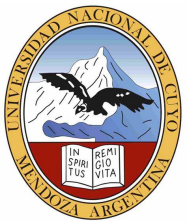
Analizador Léxico



Flores, D. E.[1]
[1]Facultad de Ingeniería
Universidad Nacional de Cuyo

26 de abril de 2022

1. Introducción

En el presente informe se expone características del analizador léxico desarrollado. El mismo, desarrollado en Java, pretende emular el comportamiento deseado por la cátedra aunque con ligeras particularidades que se espera no afecten el desarrollo de etapas posteriores.

2. Diseño

En un principio, el algoritmo general iba a estar conformado por dos clases: *Ejecutador* y *AnalizadorLexico* (clase principal). Sin embargo, a medida que el desarrollo avanzaba, se optó por modularizar más la clase *AnalizadorLexico*; están presentes diferentes módulos que asisten a la clase principal (*Manager*, *CommentsManager*, etc.), evitando redundancia de código, aislamiento de errores y manejo de avanzado de lexemas.

A tal efecto, existen otras clases como *Lexema*, *Token* y *Location* cuya única función es proporcionar objetos que retornen dos o más valores relevantes a la función que lo requiera.

Por otro lado, el código se encuentra comentado en partes que se creen fundamentales. Se exportó el contenido con Doxygen y es accesible desde *./doc/index.html*.

2.1. Errores

Únicamente existen dos clases que heredan de *Exception*: *NoSuchTokenException* y *IllegalTokenException*. La primera se lanza al detectar un fin de archivo inesperado (por ejemplo, no cierre de bloque de comentarios) y es de tratamiento interno (no será visible al usuario). La segunda excepción se lanza al detectar errores en formación de lexemas, y reporta adecuadamente la primera excepción descrita.

Como mensajes a tener en cuenta, se encuentran:

- *"Token not valid"*. Cuando se espera un lexema de doble símbolo obligatorio, pero solo encuentra uno (Por ejemplo, ||).

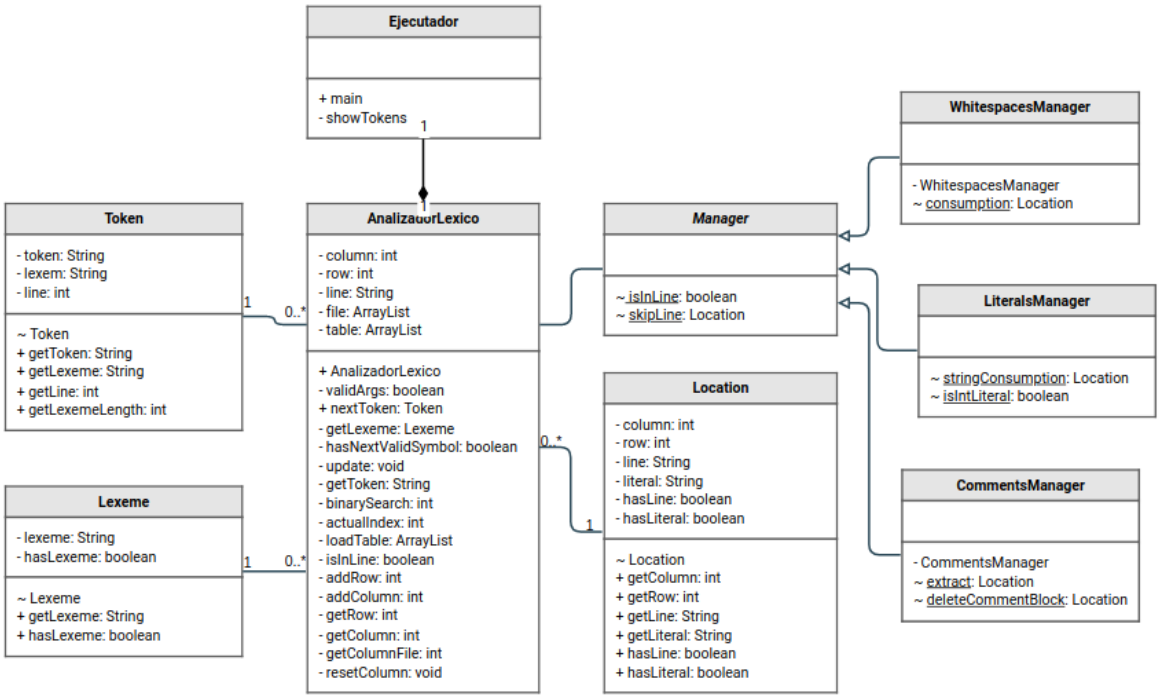


Figura 1. Diagrama de clases para el proyecto Analizador Léxico

- *"Invalid character"*. Cuando se encuentra un caracter que no es aceptado. Generalmente ocurre con símbolos que pueden estar en un literal string pero no en un identificador, por ejemplo.

- *"Invalid escape sequence in literal"*. Se presenta cuando no es posible asociar correctamente el caracter \con otro caracter admitido en un literal.

- *"Unterminated string literal"*. No se halla la comilla doble que indica fin del literal string en la misma línea del literal.

- *"Illegal char literal"*. No se encuentra la comilla simple indicadora de fin del literal character.

3. Características

3.1. Alfabeto de entrada

El alfabeto de entrada está dado por el conjunto:

$$\Sigma = \{0, 1, 2, ..., 9, a, b, c, ..., z, A, B, C, ...Z, \_, , , ., :, ; , \\ ", ', +, -, *, /, <, >, =, \{, \}, (,), [,], !, \#, \$, \%, \&, @, \wedge, \sim, \backslash, \ddot{}, EOF\} \cup Restantes$$

donde $Restantes$ es aquel conjunto descrito por todos los símbolos que puedan ser generados por una computadora.

3.2. Tokens reconocidos

Se presentan los tokens reconocidos –junto a la expresión regular que los define– por el analizador léxico:

$$
\begin{array}{rcl}
lbrace & | & \{ \\
rbrace & | & \} \\
lparent & | & (\\
rparent & | &) \\
semicolon & | & ; \\
colon & | & : \\
comma & | & , \\
dot & | & . \\
mod & | & \% \\
assign & | & = \\
lbracket & | & [\\
rbracket & | &] \\
plus & | & + \\
minus & | & - \\
ast & | & * \\
div & | & / \\
excmark & | & ! \\
less & | & < \\
greater & | & > \\
leq & | & <= \\
geq & | & >= \\
or & | & || \\
and & | & \&\& \\
noteq & | & != \\
equal & | & == \\
if & | & if
\end{array}
$$

$$
\begin{array}{rcl}
null & | & nil \\
var & | & var \\
new & | & new \\
int & | & Int \\
init & | & init \\
self & | & self \\
void & | & void \\
func & | & func \\
else & | & else \\
char & | & Char \\
true & | & true \\
bool & | & Bool \\
false & | & false \\
array & | & Array \\
while & | & while \\
class & | & class \\
static & | & static \\
string & | & String \\
return & | & return \\
private & | & private \\
charlit & | & '\bullet(a+b+c+...+z+A+B+C+...+Z+ \\
 & | & 0+1+...+9+\_+,+.+:+;+"+ \\
 & | & \ +\ +-+*+/+<+>+=+\{+\}+(+)+ \\
 & | & [+]+!+\#+\$+\%+\&+@+\wedge+\sim+\ddot{}+\backslash\backslash+\backslash'+\backslash n+\backslash r+\backslash t)\bullet' \\
stringlit & | & "\bullet(a+b+c+...+z+A+B+C+...+Z+ \\
 & | & 0+1+...+9+\_+,+.+:+;+'+ \\
 & | & \ +\ +-+*+/+<+>+=+\{+\}+(+)+ \\
 & | & [+]+!+\#+\$+\%+\&+@+\wedge+\sim+\ddot{}+\backslash\backslash+\backslash"+\backslash n+\backslash r+\backslash t)^*\bullet" \\
intlit & | & (0+1+2+3+4+5+6+7+8+9)\bullet \\
 & | & (0+1+2+3+4+5+6+7+8+9)^* \\
id & | & (a+b+...+z+\_)\bullet(a+b+...+z+ \\
 & | & A+B+...+Z+0+1+...+9+\_)^* \\
idclass & | & (A+B+...+Z)\bullet(a+b+...+z+ \\
 & | & A+B+...+Z+0+1+...+9+\_)^*
\end{array}
$$

4. Pruebas **.swift*

En el directorio *./test* existen diversos archivos de prueba exitosos (*test_xy.swift*) y no exitosos (*err_xy.swift*).

En cada uno se ha evaluado la respuesta acorde de los autómatas cuando se debe transicionar entre ellos, ya que es el lugar donde se halló más errores.

5. Consideraciones

A continuación se enumeran algunas particularidades del analizador léxico que repercutirán en las etapas posteriores de la construcción del compilador:

- El literal cadena de caracteres se formula con comillas dobles (").
- El literal caracter se formula con comilla simple (').
- Los literales caracter admitirá el símbolo secuencia de escape (\) solo si este se encuentra acompañado de n, r, t, ' o \.
- Exceptuando lo anterior, un literal caracter admite cualquier otro símbolo del alfabeto.
- Además, no se admite un literal caracter nulo. Es menester aclarar que ante una entrada del tipo ''', se reportará que el literal caracter no debe ser nulo, lo cual es correcto ya que está evaluando lo que hay dentro de la primera y segunda comilla simple.
- Un literal string solo admitirá el símbolo secuencia de escape (\) solo si este se encuentra acompañado de n, r, t, ", \.
- Exceptuando lo descrito previamente, un literal string admite cualquier otro símbolo.
- Un literal string puede estar conformado por ningún símbolo.
- Lo anterior no se aplica para literales caracter ni entero.
- El token id refiere a identificadores de variables y métodos.
- El token $idclass$ refiere a identificadores de clase, que comienzan con mayúscula.
- Si se encuentra una cadena dada por

  $$(0+1+...+9)+(a+b+...+z+A+B+...+Z+\_)\bullet(a+b+...+z+A+B+...+Z+\_+0+1+...+9)^*$$

  devolverá un error léxico: *"Invalid number"*.
- Finalmente, un literal string siempre debe terminar en la misma línea que comenzó.

6. Conclusión

En las pruebas realizadas se obtuvo el comportamiento esperado, recolectando adecuadamente los tokens. Sin embargo, la construcción carece de revisiones de optimización que pudieran acelerar la respuesta del programa, tal como la concatenación de objetos String utilizando StringBuilder y no la concatenación clásica de Java (+). Se espera, sin entorpecer futuras entregas, someter las mencionadas revisiones.