Universidad de San Carlos de Guatemala

Centro Universitario de Occidente

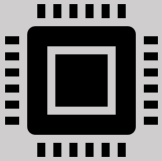
División de Ciencias de la ingeniería

Ingeniería

Lenguajes Formales y de Programación Sección “A”

Ing. Daniel González

Segundo Semestre 2025

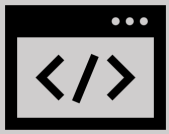


Estudiante:

Pablo Alejandro Maldonado de León

Carné: 202430233

Manual Técnico Proyecto #2 2025



Descripción:

En el siguiente manual se le describirá paso a paso como fue el proceso para la realización del programa, detallando los algoritmos más importantes como diagramas de clase, explicación del funcionamiento del programa y funcionalidades del programa, describiendo la importancia y el uso que se le da al momento de ejecutar el programa.

"Bienvenido programador"

Introducción:

En el presente manual se le mencionaran aspectos importantes sobre cómo fue estructurado el código fuente del proyecto, al igual que las técnicas utilizadas para permitir inicializar ciertos procesos que permiten la interacción del usuario, sobre todo las clases utilizadas para reducir la cantidad de código en las clases. Al igual que resaltar algunas validaciones importantes para disminuir al máximo los posibles errores que se puedan causar por el usuario.

Características del programa

- **Creado en:** NetBeans 22
- **SO utilizado:** Ubuntu 22.04
- **Lenguaje:** JAVA.
- **Versión JDK:** 21.01
- **Técnicas utilizadas:** Programación Orientada a Objetos Avanzada, uso de Estructuras dinámicas y expresiones regulares
- **Interfaz Gráfica:** Java Swing. Elaboración de backend y frontend.
- **Componentes utilizados:** JTable, JButton, JTextField, JLabel, JPanel, JFrame, JTextArea, JTextPane y JFileChooser.
- **Tipo de archivo de lectura:** .txt
- **Nombre del ejecutable:** "java -jar ProyectoNo2LFP-1.0-SNAPSHOT.jar"

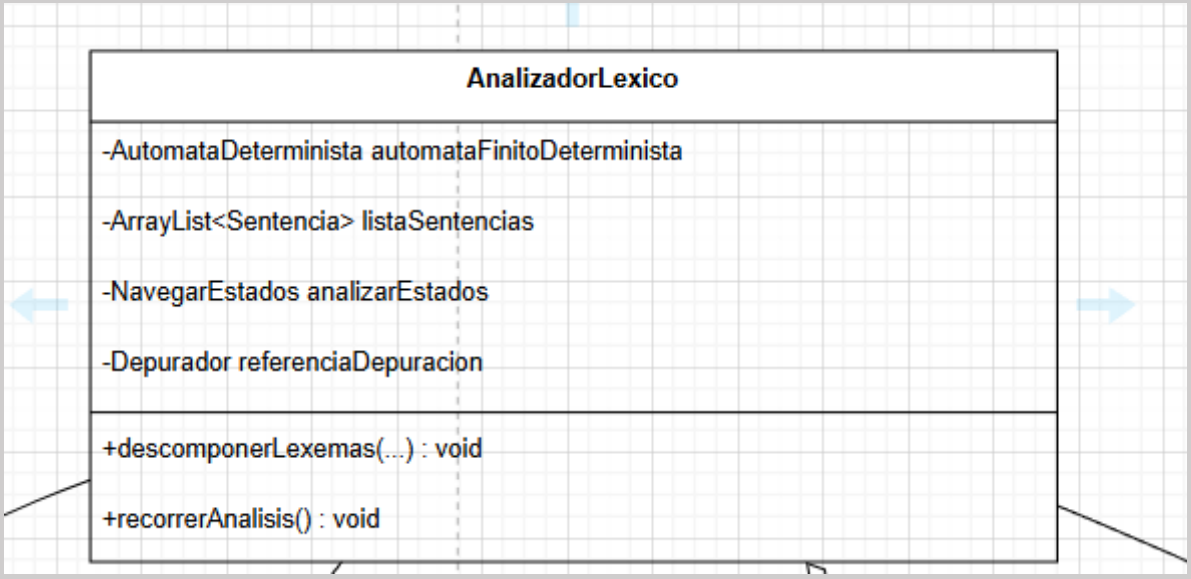
Es importante resaltar que las rutas de ciertos iconos estan ubicadas de forma relativa al proyecto, sin embargo utiliza directorios basados en unix. Por lo tanto si se ejectua en Windows, el programador es responsable de cambiar las rutas para poder visualizar los iconos. Pero esto no se vera afectado de ninguna forma en el procesamiento de la aplicación.

Diagramas de clases

En el siguiente apartado se le da a conocer al usuario todas las clases más importantes utilizadas para llevar a cabo el proyecto y poder desarrollarlo. Se le presenta la relación de todas las clases plasmadas como objetos utilizados para brindar un mejor acoplamiento y aumentar la cohesion del código y las clases más importantes que permiten armar la clase de JFlex que permite leer por completo las sentencias y poder interpretar el código para poderlo tokenizar y clasificar el tipo de lexemas que se le va ingresando.

Analizador Lexico:

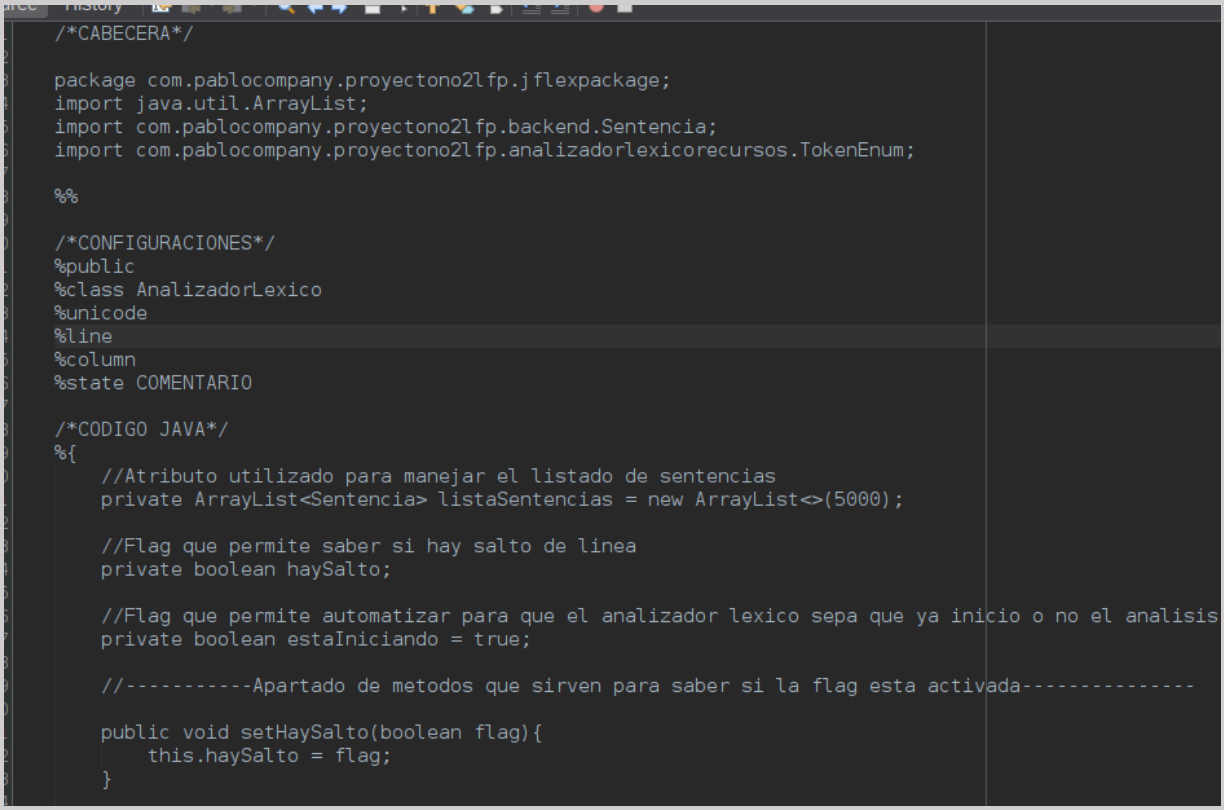
Es la clase que permite analizar por completo todas las sentencias y que esten gramaticalmente escritas correctamente lo que permite poder interpretar las sentencias, en base a métodos de lectura que permiten visualizar y poder interpretar que los lexemas que se ingresen o se escriban se vayan clasificando y tokenizando a tiempo real.



Este analizador lexico esta montado gracias a Jflex y su fácil estructura para poder generar los autómatas y poder llevar a cabo un analizador lexico de una forma más comoda en base a expresiones regulares.

Uso de Jflex:

Este es un archivo de texto tipo .flex que permite generar un Autómata finito determinista en base a expresiones regulares. Que al momento de compilar el archivo este autómata funcionará de tal forma que reconoce automáticamente todo tipo de tokens sin importar si están espaciados o no.



```
/*CABECERA*/

package com.pablocompany.proyectono2lfp.jflexpackage;
import java.util.ArrayList;
import com.pablocompany.proyectono2lfp.backend.Sentencia;
import com.pablocompany.proyectono2lfp.analizadorlexicorecursos.TokenEnum;

%%

/*CONFIGURACIONES*/
%public
%class AnalizadorLexico
%unicode
%line
%column
%state COMENTARIO

/*CODIGO JAVA*/
%{
    //Atributo utilizado para manejar el listado de sentencias
    private ArrayList<Sentencia> listaSentencias = new ArrayList<>(5000);

    //Flag que permite saber si hay salto de linea
    private boolean haySalto;

    //Flag que permite automatizar para que el analizador lexico sepa que ya inicio o no el analisis
    private boolean estaIniciando = true;

    //-----------Apartado de metodos que sirven para saber si la flag esta activada---------------

    public void setHaySalto(boolean flag){
        this.haySalto = flag;
    }
```

Expresiones regulares registradas:

- **LETRAS**
- **NUMEROS**
- **PALABRAS RESERVADAS**
- **SIGNOS DE PUNTUACION**
- **AGRUPACION**
- **OPERADORES MATEMATICOS**
- **COMENTARIOS DE LINEA**
- **CIERRE DE COMENTARIOS DE BLOQUE**
- **APERTURA COMENTARIOS DE BLOQUE**

ES MUY IMPORANTE RESALTAR QUE ESTAS EXPRESIONES NO PUEDEN SER ELIMINADAS YA QUE ESTA COMPOSICION DE EXPRESIONES ESTA BASADA EN LOS TOKENS. SI SE DESEA AGREGAR ALGO A LA GRAMATICA HAY QUE SER CONSCIENTE Y REALISTA CON LO QUE SE DEJARA A DISPOSICION EN LA NUEVA GRAMATICA.

Expresiones regulares utilizadas:

```
/*EXPRESIONES REGULARES DEL AUTOMATA*/

Espacio = " "
Tab = \t

Salto = (\r\n | \n | \r)

LineaVacia = [ \t]* {Salto}+



/* Apartado de tokens normales */
Identificador = [:jletter:] [:jletterdigit:]*
Numero = 0 | [1-9][0-9]*
Decimal = {Numero}"."{Numero}
Puntuacion = [.,;:]
OperadorAritmetico = [+\-*/%]
Agrupacion = [\(\)\[\]\{\}]

CadenaTexto = \"([^\"\n\r])*\"

/*Apartado de palabras reservadas */

PalabraReservada = "SI" | "si" |"ENTONCES" | "entonces" |"ENTERO" | "entero" |"NUMERO" | "numero" |"CADENA" | "cadena" |"ESCRIBIR" | "es
```

Reconocimiento de palabras reservadas con significado sintactico:

Estas son expresiones regulares que permiten poder reconocer ese conjunto de caracteres cuyo contexto es muy importante en el analisis sintactico por lo tanto se generaron excepciones cuya expresion regular tiene un significado sintactico para poder armar expresiones.

```
Igual              = "="
Como               = "COMO"
Definir            = "DEFINIR"
PuntoComa          = ";"

Entero             = "entero"
Cadena             = "cadena"
NumeroEspecial     = "numero"

Escribir           = "ESCRIBIR"
ParentesisCierre   = ")"
ParentesisApertura = "("


Suma = "+"
Resta = "-"
Multiplicacion = "*"
Division = "/"
```

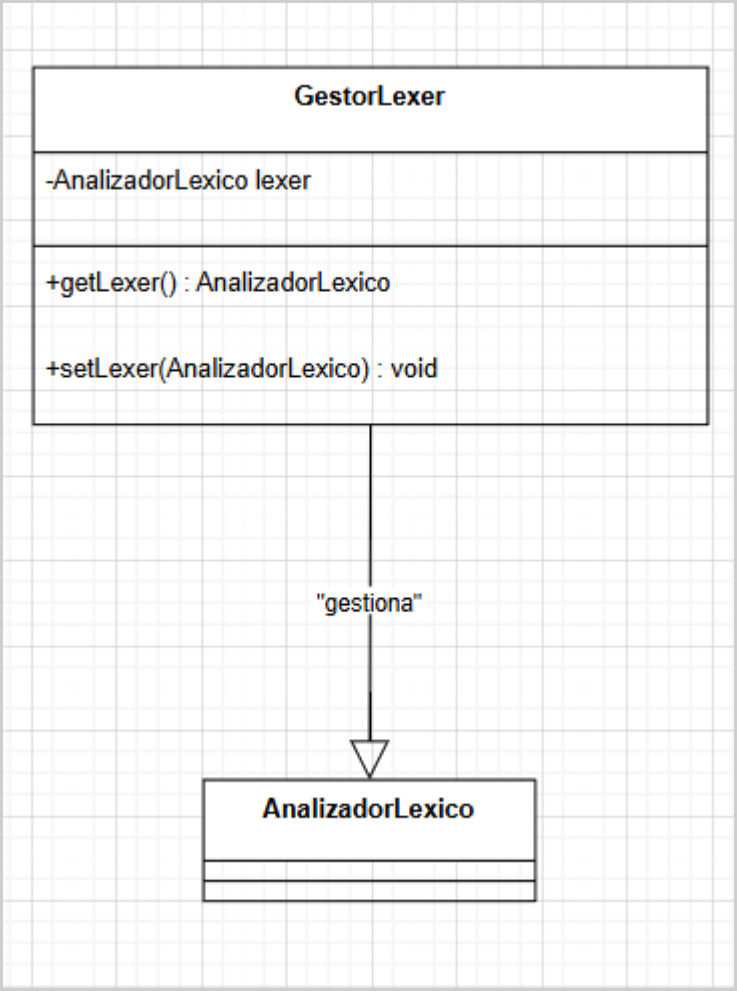
ESTA ES UNA DE LAS CLASES MAS GRANDES QUE EXISTEN POR LO TANTO SE LE SUGIERE AL PROGRAMADOR NUNCA TOCAR LA CLASE QUE JFLEX GENERA, A NO SER QUE CONSULTE LA DOCUMENTACION OFICIAL DE JFLEX.

Se le advierte al programador lector de esta documentacion que evite tocar esta clase puesto a que esta se genera sola siempre y cuando se defina de forma correcta la tactica para poder reconocer los diferentes tipos de tokens.

Jerarquia de clases para el desarrollo de los analizadores

Gestor Lexer:

Esta es una de las clases mas importantes no por su complejidad. Sino porque esta genera un juego de referencias que permite comunicar a las demás funcionalidades que son posteriores al análisis. Ya que esta conserva la referencia viva del Analizador Lexico para poder usar todos sus recursos desde las otras funcionalidades dependientes de su análisis.



Kernel del analizador léxico

Al momento de llevar a cabo el planteamiento de realización se pensó una estrategia base para brindar modularidad y control completo de cada movimiento del AFD una serie de clases cuya estructura dan vida a todas las funcionalidades. De tal forma que se puede confirmar con total seguridad que si en dado caso se requiere remasterizar por completo la aplicación teniendo la lógica de estas 3 clases se puede montar un analizador léxico e ir brindando las capacidades que este debe poseer para recuperarse de errores o saber en todo momento en que fila y columna va.

Jerarquía de Análisis:

- **Sentencia**
- **Lexema**

JERARQUIA DE PROCESAMIENTO LEXICO-SINTACTICO

PROCESO LEXICO:

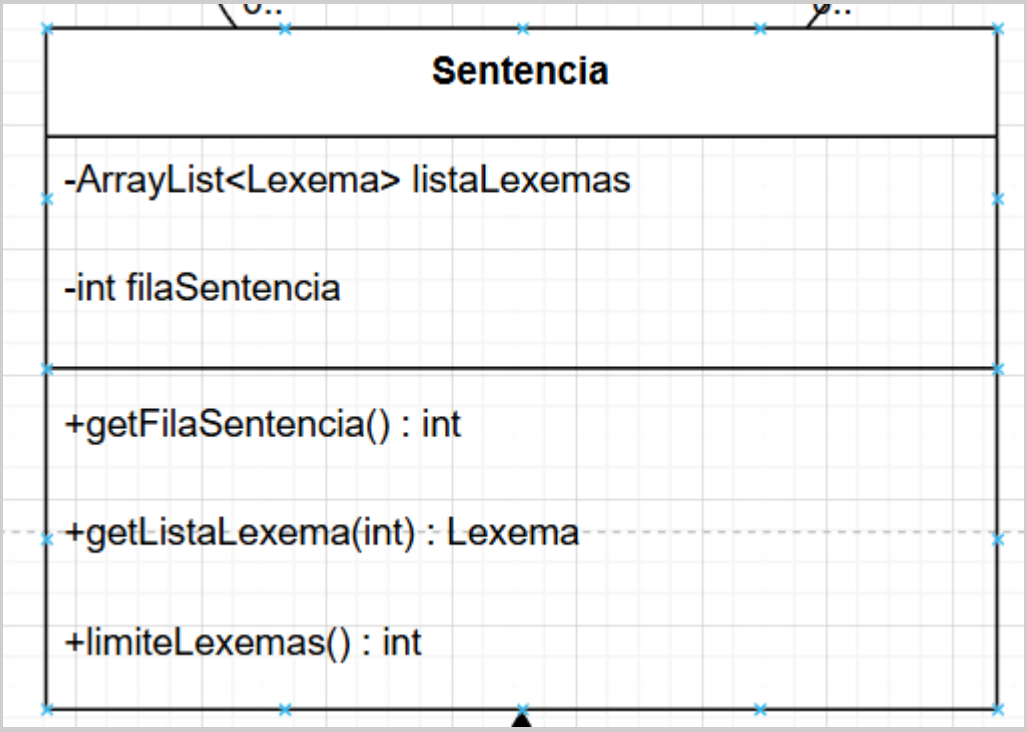
- Tokenizacion y generacion del listado de tokens. Se basa en seprar o crear nuevas columnas en cuanto se encuentra un salto de linea es decir se da un **“Enter”.**
- Recorrer la lista en busqueda de posibles errores generados por JFlex.

PROCESO SINTACTICO:

- Analizar en búsqueda de palabras reservadas DEFINIR y clasificar las variables que se plantean de los diferentes tipos existentes y disponibles
- Analizar el texto en búsqueda de expresiones. Cuyo valor sea la estructura definida para generar una variable
- Busqueda de la palabra reservada para poder escribir las sentencias o mostrar en pantalla el contenido que se encuentra especificado dentro de la expresión definida.
- Buscar errores y en caso contrario de no haber mostrar todo correctamente

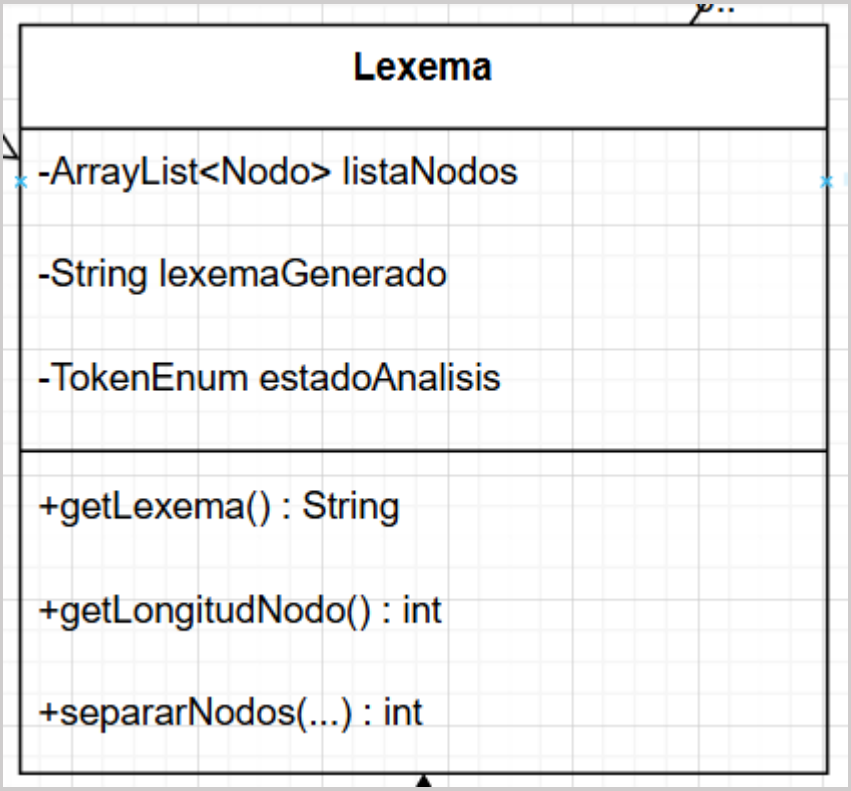
Sentencia:

Como su mismo nombre lo dice esta es encargada de manejar una sentencia por completo. Sin embargo, esta tiene la capacidad de conocer **por fila** cual es la serie de lexemas que este posee por lo tanto solo basta con ir manejando tal cual por fila el análisis. Es decir, permite simular la misma entrada. Leyendo por filas cada sentencia.



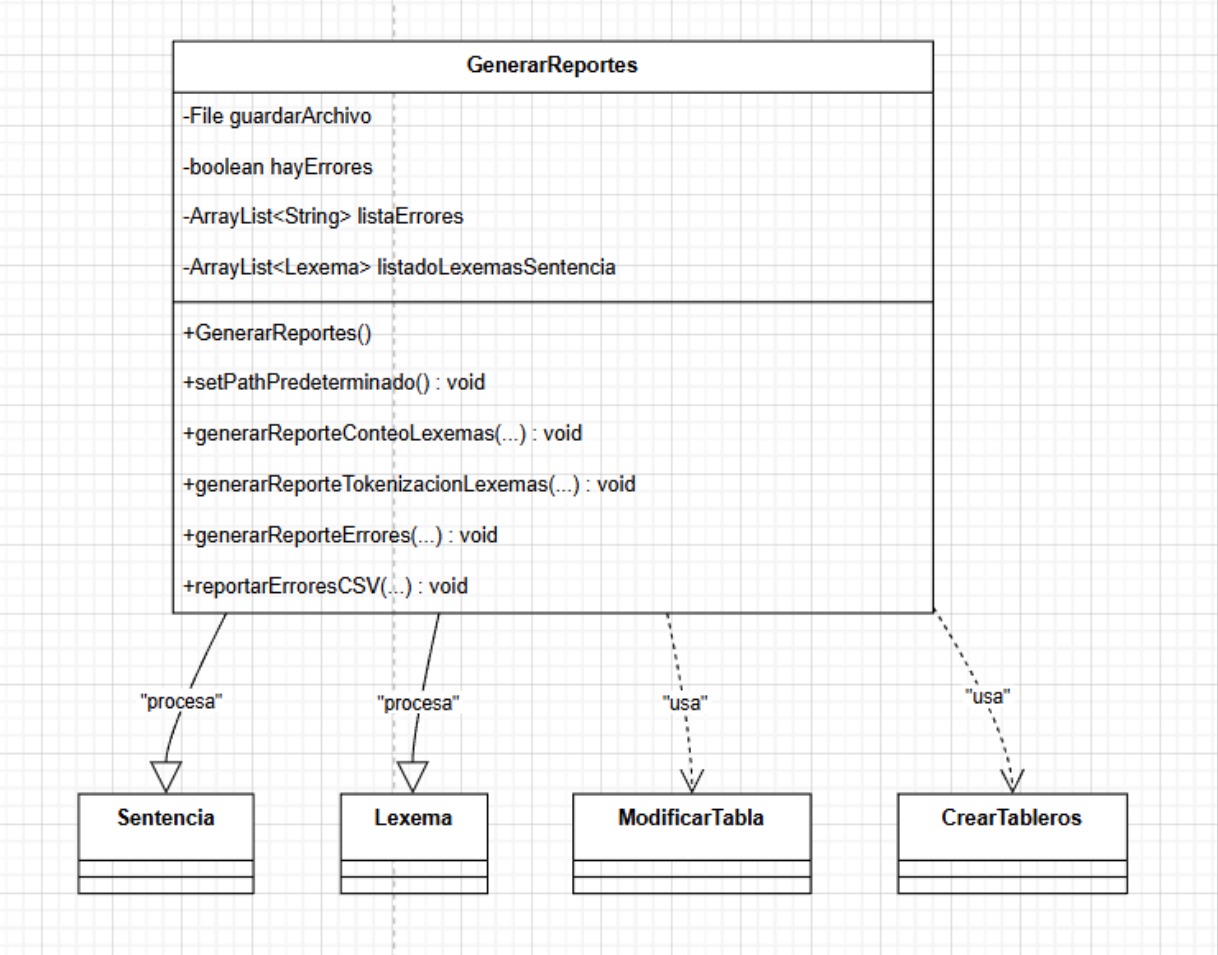
Lexema:

Como su mismo nombre lo dice esta es encargada de conocer el lexema, es decir la palabra sola brindando la facilidad que **Cada índice del listado de lexemas que contiene la sentencia sea una palabra completa**. Permitiendo de esta forma llevar a cabo un fácil manejo y evaluación de la palabra que se encuentra dentro de cada índice de la sentencia.



Reportes del analizador léxico

Esta es la ultima de las clases importantes del Analizador léxico. Se basa solamente en la interpretación de las sentencias, lexemas y nodos. Para poder determinar los reportes que generara. Por ello la importancia y la decisión de montar de esta forma el analizador lexico ya que permite modularidad y conocimiento completo de cada uno.



Decisiones de diseño del analizador léxico

Para justificar el código y funcionamiento del mismo se deja este apartado para que el programador sea consciente y sepa el motivo del porque se llevo a cabo de esta forma. Y cuáles son las proyecciones para continuar con el desarrollo de la aplicación.

- **¿Por qué la composición se basa en 2 clases principales?**

Desde un principio se busco poder diseñar un analizador léxico lo mas practico posible de implementar. Por lo tanto, para evitar dolores de cabeza se planteo que el kernel del analizador léxico sean 2 clases cuyo valor es tan grande que hace funcionar el analizador léxico.

- **¿Porque la interfaz se fragmenta en diferentes logs de salida?**

El motivo de estos logs de salida es permitir que el usuario pueda visualizar y tener consciencia de como es que funciona el analizador léxico. De esta forma se puede ilustrar para aquellos que la consuman el principio de un analizador léxico.

- **¿Por qué un modo depuración?**

El motivo del modo depuración es ilustrar como funciona paso a paso el análisis léxico. Es decir que el fin de este es ilustrar por completo cada movimiento que va realizando en el instante el analizador léxico para poder clasificar un lexema.