TEORÍA DE AUTÓMATAS Y COMPUTACIÓN

PRÁCTICA

ANALIZADOR LÉXICO JFLEX & CUP

GRUPO E04

ÍNDICE

1. Información de elementos léxicos:

Token	Descripción	Expresión regular	Ejemplo
BUCLE	Reconoce las directivas de cada bucle	while | for | do	while
NUMERO	Reconoce números enteros sin signos	[0-9]+	45
VARIABLE	Reconoce cadenas de caracteres como nombres de variables	[A-Za-z][a-zA-Z0-9_]*	Ac_D9
TRUE	Reconoce la directiva true	true	true
FALSE	Reconoce la directiva false	false	false
ENTERO	Reconoce la directiva int	int	int
BOOLEAN	Reconoce la directiva boolean	boolean	boolean
NORETORNO	Reconoce la directiva void	void	void
RETORNO	Reconoce la directiva return	return	return
SUMA	Reconoce el símbolo suma	+	+
RESTA	Reconoce el símbolo resta	-	-
POR	Reconoce el símbolo por	*	*
ENTRE	Reconoce el símbolo entre	/	/
RELACIONAL	Reconoce los símbolos relacionales de las expresiones	<|<=|>|>=|==|!=	<
LOGICA	Reconoce los símbolos lógicos de las expresiones	&&| || |!	&&
PARENTESIS	Reconoce el paréntesis de apertura y cierre en una expresión	(|)	(

LLAVE	Reconoce la llave abierta y cerrada al comienzo y al final de un método o clase	{ | }	{
PUNTUACIÓN	Reconoce símbolos de puntuación	, | ; | .	;
CLASE	Reconoce la directiva class	class	class
PUBLIC	Reconoce la directiva public	public	public
STATIC	Reconoce la directiva static	static	static
INCREMENTO	Reconoce una variable que se incrementa	Variable ++ | variable +=	Var++
DECREMENTO	Reconoce una variable que se decrementa	Variable -- | variable -=	Var--
ASIGNACION	Reconoce el símbolo de asignación	=	=
SALTODELINEA	Reconoce los saltos de línea	\n|\r|\r\n	
ESPACIO	Reconoce los espacios en blanco	“ ”	
MAIN	Reconoce la cabecera de los métodos main	main(	main(

2. Manual de usuario:

1. Crear el archivo .java utilizando jflex:
   > jflex AnalizadorLexicoCup.flex
2. Crear los archivos parser.java y sym.java utilizando cup:
   > cup Sintaxis.cup
3. Compilar todos los .java generados anteriormente:
   > javac AnalizadorLexicoCup.java parser.java sym.java
4. Ejecutamos el parser compilado introduciendo como argumento el fichero que se quiere leer:
   > java parser <archivo>

3. Código Jflex

```
/* Sección de declaraciones de JFlex */
import java_cup.runtime.Symbol;
%%
%class AnalizadorLexicoCup
%type java_cup.runtime.Symbol
%cup
%full
%line
%char
/* Inicio de Expresiones regulares */
 NUMERO = [0-9]+
 ENTERO = "int"
 BOOLEAN = "boolean"
 RETORNO = "return"
 NORETORNO = "void"
 VARIABLE = [a-zA-Z] [a-zA-Z0-9_]*
 SALTODELINEA = \n|\r|\r\n
 ASIGNACION = "="
 PUNTUACION = ";" | "," |"."
 BUCLE = "for" | "while" | "do"
 SUMA = "+"
 RESTA = "-"
 POR = "*"
 ENTRE = "/"
 RELACIONAL = "<"|"<="|">"|">="|"=="|"!="
 LOGICA = "&&"|"||"|"!"
 ESPACIO = " "
 PARENTESIS = "(" | ")"
 LLAVE = "{" | "}"
 VERDADERO = "True"
```

FALSO = "False"

MAIN = "main("

PUBLIC = "public"

STATIC = "static"

CLASS = "class"

/* Finaliza expresiones regulares */

%%

/* Finaliza la sección de declaraciones de JFlex */

/* Inicia sección de reglas */

// Cada regla está formada por una {expresión} espacio {código}

{NUMERO} { return new Symbol(sym.NUMERO, yytext()); }

{SALTO} { }

{ASIGNACION} { return new Symbol(sym.ASIGNACION, yytext()); }

{PUNTUACION} { return new Symbol(sym.PUNTUACION, yytext()); }

{BUCLE} { return new Symbol(sym.BUCLE, yytext()); }

{SUMA} { return new Symbol(sym.SUMA, yytext()); }

{RESTA} { return new Symbol(sym.RESTA, yytext()); }

{POR} { return new Symbol(sym.POR, yytext()); }

{ENTRE} { return new Symbol(sym.ENTRE, yytext()); }

{LOGICA} { return new Symbol(sym.LOGICA, yytext()); }

{RELACIONAL} { return new Symbol(sym.RELACIONAL, yytext());}

{ESPACIO} { }

{ENTERO} { return new Symbol(sym.ENTERO, yytext()); }

{BOOLEAN} { return new Symbol(sym.BOOLEAN, yytext()); }

{RETORNO} { return new Symbol(sym.RETORNO, yytext()); }

{NORETORNO} { return new Symbol(sym.NORETORNO, yytext()); }

{CLASS} { return new Symbol(sym.CLASS, yytext()); }

{STATIC} { return new Symbol(sym.STATIC, yytext()); }

{PUBLIC} { return new Symbol(sym.PUBLIC, yytext()); }

{PARENTESIS} { return new Symbol(sym.PARENTESIS, yytext()); }

{LLAVE} { return new Symbol(sym.LLAVE, yytext()); }

{VERDADERO} { return new Symbol(sym.VERDADERO, yytext()); }

{FALSO} { return new Symbol(sym.FALSO, yytext()); }

{MAIN} { return new Symbol(sym.MAIN, yytext()); }

{VARIABLE} { return new Symbol(sym.VARIABLE, yytext()); }

/* Finaliza la sección de reglas*/

4.Código Cup

```
import java_cup.runtime.*;

import java.io;

parser code {:

   public static void main (String args[]) throws Exception{

      FileInputStream f = new FileInputStream(args[0]);

      DataInputStream r = new DataInputStream(f);

      try{

         new parser(new Yylex(r)).parse();

      }

      catch (Exception e){

         System.out.println("Análisis incorrecto");

         System.exit(1);

      }

      System.out.println("Análisis correcto");

   }

:}

terminal NUMERO, ENTERO, BOOLEAN, RETORNO, NORETORNO, VARIABLE, SALTODELINEA,
ASIGNACION, PUNTUACION, BUCLE, SUMA, RESTA, POR, ENTRE, RELACIONAL, LOGICA, ESPA
CIO, PARENTESIS, LLAVE, VERDADERO, FALSO, MAIN, PUBLIC, STATIC, CLASS;

non terminal incremento, decremento, metodo, clase, principal, llamada;

precedence left SUMA, RESTA;

precedence left POR, ENTRE;

incremento ::= VARIABLE SUMA SUMA;

decremento ::= VARIABLE RESTA RESTA;

metodo ::= PUBLIC STATIC ENTERO VARIABLE | PUBLIC STATIC BOOLEAN VARIABLE | PUBLIC
 STATIC NORETORNO VARIABLE;

clase ::= PUBLIC CLASS;

principal ::= PUBLIC STATIC NORETORNO MAIN;

llamada ::= VARIABLE PARENTESIS;
```

5.Explicación non terminal:

- Incremento: Sirve para reconocer cuando una variable tiene que ser incrementada. Está compuesto por los terminal VARIABLE (pudiendo ser cualquier cadena válida recogida en la expresión regular escrita) y por dos veces el terminal SUMA (signo +). Ej: *var++*.
- Decremento: Sirve para reconocer cuando una variable tiene que ser incrementada. Está compuesto por los terminal VARIABLE (pudiendo ser cualquier cadena válida recogida en la expresión regular escrita) y por dos veces el terminal Resta (signo -). Ej: *var--*.
- Metodo: Sireve para reconocer la cabecera de un método. Está compuesto por los terminal PUBLIC (reconoce la cadena public), el terminal STATIC (reconoce la cadena static), ENTERO, BOOLEAN o NORETORNO (reconoce cuando el método devuelve un int, un boolean o no devuelve nada (void)) y el terminal VARIABLE (nombre del método). Ej: *public static int método*.
- Clase: Define lo que es una clase y reconoce la cabecera de esta. Esta formada por los terminal PUBLIC (reconoce la cadena public) y el terminal CLASS (reconoce la cadena class). Ej: *public class*.
- Principal: Sirve para reconocer el método main del programa. Esta fomardo por los terminal PUBLIC (reconoce la cadena public), el terminal STATIC (reconoce la cadena static), el terminal NORETORNO (reconoce la cadena void) y el terminal MAIN (reconoce la cadena main). Ej: *public static void main*.
- Llamada: Sirve para reconocer cuando se está produciendo una llamada a un método. Está compuesta por los terminal VARIABLE (cualquier cadena válida recogida dentro de la expresión regular que actúa como nombre del método) y el terminal PARENTESIS (reconoce el paréntesis de abertura delante del nombre del método). Ej: método(.