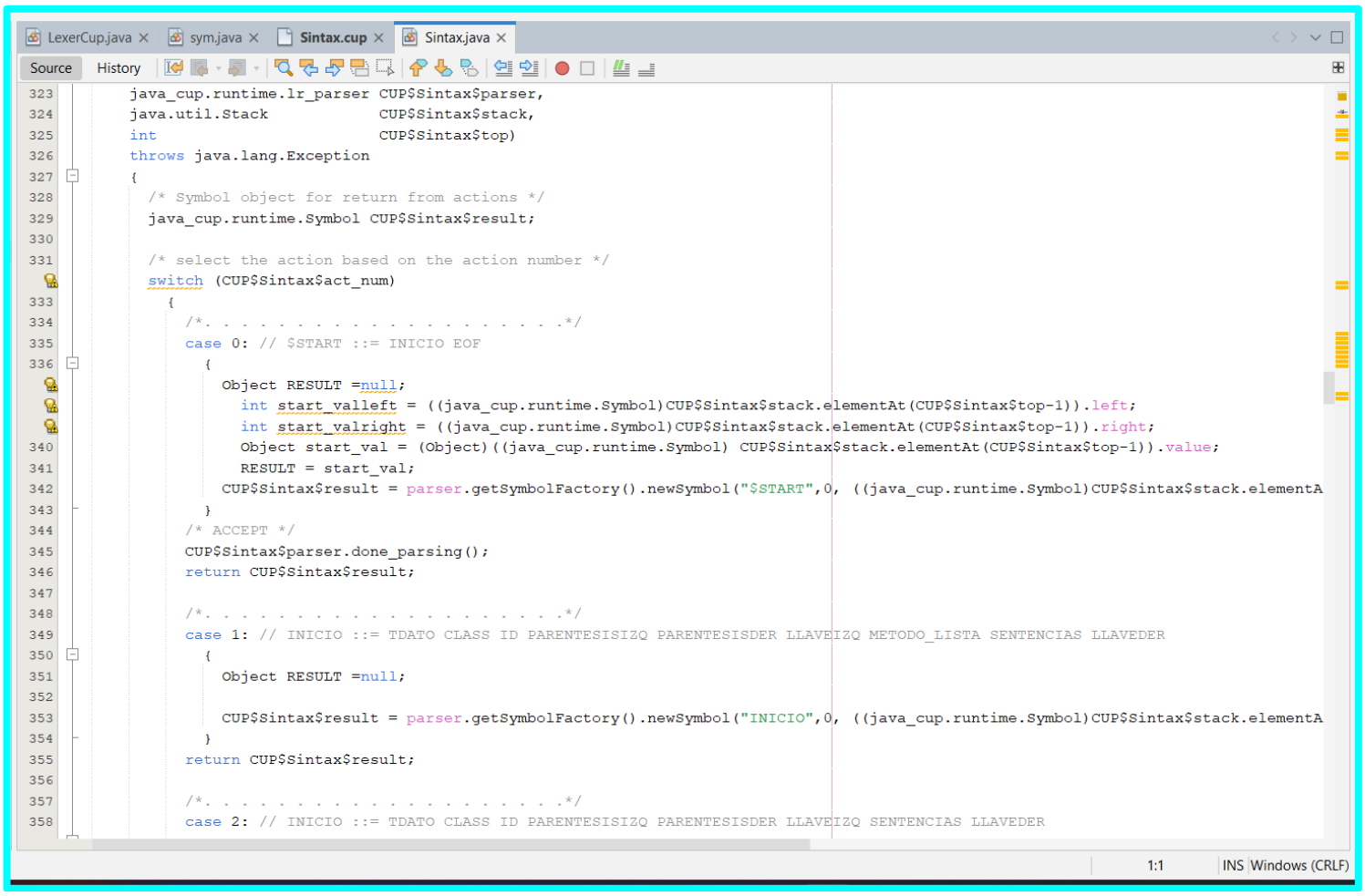


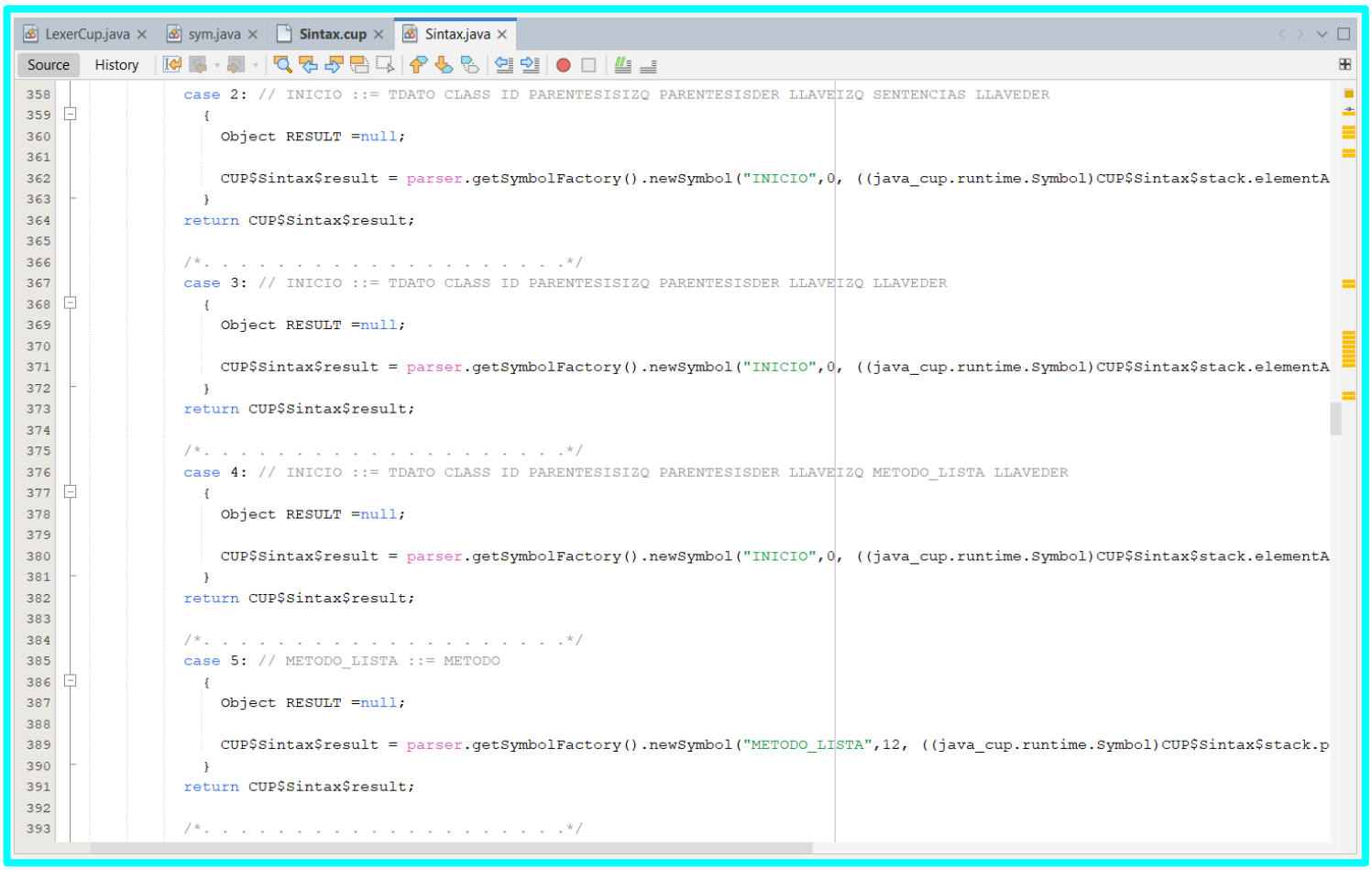

LexerCup.java
sym.java
Sintax.cup
Sintax.java
Source
History
"\052\035\001\002\000\004\016\170\001\002\000\004\041" +
"\171\001\002\000\010\015\uffdf\036\uffdf\054\uffdf\001\002" +
"\000\034\004\uffe6\012\uffe6\014\uffe6\016\uffe6\017\uffe6\020" +
"\uffe6\021\uffe6\023\uffe6\024\uffe6\036\uffe6\047\uffe6\050\uffe6" +
"\052\uffe6\001\002\000\030\004\027\012\024\014\023\017" +
"\020\020\015\021\031\023\025\024\036\047\037\050\026" +
"\052\035\001\002\000\004\016\175\001\002\000\004\041" +
"\176\001\002\000\004\036\uffe0\001\002\000\004\002\uffff" +
"\001\002\000\012\027\054\030\053\032\055\041\201\001" +
"\002\000\034\004\ufff3\012\ufff3\014\ufff3\016\ufff3\017\ufff3" +
"\020\ufff3\021\ufff3\023\ufff3\024\ufff3\036\ufff3\047\ufff3\050" +
"\ufff3\052\ufff3\001\002\000\006\016\ufff9\036\ufff9\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\205\001\002" +
"\000\004\035\064\001\002\000\034\004\uffe8\012\uffe8\014" +
"\uffe8\016\uffe8\017\uffe8\020\uffe8\021\uffe8\023\uffe8\024\uffe8" +
"\036\uffe8\047\uffe8\050\uffe8\052\uffe8\001\002\000\004\002" +
"\ufffd\001\002\000\004\036\211\001\002\000\004\002\000" +
"\001\002" });
/** Access to parse-action table. */
public short[][] action_table() {return _action_table;}
/** <code>reduce_goto</code> table. */
protected static final short[][] _reduce_table =
unpackFromStrings(new String[] {
"\000\207\000\004\002\004\001\001\000\002\001\001\000" +
"\002\001\001\000\002\001\001\000\002\001\001\000\002" +
"\001\001\000\002\001\001\000\002\001\001\000\020\004" +
"\020\007\015\010\016\011\031\012\021\015\032\016\013" +
"\001\001\000\014\004\207\007\015\010\016\011\031\012" +
"\021\001\001\000\002\001\001\000\014\004\201\007\015" +
"\010\016\011\031\012\021\001\001\000\002\001\001\000" +
"\004\013\177\001\001\000\002\001\001\000\002\001\001" +
"\000\002\001\001\000\002\001\001\000\002\001\001\000" +
"\002\001\001\000\002\001\001\000\004\017\107\001\001" +
1:1
INS
Windows (CRLF)



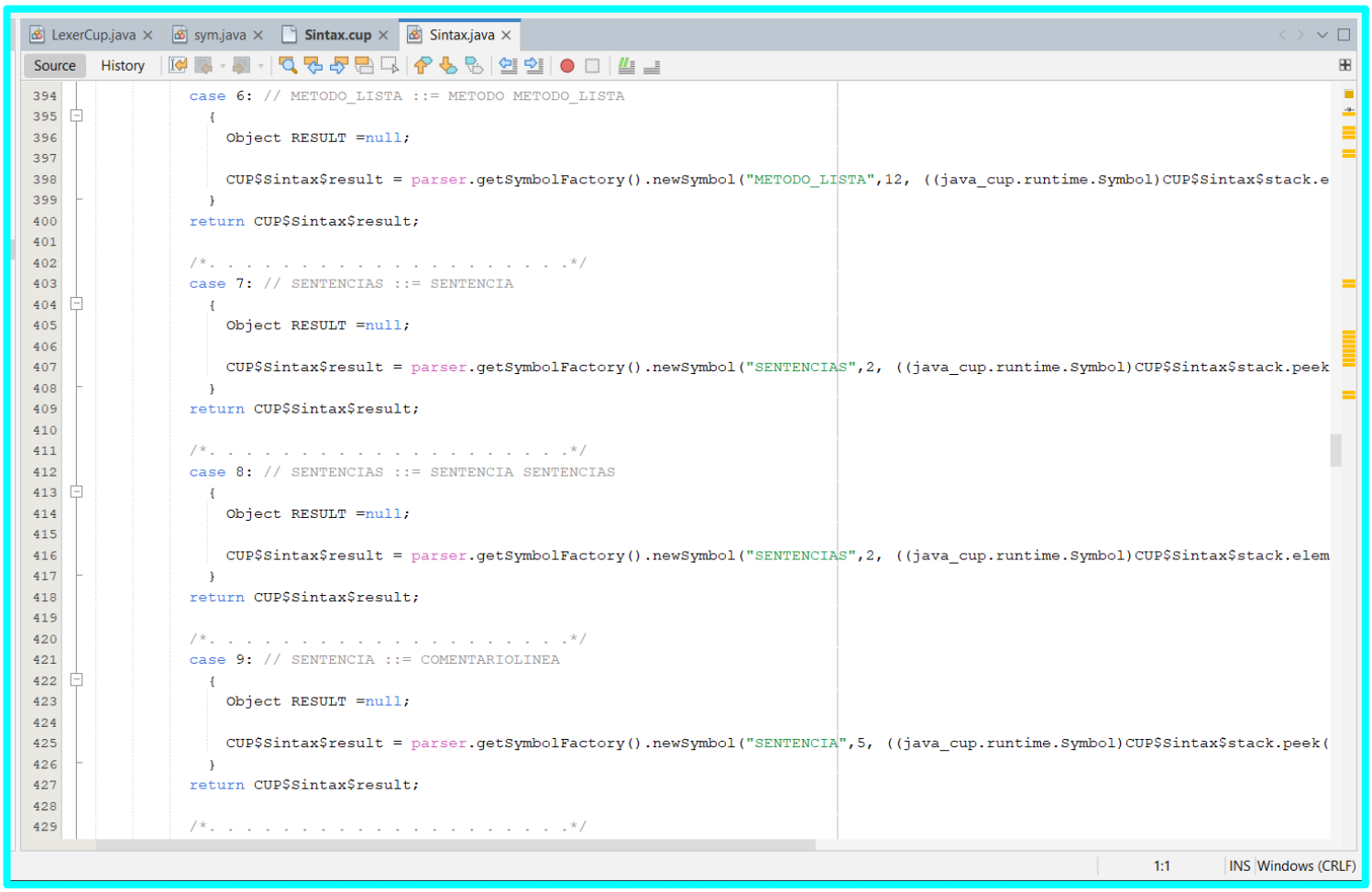

LexerCup.java
sym.java
Sintax.cup
Sintax.java
Source
History
"\001\000\002\001\001\000\014\004\166\007\015\010\016" +
"\011\031\012\021\001\001\000\002\001\001\000\002\001" +
"\001\000\002\001\001\000\002\001\001\000\014\004\173" +
"\007\015\010\016\011\031\012\021\001\001\000\002\001" +
"\001\000\002\001\001\000\002\001\001\000\002\001\001" +
"\000\002\001\001\000\002\001\001\000\002\001\001\000" +
"\004\013\203\001\001\000\002\001\001\000\004\014\205" +
"\001\001\000\002\001\001\000\002\001\001\000\002\001" +
"\001\000\002\001\001" });
/** Access to <code>reduce_goto</code> table. */
public short[][] reduce_table() {return _reduce_table;}
/** Instance of action encapsulation class. */
protected CUP$Sintax$actions action_obj;
/** Action encapsulation object initializer. */
protected void init_actions()
{
action_obj = new CUP$Sintax$actions(this);
}
/** Invoke a user supplied parse action. */
public java_cup.runtime.Symbol do_action(
int act_num,
java_cup.runtime.lr_parser parser,
java.util.Stack stack,
int top)
throws java.lang.Exception
{
/* call code in generated class */
return action_obj.CUP$Sintax$do_action(act_num, parser, stack, top);
}
/** Indicates start state. */
public int start_state() {return 0;}

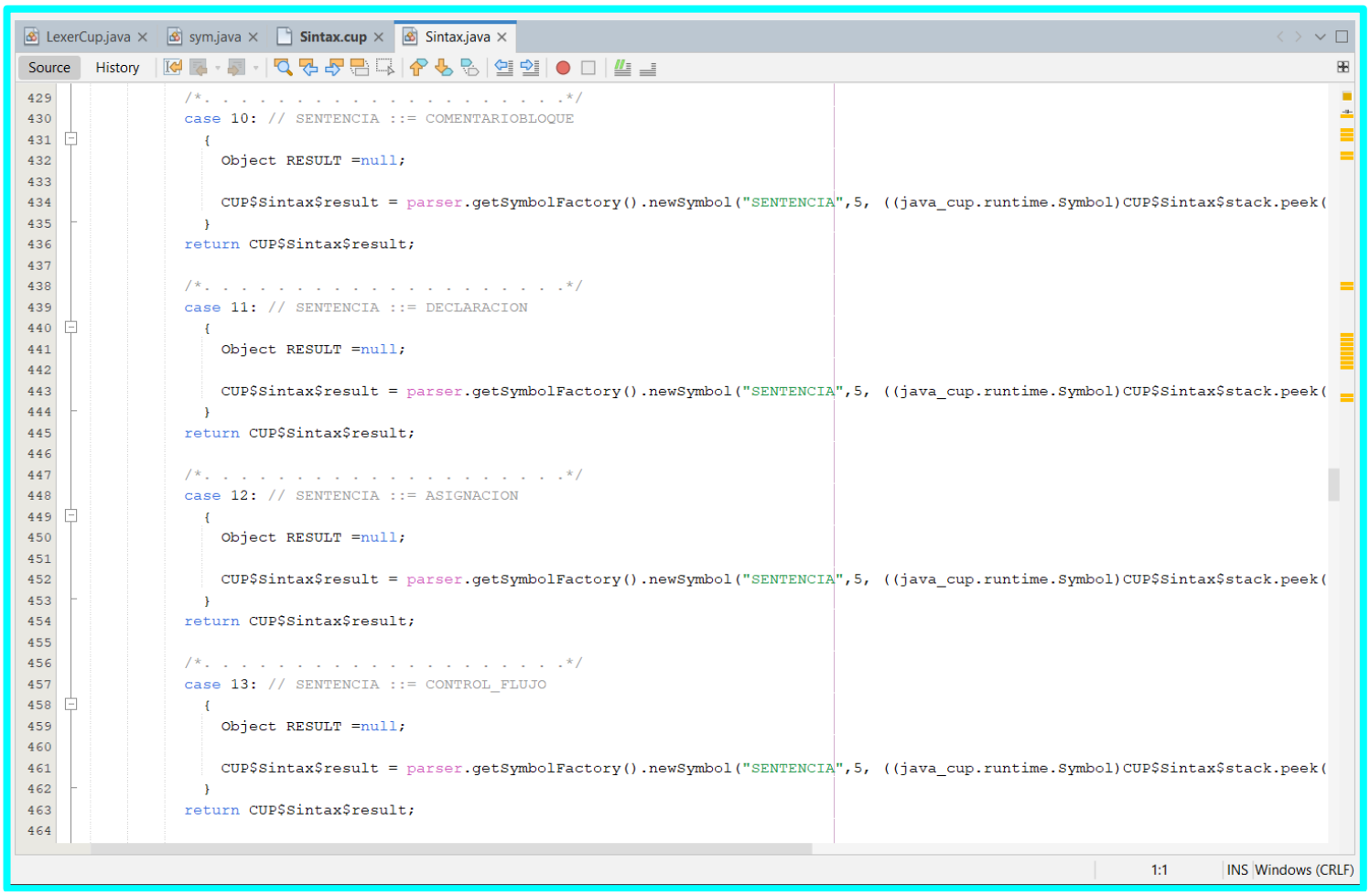

LexerCup.java
sym.java
Sintax.cup
Sintax.java
Source
History
/** Indicates start production. */
public int start_production() {return 0;}
/** <code>EOF</code> Symbol index. */
public int EOF_sym() {return 0;}
/** <code>error</code> Symbol index. */
public int error_sym() {return 1;}
private Symbol s;
public void syntax_error(Symbol s) {
this.s = s;
System.err.println("Syntax error at line: " + s.left + ", column: " + s.right + ", Text: \"" + s.value + "\"");
}
public Symbol getS() {
return this.s;
}
/** Cup generated class to encapsulate user supplied action code.*/
@SuppressWarnings({"rawtypes", "unchecked", "unused"})
class CUP$Sintax$actions {
private final Sintax parser;
/** Constructor */
CUP$Sintax$actions(Sintax parser) {
this.parser = parser;
}
/** Method 0 with the actual generated action code for actions 0 to 300. */
public final java_cup.runtime.Symbol CUP$Sintax$do_action_part00000000(
int CUP$Sintax$act_num,

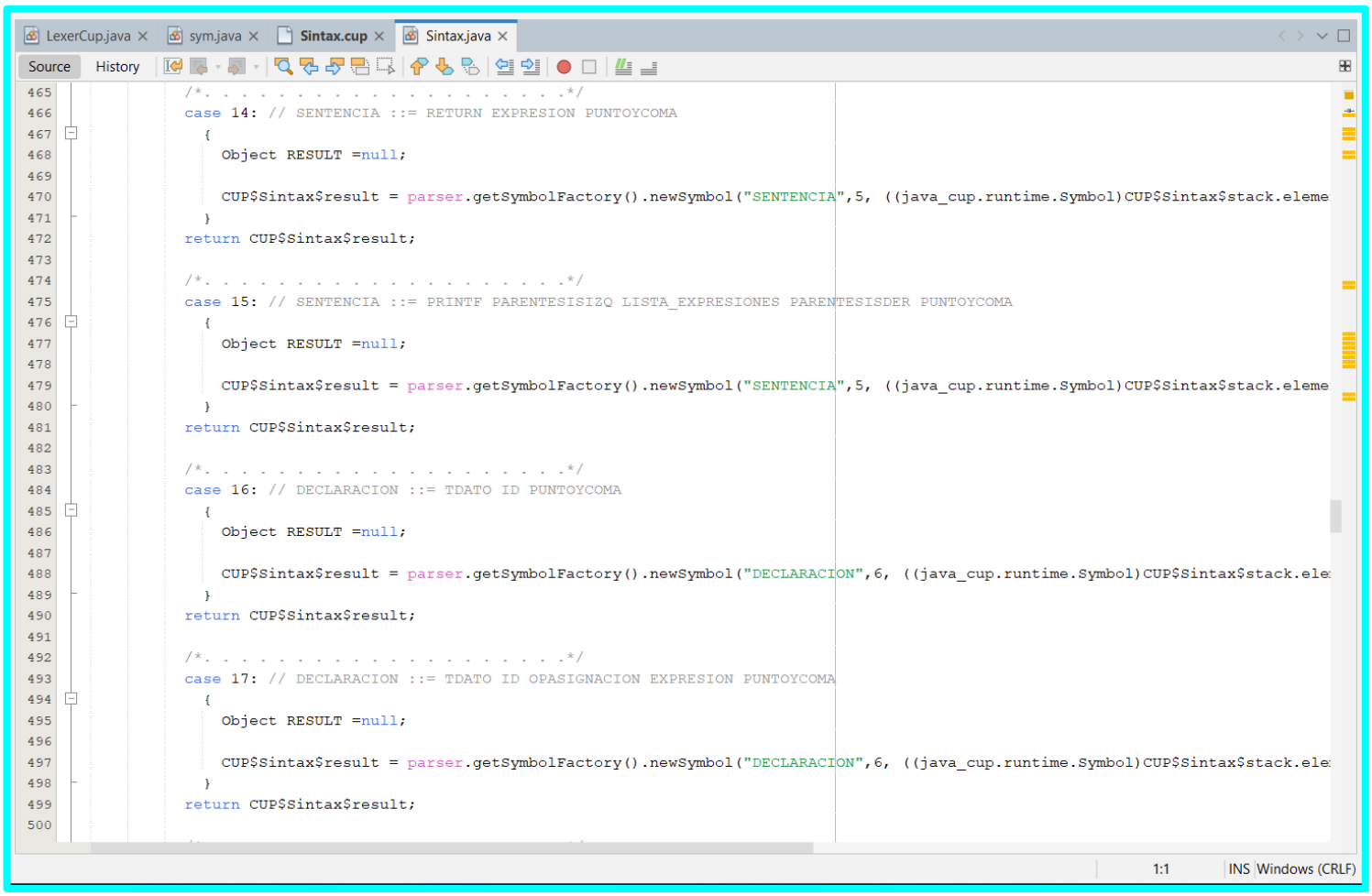

LexerCup.java
sym.java
Sintax.cup
Sintax.java
Source
History
java_cup.runtime.lr_parser CUP$Sintax$parser,
java.util.Stack CUP$Sintax$stack,
int CUP$Sintax$top)
throws java.lang.Exception
{
/* Symbol object for return from actions */
java_cup.runtime.Symbol CUP$Sintax$result;
/* select the action based on the action number */
switch (CUP$Sintax$act_num)
{
/*. . . . . .*/
case 0: // $START ::= INICIO EOF
{
Object RESULT =null;
int start_valleft = ((java_cup.runtime.Symbol)CUP$Sintax$stack.elementAt(CUP$Sintax$top-1)).left;
int start_valright = ((java_cup.runtime.Symbol)CUP$Sintax$stack.elementAt(CUP$Sintax$top-1)).right;
Object start_val = (Object)((java_cup.runtime.Symbol) CUP$Sintax$stack.elementAt(CUP$Sintax$top-1)).value;
RESULT = start_val;
CUP$Sintax$result = parser.getSymbolFactory().newSymbol("$START",0, ((java_cup.runtime.Symbol)CUP$Sintax$stack.elementA
}
/* ACCEPT */
CUP$Sintax$parser.done_parsing();
return CUP$Sintax$result;
/*. . . . . .*/
case 1: // INICIO ::= TDATO CLASS ID PARENTESISIZQ PARENTESISDER LLAVEIZQ METODO_LISTA SENTENCIAS LLAVEDER
{
Object RESULT =null;
CUP$Sintax$result = parser.getSymbolFactory().newSymbol("INICIO",0, ((java_cup.runtime.Symbol)CUP$Sintax$stack.elementA
}
return CUP$Sintax$result;
/*. . . . . .*/
case 2: // INICIO ::= TDATO CLASS ID PARENTESISIZQ PARENTESISDER LLAVEIZQ SENTENCIAS LLAVEDER
1:1
INS Windows (CRLF)

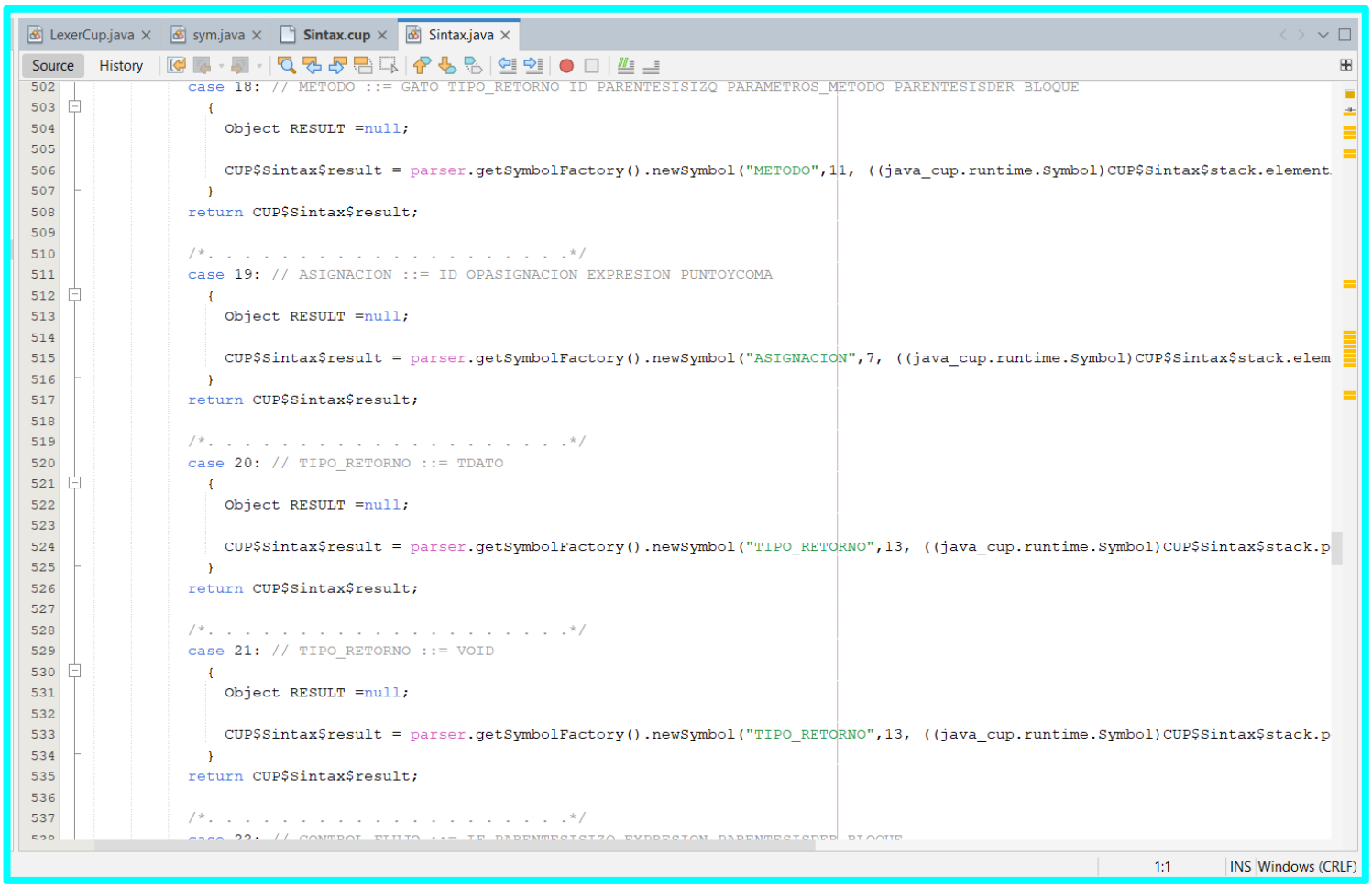


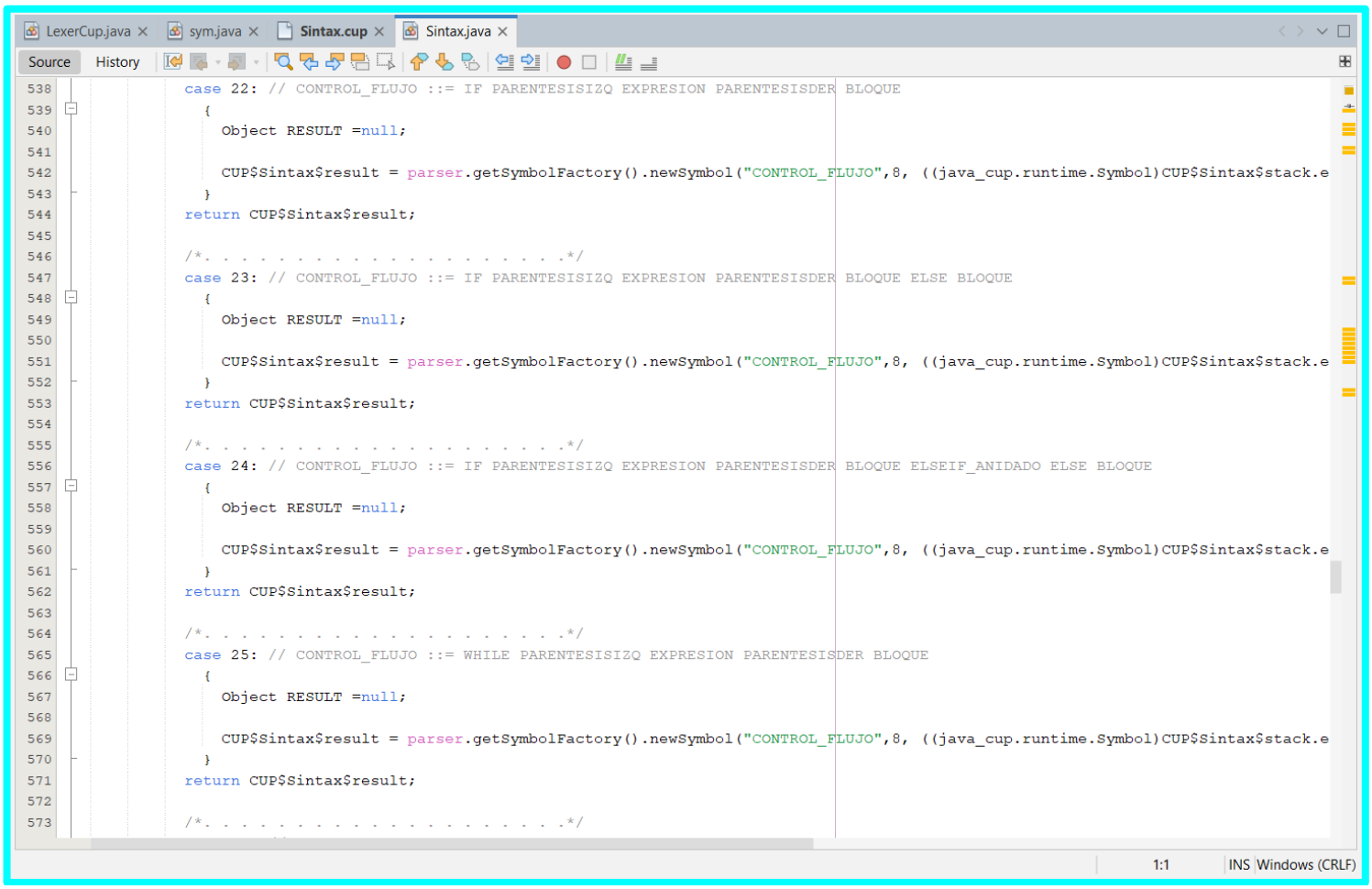

```
358             case 2: // INICIO ::= TDATO CLASS ID PARENTESISIZQ PARENTESISDER LLAVEIZQ SENTENCIAS LLAVEDER
359               {
360                 Object RESULT =null;
361
362                 CUP$Sintax$result = parser.getSymbolFactory().newSymbol("INICIO",0, ((java_cup.runtime.Symbol)CUP$Sintax$stack.elementA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

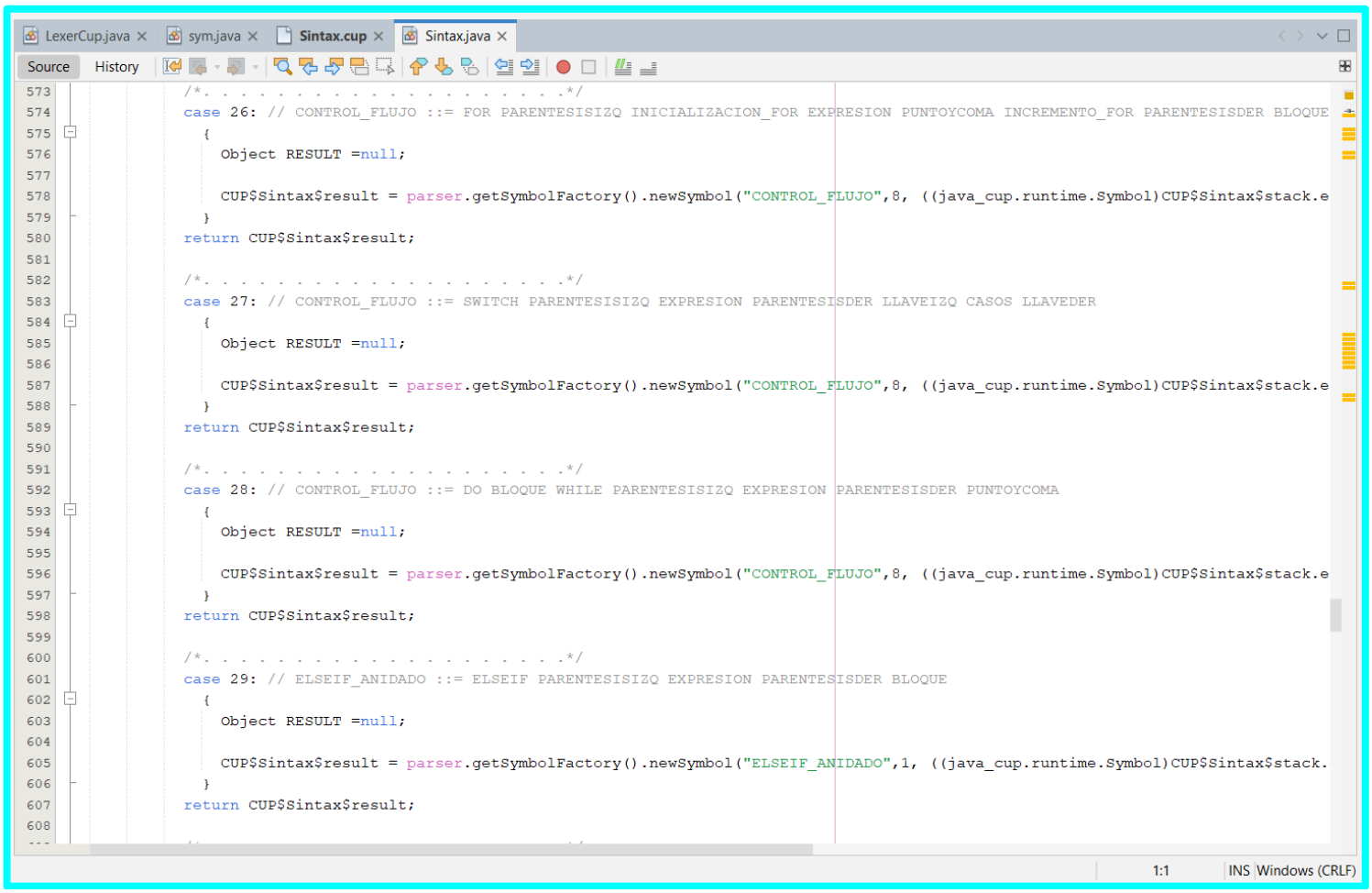


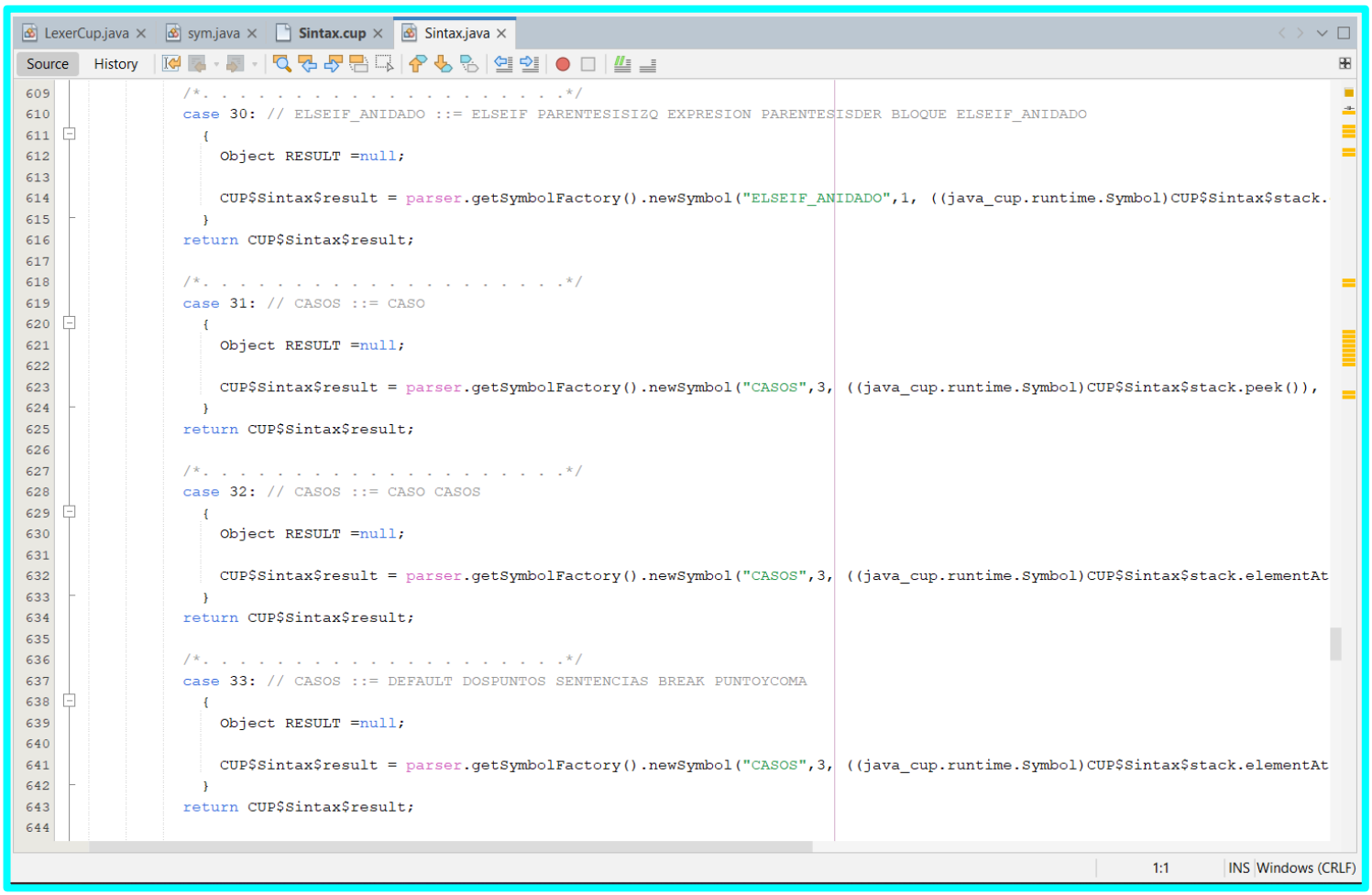


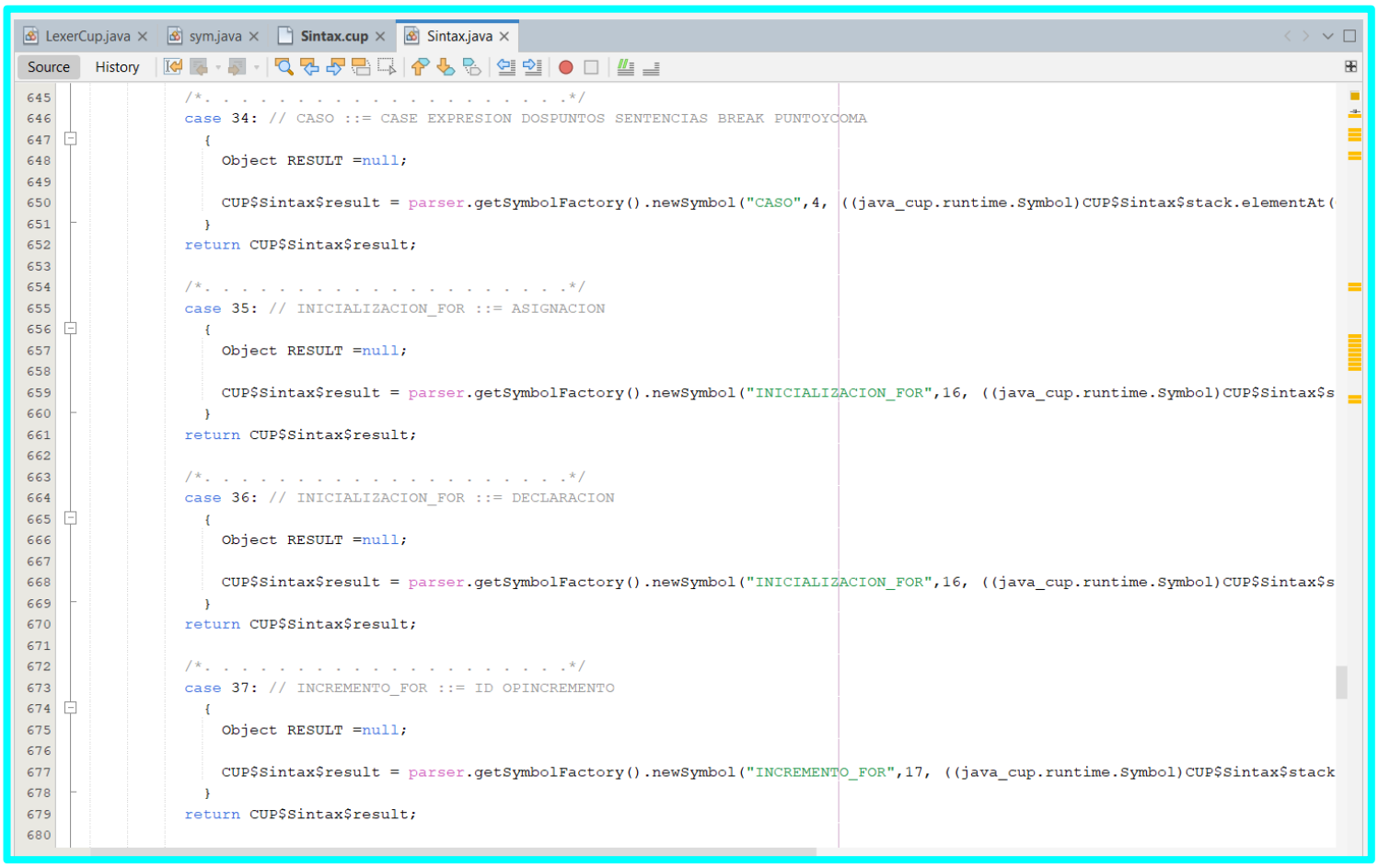



LexerCup.java
sym.java
Sintax.cup
Sintax.java
Source
History
case 18: // METODO ::= GATO TIPO_RETORNO ID PARENTESISIZQ PARAMETROS_METODO PARENTESISDER BLOQUE
{
Object RESULT =null;
CUP$Sintax$result = parser.getSymbolFactory().newSymbol("METODO",11, ((java_cup.runtime.Symbol)CUP$Sintax$stack.element
}
return CUP$Sintax$result;
/*. . . . . .*/
case 19: // ASIGNACION ::= ID OPASIGNACION EXPRESION PUNTOYCOMA
{
Object RESULT =null;
CUP$Sintax$result = parser.getSymbolFactory().newSymbol("ASIGNACION",7, ((java_cup.runtime.Symbol)CUP$Sintax$stack.elem
}
return CUP$Sintax$result;
/*. . . . . .*/
case 20: // TIPO_RETORNO ::= TDATO
{
Object RESULT =null;
CUP$Sintax$result = parser.getSymbolFactory().newSymbol("TIPO_RETORNO",13, ((java_cup.runtime.Symbol)CUP$Sintax$stack.p
}
return CUP$Sintax$result;
/*. . . . . .*/
case 21: // TIPO_RETORNO ::= VOID
{
Object RESULT =null;
CUP$Sintax$result = parser.getSymbolFactory().newSymbol("TIPO_RETORNO",13, ((java_cup.runtime.Symbol)CUP$Sintax$stack.p
}
return CUP$Sintax$result;
/*. . . . . .*/
1:1
INS
Windows (CRLF)

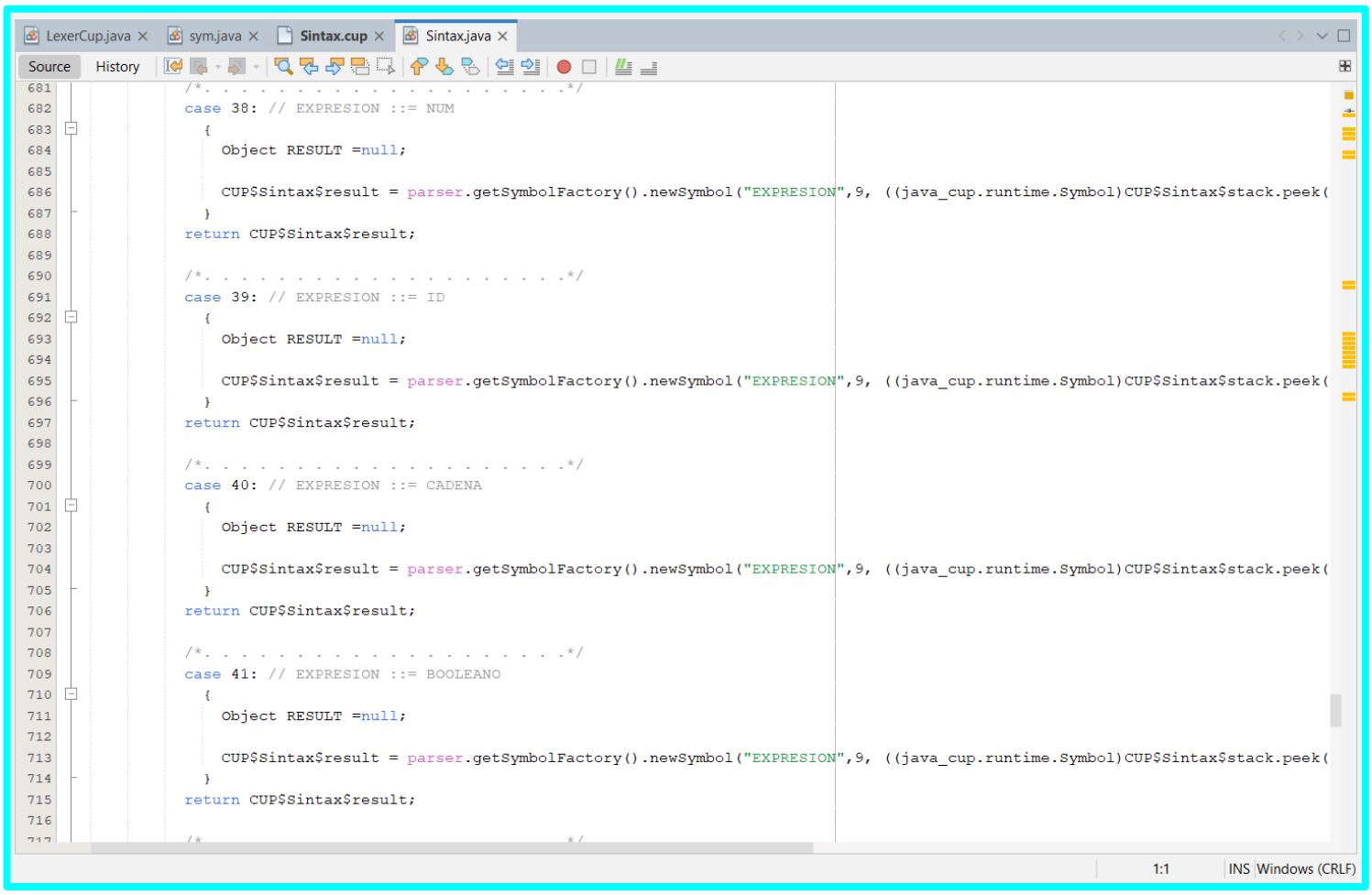


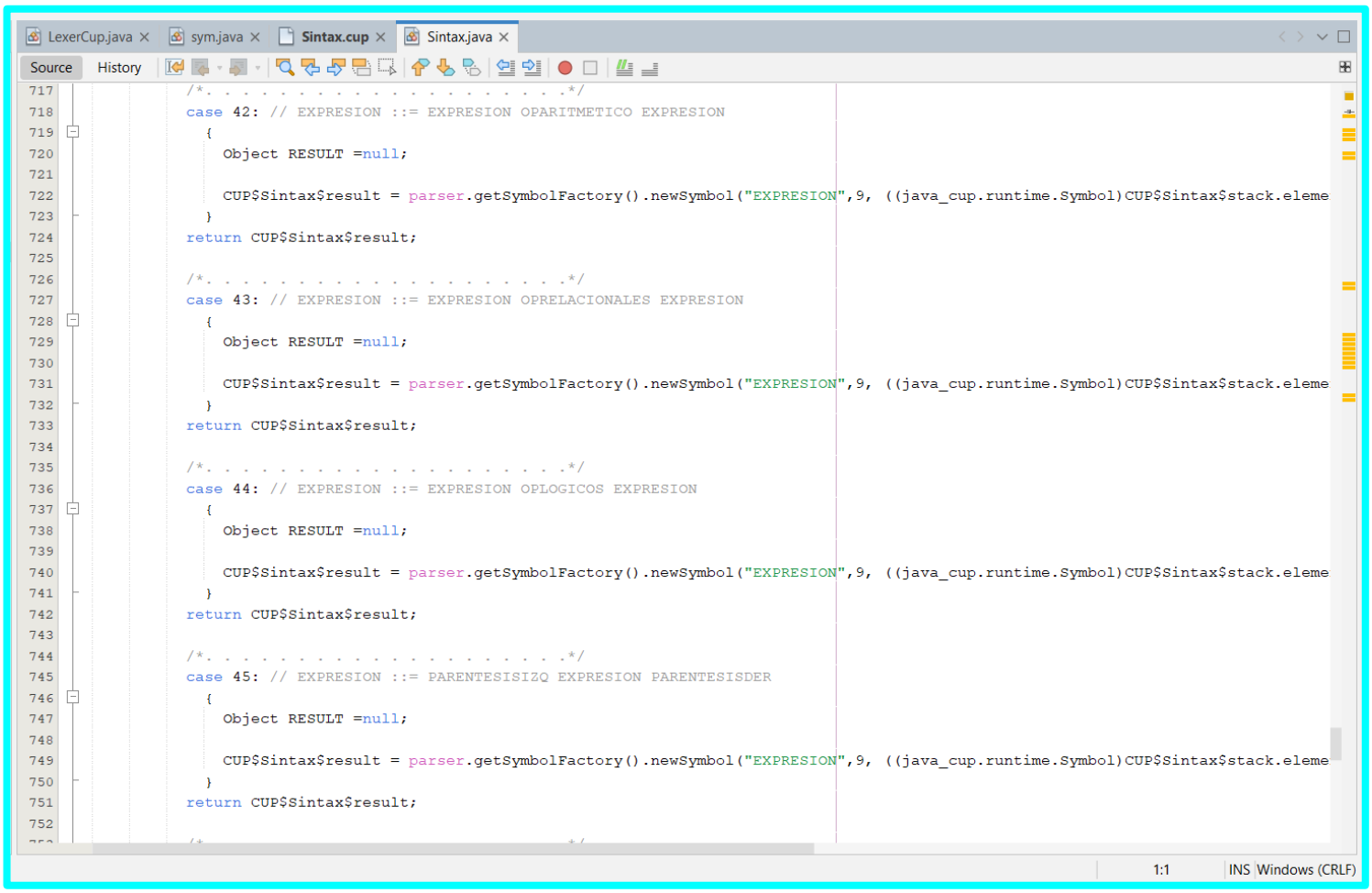


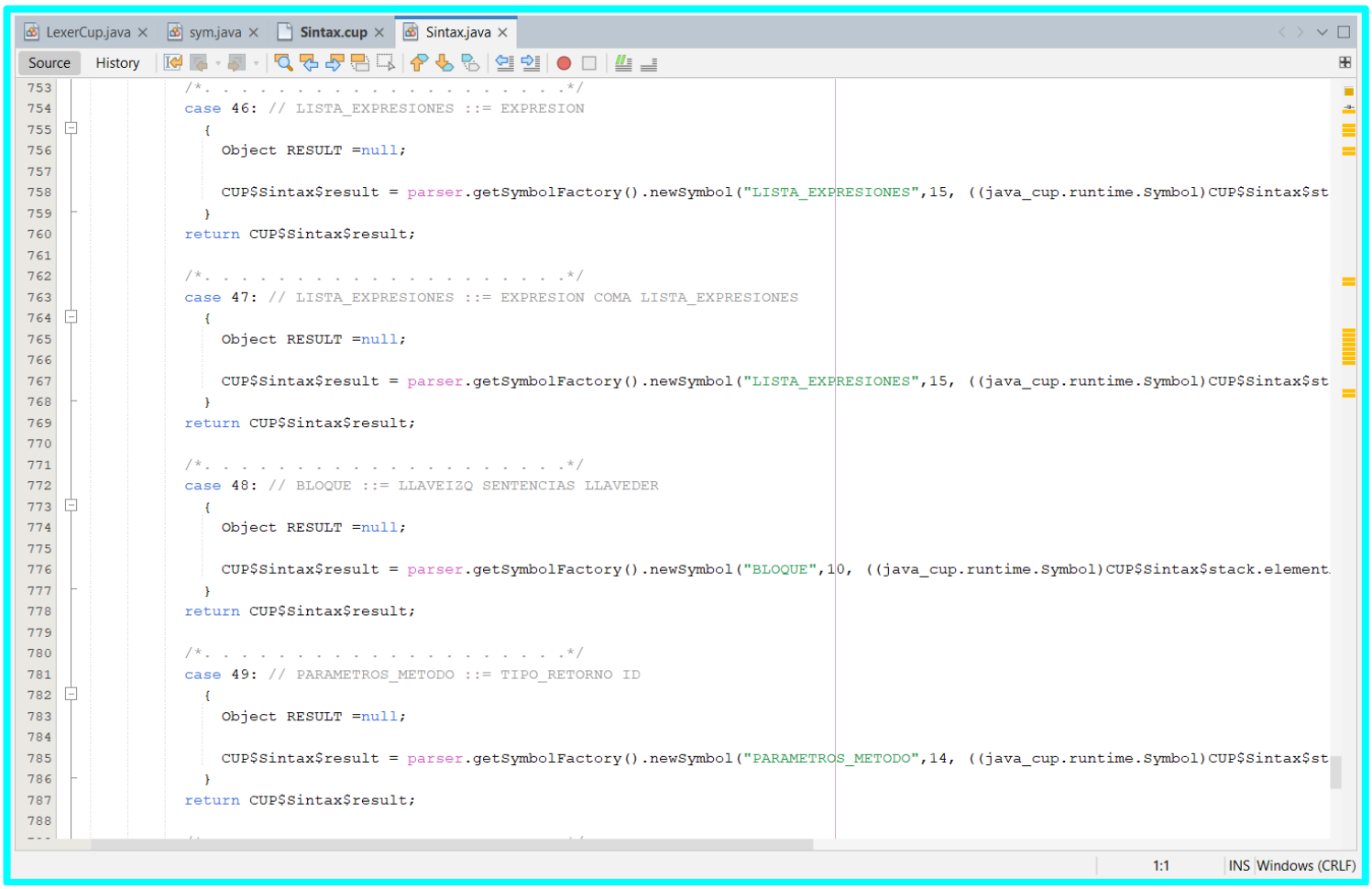

LexerCup.java
sym.java
Sintax.cup
Sintax.java
Source
History
/*. . . . . .*/
case 30: // ELSEIF_ANIDADO ::= ELSEIF PARENTESISIZQ EXPRESION PARENTESISDER BLOQUE ELSEIF_ANIDADO
{
Object RESULT =null;
CUP$Sintax$result = parser.getSymbolFactory().newSymbol("ELSEIF_ANIDADO",1, ((java_cup.runtime.Symbol)CUP$Sintax$stack.
}
return CUP$Sintax$result;
/*. . . . . .*/
case 31: // CASOS ::= CASO
{
Object RESULT =null;
CUP$Sintax$result = parser.getSymbolFactory().newSymbol("CASOS",3, ((java_cup.runtime.Symbol)CUP$Sintax$stack.peek()),
}
return CUP$Sintax$result;
/*. . . . . .*/
case 32: // CASOS ::= CASO CASOS
{
Object RESULT =null;
CUP$Sintax$result = parser.getSymbolFactory().newSymbol("CASOS",3, ((java_cup.runtime.Symbol)CUP$Sintax$stack.elementAt
}
return CUP$Sintax$result;
/*. . . . . .*/
case 33: // CASOS ::= DEFAULT DOSPUNTOS SENTENCIAS BREAK PUNTOYCOMA
{
Object RESULT =null;
CUP$Sintax$result = parser.getSymbolFactory().newSymbol("CASOS",3, ((java_cup.runtime.Symbol)CUP$Sintax$stack.elementAt
}
return CUP$Sintax$result;
1:1
INS
Windows (CRLF)

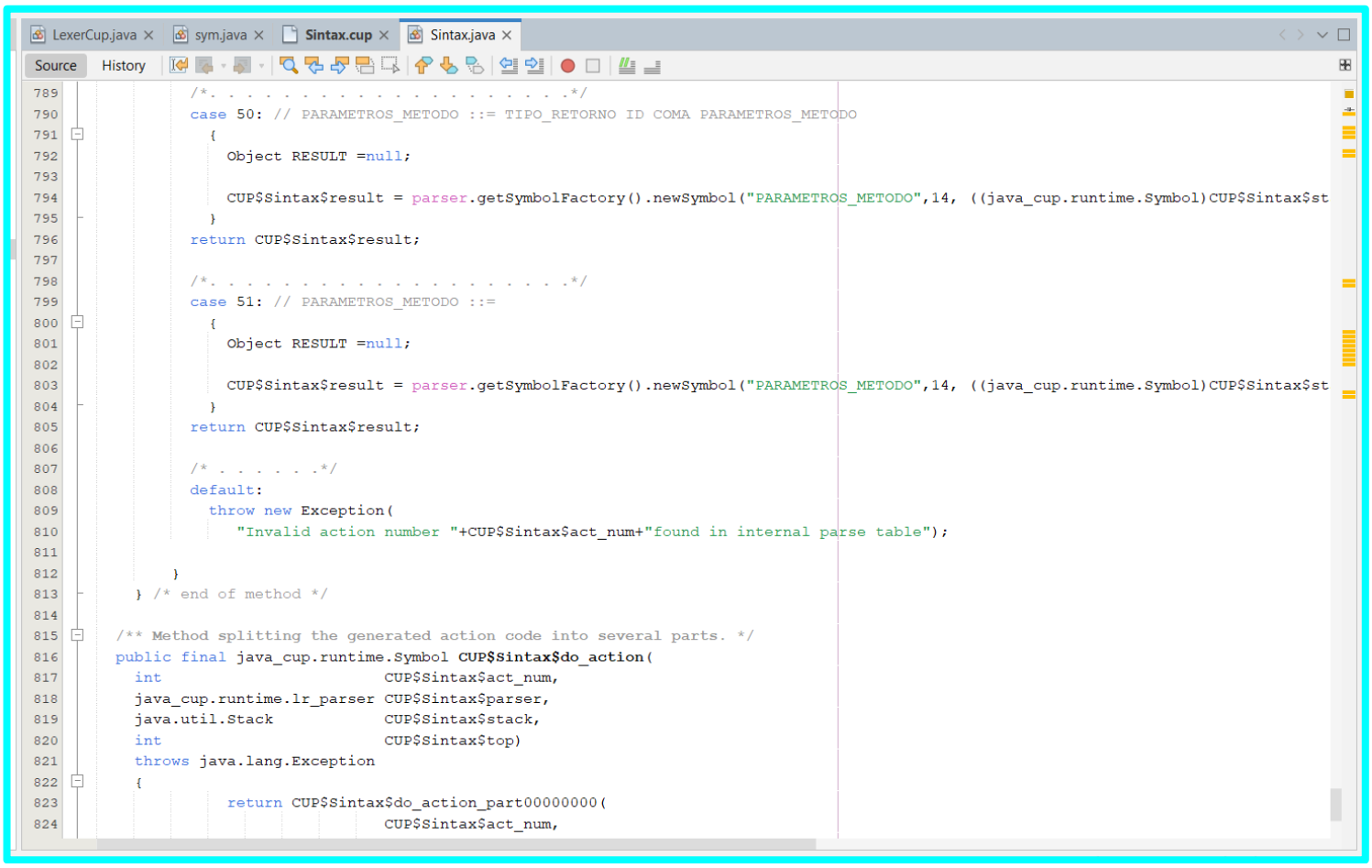



```
645             /*. . . . . .*/
646             case 34: // CASO ::= CASE EXPRESION DOSPUNTOS SENTENCIAS BREAK PUNTOYCOMA
647               {
648                 Object RESULT =null;
649
650                 CUP$Sintax$result = parser.getSymbolFactory().newSymbol("CASO",4, ((java_cup.runtime.Symbol)CUP$Sintax$stack.elementAt(
651               }
652             return CUP$Sintax$result;
653
654             /*. . . . . .*/
655             case 35: // INICIALIZACION_FOR ::= ASIGNACION
656               {
657                 Object RESULT =null;
658
659                 CUP$Sintax$result = parser.getSymbolFactory().newSymbol("INICIALIZACION_FOR",16, ((java_cup.runtime.Symbol)CUP$Sintax$s
660               }
661             return CUP$Sintax$result;
662
663             /*. . . . . .*/
664             case 36: // INICIALIZACION_FOR ::= DECLARACION
665               {
666                 Object RESULT =null;
667
668                 CUP$Sintax$result = parser.getSymbolFactory().newSymbol("INICIALIZACION_FOR",16, ((java_cup.runtime.Symbol)CUP$Sintax$s
669               }
670             return CUP$Sintax$result;
671
672             /*. . . . . .*/
673             case 37: // INCREMENTO_FOR ::= ID OPINCREMENTO
674               {
675                 Object RESULT =null;
676
677                 CUP$Sintax$result = parser.getSymbolFactory().newSymbol("INCREMENTO_FOR",17, ((java_cup.runtime.Symbol)CUP$Sintax$stack
678               }
679             return CUP$Sintax$result;
680
```

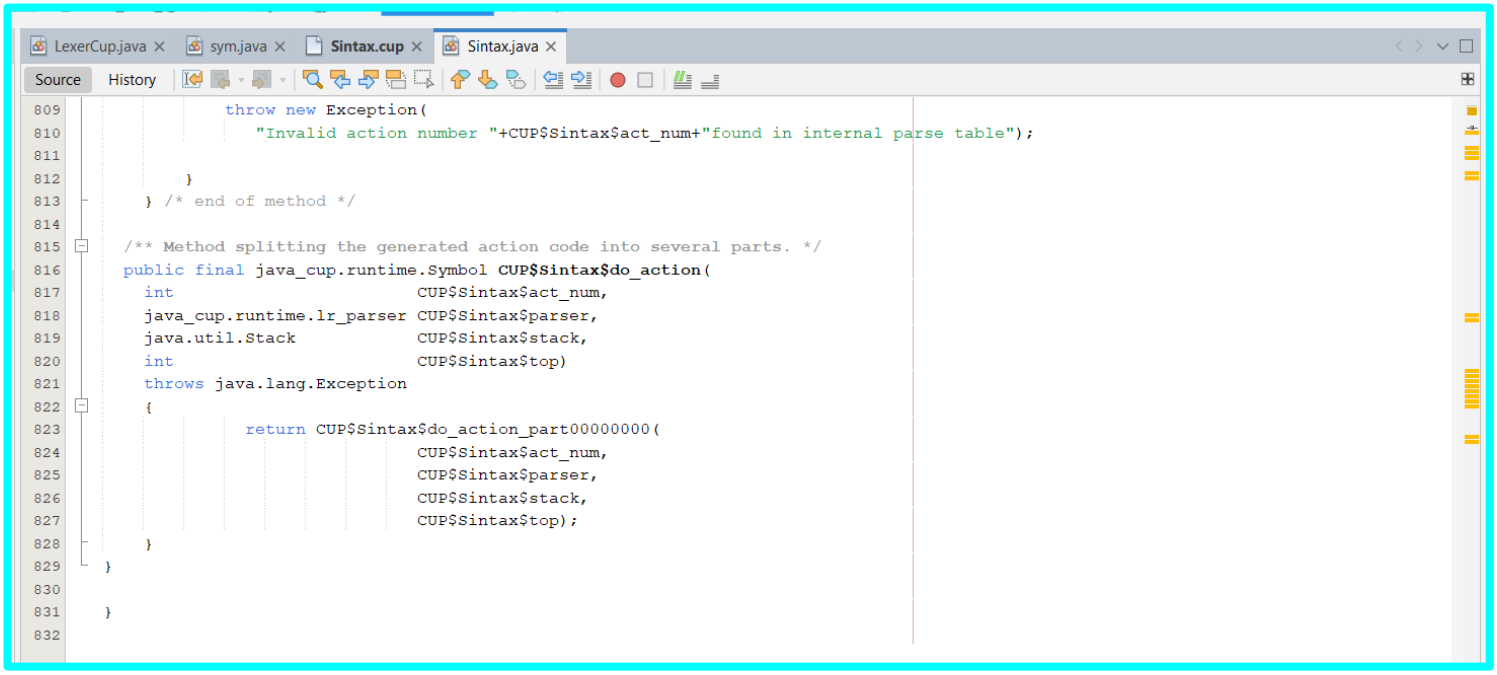









```
789             /*. . . . . .*/
790             case 50: // PARAMETROS_METODO ::= TIPO_RETORNO ID COMA PARAMETROS_METODO
791               {
792                 Object RESULT =null;
793
794                 CUP$Sintax$result = parser.getSymbolFactory().newSymbol("PARAMETROS_METODO",14, ((java_cup.runtime.Symbol)CUP$Sintax$st
795               }
796             return CUP$Sintax$result;
797
798             /*. . . . . .*/
799             case 51: // PARAMETROS_METODO ::=
800               {
801                 Object RESULT =null;
802
803                 CUP$Sintax$result = parser.getSymbolFactory().newSymbol("PARAMETROS_METODO",14, ((java_cup.runtime.Symbol)CUP$Sintax$st
804               }
805             return CUP$Sintax$result;
806
807             /* . . . . . .*/
808             default:
809               throw new Exception(
810                  "Invalid action number "+CUP$Sintax$act_num+"found in internal parse table");
811
812           }
813       } /* end of method */
814
815     /** Method splitting the generated action code into several parts. */
816     public final java_cup.runtime.Symbol CUP$Sintax$do_action(
817       int                        CUP$Sintax$act_num,
818       java_cup.runtime.lr_parser CUP$Sintax$parser,
819       java.util.Stack            CUP$Sintax$stack,
820       int                        CUP$Sintax$top)
821       throws java.lang.Exception
822       {
823               return CUP$Sintax$do_action_part00000000(
824                                  CUP$Sintax$act_num,
```




```
809               throw new Exception(
810                  "Invalid action number "+CUP$Sintax$act_num+"found in internal parse table");
811
812           }
813       } /* end of method */
814
815     /** Method splitting the generated action code into several parts. */
816     public final java_cup.runtime.Symbol CUP$Sintax$do_action(
817       int                        CUP$Sintax$act_num,
818       java_cup.runtime.lr_parser CUP$Sintax$parser,
819       java.util.Stack            CUP$Sintax$stack,
820       int                        CUP$Sintax$top)
821       throws java.lang.Exception
822       {
823               return CUP$Sintax$do_action_part00000000(
824                                  CUP$Sintax$act_num,
825                                  CUP$Sintax$parser,
826                                  CUP$Sintax$stack,
827                                  CUP$Sintax$top);
828       }
829   }
830
831   }
832
```

TermSintac.java

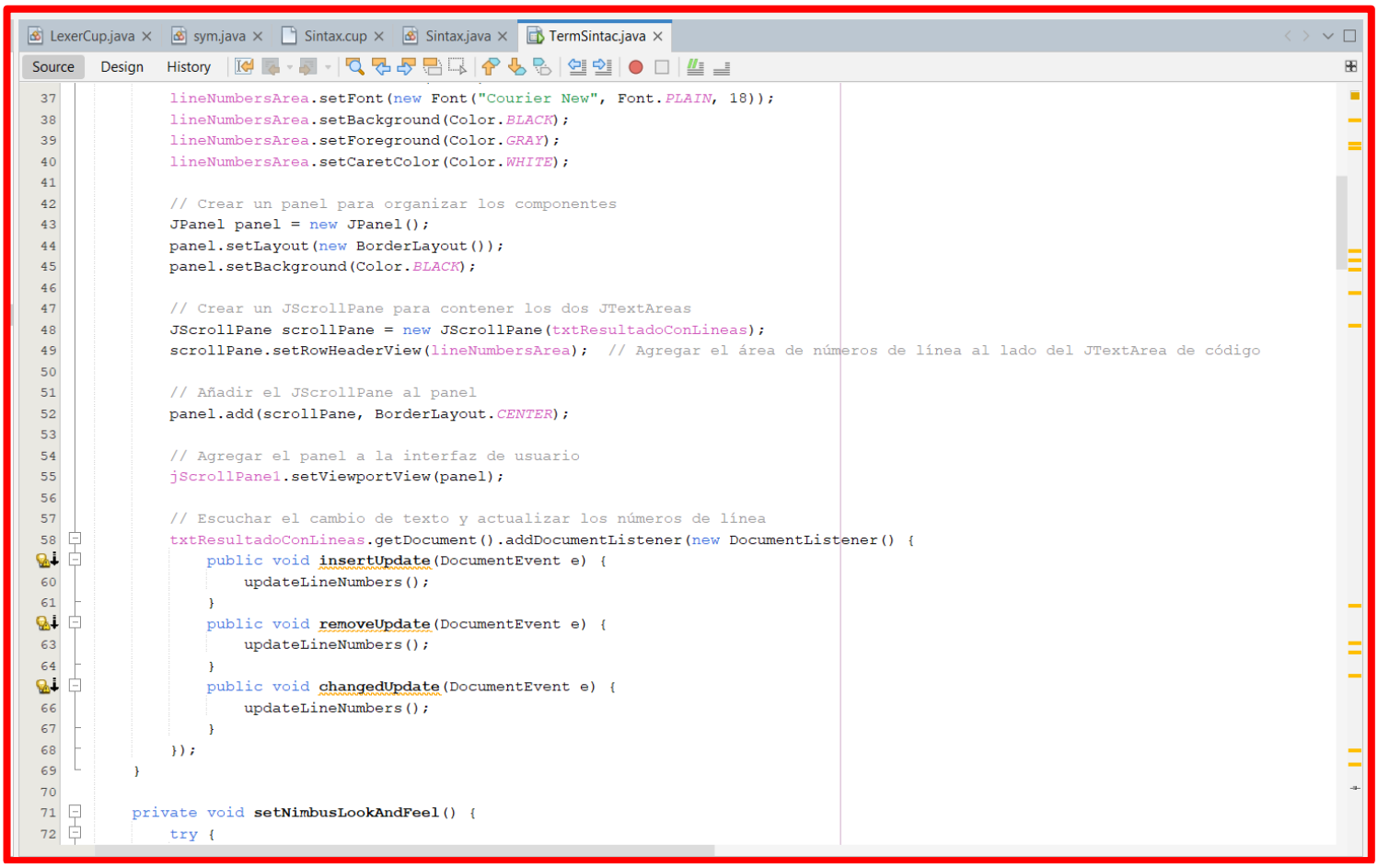
Este código implementa una aplicación gráfica en Java Swing llamada TermSintac, que sirve como una interfaz para un analizador sintáctico. La aplicación permite cargar archivos de texto, visualizar el contenido con números de línea, realizar análisis sintácticos y mostrar errores o resultados del análisis. 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.

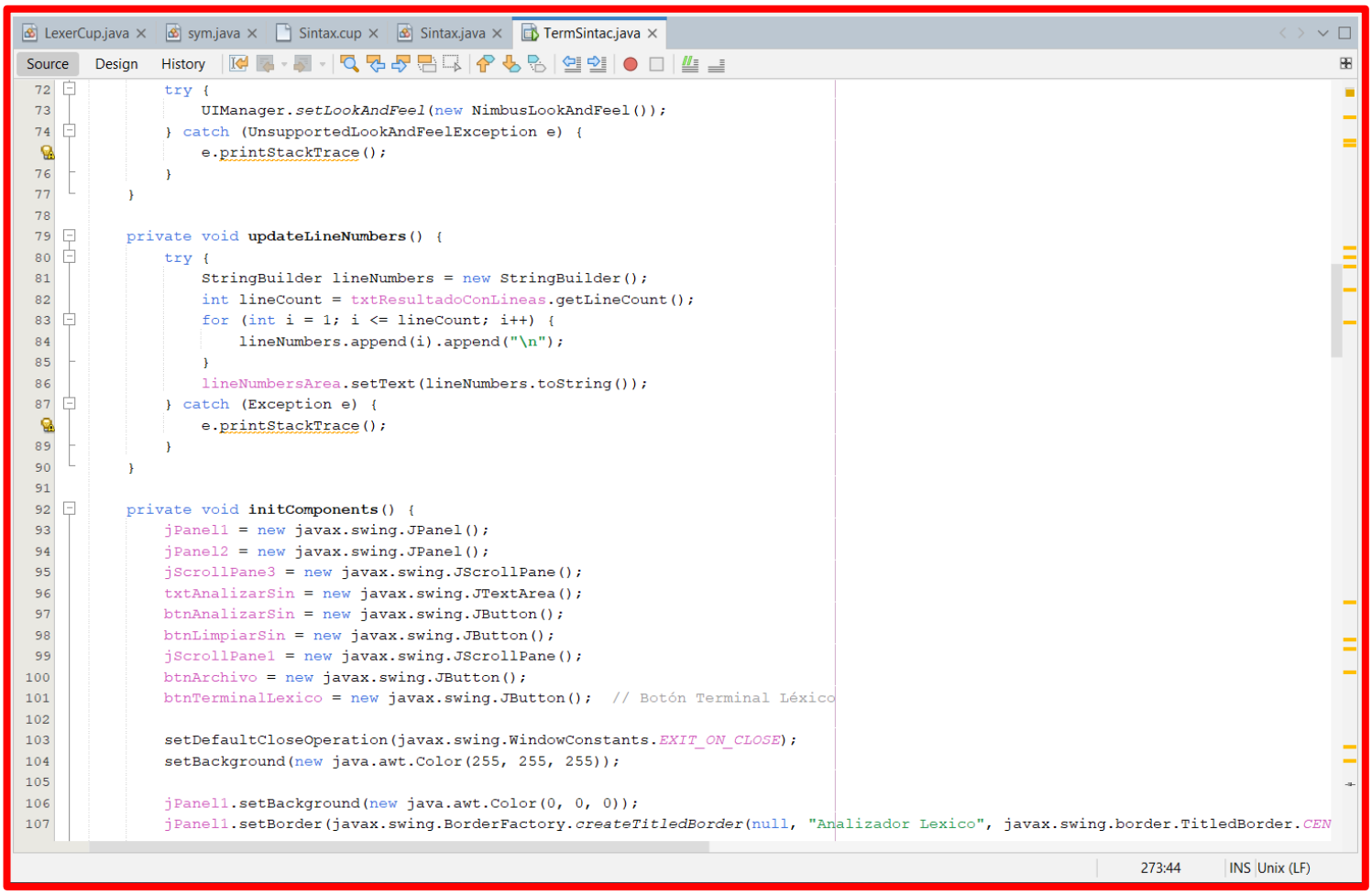
LexerCup.java | sym.java | Sintax.cup | Sintax.java | TermSintac.java

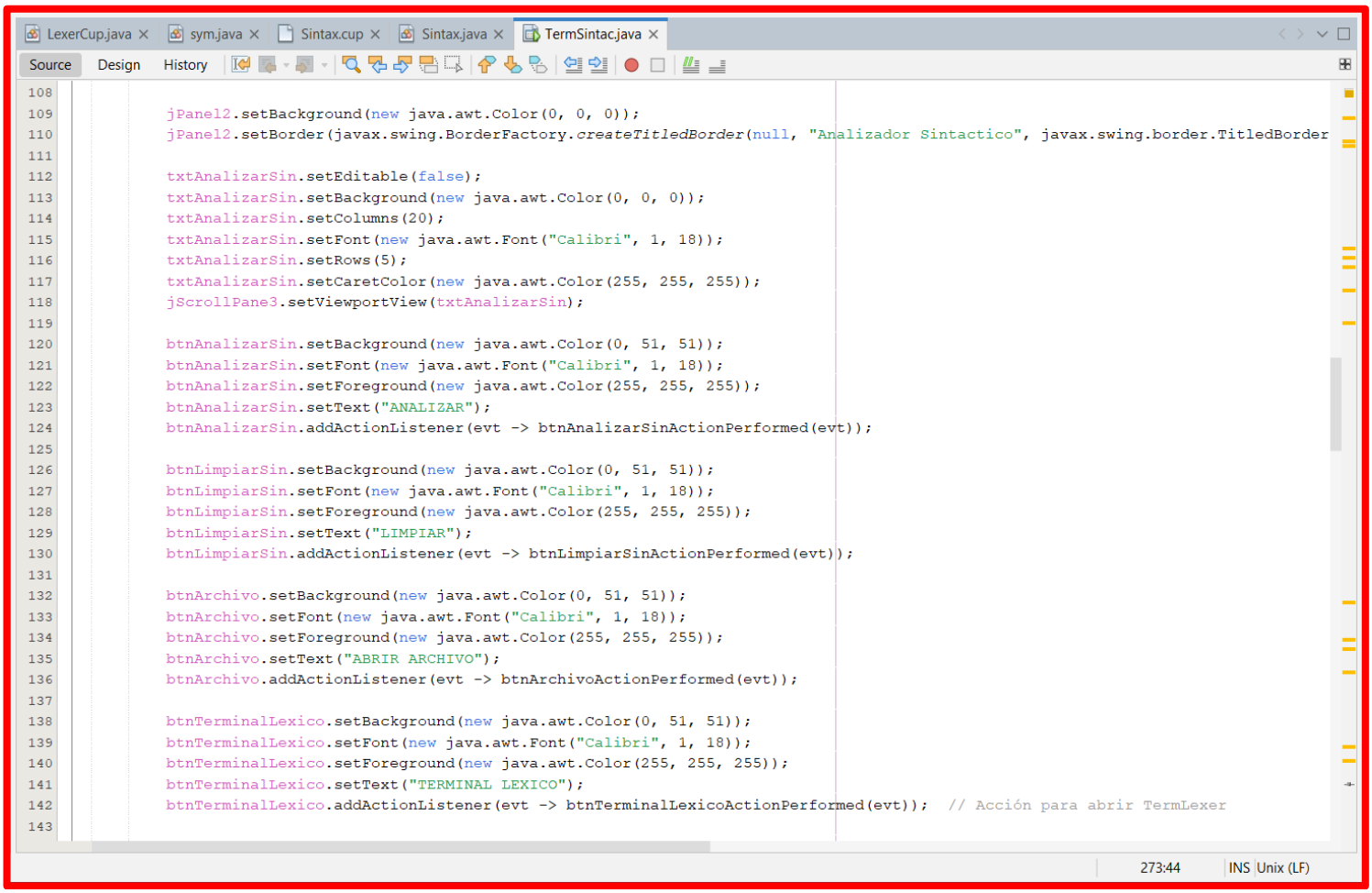
Source | Design | History

```
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);
```

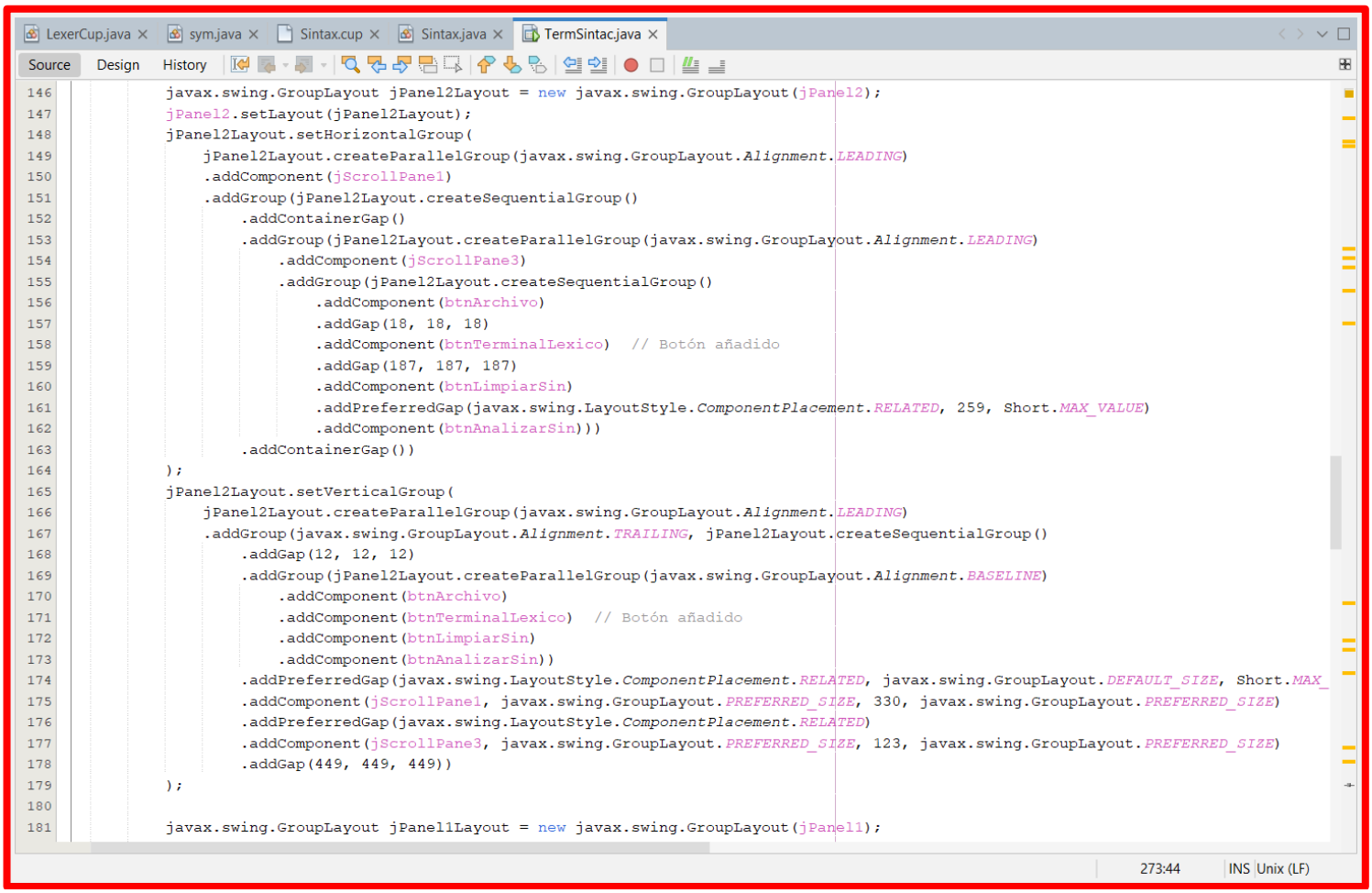
273:44 | INS | Unix (LF)


LexerCup.java
sym.java
Sintax.cup
Sintax.java
TermSintac.java
Source
Design
History
lineNumbersArea.setFont(new Font("Courier New", Font.PLAIN, 18));
lineNumbersArea.setBackground(Color.BLACK);
lineNumbersArea.setForeground(Color.GRAY);
lineNumbersArea.setCaretColor(Color.WHITE);
// Crear un panel para organizar los componentes
JPanel panel = new JPanel();
panel.setLayout(new BorderLayout());
panel.setBackground(Color.BLACK);
// Crear un JScrollPane para contener los dos JTextAreas
JScrollPane scrollPane = new JScrollPane(txtResultadoConLineas);
scrollPane.setRowHeaderView(lineNumbersArea); // Agregar el área de números de línea al lado del JTextArea de código
// Añadir el JScrollPane al panel
panel.add(scrollPane, BorderLayout.CENTER);
// Agregar el panel a la interfaz de usuario
jScrollPane1.setViewportView(panel);
// Escuchar el cambio de texto y actualizar los números de línea
txtResultadoConLineas.getDocument().addDocumentListener(new DocumentListener() {
public void insertUpdate(DocumentEvent e) {
updateLineNumbers();
}
public void removeUpdate(DocumentEvent e) {
updateLineNumbers();
}
public void changedUpdate(DocumentEvent e) {
updateLineNumbers();
}
});
}
private void setNimbusLookAndFeel() {
try {

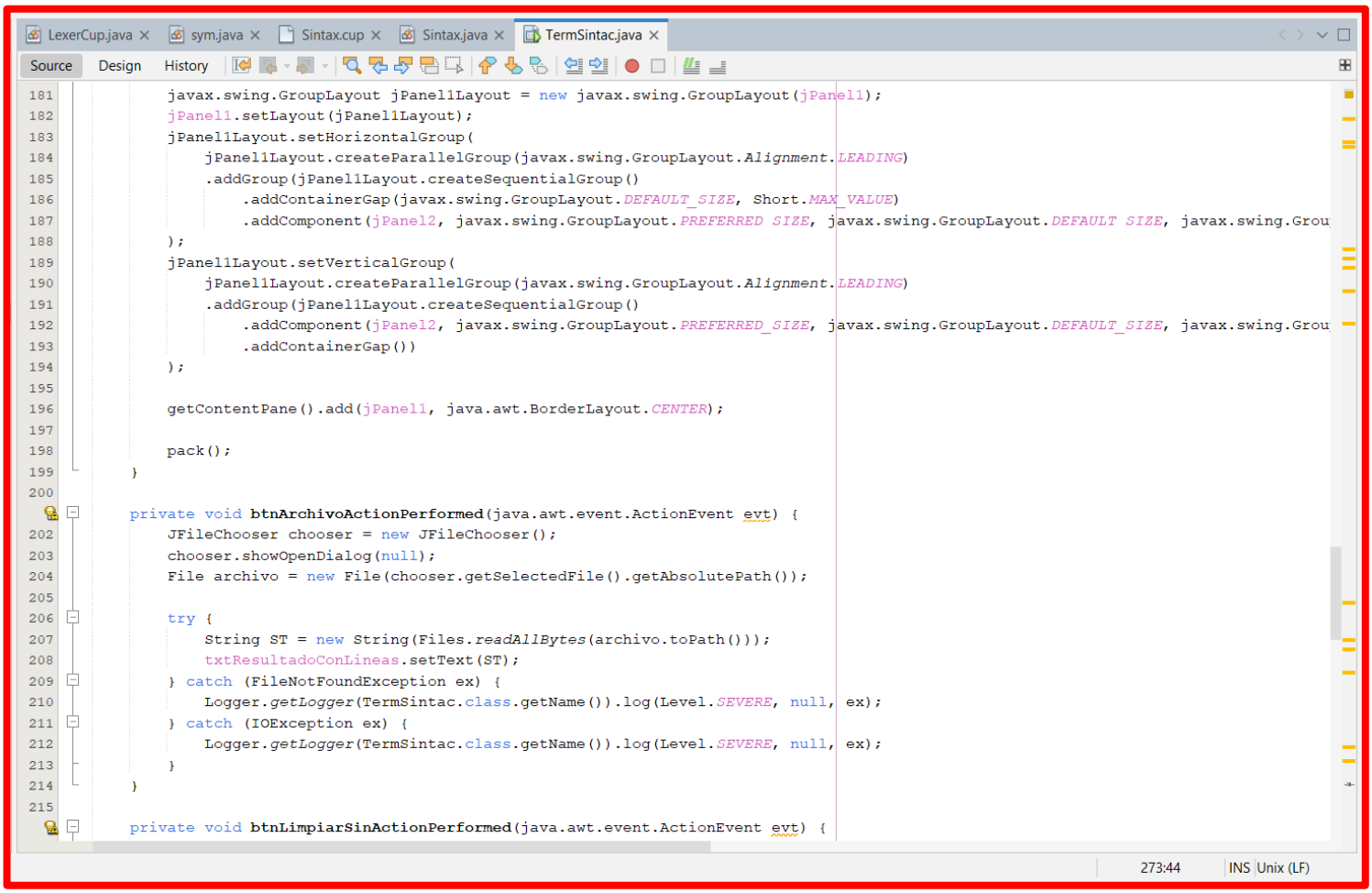


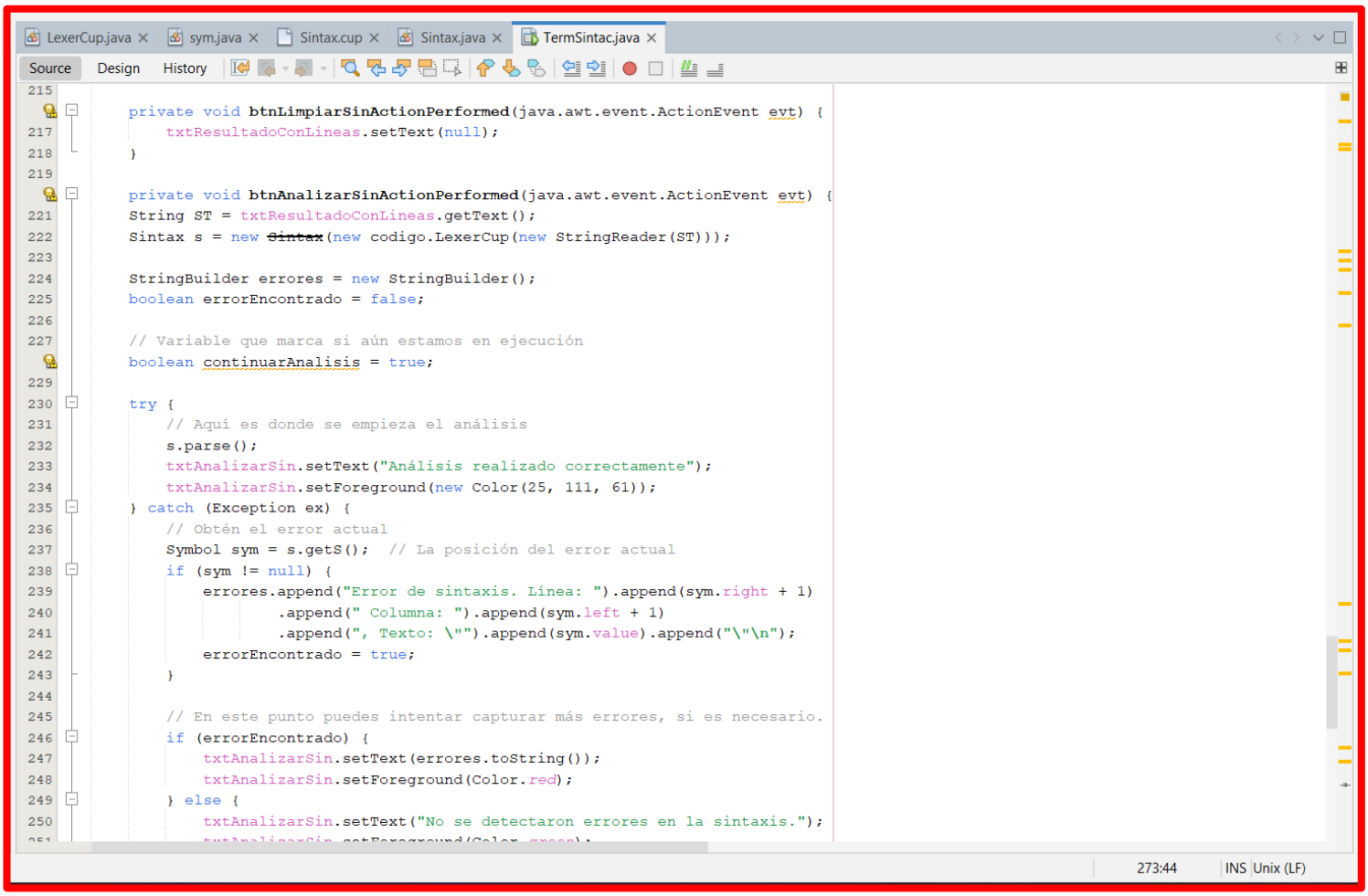


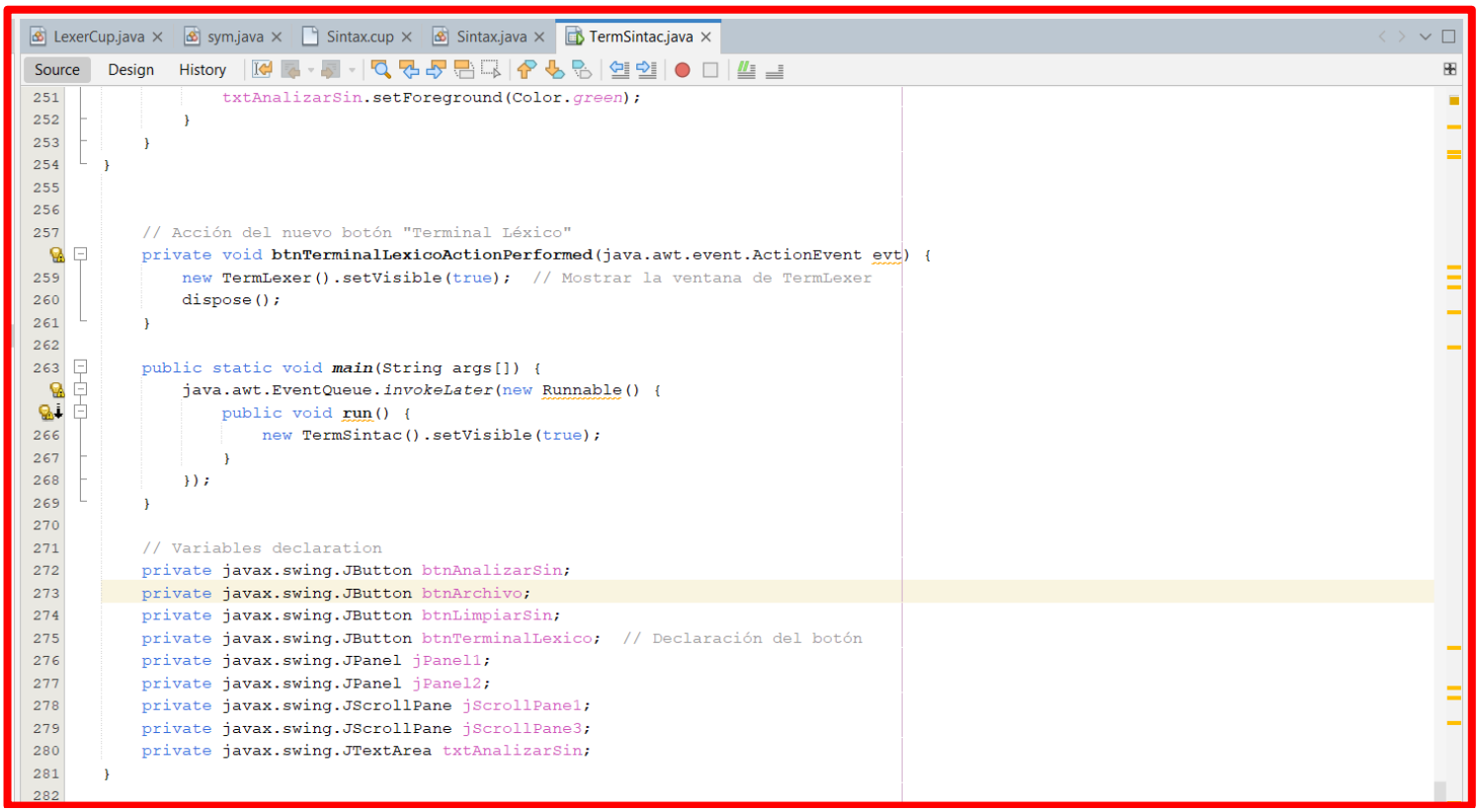

```
108
109         jPanel2.setBackground(new java.awt.Color(0, 0, 0));
110         jPanel2.setBorder(javax.swing.BorderFactory.createTitledBorder(null, "Analizador Sintactico", javax.swing.border.TitledBorder
111
112         txtAnalizarSin.setEditable(false);
113         txtAnalizarSin.setBackground(new java.awt.Color(0, 0, 0));
114         txtAnalizarSin.setColumns(20);
115         txtAnalizarSin.setFont(new java.awt.Font("Calibri", 1, 18));
116         txtAnalizarSin.setRows(5);
117         txtAnalizarSin.setCaretColor(new java.awt.Color(255, 255, 255));
118         jScrollPane3.setViewportView(txtAnalizarSin);
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         btnLimpiarSin.setBackground(new java.awt.Color(0, 51, 51));
127         btnLimpiarSin.setFont(new java.awt.Font("Calibri", 1, 18));
128         btnLimpiarSin.setForeground(new java.awt.Color(255, 255, 255));
129         btnLimpiarSin.setText("LIMPIAR");
130         btnLimpiarSin.addActionListener(evt -> btnLimpiarSinActionPerformed(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
```




```
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(jScrollPane1)
151             .addGroup(jPanel2Layout.createSequentialGroup()
152                 .addContainerGap()
153                 .addGroup(jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
154                     .addComponent(jScrollPane3)
155                     .addGroup(jPanel2Layout.createSequentialGroup()
156                         .addComponent(btnArchivo)
157                         .addGap(18, 18, 18)
158                         .addComponent(btnTerminalLexico)  // Botón añadido
159                         .addGap(187, 187, 187)
160                         .addComponent(btnLimpiarSin)
161                         .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, 259, Short.MAX_VALUE)
162                         .addComponent(btnAnalizarSin)))
163                 .addContainerGap())
164         );
165         jPanel2Layout.setVerticalGroup(
166             jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
167             .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, jPanel2Layout.createSequentialGroup()
168                 .addGap(12, 12, 12)
169                 .addGroup(jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
170                     .addComponent(btnArchivo)
171                     .addComponent(btnTerminalLexico)  // Botón añadido
172                     .addComponent(btnLimpiarSin)
173                     .addComponent(btnAnalizarSin))
174                 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_
175                 .addComponent(jScrollPane1, javax.swing.GroupLayout.PREFERRED_SIZE, 330, javax.swing.GroupLayout.PREFERRED_SIZE)
176                 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
177                 .addComponent(jScrollPane3, javax.swing.GroupLayout.PREFERRED_SIZE, 123, javax.swing.GroupLayout.PREFERRED_SIZE)
178                 .addGap(449, 449, 449))
179         );
180
181         javax.swing.GroupLayout jPanel1Layout = new javax.swing.GroupLayout(jPanel1);
```




LexerCup.java
sym.java
Sintax.cup
Sintax.java
TermSintac.java
Source
Design
History
215
private void btnLimpiarSinActionPerformed(java.awt.event.ActionEvent evt) {
txtResultadoConLineas.setText(null);
}
private void btnAnalizarSinActionPerformed(java.awt.event.ActionEvent evt) {
String ST = txtResultadoConLineas.getText();
Sintax s = new Sintax(new codigo.LexerCup(new StringReader(ST)));
StringBuilder errores = new StringBuilder();
boolean errorEncontrado = false;
// Variable que marca si aún estamos en ejecución
boolean continuarAnalisis = true;
try {
// Aquí es donde se empieza el análisis
s.parse();
txtAnalizarSin.setText("Análisis realizado correctamente");
txtAnalizarSin.setForeground(new Color(25, 111, 61));
} catch (Exception ex) {
// Obtén el error actual
Symbol sym = s.getS(); // La posición del error actual
if (sym != null) {
errores.append("Error de sintaxis. Línea: ").append(sym.right + 1)
.append(" Columna: ").append(sym.left + 1)
.append(", Texto: \"").append(sym.value).append("\"\n");
errorEncontrado = true;
}
// En este punto puedes intentar capturar más errores, si es necesario.
if (errorEncontrado) {
txtAnalizarSin.setText(errores.toString());
txtAnalizarSin.setForeground(Color.red);
} else {
txtAnalizarSin.setText("No se detectaron errores en la sintaxis.");
273:44
INS
Unix (LF)



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 teorico de un inicio se concirtio

en un compilador.