Universidad del Quindío
Programa de Ingeniería de Sistemas y Computación
Espacio académico Teoría de Lenguajes Formales

Guía de la parte 2 de la fase 2 del proyecto de asignatura
Implementación de un analizador léxico

1 Objetivo

Implementar un analizador léxico para el lenguaje de programación especificado por el profesor

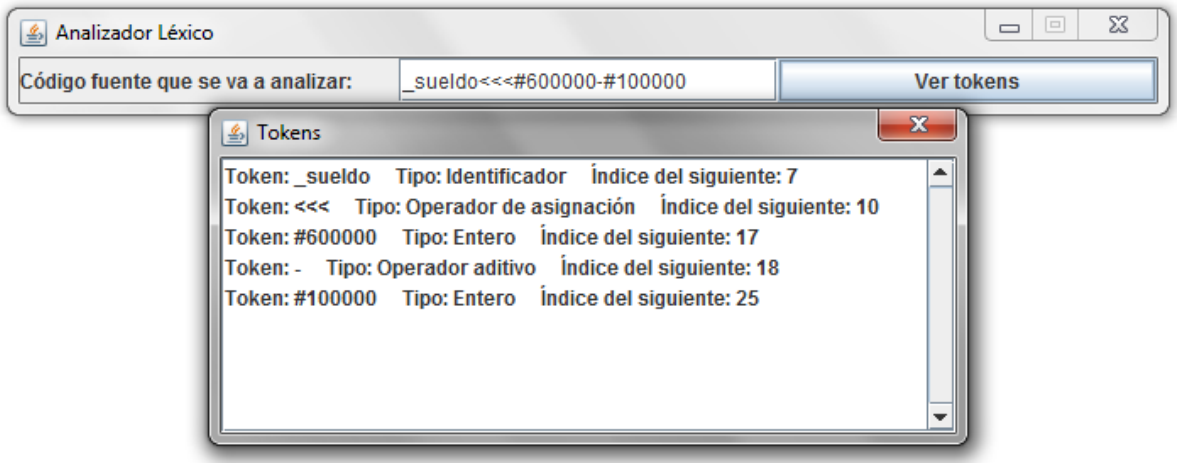
El analizador debe implementarse en Java. Pueden implementar el analizador léxico desde cero, o adaptar el mini-analizador léxico suministrado por la profesora, el cual fue creado por el profesor Leonardo Hernandez y el cual se explica en la sección 2

No se puede cambiar la especificación de los tokens entregada previamente, sin el visto bueno de la profesora.

2 Código inicial suministrado

Como base o referencia para la elaboración del analizador léxico, se suministra un proyecto Eclipse, correspondiente a un mini-analizador léxico, cuya interfaz puede observarse en la Figura 1.

Figura 1. Interfaz del mini analizador léxico



El mini-analizador reconoce los tokens de LeDiAn, un mini-lenguaje creado por el profesor que diseño el proyecto, con solo 4 tipos de tokens:

- Enteros, dados por la expresión regular $\#DD^*$
- Identificadores, dados por la expresión regular $\_L^*$
- Dos operadores aditivos, dados por la expresión regular $+ \quad \cup \quad -$

- Dos operadores de asignación, dados por la expresión regular <<< ∪ < −<

El código incluye los siguientes cuatro métodos, correspondientes a los cuatro tipos de tokens mencionados.

- public Token extraerEntero (String cod, int i)
- public Token extraerIdentificador (String cod, int i)
- public Token extraerOperadorAditivo (String cod, int i)
- public Token extraerOperadorAsignacion (String cod, int i)

`extraerEntero()` reconoce los números enteros, a los que en LeDiAn debe antepornerse un símbolo de numeral.

`extraerIdentificador()` reconoce los identificadores, que en LeDiAn consisten en una raya baja seguida de cero o más letras.

`extraerOperadorAditivo( )` reconoce el signo más (+) y el signo menos (−)

`extraerOperadorAsignación( )` reconoce los dos operadores de asignación de LeDiAn: `<<<` y `<-<`

3 Partes del código que se deben adaptar para el reconocimiento de nuevos tokens

Para que en mini-analizador léxico reconozca nuevos tipos de token, se debe adaptar el código básicamente en tres lugares diferentes.

1) En el paquete *mundo*, clase *Token.java*, debe adicionarse una constante para cada tipo de token. Véase la Figura 2

Figura 2. Declaración de constantes para cada tipo de token

```
/**
 * Constantes para modelar los posibles tipos de token que se van a analizar
 */
final public static String ENTERO = "Entero";
final public static String OPERADORADITIVO = "Operador aditivo";
final public static String OPERADORASIGNACION = "Operador de asignación";
final public static String IDENTIFICADOR = "Identificador";
final public static String NORECONOCIDO = "No reconocido";
```

2) En el paquete *mundo*, clase *AnalizadorLexico.java*, debe adicionarse un método para cada tipo de token. Véase la Figura 3

Figura 3. Método para extraer un entero

```
/**
 * Intenta extraer un entero de la cadena cod a partir de la posición i,
 * basándose en el Autómata
 * @param cod - código al cual se le va a intentar extraer un entero - codigo!=null
 * @param i - posición a partir de la cual se va a intentar extraer un entero  - 0<=indice<codigo.length()
 * @return el token entero o NULL, si el token en la posición dada no es un entero. El Token se compone de
 * el lexema, el tipo y la posición del siguiente lexema.
 */

// Este método usa el método substring(), que se explica a continuación:
// x.substring( i, j ) retorna una nueva cadena que es una subcadena de la cadena x.
// La subcadena comienza en la posición i y
// se extiende hasta el carácter en la posición j-1.
// Ejemplo: "universidad".substring(3,6) retorna "ver",


public Token extraerEntero ( String cod, int i)
{

    int j;
    String lex;
    if( cod.charAt(i)=='#' ){
        j=i+1;
        if( j<cod.length() && esDigito(cod.charAt(j)) ){
            do
                j++;
            while (  j<cod.length( ) && esDigito(cod.charAt(j)) );
            lex =  cod.substring( i, j);
            Token token = new Token( lex, Token.ENTERO, j );
            return token;
        }
    }

    return null;
}
```

3) Finalmente, también en el paquete *mundo*, clase *AnalizadorLexico.java*, debe adicionarse una llamada al método que se cree en el paso 2). Véase la Figura 4

Figura 4. Llamada a los métodos que extraen los tokens

```
/**
 * Extrae el token de la cadena cod a partir de la posición i, basándose en el Autómata
 * @param cod - código al cual se le va a extraer un token - codigo!=null
 * @param i - posición a partir de la cual se va a extraer el token  - i>=0
 * @return token que se extrajo de la cadena
 */
public Token extraerSiguienteToken( String cod, int i )
{
    Token token;

    // Intenta extraer un entero
    token = extraerEntero( cod, i);
    if ( token != null )
        return token;

    // Intenta extraer un operador aditivo
    token = extraerOperadorAditivo( cod, i);
    if ( token != null )
        return token;

    // Intenta extraer un operador de asignación
    token = extraerOperadorAsignacion( cod, i);
    if ( token != null )
        return token;

    // Intenta extraer un identificador
    token = extraerIdentificador( cod, i);
    if ( token != null )
        return token;

    // Extrae un token no reconocido
    token = extraerNoReconocido( cod, i);
    return token;
}
```

4 Buenas prácticas de programación

En la implementación del analizador léxico deben usar estándares de programación apropiados, entre otros:

- Uso de comentarios
- Uso de documentación Javadoc
- Estructuración del programa en varias clases
- Clases diferentes para implementar el mundo y para implementar la interfaz
- No abreviar los nombres de los atributos ni de los métodos.
- No sobrecargar de responsabilidades los métodos ni las clases. En general, una clase o un método no debe tener a la vez responsabilidades relacionadas con la interfaz y relacionadas con la lógica del problema.