

Tutorial Jlex

Procesadores del Lenguaje

Roberto de la Torre Rivero

Exp: 20212658

Introducción

El Jlex es un generador de analizadores léxicos diseñado para Java.

Jlex es capaz de generar automáticamente un analizador léxico a partir de una especificación léxica del lenguaje.

Un analizador léxico toma como entrada una cadena de caracteres y la transforma en una secuencia de tokens.

Jlex toma como entrada un archivo con una especificación y con ella crea un archivo fuente Java correspondiente al analizador léxico, que tras ser compilado da como resultado el analizador.

Archivo de especificación

Las especificaciones Jlex están organizadas en tres secciones, separadas por la directiva "%%", como se muestra a continuación:

código de usuario

%%

directivas Jlex

%%

reglas de expresiones regulares

La directiva "%%" distingue las diferentes secciones y debe ser ubicada en el comienzo de la línea, y el espacio restante de dicha línea debería permanecer en blanco.

Código de usuario

La sección de código de usuario, contiene el código que se desea incluir al comienzo del archivo de salida, tal como el paquete en el que se desea almacenar, o las clases que se importarán. Esta área de la especificación provee espacio para la implementación de clases de utilidad o de tipos de retorno.

Directivas Jlex

La sección de directivas Jlex es la segunda parte del archivo de entrada. En ella se declaran las definiciones de macros y los nombres de los estados. Cada directiva Jlex debería estar contenida en una línea simple y debería comenzar esa línea.

Reglas de Expresiones Regulares

La sección de reglas de expresiones regulares es la tercera y última parte de la especificación, y contiene las reglas necesarias para el análisis léxico; las reglas tienen tres partes diferentes: una lista opcional de estados léxicos, una expresión regular, y la acción asociada. Tal formato, puede representarse como sigue:

[<states>] <expression> { <action> }

Estas reglas son las que permiten dividir la secuencia de caracteres de entrada en una secuencia de tokens. Estas reglas especifican las expresiones regulares, y les asocian acciones escritas con código fuente Java

Reconocimiento de Tokens

Si la entrada coincide con más de una regla, el analizador resuelve el conflicto entre las reglas, escogiendo la regla que reconoce la cadena más larga, y en caso de que, a su vez, haya varias reglas que reconocen cadenas de la misma longitud máxima, elige la regla que aparece primero en la especificación. Entonces, las reglas que aparecen primero en la especificación son las que tienen mayor prioridad en el scanner generado.

Para evitar errores, causados por el no reconocimiento de una entrada, ya sea porque se omitió alguna regla del lenguaje, o porque la entrada no pertenece al lenguaje, debe agregarse al final de la especificación de las reglas, la siguiente cláusula:

. { java.lang.System.out.println("Unmatched input: " + yytext()); }

donde el punto reconoce cualquier entrada excepto la línea nueva.

Ejemplo:

Primeros pasos:

Debemos tener instalado J2SE(TM) Development Kit

Descargamos JLex de su página:

<http://www.cs.princeton.edu/~appel/modern/java/JLex/>

Source Code: Main.java

Creamos una carpeta en el directorio de Jdk en LIB con el nombre de JLex y copiamos el archivo Main.java del JLEX a esta carpeta

A continuación compilamos este archivo y tendremos la clase Jlex

Javac Main.java

Resultado: Main.class

Archivo de especificación:

Lo llamaremos especificación.lex

Código de usuario:

Definimos la librerías

import java.lang.System;

y la clase yytoken que es la que devolverá los datos del token encontrado:

class Yytoken {

Yytoken (int numToken,String text, String compo, int line, int charBegin){

//Contador para el número de tokens reconocidos

m_numToken = numToken;

//String del token reconocido

m_text = new String(text);

//Tipo de componente léxico encontrado

m_compo = compo;

//Número de línea

m_line = line;

//Columna donde empieza el primer carácter del token

m_charBegin = charBegin;

}

//Métodos de los atributos de la clase

public int m_numToken;

```

public String m_text;

public String m_compo;

public int m_line;

public int m_charBegin;

//Metodo que devuelve los datos necesarios que escribiremos en un archive de salida

public String toString() {

    return "Token #"+m_numToken+": "+m_text+" C.Lexico: "+m_compo+" Line: "+m_line+" Column:
"+m_charBegin;

}

}

```

Directivas JLex

```

%%
//permite cambiar el nombre de la función de yylex que reconoce los tokens (next token)
%function nextToken
//permite cambiar el nombre de la clase del analizador léxico, desde Yylex.
%class Analizador
//se define in integer para el contador de tokens
%{
    private int contador;
}%
//con init inicializamos variables
%init{
    contador = 0;
%init}
//fin de fichero
%eof{
%eof}
//Activa el contador de líneas, almacena en al variable entera yyline el índice de la primera //línea del token
reconocido
%line
//activa el contador de caracteres, por defecto desactivado, almacena en al variable entera yychar el índice
del primer caracter del token reconocido
%char

```

Expresiones regulares acciones

//Definimos los patrones para los tipos de datos a reconocer requeridos

EXP_ALPHA=[A-Za-z]

EXP_DIGITO=[0-9]

EXP_ALPHA_NUMERIC={EXP_ALPHA}{EXP_DIGITO}

NUMERO={EXP_DIGITO}*

EXP_ENTERO=[1-9]

NUMERO_ENTERO="0" | {EXP_ENTERO}{EXP_DIGITO}*

EXP_OCTAL=[0-8]

NUMERO_OCTAL="0"({EXP_OCTAL})+

EXP_HEX=[0-9a-fA-F]

NUMERO_HEX="0x"({EXP_HEX})+

IDENTIFICADOR={EXP_ALPHA}{EXP_ALPHA_NUMERIC}*

%%

//Acciones a realizar cuando encontramos un token que concuerda con el patrón

//en cada caso delvemos una instancia de la clase yytoken con los datos requeridos para escribirlos en una archivo de salida

//contador

// yytext

//componente léxico

//yyline

/yychar

```
{NUMERO_HEX} {   contador++;  
                  return new Yytoken(contador, yytext(),"Hexadecimal", yyline, yychar);  
                }
```

```
{NUMERO_ENTERO} {   contador++;  
                    return new Yytoken(contador, yytext(),"Entero", yyline, yychar);  
                  }
```

```
{NUMERO_OCTAL} {   contador++;  
                   return new Yytoken(contador, yytext(),"Octal", yyline, yychar);  
                 }
```

```
{IDENTIFICADOR} {   contador++;  
                    return new Yytoken(contador, yytext(),"Identificador", yyline, yychar);  
                  }
```

```
{NUMERO} {   contador++;  
              return new Yytoken(contador, yytext(),"Numero", yyline, yychar);  
            }
```

//Se podrán definir mas acciones para diferentes situaciones o patrones encontrados como salto de línea, fin de fichero, caracteres de puntuación....

```
" "      {}  
[\n]     {}  
[\t\r]   {}
```

Ahora crearemos la clase principal en java que realizara la lectura del archivo entrada.txt, ordenará al analizador reconocer los tokens, y la salida será escrita en un archivo de salida.txt

Archivo Jlex.java

```
import java.io.BufferedReader;  
import java.io.File;  
import java.io.FileNotFoundException;  
import java.io.FileReader;  
import java.io.FileWriter;  
import java.io.IOException;  
import java.io.InputStreamReader;  
import java.io.PrintWriter;  
  
public class Jlex {  
  
    protected FileReader fichero = null;  
    private static BufferedReader stdIn = new BufferedReader(new  
InputStreamReader(System.in));  
    private static PrintWriter stdout = new PrintWriter(System.out, true);  
    private static PrintWriter stderr = new PrintWriter(System.err, true);  
    static String nombre;  
    static File filein;  
    File fileout;  
    static FileReader filereader;  
  
    public static void main(String argv[]) throws IOException{  
  
        stdout.println("-----");  
        stdout.println("Escriba el nombre del archivo a analizar");  
        stdout.println("-----");  
  
        stderr.flush();  
        nombre = stdIn.readLine();  
  
        filein = open(nombre);  
  
        filereader = CrearFileReader(filein);  
  
        alexico(filereader);  
        System.out.println("Se ha creado un archivo de salida.txt con el resultado");  
  
    }  
  
    public static File open(String nombre){  
        File archivo = null;  
        archivo = new File ("\\"+nombre);  
        return archivo;  
    }  
  
    public static FileReader CrearFileReader(File archivo){  
        FileReader fr = null;
```

```

        try {
            fr = new FileReader (archivo);
        } catch (FileNotFoundException ioe) {
            System.out.println(ioe);
        }
        return fr;
    }
}
//Escribimos el objeto token en el archivo de salida
public static void Salida(Ytoken token){
    Ytoken stoken = token;
    FileWriter ficherosalida = null;
    PrintWriter printwriter = null;

    try
    {

        ficherosalida = new FileWriter("salida.txt", true);
        printwriter = new PrintWriter(ficherosalida);

        printwriter.println(token);

    } catch (Exception e) {
        e.printStackTrace();
    } finally {
        try {
            if (null != ficherosalida)
                ficherosalida.close();
        } catch (Exception e2) {
            e2.printStackTrace();
        }
    }
}

public static void alexico(FileReader fichero) {

    //Creamos la instancia de la clase analizador (yylex)
    Analizador sampleLex = new Analizador(fichero);
    Ytoken token = null;
    String buff;

    do{
        try{
            //como hemos renombrado el metodo en las directivas ahora es nextToken
            token = sampleLex.nextToken();
        }catch(java.io.IOException e){

            System.out.println("Error en nextToken()");
        }
        //Mientras no este vacio
        if (token!=null)
            //Escribimos en fichero

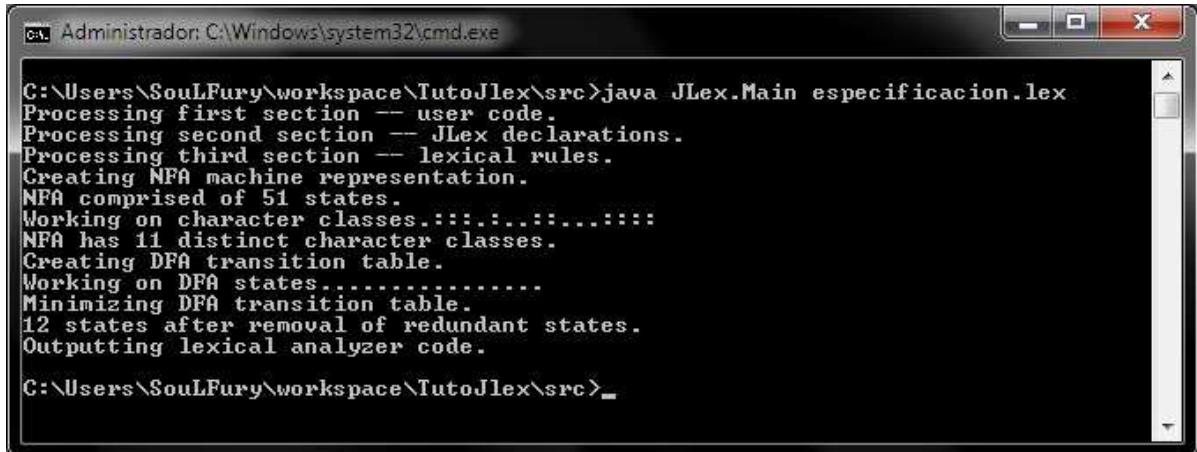
            Salida(token);
    }while(token!=null);
}
}

```

Pasos para ejecutar la aplicación:

1. Primero compilamos el archivo de especificacion.txt con JLex:

java JLex.Main especificación.lex



```
C:\Users\SoulFury\workspace\TutoJlex\src>java JLex.Main especificacion.lex
Processing first section -- user code.
Processing second section -- JLex declarations.
Processing third section -- lexical rules.
Creating NFA machine representation.
NFA comprised of 51 states.
Working on character classes.:.:.:.:.:
NFA has 11 distinct character classes.
Creating DFA transition table.
Working on DFA states.:.:.:.:.:
Minimizing DFA transition table.
12 states after removal of redundant states.
Outputting lexical analyzer code.
C:\Users\SoulFury\workspace\TutoJlex\src>_
```

2. Obtendremos el archivo **especificación.lex.java** que debemos compilar con javac



especificacion.lex.java

javac especificación.lex.java

Si todo ha ido bien, obtendremos las clases especificadas en el archivo:



Yytoken.class



Analizador.class

3. Ahora compilamos el archivo Jlex.java con nuestro código que utiliza estas clases:

Javac Jlex.java

Obtendremos el class:



Jlex.class

4. Ejecutamos el archivo generado con:

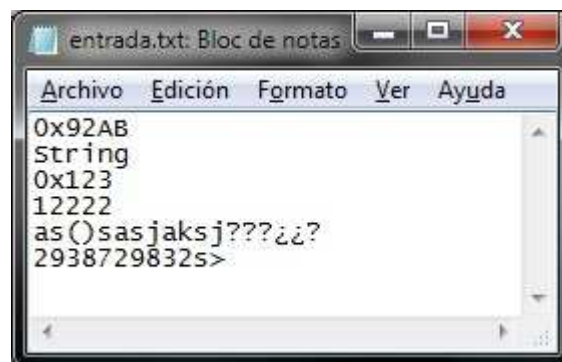
Java Jlex

Se mostrara el programa principal, pidiéndonos el archivo de entrada con los caracteres a reconocer.

Este archivo debe de localizarse en el directorio donde estén estos archivos (src)

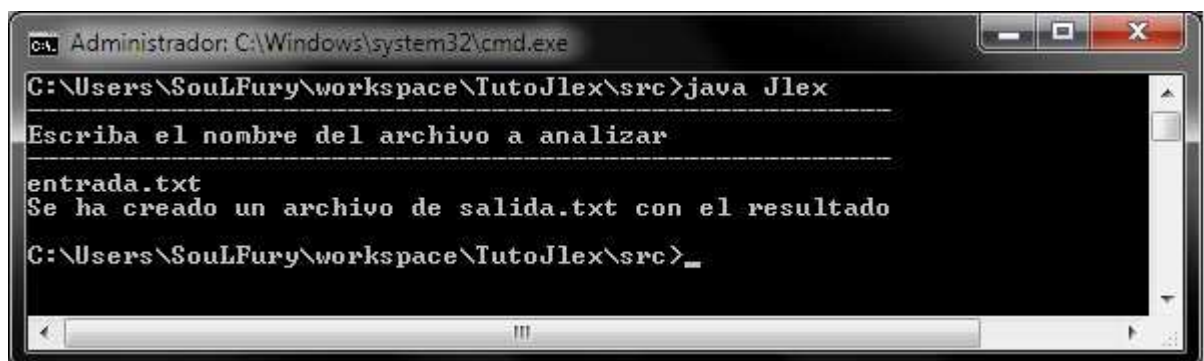


```
C:\Users\SoulFury\workspace\TutoJlex\src>java Jlex
Escriba el nombre del archivo a analizar
entrada.txt_
```



```
Archivo  Edición  Formato  Ver  Ayuda
0x92AB
String
0x123
12222
as()sasjaksj???¿¿?
2938729832s>
```

A continuación el programa abrirá el fichero, leerá su contenido y lo pasará al analizador, cuando se encuentre una coincidencia definida en nuestro archivo de especificación lo escribirá en un archivo de salida con la información también definida en este archivo de especificación.



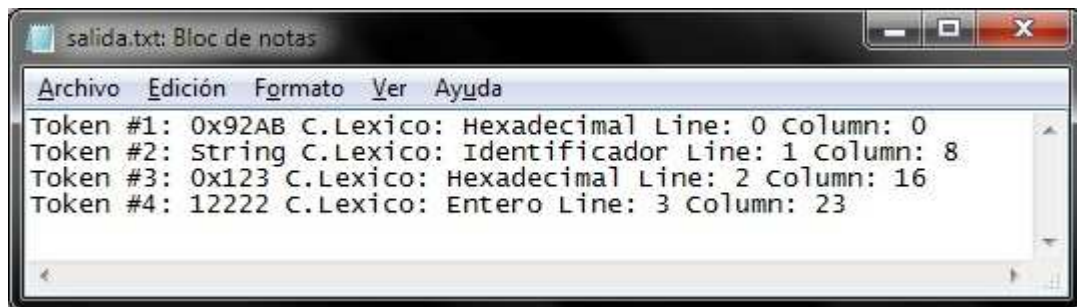
```
C:\Users\SoulFury\workspace\TutoJlex\src>java Jlex
Escriba el nombre del archivo a analizar
entrada.txt
Se ha creado un archivo de salida.txt con el resultado
C:\Users\SoulFury\workspace\TutoJlex\src>
```

En el directorio de nuestro archivos encontraremos un archivo llamado salida.txt



salida.txt

En este archivo se encuentra la información sobre los tokens encontrados, componente léxico, línea y columna.



En nuestro caso se nos pedía que reconociese los tokens de ENTEROS, HEXDECIMALES, OCTALES e IDENTIFICADORES, podemos comprobar con diferentes archivo de entrada que funciona correctamente.

Se puede completar el sistema, añadiendo al archivo de especificacion.lex más conjuntos de expresiones regulares y patrones para reconocer diferentes tipos de tokens, caracteres, símbolos etc...

REFERENCIAS

Jlex Manual

<http://www.cs.princeton.edu/~appel/modern/java/JLex/current/manual.html>

www.cc.uah.es/ie/docencia/ProcesadoresDeLenguaje

<http://talf.wikispaces.com/JLex>

Directivas Jlex

<http://www.lcc.uma.es/~galvez/theme/cup/anlexcup/especificacion/directivas.html>