INSTITUTO TECNOLOGICO DE PACHUCA

"El hombre alimenta el ingenio en contacto con la Ciencia"

"DOCUMENTACION ER Y AF "

INGENIERIA EN SISTEMAS COMPUTACIONALES

NOMBRE DE LA ASIGNATURA

LENGUAJES Y ATOMATA 1

INTEGRANTES DEL EQUIPO

Daniel Flores Bautista

Sergio Enrique Torres Martínez

Víctor Gabriel Gutiérrez Hernández

Mariana Mendoza Gutiérrez

Reyes Hernández Reyes

PROFESOR DE LA MATERIA

Ing. Baume Lazcano Rodolfo

PACHUCA, HIDALGO, 14 DE MAYO DEL 2024

```
package codigo; // Declaración del paquete "codigo"
import static codigo.Tokens.*; // Importación de los tokens definidos en el archivo Tokens.java
%%
%class Lexer // Declaración del nombre de la clase Lexer
%type Tokens // Declaración del tipo de los tokens
L=[a-zA-Z_]+ // Definición de una expresión regular para identificar letras y guiones bajos como nombres de variables o palabras clave
D=[0-9]+ // Definición de una expresión regular para identificar números enteros
espacio=[*,\t,\r,\n]+ // Definición de una expresión regular para identificar espacios en blanco
%{
    public String lexeme; // Declaración de una variable "lexeme" para almacenar el lexema actual
%}
%%

// Definición de las reglas de tokenización:
int | // Palabra clave 'int'
if | // Palabra clave 'if'
else | // Palabra clave 'else'
while {lexeme=yytext(); return Reservadas;} // Palabra clave 'while'. Se asigna el lexema actual a la variable 'lexeme' y se retorna el t

{espacio} {/ignore/} // Espacios en blanco. Se ignoran.
"//".* {/Ignore/} // Comentarios de una línea. Se ignoran.
"=" {return Igual;} // Asignación. Se retorna el token 'Igual'
"+" {return Suma;} // Operador de suma. Se retorna el token 'Suma'
"-" {return Resta;} // Operador de resta. Se retorna el token 'Resta'
"*" {return Multiplicacion;} // Operador de multiplicación. Se retorna el token 'Multiplicacion'
"/" {return Division;} // Operador de división. Se retorna el token 'Division'
{L}({L}|{D})* {lexeme=yytext(); return Identificador;} // Identificador. Se asigna el lexema actual a la variable 'lexeme' y se retorna el
("(-"{D}+")")|{D}+ {lexeme=yytext(); return Numero;} // Número entero. Se asigna el lexema actual a la variable 'lexeme' y se retorna el
. {return ERROR;} // Cualquier otro carácter que no se ajuste a ninguna de las reglas anteriores. Se retorna el token 'ERROR'
```

Lexer.flex

Enel código se definen varias reglas de coincidencia utilizando expresiones regulares y acciones específicas para cada una. Aquí te detallo las reglas de coincidencia y su propósito:

Definición de expresiones regulares:

L=[a-zA-Z_]+: Identifica secuencias de una o más letras (mayúsculas o minúsculas) o guiones bajos. Esto generalmente se usa para identificar nombres de variables o palabras clave.

D=[0-9]+: Identifica secuencias de una o más cifras numéricas. Esto se usa para identificar números enteros.

espacio=[\t\r\n]+: Identifica secuencias de uno o más espacios, tabulaciones, retornos de carro o nuevas líneas. Esto se usa para identificar espacios en blanco.

Reglas de tokenización:

int |: Coincide con la palabra clave "int".

if |: Coincide con la palabra clave "if".

else |: Coincide con la palabra clave "else".

while {lexeme=yytext(); return Reservadas;}: Coincide con la palabra clave "while", asigna el texto coincidente a lexeme y retorna el token Reservadas.

{espacio} {/ignore/}: Coincide con cualquier secuencia de espacios en blanco y los ignora.

"//".* {/Ignore/}: Coincide con comentarios de una línea que comienzan con // y los ignora.

"=" {return Igual;}: Coincide con el signo de asignación = y retorna el token Igual.

"+" {return Suma;}: Coincide con el operador de suma + y retorna el token Suma.

"-" {return Resta;}: Coincide con el operador de resta - y retorna el token Resta.

"*" {return Multiplicacion;}: Coincide con el operador de multiplicación * y retorna el token Multiplicacion.

"/" {return Division;}: Coincide con el operador de división / y retorna el token Division.

{L}({L}{D})* {lexeme=yytext(); return Identificador;}: Coincide con un identificador que empieza con una letra o guion bajo, seguido opcionalmente por más letras y números. Asigna el texto coincidente a lexeme y retorna el token Identificador.

("(-"{D}+")")|{D}+ {lexeme=yytext(); return Numero;}: Coincide con números enteros, ya sean positivos o negativos. Asigna el texto coincidente a lexeme y retorna el token Numero.

. {return ERROR;}: Coincide con cualquier otro carácter que no se ajuste a ninguna de las reglas anteriores y retorna el token ERROR.

Asociación de acciones:

```
    private void btnAnalizarActionPerformed(java.awt.event.ActionEvent evt) {
        // Se crea un objeto File para el archivo de texto "Archivo.txt"
    File archivo = new File( pathname: "Archivo.txt");
    PrintWriter escribir;
    try {
        // Se crea un objeto PrintWriter para escribir en el archivo
        escribir = new PrintWriter( file: archivo);
        // Se escribe el contenido del campo de texto txtEntrada en el archivo
        escribir.print( s: txtEntrada.getText());
        // Se cierra el objeto PrintWriter
        escribir.close();
    } catch (FileNotFoundException ex) {
        // Se maneja la excepción si no se puede encontrar el archivo
        Logger.getLogger( name: FrmPrincipal.class.getName()).log( level: Level.SEVERE, msg: null, thrown: ex);
    }

    try {
        // Se crea un objeto Reader para leer el archivo de texto
        Reader lector = new BufferedReader(new FileReader( fileName: "Archivo.txt"));
        // Se crea un objeto Lexer utilizando el lector
        Lexer lexer = new Lexer( in: lector);
        String resultado = ""; // Se inicializa una cadena para almacenar los resultados del análisis
        while (true) {
            // Se obtiene el siguiente token del analizador léxico
            Tokens tokens = lexer.yylex();
            if (tokens == null) {
                // Si no hay más tokens, se agrega "FIN" al resultado y se muestra en txtResultado
                resultado += "FIN";
                txtResultado.setText( t: resultado);
                return;
            }
            // Se verifica el tipo de token obtenido y se agrega al resultado correspondiente
            switch(tokens) {
                case ERROR -> resultado += "Simbolo no definido\n"; // Token de error
                case Identificador, Numero, Reservadas -> resultado += lexer.lexeme + ": Es una " + tokens + "\n"; // Tokens válidos
                default -> resultado += "Token: " + tokens + "\n"; // Otros tokens no manejados
            }
        }
    } catch (FileNotFoundException ex) {
        // Se maneja la excepción si no se puede encontrar el archivo
        Logger.getLogger( name: FrmPrincipal.class.getName()).log( level: Level.SEVERE, msg: null, thrown: ex);
    } catch (IOException ex) {
        // Se maneja la excepción de entrada/salida
        Logger.getLogger( name: FrmPrincipal.class.getName()).log( level: Level.SEVERE, msg: null, thrown: ex);
    }
    }
```

Lectura del archivo y creación del lexer:

```
try  {
    // Se crea un objeto Reader para leer el archivo de texto
    Reader lector =  new BufferedReader ( new FileReader ( "Archivo.txt"
));
    // Se crea un objeto Lexer utilizando el lector
    Lexer lexer =  new Lexer ( lector );
    Cadena resultado =  "" ;  // Se inicializa una cadena para almacenar
los resultados del análisis
```

Aquí se crea un Reader para leer el archivo Archivo.txt y se inicializa el lexer con este Reader.

Bucle para obtener tokens:

```
while  ( true )  {
    // Se obtiene el siguiente token del analizador léxico
    Tokens tokens = lexer . yylex ();
    if  ( tokens ==  null )  {
        // Si no hay más tokens, se agrega "FIN" al resultado y se
muestra en txtResultado
        resultado +=  "FIN" ;
        txtResultado . setText ( resultado );
        devolver ;
    }
```

En este bucle infinito, se llama a lexer.yylex() para obtener el siguiente token.

Acciones basadas en el tipo de token:

```
// Se verifica el tipo de token obtenido y se agrega al resultado
correspondiente
switch ( tokens )  {
    case ERROR -> resultado +=  "Simbolo no definido\n" ;  // Token de
error
    case Identificador , Numero , Reservadas -> resultado += lexer .
lexema  +  ": Es una "  + tokens +  "\n" ;  // Tokens válidos
    default  -> resultado +=  "Token: "  + tokens +  "\n" ;  // Otros
tokens no manejados
}
```

Dependiendo del tipo de token obtenido, se realizan diferentes acciones:

ERROR: Si el token es ERROR, se agrega un mensaje indicando que se encontró un símbolo no definido.

Identificador, Numero, Reservadas: Para estos tokens válidos, se agrega al resultado el lexema asociado (almacenado en lexer.lexeme) y el tipo de token.

Otros tokens: Para cualquier otro tipo de token no manejado específicamente, se agrega un mensaje genérico.

Mostrar resultado en el campo de texto:

```
if  ( tokens ==  null )  {
    resultado +=  "FIN" ;
    txtResultado . setText ( resultado );
    devolver ;
}
```

Cuando no hay más tokens (es decir, tokens es null), se agrega "FIN" al resultado y se muestra en el campo de texto txtResultado.

Manejo de casos especiales

1. Prioridad en las Reglas de Coincidencia

Cuando un mismo patrón puede corresponder a múltiples tokens, es importante definir el orden de las reglas de coincidencia de manera que las reglas más específicas se evalúen antes que las más generales. Esto es especialmente relevante para distinguir entre palabras clave e identificadores.

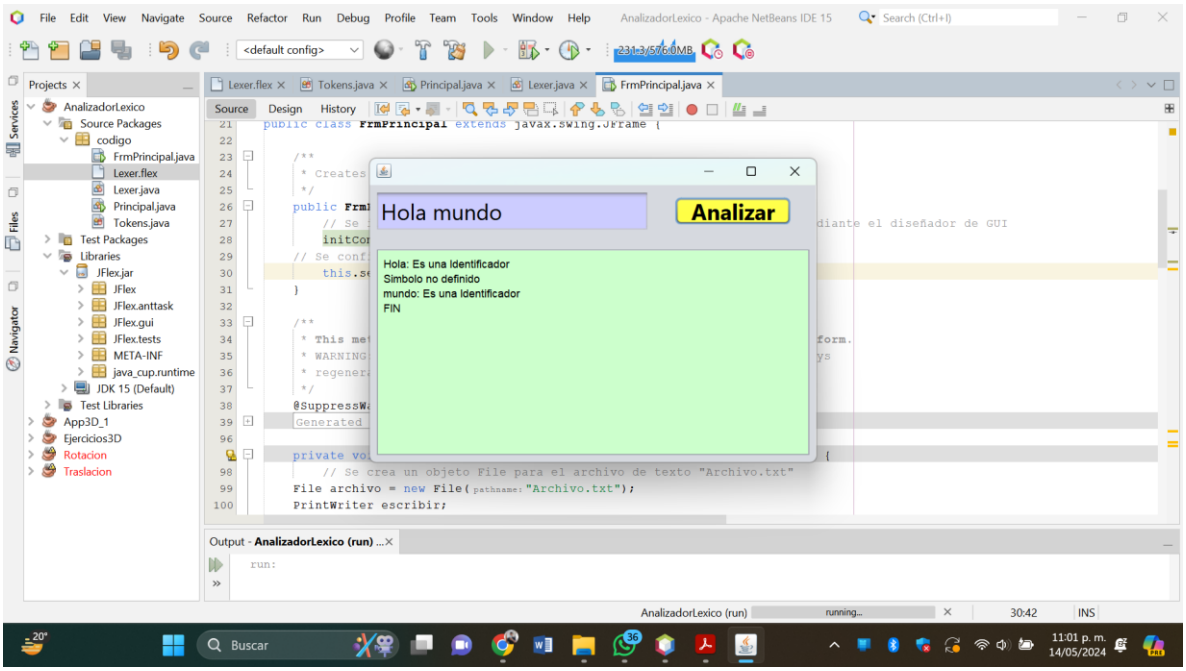
Por ejemplo, las palabras clave como int, if, else, y while deben definirse antes que la regla para identificadores. Así, si el patrón int aparece en el código fuente, se reconocerá como una palabra clave y no como un identificador.

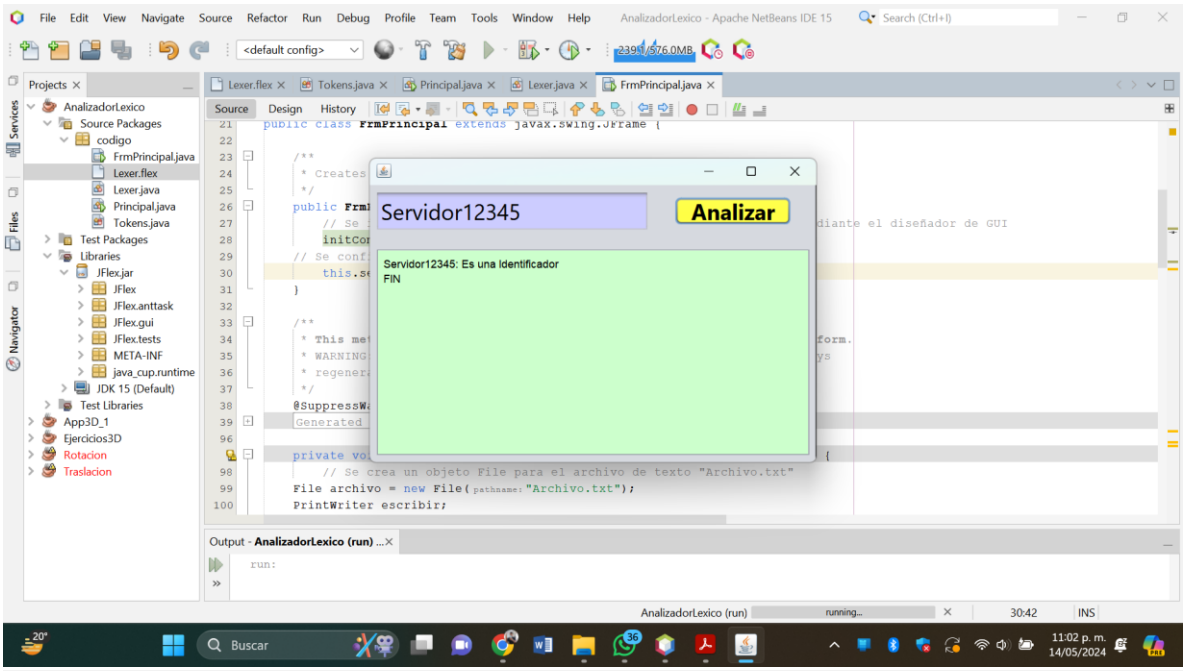
2. Uso de Acciones Específicas

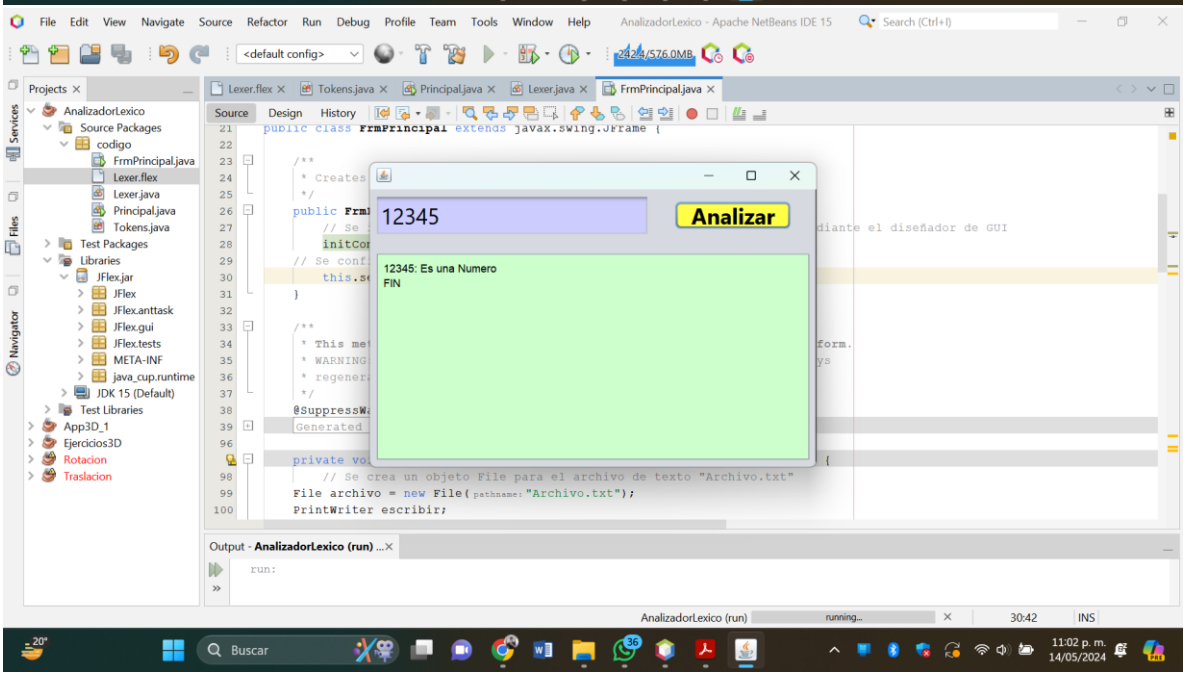
Dentro de las reglas de coincidencia, puedes usar acciones específicas para manejar casos especiales. Estas acciones pueden incluir la asignación de lexemas, el manejo de errores, y la definición de tokens específicos.

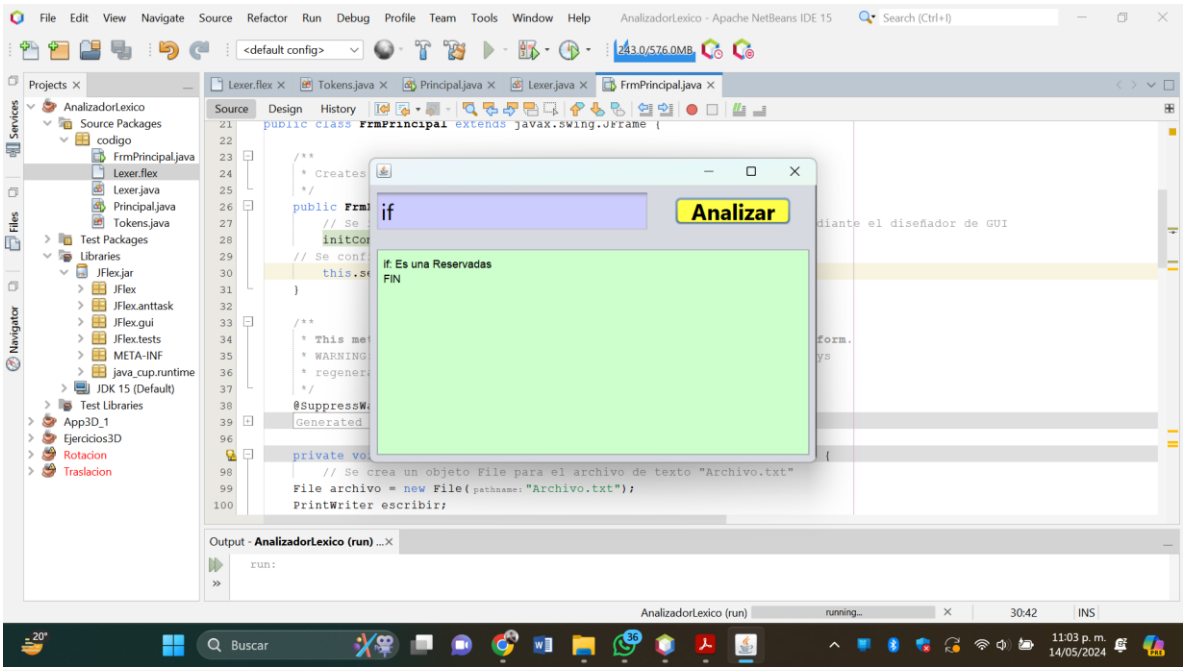
Pruebas de las reglas

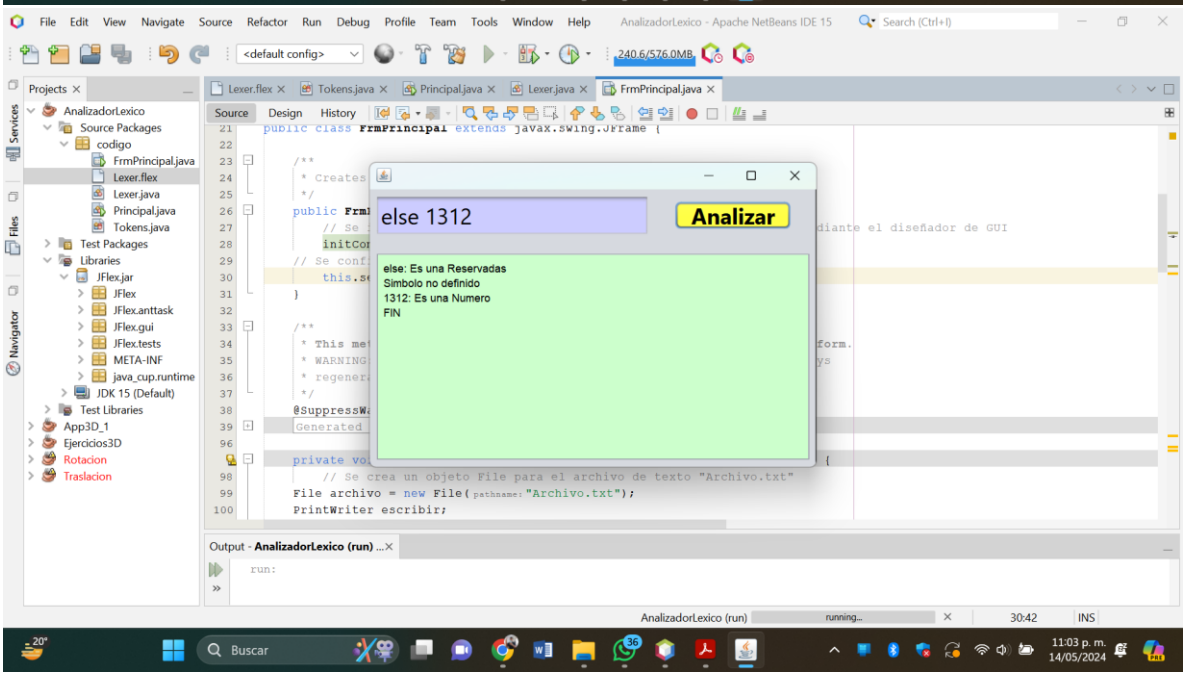
Para asegurar que las reglas de análisis léxico funcionen correctamente, es esencial realizar pruebas exhaustivas utilizando diferentes ejemplos de texto de entrada

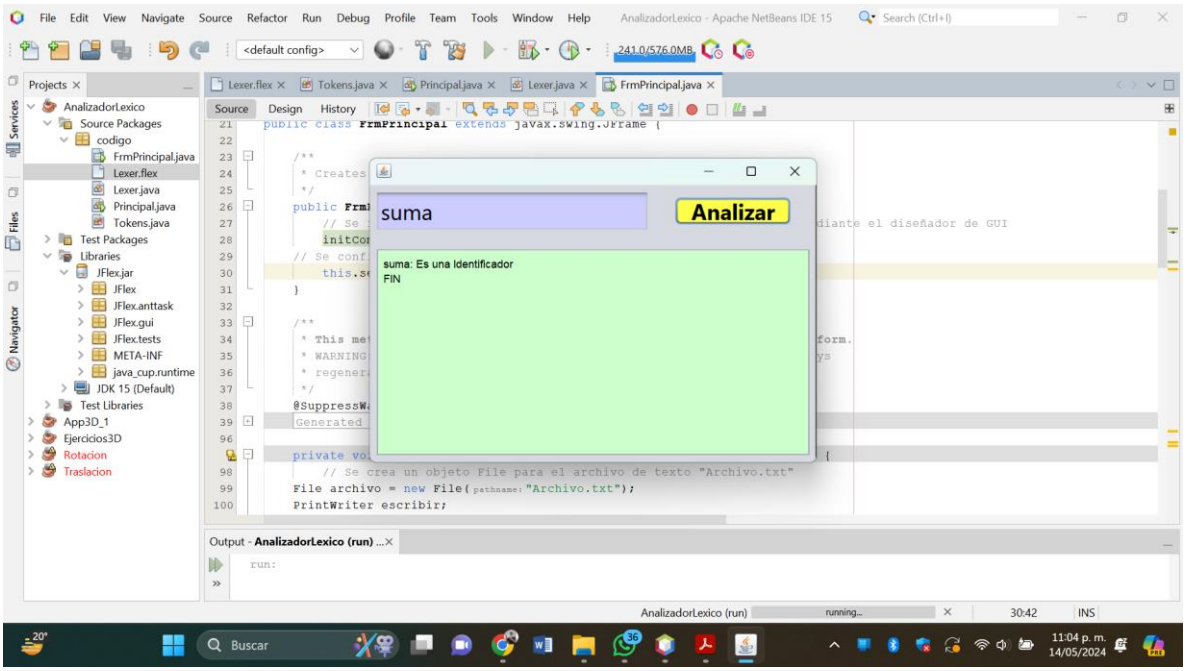

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
AnalizadorLexico - Apache NetBeans IDE 15
Search (Ctrl+I)
<default config>
Projects
AnalizadorLexico
Source Packages
codigo
FrmPrincipal.java
Lexer.flex
Lexer.java
Principal.java
Tokens.java
Test Packages
Libraries
JFlex.jar
JFlex
JFlex.anttask
JFlex.gui
JFlex.tests
META-INF
java_cup.runtime
JDK 15 (Default)
Test Libraries
App3D_1
Ejercicios3D
Rotacion
Traslacion
Lexer.flex
Tokens.java
Principal.java
Lexer.java
FrmPrincipal.java
Source
Design
History
public class FrmPrincipal extends javax.swing.JFrame {
Hola mundo
Analizar
Hola: Es una Identificador
Simbolo no definido
mundo: Es una Identificador
FIN
// Se crea un objeto File para el archivo de texto "Archivo.txt"
File archivo = new File(pathname: "Archivo.txt");
PrintWriter escribir;
Output - AnalizadorLexico (run)
run:
AnalizadorLexico (run)
running...
30:42
INS
Buscar
11:01 p. m.
14/05/2024

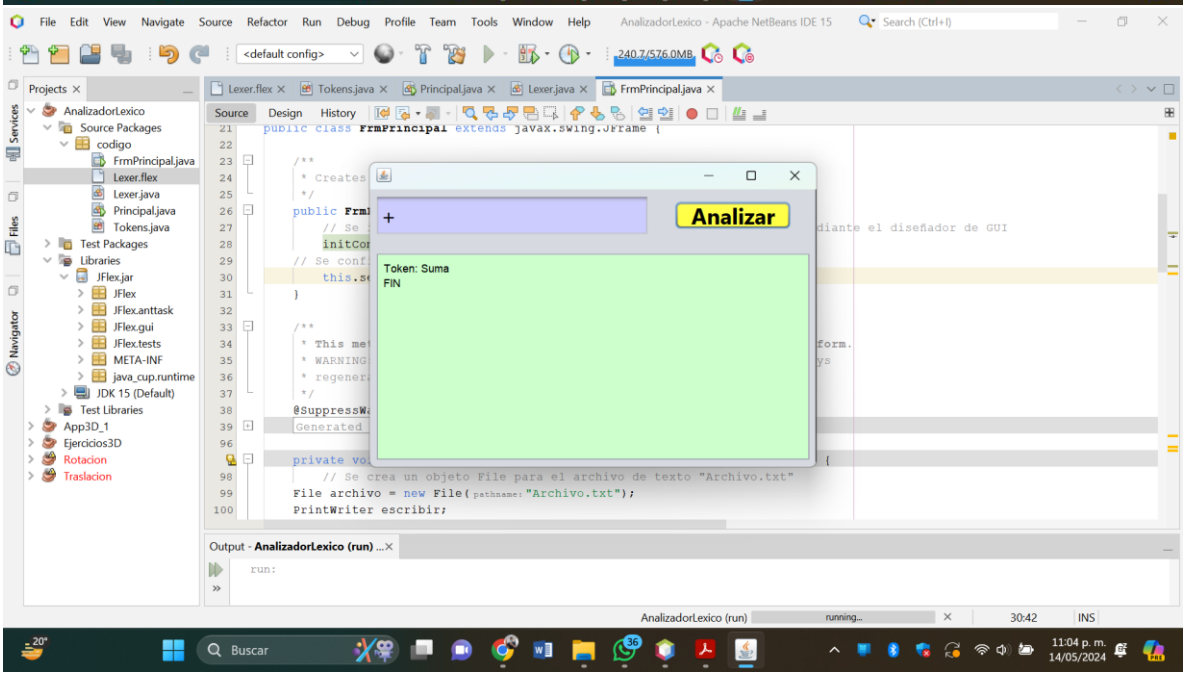

Servidor12345
Analizar
Servidor12345: Es una Identificador
FIN

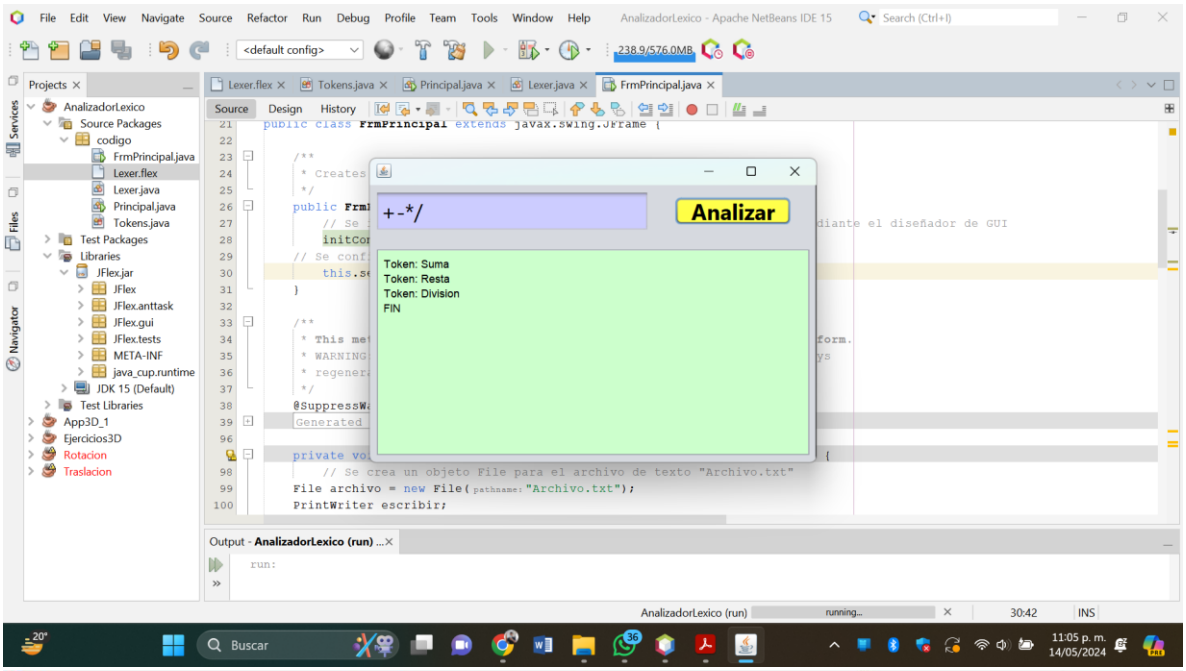

12345
Analizar
12345: Es una Numero
FIN

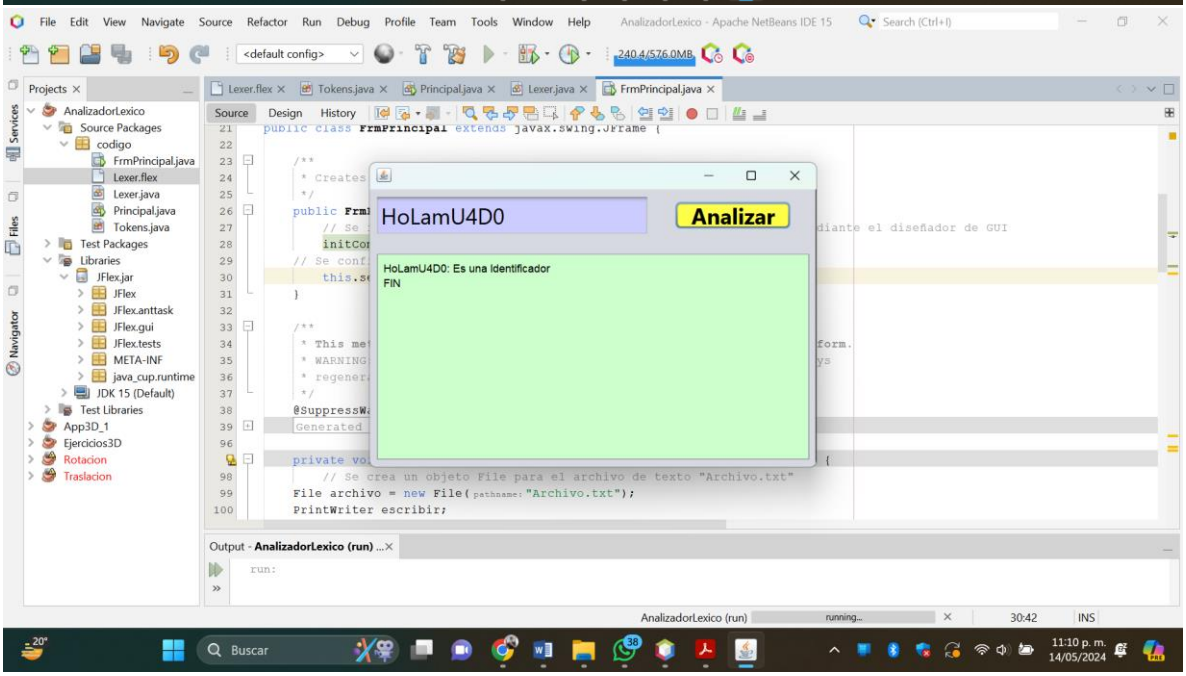

if
Analizar
If: Es una Reservadas
FIN


else 1312
Analizar
else: Es una Reservadas
Simbolo no definido
1312: Es una Numero
FIN


File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
AnalizadorLexico - Apache NetBeans IDE 15
Search (Ctrl+I)
<default config>
241.0/576.0MB
Projects
AnalizadorLexico
Source Packages
codigo
FrmPrincipal.java
Lexer.flex
Lexer.java
Principal.java
Tokens.java
Test Packages
Libraries
JFlex.jar
JFlex
JFlex.anttask
JFlex.gui
JFlex.tests
META-INF
java_cup.runtime
JDK 15 (Default)
Test Libraries
App3D_1
Ejercicios3D
Rotacion
Traslacion
Lexer.flex
Tokens.java
Principal.java
Lexer.java
FrmPrincipal.java
Source
Design
History
public class FrmPrincipal extends javax.swing.JFrame {
suma
Analizar
suma: Es una Identificador
FIN
Output - AnalizadorLexico (run)
run:
AnalizadorLexico (run)
running...
30:42
INS
Buscar
11:04 p. m.
14/05/2024


File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
AnalizadorLexico - Apache NetBeans IDE 15
Search (Ctrl+I)
<default config>
240.7/576.0MB
Projects
AnalizadorLexico
Source Packages
codigo
FrmPrincipal.java
Lexer.flex
Lexer.java
Principal.java
Tokens.java
Test Packages
Libraries
JFlex.jar
JFlex
JFlex.anttask
JFlex.gui
JFlex.tests
META-INF
java_cup.runtime
JDK 15 (Default)
Test Libraries
App3D_1
Ejercicios3D
Rotacion
Traslacion
Lexer.flex
Tokens.java
Principal.java
Lexer.java
FrmPrincipal.java
Source
Design
History
public class FrmPrincipal extends javax.swing.JFrame {
+
Analizar
Token: Suma
FIN
Output - AnalizadorLexico (run)
run:
AnalizadorLexico (run)
running...
30:42
INS
Buscar
11:04 p. m.
14/05/2024


+-*/
Analizar
Token: Suma
Token: Resta
Token: Division
FIN


HoLamU4D0
Analizar
HoLamU4D0: Es una Identificador
FIN