Manual técnico

PRACTICA_1_LENGUAJES

CENTRO UNIVERSITARIO DE OCCIDENTE

CUNOC

Desarrollador: Anthony Gutiérrez

28/08/2025

1. Información General

Nombre del Programa: Practica_1_lenguajes
Versión: 2.0
Descripción:
Esta es una aplicación para el análisis léxico de texto, puede reconocer patrones de texto y errores mediante la creación de tokens de distintos tipos definidos por un archivo Json que tiene las instrucciones para reconocer todos los patrones que se evalúan en el programa, puede buscar cadenas de texto introducidas por el usuario y generar reportes de la cantidad de tokens validos o errores que se obtengan tras analizar el texo.

En la actualización 2.0 se hagrega una función similar a la de analizar texto pero esta vez utilizando un automata y la capacidad de generar diagramas AFD para el texto evaluado

2. Requisitos del Sistema

Hardware

- **Procesador:** Intel Core i3 o superior
- **Memoria RAM:** 4GB o más
- **Almacenamiento disponible:** -- KB o más
- **Tarjeta gráfica:** Compatible con Java

Software

- **Sistema Operativo:** Windows, macOS o Linux
- **Java Development Kit (JDK):** Versión 21 o superior

3. Instalación

1. **Descargar el programa:**
   Asegúrate de que los archivos del programa estén disponibles en tu computadora.
2. **Instalar Java:**
   Descarga e instala la última versión del Java Development Kit (JDK) desde la página oficial de Oracle o desde el gestor de paquetes de tu sistema operativo.
3. **Ejecutar el programa:**

   Entra a la terminar de tu ordenador y ejecuta el siguiente comando dentro de la carpeta target del archivo
   Practica_1_lenguajes-1.0-SNAPSHOT-jar-with-dependencies

2. Clases Principales

- **Practica_1_lenguajes.java**

Esta clase es la que lleva el método main, es imprescindible para el programa.

- **FileHandler**

Esta clase es la encargada de cargar y guardar los archivos de texto hacia el programa.

1. *public void buscarArchivo()*
2. *private String getFileContent()*
3. *public void sobreescribirContenido(String nuevoConetnido)*
4. *public void guardarComo(texto)*

- **TextPainter**

Esta clase es la encargada de crear los estilos para el JTextPane y también la encargada de "colorear" el pane, ya sean las letras o el fondo.

- **LexicalAnalyzer**

Este es el método principal del programa, pues todo gira en torno a el, es el encargado de analizar carácter por carácter del texto e ir evaluando si pertenece a algún tipo de los indicados en el JSon, al ir analizando crea los tokens en un arrayList dándoles todo lo necesario para existir.

1. *Analizar(String archivo, Textpane textPane)*
2. *Varias clases de boleanos que comprueban si los lexemas pertenecen a algo*

- **WordFinder**

Clase encargada comparar carácter por carácter lo que el usuario esta buscando con el teto en el pane.

- **ConfigLoader**

Esta clase es la encargada de leer el JSON que se encuentra en src/main/resources y lo carga en la clase **Config.java** la cual es la encargada de encapsular todos los datos del json y así poder ser utilizados en el analizador léxico.

- **Token**

Esta clase solo se encarga de encapsular todos los datos del token, como el tipo (**TokenTipo.java; es un enum**) o la clase **Posicion.java** que es la encargada de generar la posición del lexema dentro del pane

- **ReportesExporter**

Esta clase se encarga de guardar los reportes en un archivo CSV y lo guarda en el dispositivo.

- **Frontend**

Las calses encargadas de mostrar visualmente lo que sucede al usuario.

3. Actualización 2.0

- **Automata**
    1. **Automata(textPane1, textPane2):**
       método constructor, es el encargado de iniciar correctamente el arreglo de transiciones que se usaran durante el análisis, también inicia los paneles que se van a utilizar para mostrar los resultados.
    2. **Analizar(String text);**
       es el el método principal, recibe un texto lo normaliza y lo analiza carácter por carácter para formar un lexema y posteriormente identificar si esta en un estado de aceptación o es un error. También pinta el panel principal para mostrar los textos.
    3. **reiniciarAutomata(String, int, Posicion);**
       este es el método encargado de reiniciar el automata cuando se termina un análisis o cuando se encuentra un error.
- **AutomataFrontend**

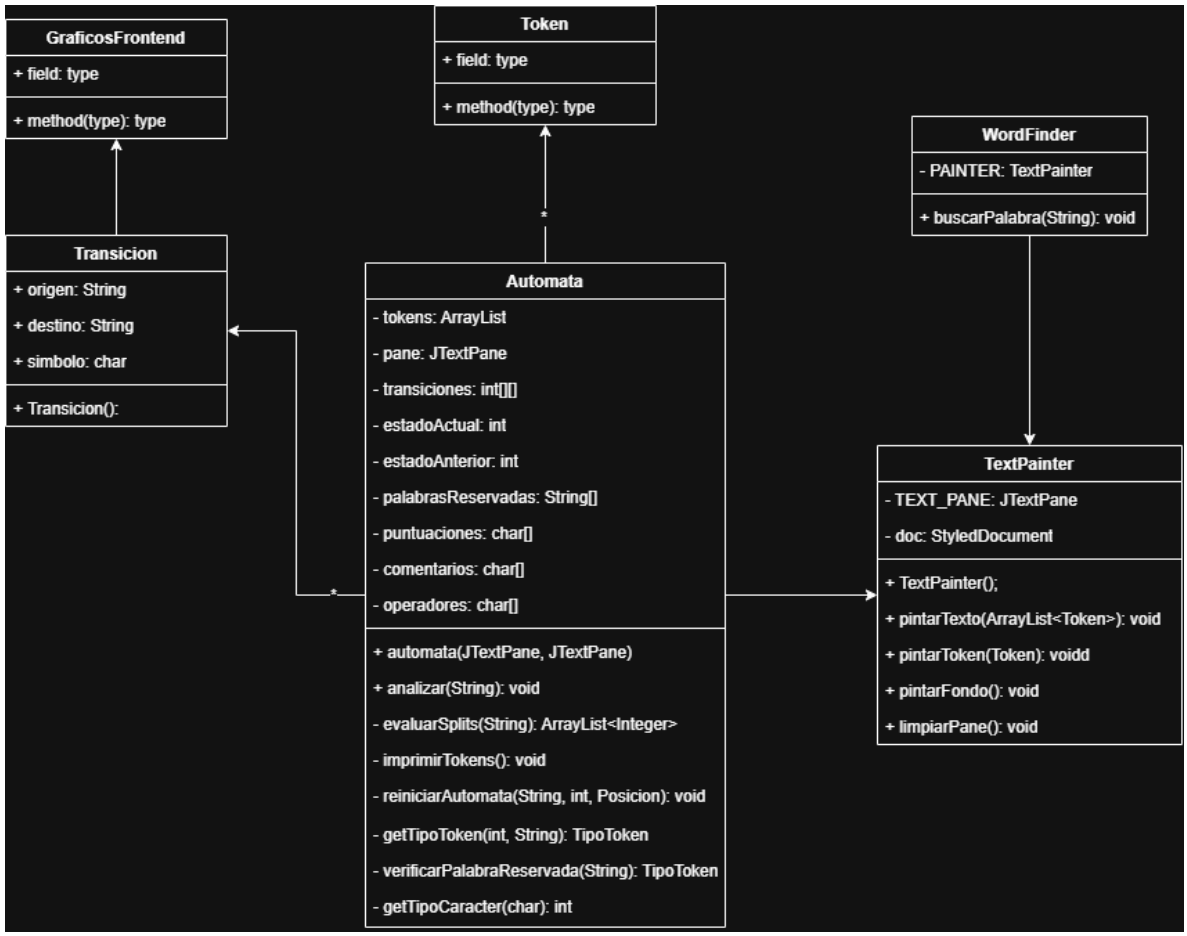
Es el encargado de mostrar el frontend para que se muestre el texto, la terminar y el resto de opciones sobre el analizador.

- **FrameGrafo y PanelGrafico**

Ambos en el frontend son los encargados de mostrar el diagrama afd que se crea tras un análisis

Diagrama de clases

1. Diagrama de actualización 2.0


GraficosFrontend
+ field: type
+ method(type): type
Token
+ field: type
+ method(type): type
WordFinder
- PAINTER: TextPainter
+ buscarPalabra(String): void
Transicion
+ origen: String
+ destino: String
+ simbolo: char
+ Transicion():
Automata
- tokens: ArrayList
- pane: JTextPane
- transiciones: int[][]
- estadoActual: int
- estadoAnterior: int
- palabrasReservadas: String[]
- puntuaciones: char[]
- comentarios: char[]
- operadores: char[]
+ automata(JTextPane, JTextPane)
+ analizar(String): void
- evaluarSplits(String): ArrayList<Integer>
- imprimirTokens(): void
- reiniciarAutomata(String, int, Posicion): void
- getTipoToken(int, String): TipoToken
- verificarPalabraReservada(String): TipoToken
- getTipoCaracter(char): int
TextPainter
- TEXT_PANE: JTextPane
- doc: StyledDocument
+ TextPainter();
+ pintarTexto(ArrayList<Token>): void
+ pintarToken(Token): voidd
+ pintarFondo(): void
+ limpiarPane(): void
*
*