Analizador Léxico JFLEX

INTRODUCCIÓN:

JFlex es un generador de analizadores lexicográficos desarrollado por Grewin Klein, como extensión a la herramienta JLex. Está desarrollado en Java y genera código Java.

Genera un analizador léxico para un lenguaje determinado. Se caracteriza porque utiliza expresiones regulares (regex) para reconocer los lexemas del lenguaje. Puede incluir código para ser invocado cuando esas expresiones son reconocidas y, además, puede incluir código auxiliar para el soporte del código de entrada.

La función yylex() es propia de JFlex y al ser invocada comienza a leer carácter a carácter hasta que encuentra el mayor prefijo en la entrada que concuerde con una de las expresiones regulares, una vez que reconoce una regex ejecuta la acción indicada para el lexema detectado.

ESTRUCTURA DEL ARCHIVO:

Un programa Flex se organiza en tres secciones:

1) **Importaciones y código del usuario**
2) **Directivas JFLex (opciones y declaraciones)**
3) **Reglas**

Importaciones y código del usuario:

En esta área se debe incluir sentencias Java de importación de los paquetes que se vayan a utilizar en las acciones regulares situadas al lado de cada patrón en la zona de reglas, como también escribir clases completas e interfaces. Cualquier cosa que se escriba en esta área se copiará textualmente al archivo java generado por JFlex.

El código que incorpore el programador va entre **%{** y **%}**

Ejemplo:

```
import java.io.*;
%{
   private MiToken token(String nombre) {
      return new MiToken(nombre, this.yyline, this.yycolumn);
   }
   private MiToken token(String nombre, Object valor) {
      return new MiToken(nombre, this.yyline, this.yycolumn, valor);
   }
%}
```

Directivas JFLex (opciones y declaraciones)

Es la segunda parte del archivo de entrada, comienza y finaliza con los símbolos '**%%**' .
En esta sección se definen:

- Opciones
- Declaraciones

Opciones

Pueden ser: de clase, de función de análisis, de fin de fichero, de juego de caracteres y de contadores. Todas las opciones comienzan con el carácter '**%**'.

Ejs.

%public –> genera una clase publica.

%final –> genera una clase final (sin herencia)

%int –> modifica la función de análisis yylex() la cual devolverá valores de tipo int en lugar de Yytoken

%eofclose -> provoca que yylex() finalice la lectura en cuanto encuentre EOF.

%unicode –> la entrada estará formada por caracteres Unicode.

%line –> guarda en la variable yyline el numero de línea en que comienza el lexema actual. Puede consultarse con this.yyline

%class Nombre -> Nombre de la clase generada

%type Nombre -> Nombre de la clase usada para representar token. Tipo de los objetos retornados por yylex()

%column -> Numero de columna que esta analizando. Puede consultarse con this.yycolumn

%char -> Numero de carácter que esta analizando. Puede consultarse con this.yychar

Declaraciones

Pueden ser:

- Declaraciones de estados léxicos
- Declaraciones de reglas

Los estados léxicos (también llamados condiciones start) se declaran de la forma:

%state estado1, estado2, etc

Las expresiones regulares se pueden definir como patrones que tienen un lenguaje propio y describen el formato del lexema. Los patrones definidos podrán ser utilizados posteriormente

en cualquier otra regla encerrados entre llaves. Las declaraciones de reglas se hacen con el siguiente formato:

nombre **=** patrón

Ej:

DIGITO = [0 - 9]

LETRA = [a - z A-Z]

Área de reglas

Es la tercera y última parte de la especificación, contiene todos los patrones necesarios para el realizar el análisis léxico a través de las acciones semánticas. Está conformada por tres partes: una lista opcional de estados léxicos, una regex o nombre (definido en el área de definiciones) y la acción asociada.

Las acciones son ejecutadas cada vez que una regla es reconocida con el proceso de entrada. Estas acciones pueden retornar valores, componentes léxicos (tokens) y pueden ejecutar algún código.

Una acción es una sentencia escrita en el lenguaje Java, lenguaje que compila JFLex. Deben aparecer encerradas entre llaves y finaliza con '**;**' .

Ej:

<YYINITIAL> "=" {System.out.println("Token '=');}

<YYINITIAL> "!=" {System.out.println("Token distintio");}

También puede escribirse:

<YYINITIAL> {

"=" {System.out.println("Token '=');}

"!=" {System.out.println("Token distintio");}

}

Si en el área de reglas utilizamos un nombre (que previamente definimos en la sección anterior) ese nombre deberá ir entre llaves.

Ej:

{IDENTIFICADOR} { System.out.println ("IDENTIFICADOR"); }

Una característica importante de JFlex, es que incluye algunas clases de caracteres predefinidas. Estas clases se invocan entre los símbolos **[:** y **:]** .

La lista de patrones es:

Patrón	Predicado asociado
[:jletter:]	isJavaIdentifierStart()
[:jletterdigit:]	isJavaIdentifierPart()
[:letter:]	isLetter()
[:digit:]	isDigit()
[:uppercase:]	isUpperCase()
[:lowercase:]	isLowerCase()

FUNCIONES Y VARIABLES DE LA CLASE YYLEX:

Yylex(Reader r): es el constructor del analizador léxico. Toma como parámetro la vía de entrada del cual se leerán los caracteres.

Yytoken yylex(): función principal que implementa el analizador léxico. Toma la entrada del parámetros especificado en la llamada al construcción de Yylex.

String yytect(): devuelve el lexema actual.

Int yylength(): devuelve el número de caracteres del lexema actual.

void yyreset(Reader r): cierra la via de entrada actual y redirige la entrada hacia el n uevo canal especificado como parámetro.

void yypushStream(Reader r): guarda el canal de entrada actual en una pila y continua la lectura por el nuevo canal especificado. Cuando este último finalice continuará por el anterior extrayéndolo de la pila.

yyline, yychar e yycolumn: son las variables generadas por las opciones %line, %char y %column respectivamente.

EJECUCIÓN DE JFLEX: EJEMPLO

- **Generar el programa con extensión FLEX**: es el programa que tiene la estructura que se explica en este documento (Estructura del archivo Flex), a partir del cual se genera el analizador lexicográfico, por ejemplo, lo llamamos lexer.flex. Dentro de las directivas del mismo, hay una opción, **%class**, que indica cómo se llamará el programa en código Java que se arma en base al programa Flex (por ejemplo %class MiLexer)

- **Generar el programa en código Java**: Ejecutar el aplicativo JFlex, como se indica:

JFlex lexer.flex

Al ejecutarse el aplicativo JFlex, se obtiene el programa MiLexer.java, que es el código java necesario para hacer el análisis léxico en base a las reglas que se establecieron en el programa lexer.flex.