uc3m

Procesadores del Lenguaje

Curso 2020-2021

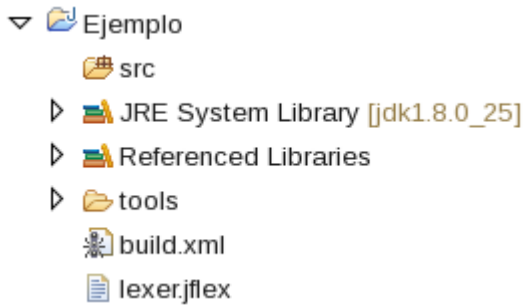
Práctica 1.1: Análisis léxico: analizador léxico independiente

1 Introducción

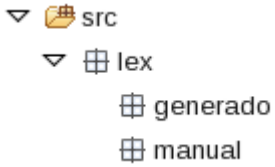
En esta primera práctica, una vez familiarizados con la herramienta JFlex que permite representar expresiones regulares para construir analizadores léxicos, partiendo del ejemplo de la calculadora, se va a desarrollar un proyecto desacoplado del analizador sintáctico.

2 Construcción del proyecto únicamente con análisis léxico (JFlex)

En primer lugar, vamos a modificar el proyecto para que solo realice la parte léxica. Eliminamos el fichero cup y el paquete cup (dejando src vacío)



Ahora, para ver más claramente el funcionamiento, creamos la estructura de paquetes siguiente, diferenciando entre ficheros java manuales (que editaremos libremente) y ficheros java generados por las herramientas (que no podremos cambiar directamente):



El segundo paso es modificar el script de ant "build.xml" para reflejar estos cambios:

build.xml

```
<project name="LexProject" default="compile" basedir=".">

        <property name="java"    location="src"     />
        <property name="classes"  location="bin/cls"  />
        <property name="result"   location="bin/jar"  />
        <property name="lib"     location="lib"     />
        <property name="tools"    location="tools"    />
        <property name="base"    location="."    />

        <taskdef name="jflex" classname="JFlex.anttask.JFlexTask"  classpath="${tools}/JFlex.jar"   />

        <target name="generate">
                <jflex  file="lexer.jflex" destdir="${java}" />
        </target>
```

```
        <target name="compile" depends="generate">
                <mkdir dir="${classes}"/>
                <mkdir dir="${result}"/>
          <javac includeantruntime="false" srcdir="${java}" destdir="${classes}" />
        </target>

        <target name="clean">
          <delete file="${java}/generado/Lexer.java" />
          <delete dir="${classes}" />
          <delete dir="${result}" />
        </target>
</project>
```

En este punto ya podemos empezar a rellenar el archivo jflex de acuerdo a lo especificado en el manual (http://jflex.de/manual.html).

- Crear una clase Java para representar los símbolos:

src/lex/manual/Symbol.java

```
package lex.manual; // paquete en el que creemos el archivo

public class Symbol {

        /* Atributos */
        private int type;
        private Object value;

        public Symbol(int tipo) {
                this.type = tipo;
                this.value = null;
        }

        public Symbol(int tipo, Object valor) {
                this.type = tipo;
                this.value = valor;
        }

        public int type() {
                return this.type;
        }

        public Object value() {
                return this.value;
        }
}
```

- Crear un nuevo fichero Java de símbolos terminales, partiendo de la siguiente plantilla vacía:

src/lex/manual/SimbolosTerminales.java

```
package lex.manual; // paquete en el que creemos el archivo

public interface SimbolosTerminales {
  /* terminals */
              public static final int EOF = 0;
              public static final int error = 1;
//            public static final int nombre = 2;
//            public static final int numero = 3;
//            ...

  /* lista de nombres, util para devolver información por pantalla */
  public static final String[] terminalNames = new String[] {
  "EOF",
  "error"
// ,"nombre",
// "numero"
//....
  };
}
```

- Debemos incluir en el fichero jflex las siguientes declaraciones:

Lexer.jflex

```
package lex.generado; // paquete en el que se genera el fichero java
import lex.manual.SimbolosTerminales; //Simbolos terminales definidos
import lex.manual.Symbol;

%%

%class Lexer
%implements SimbolosTerminales
%public
%unicode

%line
%column
%char

%function next_token // Nombre del método que escanea la entrada y
                     // devuelve el siguiente token
%type Symbol // Tipo de retorno para la función de scan


%eofval{
   return new Symbol(EOF);
%eofval}


/* Macros para expresiones regulares (para simplificar reglas) */

%%

/* Reglas para detectar los tokens y acciones asociadas */

// error fallback
```

```
.|\n        {System.err.println("warning: Unrecognized character '"
              + yytext()+"' -- ignored" + " at : "+ (yyline+1) + " " +
              (yycolumn+1) + " " + yychar); }
```

- Realizar los ajustes que sean convenientes mediante las directivas JFlex (comienzan por el símbolo de porcentaje '%')

- Crear un método Main que nos permita ejecutar la aplicación. Utilizar el siguiente código de ejemplo:

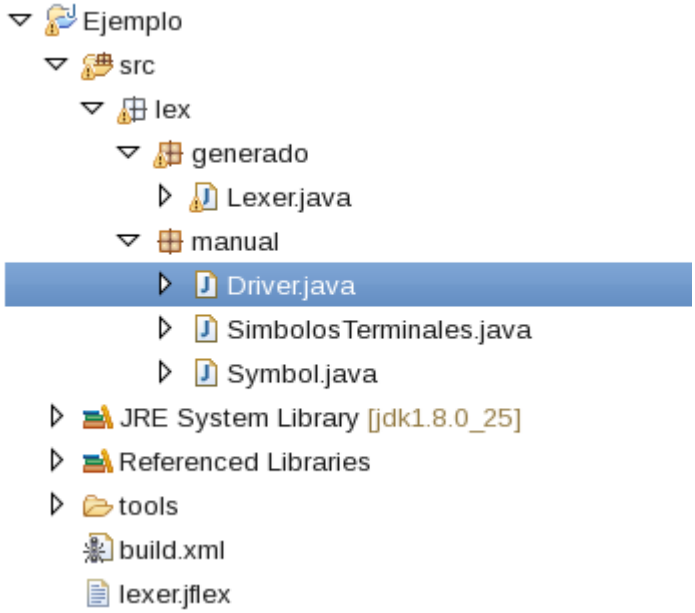
src/lex/manual/Driver.java

```
package lex.manual; // paquete en el que creemos el archivo

import lex.generado.*;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.util.ArrayList;

class Driver {

        public static void main(String args[]) throws IOException {
        // Entrada de datos: teclado por defecto, fichero si hay argumento
                InputStream dataStream = System.in;

                if (args.length >= 1) {
                        System.out.println("Leyendo entrada de fichero... ");
                        dataStream = new FileInputStream(args[0]);
                } else {
                        System.out.println("Inserta expresiones a reconocer,"+
                  " pulsando <ENTER> entre ellas");
                }
                // Creamos el objeto scanner
                Lexer scanner = new Lexer(dataStream);
                ArrayList<Symbol> symbols = new ArrayList<Symbol>();
                // Mientras no alcancemos el fin de la entrada
                boolean end = false;
                while (!end) {
                        try {
                                Symbol token = scanner.next_token();
                                symbols.add(token);
                                end = (token == null);
                                if (!end) {
                                 System.out.println("Encontrado: {" + token.type() + " - "
                                    +SimbolosTerminales.terminalNames[token.type()]+
                                    +  "} >>            " + token.value());
                                 }
                        } catch (Exception x) {
                                System.out.println("Ups... algo ha ido mal");
                                x.printStackTrace();
                        }
                }
                symbols.trimToSize();
                System.out.println("\n\n -- Bye-bye -- ");
```

```
        }
}
```

2.1 Generar Código, Compilar y Ejecutar

En este momento, tendremos el proyecto listo. Si ejecutamos la clase "Driver" como una aplicación java, el script de ant, gererará el fichero "src/lex/generado/Lexer.java" de acuerdo a las reglas establecidas en el fichero "lexer.jflex", compilará el conjunto y ejecutará el main de la clase Driver.



Recordad que los ficheros java del paquete "generado" no se deben modificar directamente, ya que se borran y crean nuevamente cada vez que ejecutamos el proyecto a partir de las reglas.

3 Especificación

En el ejemplo anterior vimos como implementar un reconocedor de expresiones matemáticas con números enteros, con posibilidad de agrupar con paréntesis.

A continuación, se solicita modificar la especificación del lenguaje utilizados en el escáner (JFlex) para realizer las siguientes tareas. Como se ha indicado, es fundamental tener a la vista los manuales de las herramientas.

3.1 Analizador Léxico

A: Aceptar e ignorar comentarios, que deben ser tratados –e ignorados– por completo en el análisis léxico. Los comentarios comenzarán con los caracteres '/*' y se extienden hasta la aparición de '*/', pudiéndose extender a varias líneas separadas. Un ejemplo sería:

```
/* Esto es un comentario en tres líneas

*/
/* Operandos individuales */*/
5
-1
/* Con mas de un caracter */
1341894
+1341894
-1341894

% Operaciones
5*-3-8; /* esta expresion debe resultar -23 */
10 / 5 * 2; /* esta expresion debe resultar 4 */
- 3 *4; /* esta expresion debe resultar -12 */
5*(4-3) * (8 - (4-1)); /* esta expresion debe resultar 25 */
```

B. Incorporar números reales (con notación científica habitual) y números hexadecimales (comienzan por la secuencia "0x" o "0X")

```
% Operaciones
5.3+.1+1e-1 /* esta expresion debe resultar 5.5 */
5.3+0X11 /* esta expresion debe resultar 22.3 */
```

C. El fichero de entrada podrá incorporar información con datos identificativos del autor, cumpliendo las siguientes especificaciones léxicas

- Nombre y apellidos (comenzando con mayúsculas)
- Una dirección de email
- Un DNI de España
- Una matrícula de turismo español
- Una fecha, en formato dd/mm/aaaa

Si el texto de entrada no puede reconocerse deberá salir el aviso correspondiente

La salida esperada del analizador léxico para sería el conjunto de tokens reconocidos en el lenguaje, mostrados tras la ejecución del programa:

OPERADORES:
* - - / …

SEPARADORES Y PARÉNTESIS:
; ; ; ()(())…
NÚMEROS:
5 1 1341894 …

IDENTIFICADORES :
Nombre y Apellidos: ...
Email: ...
DNI: ...
Matrícula: ...
Fecha: ...

Extensión: incluir el código adicional para finalmente mostrar por pantalla el número de líneas, caracteres y símbolos encontrados en el programa.

4 Entrega

Cada grupo debe entregar todo el contenido de su práctica en un único archivo comprimido –preferiblemente en formato ZIP–. El nombre del comprimido debe ser "pl_p1.1_grupo_XX", donde XX son los primeros apellidos de cada componente del grupo.
Al abrir el archivo comprimido, debe verse un único directorio con el mismo nombre que el comprimido. Este directorio debe contener un proyecto eclipse completo y funcional, siguiendo la estructura del código de base. El código fuente del proyecto incluirá al menos los archivos con la especificación léxica del scanner que satisfagan los requisitos del enunciado.
NOTAS:

- ***Se permite añadir nuevas clases Java si se considera necesario, y deberán incluirse en la entrega.***
- ***Todos los proyectos son compilados por el corrector, de forma que NO se deben realizar modificaciones a los ficheros generados por JFlex, puesto que al compilar de nuevo estos cambios se perderán.***

Se adjuntará una breve memoria con consideraciones para el corrector. El formato deber ser TXT, DOC o PDF
Los dos criterios que primarán en la corrección son:
• Funcionalidad (principalmente). Todo debe funcionar sin errores y devolver la salida esperada.
• Buen uso de JFlex, que demuestre que se domina la herramienta.

En último lugar, cada grupo debe entregar ficheros de prueba, que contendrán entradas compuesta por código bien formado, que producirá una salida correcta del programa.