



TECNOLÓGICO NACIONAL DE MÉXICO



Tecnológico Nacional de México

Instituto Tecnológico de Pachuca

Ingeniería en Sistemas Computacionales

7mo Semestre Grupo: B

Alumna: Muñoz Castillo Ariana 22-200-196

Tema: Analizador Léxico

Materia: Lenguajes y Autómatas

Profesor: Rodolfo Baume Lazcano

SEMESTRE: Febrero – Junio 2024

21 – Mayo – 2024

ESTE PROYECTO TIENE LO SIGUIENTE:

LENGUAJE USADO

PALABRAS CLAVE:

- | | |
|----------|----------|
| ✓ Import | ✓ Case |
| ✓ Class | ✓ Break |
| ✓ Enum | ✓ For |
| ✓ Static | ✓ If |
| ✓ Void | ✓ Else |
| ✓ New | ✓ return |
| ✓ Switch | |

IDENTIFICADORES:

- ✓ Token
- ✓ TokenType
- ✓ Lexer
- ✓ tokens
- ✓ input
- ✓ isVocal
- ✓ isDigit
- ✓ isOperador
- ✓ tokenize
- ✓ main

OPERADORES:

- = → Asignación
- != → Comparación de desigualdad
- + → Concatenación
- () → Control de flujo

SIMBOLOS ESPECIALES:

{ } → Llaves que determinan el bloque de código
() → Parámetros o métodos
; → Finalizar declaraciones
, → Separa elementos

Comentario:

Lo determino como JAVA lo suele utilizar con //

Reglas establecidas:

I. Regla de Identificación de Vocales:

Regla: Reconocer caracteres vocales (A, E, I, O, U) tanto en mayúsculas como en minúsculas.

Expresión Regular: [AEIOUaeiou]

Implementación: Método isVocal(char c).

II. Regla de Identificación de Dígitos:

Regla: Reconocer caracteres numéricos del 0 al 9.

Expresión Regular: [0-9]

Implementación: Método isDigit(char c).

III. Regla de Identificación de Operadores:

Regla: Reconocer los operadores aritméticos +, -, *, /.

Expresión Regular: [+\\-*/]

Implementación: Método isOperator(char c).

IV. Regla de Identificación de Tokens Erróneos:

Regla: Cualquier carácter que no sea una vocal, un dígito o un operador debe ser identificado como un valor erróneo.

Expresión Regular: [^AEIOUaeiou0-9+\\-*/]

Implementación: Se maneja en la lógica de tokenización dentro del método tokenize() cuando un carácter no coincide con las reglas de vocales, dígitos u operadores.

V. Regla de Comentarios:

Regla: En caso de ocupar comentarios dentro del código se identificarán con la siguiente expresión

Expresión Regular: //.*

Implementación: //Ariana Muñoz Castillo

VI. Regla de Creación de Tokens:

Regla: Crear objetos de token con tipo y valor para cada carácter reconocido según las reglas anteriores.

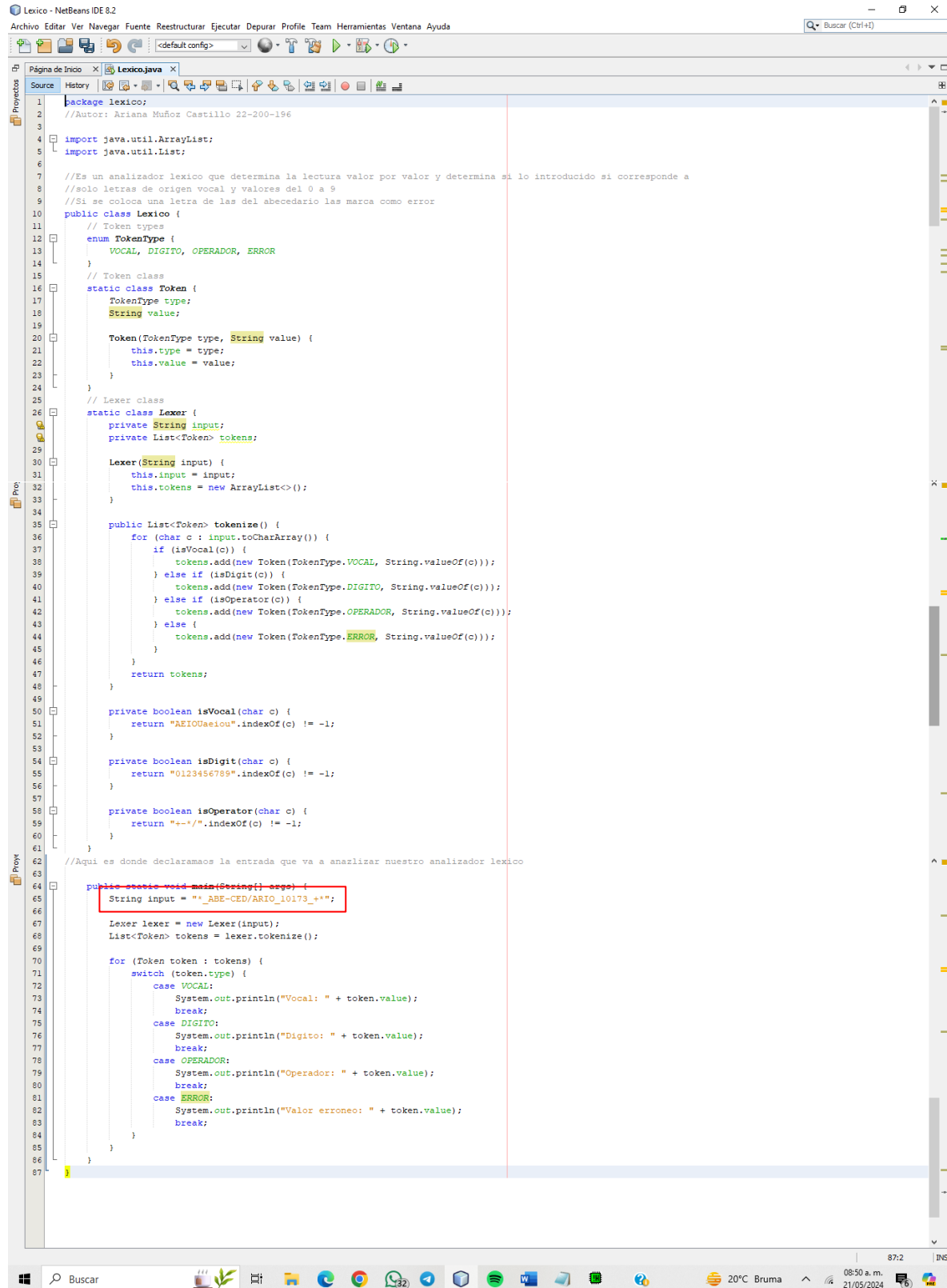
Implementación: Se maneja en el método tokenize(), donde se crean instancias de la clase Token con el tipo correspondiente (VOCAL, DIGITO, OPERADOR, ERROR).

Tabla de Tokens:

Tipo de Token	Nombre del Token	Expresión Regular
Palabra clave	import	\bimport\b
Palabra clave	class	\bclass\b
Palabra clave	enum	\benum\b
Palabra clave	static	\bstatic\b
Palabra clave	void	\bvoid\b
Palabra clave	new	\bnew\b
Palabra clave	switch	\bswitch\b
Palabra clave	case	\bcase\b
Palabra clave	break	\bbreak\b
Palabra clave	for	\bfor\b
Palabra clave	if	\bif\b
Palabra clave	else	\belse\b
Palabra clave	return	\breturn\b
Identificador	Token	[a-zA-Z_][a-zA-ZO-9_]*
Identificador	TokenType	[a-zA-Z_][a-zA-ZO-9_]*
Identificador	Lexer	[a-zA-Z_][a-zA-ZO-9_]*
Identificador	tokens	[a-zA-Z_][a-zA-ZO-9_]*
Identificador	input	[a-zA-Z_][a-zA-ZO-9_]*
Identificador	isVocal	[a-zA-Z_][a-zA-ZO-9_]*
Identificador	isDigit	[a-zA-Z_][a-zA-ZO-9_]*
Identificador	isOperator	[a-zA-Z_][a-zA-ZO-9_]*
Identificador	tokenize	[a-zA-Z_][a-zA-ZO-9_]*
Identificador	main	[a-zA-Z_][a-zA-ZO-9_]*
Vocal	A, E, I, O, U	[AEIOUaeiou]

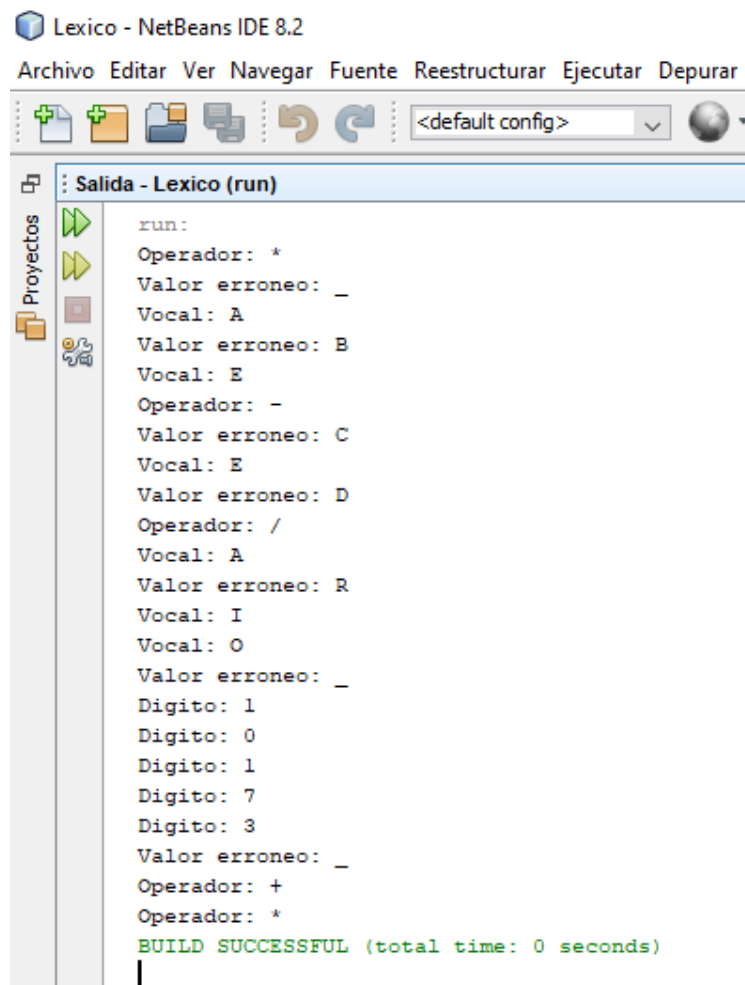
Dígito	0, 1, 2, ..., 9	[0-9]
Operador	+, -, *, /	[+\-*/]
Asignación	=	=
Comparación	==	==
Desigualdad	!=	!=
Punto	.	\.
Paréntesis	(,)	[\(\)]
Llaves	{, }	[\{\}]
Punto y coma	;	;
Coma	,	,
Comentario	//	//.*
Error	Cualquier otro	[^AEIOUaeiou0-9+\-*/\(\)\{\}\};\.,=!]

Capturas del código en java y sus ejecuciones.



```
1 package lexico;
2 //Autor: Ariana Muñoz Castillo 22-200-196
3
4 import java.util.ArrayList;
5 import java.util.List;
6
7 //Es un analizador lexico que determina la lectura valor por valor y determina si lo introducido si corresponde a
8 //solo letras de origen vocal y valores del 0 a 9
9 //Si se coloca una letra de las del abecedario las marca como error
10 public class Lexico {
11     // Token types
12     enum TokenType {
13         VOCAL, DIGITO, OPERADOR, ERROR
14     }
15     // Token class
16     static class Token {
17         TokenType type;
18         String value;
19
20         Token(TokenType type, String value) {
21             this.type = type;
22             this.value = value;
23         }
24     }
25     // Lexer class
26     static class Lexer {
27         private String input;
28         private List<Token> tokens;
29
30         Lexer(String input) {
31             this.input = input;
32             this.tokens = new ArrayList<>();
33         }
34
35         public List<Token> tokenize() {
36             for (char c : input.toCharArray()) {
37                 if (isVocal(c)) {
38                     tokens.add(new Token(TokenType.VOCAL, String.valueOf(c)));
39                 } else if (isDigit(c)) {
40                     tokens.add(new Token(TokenType.DIGITO, String.valueOf(c)));
41                 } else if (isOperator(c)) {
42                     tokens.add(new Token(TokenType.OPERADOR, String.valueOf(c)));
43                 } else {
44                     tokens.add(new Token(TokenType.ERROR, String.valueOf(c)));
45                 }
46             }
47             return tokens;
48         }
49
50         private boolean isVocal(char c) {
51             return "AEIOUaeiou".indexOf(c) != -1;
52         }
53
54         private boolean isDigit(char c) {
55             return "0123456789".indexOf(c) != -1;
56         }
57
58         private boolean isOperator(char c) {
59             return "+-*/".indexOf(c) != -1;
60         }
61     }
62
63     //Aquí es donde declaramos la entrada que va a analizar nuestro analizador lexico
64     public static void main(String[] args) {
65         String input = "A_BE-CED/ARIO_10173_+";
66
67         Lexer lexer = new Lexer(input);
68         List<Token> tokens = lexer.tokenize();
69
70         for (Token token : tokens) {
71             switch (token.type) {
72                 case VOCAL:
73                     System.out.println("Vocal: " + token.value);
74                     break;
75                 case DIGITO:
76                     System.out.println("Digito: " + token.value);
77                     break;
78                 case OPERADOR:
79                     System.out.println("Operador: " + token.value);
80                     break;
81                 case ERROR:
82                     System.out.println("Valor erroneo: " + token.value);
83                     break;
84             }
85         }
86     }
87 }
```

Es aquí donde se indica la frase a evaluar y determinar si esta cumpliendo con lo establecido.



The screenshot shows the NetBeans IDE 8.2 interface. The title bar reads "Lexico - NetBeans IDE 8.2". The menu bar includes "Archivo", "Editar", "Ver", "Navegar", "Fuente", "Reestructurar", "Ejecutar", and "Depurar". The toolbar contains icons for file operations and a configuration dropdown set to "<default config>". The "Proyectos" (Projects) sidebar on the left shows a tree view with a folder icon and a file icon. The main output window, titled "Salida - Lexico (run)", displays the following text:

```
run:
Operador: *
Valor erroneo: _
Vocal: A
Valor erroneo: B
Vocal: E
Operador: -
Valor erroneo: C
Vocal: E
Valor erroneo: D
Operador: /
Vocal: A
Valor erroneo: R
Vocal: I
Vocal: O
Valor erroneo: _
Digito: 1
Digito: 0
Digito: 1
Digito: 7
Digito: 3
Valor erroneo: _
Operador: +
Operador: *
BUILD SUCCESSFUL (total time: 0 seconds)
```

Inserto el código realizado para ejecutar en JAVA (netbeans)

- ✓ **Creo Proyecto, package y .java con el nombre de Lexico**

```
package lexico;

//Autor: Ariana Muñoz Castillo 22-200-196

import java.util.ArrayList;
import java.util.List;

//Es un analizador lexico que determina la lectura valor por valor y determina si lo
    introducido si corresponde a
//solo letras de origen vocal y valores del 0 a 9
//Si se coloca una letra de las del abecedario las marca como error
public class Lexico {
    // Token types
    enum TokenType {
        VOCAL, DIGITO, OPERADOR, ERROR
    }
    // Token class
    static class Token {
        TokenType type;
        String value;

        Token(TokenType type, String value) {
            this.type = type;
            this.value = value;
        }
    }
    // Lexer class
    static class Lexer {
        private String input;
        private List<Token> tokens;
```



```

Lexer(String input) {
    this.input = input;
    this.tokens = new ArrayList<>();
}

public List<Token> tokenize() {
    for (char c : input.toCharArray()) {
        if (isVocal(c)) {
            tokens.add(new Token(TokenType.VOCAL, String.valueOf(c)));
        } else if (isDigit(c)) {
            tokens.add(new Token(TokenType.DIGITO, String.valueOf(c)));
        } else if (isOperator(c)) {
            tokens.add(new Token(TokenType.OPERADOR, String.valueOf(c)));
        } else {
            tokens.add(new Token(TokenType.ERROR, String.valueOf(c)));
        }
    }
    return tokens;
}

private boolean isVocal(char c) {
    return "AEIOUaeiou".indexOf(c) != -1;
}

private boolean isDigit(char c) {
    return "0123456789".indexOf(c) != -1;
}

private boolean isOperator(char c) {
    return "+-*/".indexOf(c) != -1;
}
}

//Aqui es donde declaramos la entrada que va a analizar nuestro analizador lexico

```

```
public static void main(String[] args) {  
    String input = "*_ABE-CED/ARIO_10173_+*";  
  
    Lexer lexer = new Lexer(input);  
    List<Token> tokens = lexer.tokenize();  
  
    for (Token token : tokens) {  
        switch (token.type) {  
            case VOCAL:  
                System.out.println("Vocal: " + token.value);  
                break;  
            case DIGITO:  
                System.out.println("Digito: " + token.value);  
                break;  
            case OPERADOR:  
                System.out.println("Operador: " + token.value);  
                break;  
            case ERROR:  
                System.out.println("Valor erroneo: " + token.value);  
                break;  
        }  
    }  
}
```

Aquí concluye el código.