Manual Técnico Analizador Léxico

Curso: Lenguajes Formales y de Programación

Práctica 1

Autor: Julio Allan Ovalle Paz

Fecha: 28/08/2025

1. Introducción

Este documento describe la arquitectura, componentes y funcionamiento interno del Analizador Léxico implementado en Java. El objetivo es permitir que otros programadores comprendan, mantengan o extiendan el sistema.

2. Arquitectura del sistema

El proyecto está desarrollado en Java con el uso de Maven como gestor de dependencias y construcción. Se organiza en paquetes que separan las responsabilidades principales:

- Analizador léxico
- Tokens y errores
- Configuración (lectura de archivo JSON)
- Interfaz gráfica (Swing)

-Versión Java 21.0.04

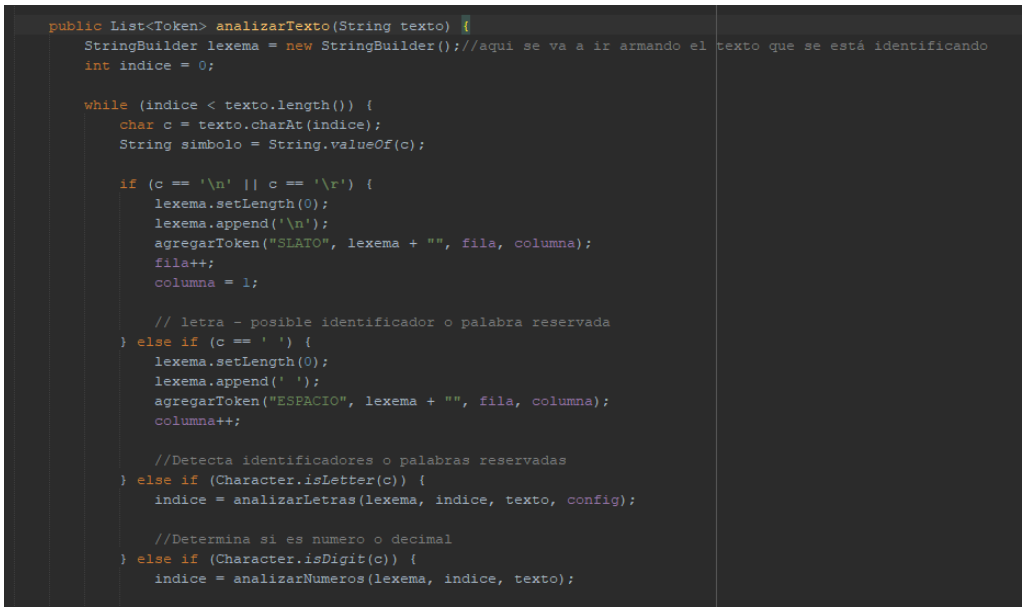
3. Descripción de componentes

Principales clases del sistema:

- Token: Representa cada unidad léxica reconocida.

```java
package Logica;

public class Token {
    private String tipo;
    private String lexema;
    private int fila,columna;

    public Token(String tipo, String lexema, int fila, int columna) {
        this.tipo = tipo;
        this.lexema = lexema;
        this.fila = fila;
        this.columna = columna;
    }
```

- Analisis: Contiene la lógica de análisis, reconoce números, decimales, cadenas, comentarios, etc.



```java
public List<Token> analizarTexto(String texto) {
    StringBuilder lexema = new StringBuilder();//aqui se va a ir armando el texto que se está identificando
    int indice = 0;

    while (indice < texto.length()) {
        char c = texto.charAt(indice);
        String simbolo = String.valueOf(c);

        if (c == '\n' || c == '\r') {
            lexema.setLength(0);
            lexema.append('\n');
            agregarToken("SLATO", lexema + "", fila, columna);
            fila++;
            columna = 1;

            // letra - posible identificador o palabra reservada
        } else if (c == ' ') {
            lexema.setLength(0);
            lexema.append(' ');
            agregarToken("ESPACIO", lexema + "", fila, columna);
            columna++;

            //Detecta identificadores o palabras reservadas
        } else if (Character.isLetter(c)) {
            indice = analizarLetras(lexema, indice, texto, config);

            //Determina si es numero o decimal
        } else if (Character.isDigit(c)) {
            indice = analizarNumeros(lexema, indice, texto);
```

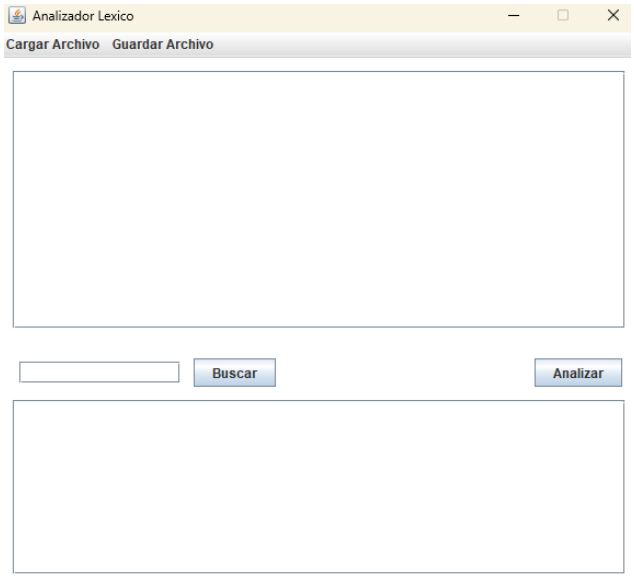
- Configuracion: Lee el archivo JSON que define los tokens válidos.

```java
public class Configuracion {//Clase que se encarga de cargar el archivo config.json

    private List<String> palabrasReservadas = new ArrayList<>();
    private List<String> operadores = new ArrayList<>();
    private List<String> puntuacion = new ArrayList<>();
    private List<String> agrupacion = new ArrayList<>();
    private String comentarioLinea;
    private String comentarioBloqueInicio;
    private String comentarioBloqueFin;
```

```java
public void cargarJSON() {

    //Obtiene la direccion del archivo .json situado en el proyecto
    try (InputStream is = getClass().getResourceAsStream("/JSON/config.json")) {
        if (is == null) {
            throw new RuntimeException("No se encontró config.json");
        }

        // Leer el archivo como String
        Scanner scanner = new Scanner(is, StandardCharsets.UTF_8.name());
        String contenido = scanner.useDelimiter("\\A").next();
        scanner.close();

        // Convertir a objeto JSON
        JSONObject json = new JSONObject(contenido);

        // Cargar los tokens del archivo
        palabrasReservadas = cargarLista(json.getJSONArray("palabrasReservadas"));
        operadores = cargarLista(json.getJSONArray("operadoresLogicos"));
        puntuacion = cargarLista(json.getJSONArray("signosPuntuacion"));
        agrupacion = cargarLista(json.getJSONArray("signosAgrupacion"));

        JSONObject comentarios = json.getJSONObject("comentarios");
        comentarioLinea = comentarios.getString("linea");
        comentarioBloqueInicio = comentarios.getString("bloqueInicio");
        comentarioBloqueFin = comentarios.getString("bloqueFin");

    } catch (Exception e) {
        System.out.println("Error leyendo el archivo config.json: " + e.getMessage());
    }
}
```

Método encargado de cargar el archivo “config.json” para luego obtener todos los tipos de tokens y así ayudar a la clase Analisis a leer los tokens

- Interfaz: Implementada en Swing (JFrame, JTextPane) con coloreado de palabras usando StyledDocument.



- CrearReportes: Recibe una lista con los tokens analzados y luego los procesa para determinar que tipo de reporte mostrar.

```
public void filtrarLista(List<Token> tokens) {
    boolean errores = false;

    for (Token t : tokens) {

        if ("ERROR".equals(t.getTipo())) {
            errores = true;
            listaErrores.add(t);
        } else {
            listaTokens.add(t);//listado con tokens validos
        }

    }
    if (errores) {//Si hay al menos un error
        mostrarErrores(listaErrores, jfErrores.getTxtArea());

    } else {// Si no hay errores
        mostrarTokens(listaTokens, jfTokens.getTxtAreaReporte(), jfTokens.getTxtAreaRecuento());
    }

    //Muestra el reporte general
    crearReporteGeneral();
}
```

Método que verifica si hay al menos un error para saber que tipo de reporte mostrar.

```
public void mostrarErrores(List<Token> errores, JTextArea areaReporte) {
    StringBuilder reporte = new StringBuilder();
    reporte.append("------------------------------------------------------------\n");
    reporte.append(String.format("| %-15s | %-7s | %-10s |\n", "Simbolo o cadena", "Fila", "Columna"));
    reporte.append("------------------------------------------------------------\n");

    for (Token e : errores) {

        reporte.append(String.format("| %15s  (%-3d , %-5d) \n",
                e.getLexema(), e.getFila(), e.getColumna()));

    }
    reporte.append("------------------------------------------------------------\n");

    //  Mostrar en el JTextArea
    areaReporte.setText(reporte.toString());

    //  Guardar en archivo .txt
    try (PrintWriter writer = new PrintWriter(new FileWriter("errores.txt"))) {
        writer.write(reporte.toString());
        System.out.println("Reporte de errores generado en errores.txt");
    } catch (IOException ex) {
        System.out.println("ERROR AL EXPORTAR REPORTE DE ERRORES " + ex.getMessage());
    }

    jfErrores.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
    jfErrores.setVisible(true);
}
```

Método que recibe una lista con los tokens ya analizados para mostrarlos en pantalla para luego escribirlos e importarlos en un archivo de texto.

4. Flujo del programa

1. El usuario ingresa o carga un archivo de texto.
2. El texto se envía al analizador léxico.
3. El analizador genera listas de tokens válidos y errores.
4. La interfaz gráfica colorea los tokens en tiempo real.
5. El usuario puede exportar los resultados a reportes.

5. Configuración del proyecto

El proyecto utiliza Maven para gestionar dependencias y construir el artefacto JAR. La dependencia principal es org.json para la lectura de archivos JSON.
Además, se configuró el plugin maven-shade-plugin para empaquetar todas las librerías en un JAR ejecutable.

6. Manejo de errores

Cuando el analizador encuentra una secuencia de caracteres inválida, genera un objeto Token con “tipo” error, que registra la fila, columna y el lexema inválido. Estos errores se muestran en la interfaz y se exportan a reportes.

7. Exportaciones y reportes

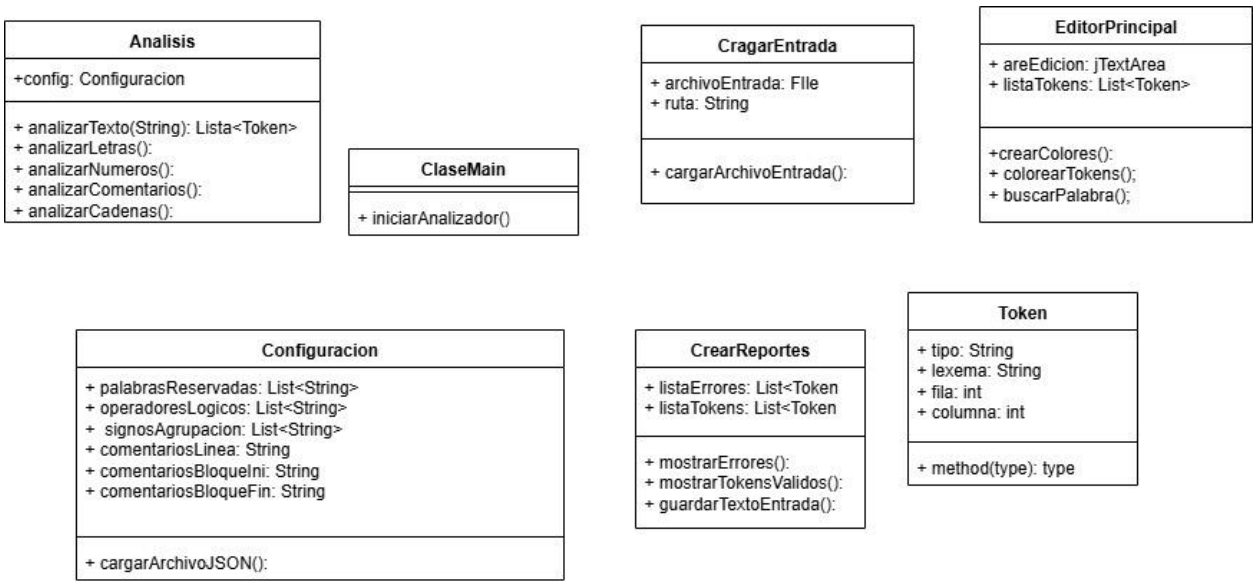
El sistema permite exportar:

- Texto de entrada en un archivo .txt

- Tokens reconocidos en .csv

- Errores encontrados en .csv

8. Diagrama de clases



9. Conclusiones

El analizador léxico implementado cumple con la detección de tokens básicos y manejo de errores. Como mejoras futuras se recomienda optimizar el reconocimiento de tokens con expresiones regulares y ampliar la compatibilidad con más tipos de lenguajes.