HITO 1 TAC

Javier Loro Carrasco

Diego Cordero Contreras

Mohamed Essalhi

Índice:

1. Tarea realizada

1.1. Enunciado

Lo que se nos pide en este primer hito es reconocer los elementos léxicos que nos encontramos en el lenguaje DOT e identificarlos. Además, debemos definir un token para cada elemento e implementar cada una de estas expresiones regulares con JFLEX para que tenga una acción asociada.

En el caso concreto de este hito, el programa que creemos como parser, la acción correspondiente será imprimir el elemento de DOT junto a su identificador.

1.2. Ejercicio 1

Ejercicio 1. Estudiar los componentes léxicos del lenguaje DOT y definir una tabla con la información de cada elemento léxico.

Expresión Regular	Token
arbol	ARBOL
node	NODO
shape	SH
label	ETIQUETA
color	COLOR
fontcolor	COLOR_FUENTE
style	FUENTE
edge	ARISTA
dir	DIRECCION
hijos	HIJOS
square	SH_CUADRADO
circle	SH_CIRCULO
doublecircle	SH_DOBLECIRCULO
rectangle	SH_RECTANGULO
blue	CF_AZUL
green	CF_VERDE
red	CF_ROJO
yellow	CF_AMARILLO
“blue”	C_AZUL
“green”	C_VERDE
“red”	C_ROJO
“yellow”	C_AMARILLO
bold	FT_BOLD
dashed	FT_DASHED
solid	FT_SOLID
none	DIR_NONE
forward	DIR_FORWARD

back	DIR_BACK
both	DIR_BOTH
{	LLAVE_A
[	CORCHETE_A
}	LLAVE_C
]	CORCHETE_C
=	IGUAL
,	COMA
;	PUNTO_COMA

Expresión Regular	Descripción	Token	Ejemplo
“([a-zA-Z]*+(" "| \t | \f | \n | \r)*+[0-9]*+(á | é | í | ó | ú)*+(_ | \! | ¡ | ¿ | \?)*)* “	Reconoce cualquier cadena comprendida entre dos comillas (“”)	ETIQUETA_VALOR	“¿Cómo?”
[a-zA-Z]([a-zA-Z]*+[0-9]*+(á | é | í | ó | ú)*+(_ | \! | ¡ | ¿ | \?)*)*	Reconoce cualquier palabra que empiece con un carácter de a-z y seguido por cualquier otro carácter	IDENTIFICADOR	Diccionario
//+([a-zA-Z]*+(" "| \t | \f | \n | \r)*+[0-9]*+(á | é | í | ó | ú)*+(_ | \! | ¡ | ¿ | \?)*)*+\n	Reconoce cualquier cadena que empiece con // y termina con un salto de línea	COMENTARIO//	//Temporal \n
/*+([a-zA-Z]*+(" "| \t | \f | \n | \r)*+[0-9]*+(á | é | í | ó | ú)*+(_ | \! | ¡ | ¿ | \?)*)*+*/	Reconoce cualquier cadena que empiece con /* y acabe con */	COMENTARIO/*	/*Funcionalidad del método*/

1.3. Ejercicio 2 y 3

Ejercicio 2. Implementar en Jflex cada una de las ERs

Ejercicio 3. Definir un nombre para cada elemento léxico. En JFLEX deberemos obtener como salida: <Nombre léxico > => <Cadena reconocida>.

Para la realización de esta tarea procedemos a convertir la información de la anterior tabla a un archivo .jflex. En este archivo declararemos, cada elemento del lenguaje a reconocer y la acción de imprimirlo en java para demostrar que se reconoce de forma correcta. Por ejemplo, para el caso de los elementos de "arbol" o "node", la sintaxis sería la siguiente:

```
arbol {System.out.println("ARBOL => " + yytext());}

node {System.out.println("NODO => " + yytext());}
```

Quedando así de esta manera definidos todos los elementos léxicos del lenguaje que deseamos compilar. En cuanto a los elementos de la segunda tabla que requieren de una detección distinta ya que son elementos que admiten distintas estructuras y no solo una palabra reservada, obtenemos la siguiente sintaxis:

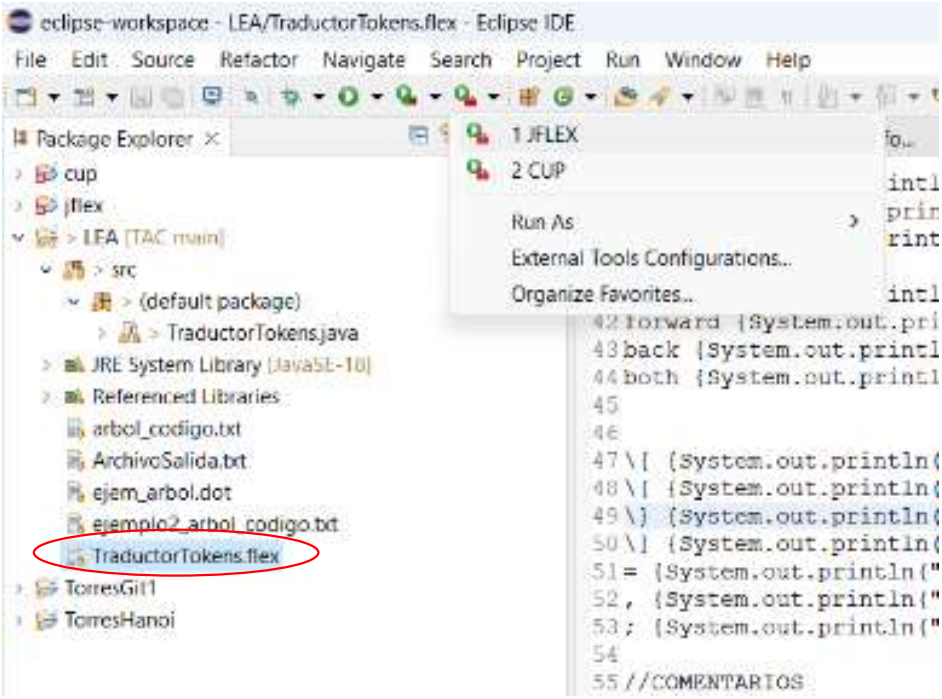
```
//COMENTARIOS

"//"+([a-zA-Z]*+(" "| \t | \f | \r )*+[0-9]*+(á | é | í | ó |
ú)*+( _ | \! | ¡ | ¿ | \? )*)*+\n
{System.out.println("COMENTARIO// => " + yytext());}

"/*"+([a-zA-Z]*+(" "| \t | \f | \n | \r )*+[0-9]*+(á | é | í |
ó | ú)*+( _ | \! | ¡ | ¿ | \? )*)*+"*/"
{System.out.println("COMENTARIO/* => " + yytext());}

//IDENTIFICADORES

([a-zA-Z]([a-zA-Z]*+[0-9]*+(á | é | í | ó | ú)*+( _ | \! | ¡ |
¿ | \? )*)*) {System.out.println("IDENTIFICADOR => " +
yytext());}

//LABELS

\"(([a-zA-Z]*+(" "| \t | \f | \n | \r )*+[0-9]*+(á | é | í | ó
| ú)*+( _ | \! | ¡ | ¿ | \? )*)*)\"
{System.out.println("ETIQUETA_VALOR => " + yytext());}
```

2. Manual de usuario

Para poder ejecutar este hito son principalmente necesarios los siguientes archivos:

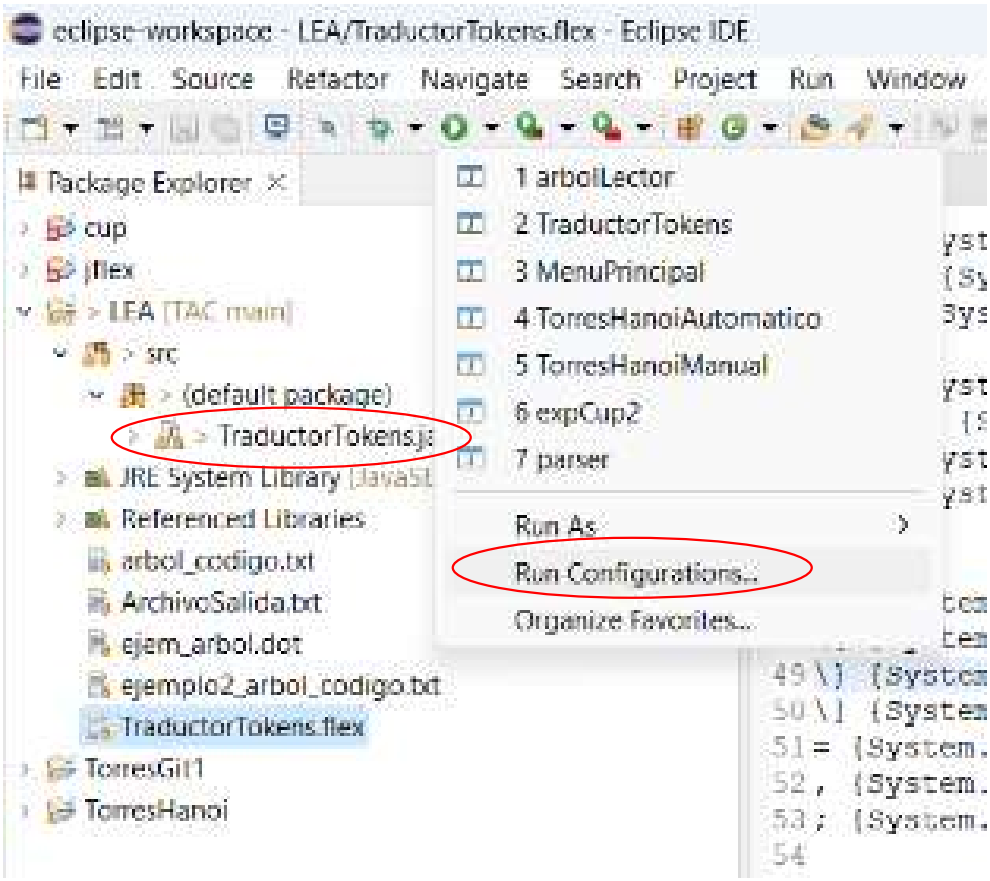
1. “TraductorTokens.jflex”
2. Código de ejemplo en formato “.txt” o “.dot”

Paso 1. El primer paso sería compilar el archivo jflex. Para ello se debe ejecutar como archivo JFLEX según se nos indica en el tutorial presente en el campus virtual

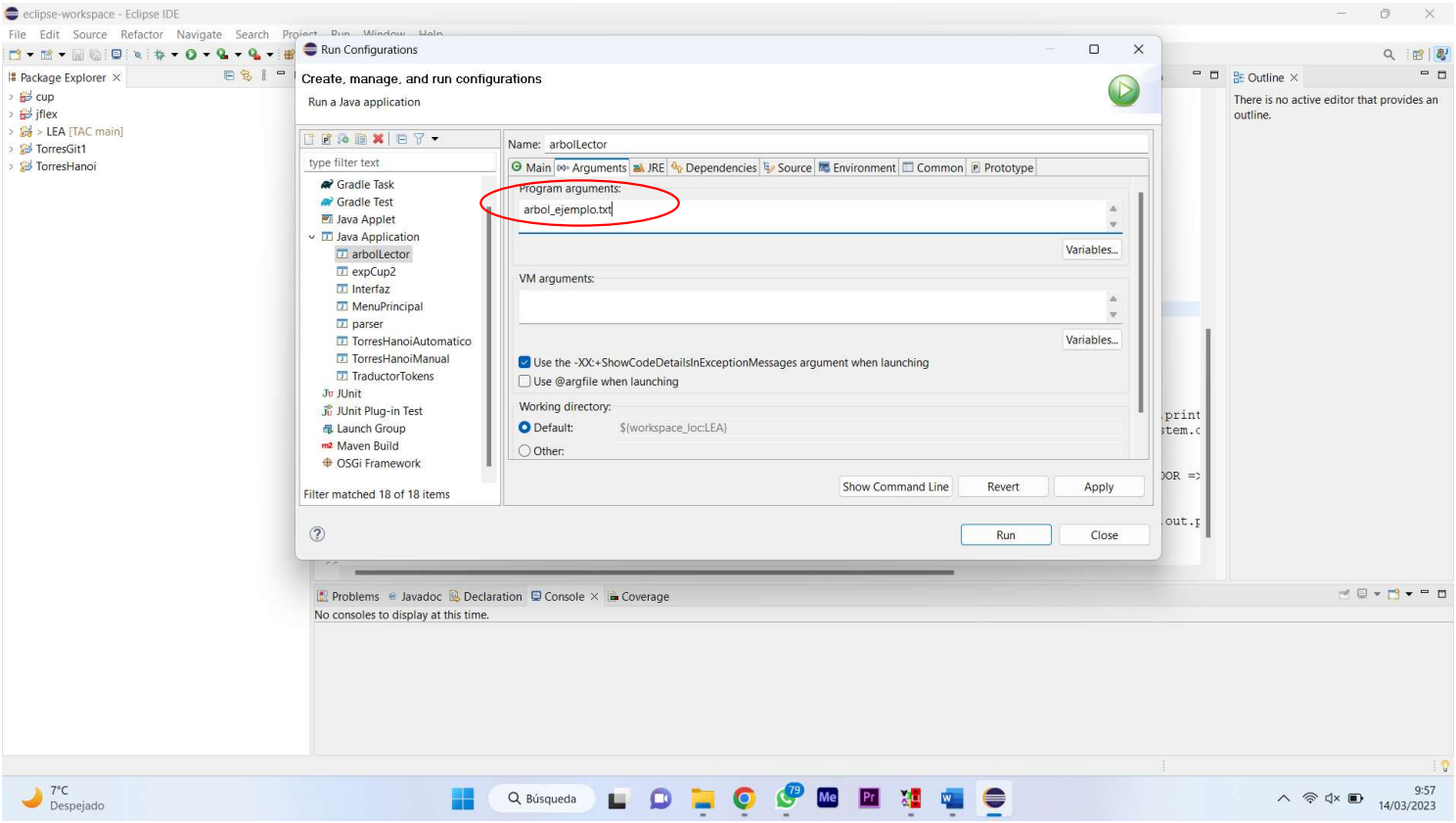


Paso 2. Una vez compilado o ejecutado como jflex, se nos debería haber generado en la carpeta SRC del proyecto un archivo llamado “TraductorTokens.java”. Este será nuestro “compilador” de los códigos de ejemplo, pero para usarlo primero debemos crear una aplicación en eclipse para su correcto comportamiento y poder así pasarle el código de ejemplo que queramos por parámetros.

Para crear la aplicación solo tendremos que irnos al apartado de ejecutar el programa, ver más y seleccionar la opción de run configurations:



Dentro debemos crear una nueva aplicación que podremos llamar como queramos, en este caso se llamará “arbolLector”. De las configuraciones que nos aparecen al crear una nueva aplicación, solo tendremos que modificar en la pestaña de argumentos el campo “program arguments”. Dentro de este campo pondremos el nombre del fichero de texto donde tengamos el código que queremos compilar (Es importante que este dichero este en la carpeta raíz del proyecto):



Una vez realizadas estas modificaciones, solo haría falta ejecutarlo con los nuevos parámetros y se nos generaría un archivo de salida tal y como el que adjuntamos en la tarea.

Con la entrega también proporcionamos las salidas tras ejecutar los archivos de ejemplo, estos son:

1- “arbol_ejemplo.txt” → “resultado_arbol_ejemplo.txt”
2- “ejem_arbol.dot” → “resultado_ejem_arbol.dot”
3- “ejemplo2_arbol_codigo.txt” → “resultado_ejemplo2_arbol_codigo.txt”