

# UN PEQUEÑO ANALIZADOR

## PARTE II: ANÁLISIS LÉXICO

### SESIÓN ANÁLISIS LÉXICO

# Análisis léxico

Un **analizador léxico** lee una cadena de texto y realiza diversas acciones sobre ella.

Ejemplos:

- Busca todas las mayúsculas de un texto y las convierte en minúsculas*

AqUI, AprEndIEndO AUtOmAtAs



aqui, aprendiendo automatas

- Cada vez que encuentre una serie de dígitos, los sustituye por “es un entero” o por “es un real” dependiendo del caso*

El numero 7 y el numero 3.8



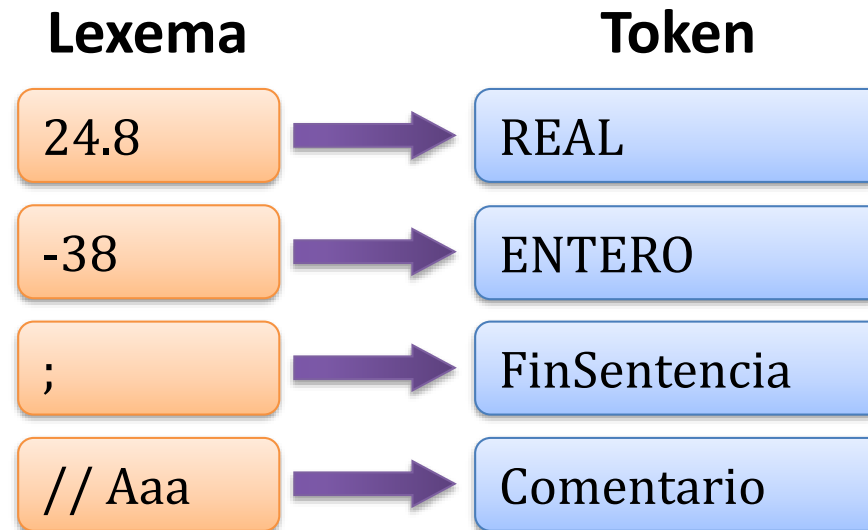
El numero es un entero y el numero es un real

# Análisis léxico

En el ámbito de los compiladores:

- La cadena de texto recibida es el código fuente.
- Identifica elementos del lenguaje (lexemas)
- Asocia un token a cada lexema

Ejemplo:



# Análisis léxico

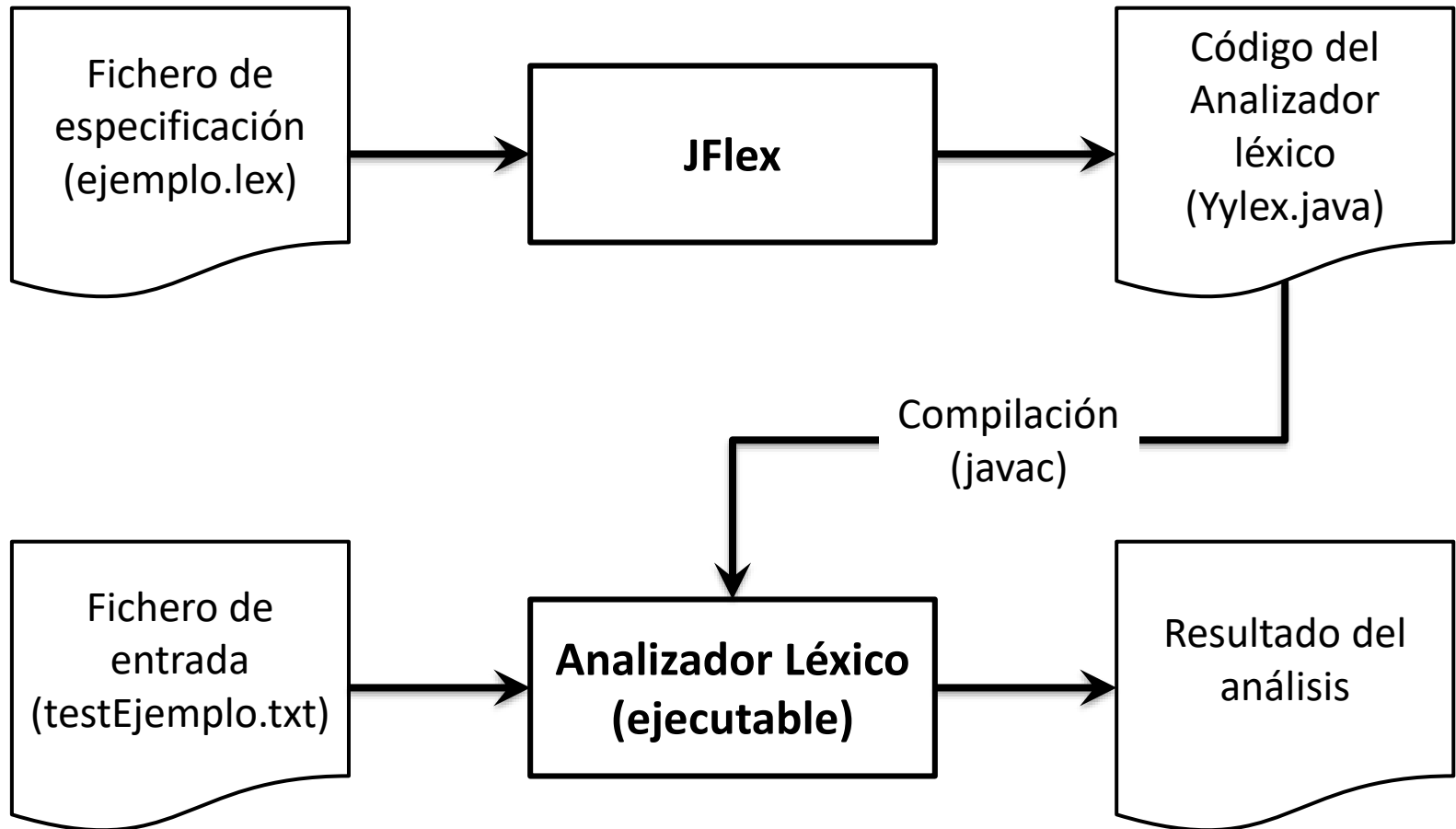
- La construcción de analizadores léxicos es muy compleja.
- Se suelen utilizar **generadores de analizadores léxicos**.

## La herramienta JFlex:

- Herramienta en Java para generar analizadores léxicos.
- A partir de un **fichero de especificación**, genera automáticamente el código fuente (en Java) de un analizador léxico para el lenguaje descrito.
- Los lexemas se identifican mediante **Expresiones Regulares**.
- A cada lexema se le asigna un token.

# La herramienta JFlex

Esquema general:



# Analizador léxico para Aemedede

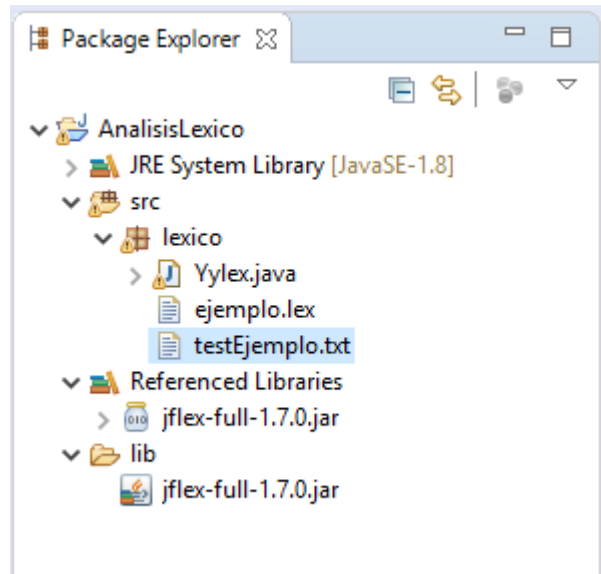
## ➤ Ejercicio:

- Descarga el fichero “AnálisisLexico” del Campus Virtual.
- Descomprime el fichero donde desees.
- Abre Eclipse e importa el proyecto que acabas de descargar.
- Para ello:
  1. Vete a File → Import... → General → Existing Projects into Workspace
  2. Opción “Select root directory”: Busca la carpeta en la que has descomprimido el proyecto.
  3. Click en “Select All” y finalmente en “Finish”

# Analizador léxico para Aemedede

## ➤ Ejercicio:

- A continuación:
  4. Vete a File → Import... → Run/Debug → Launch Configurations
  5. Opción “From directory”: Busca la carpeta en la que has descomprimido el proyecto.
  6. Marca “AnalizadorLexico” y finalmente en “Finish”.

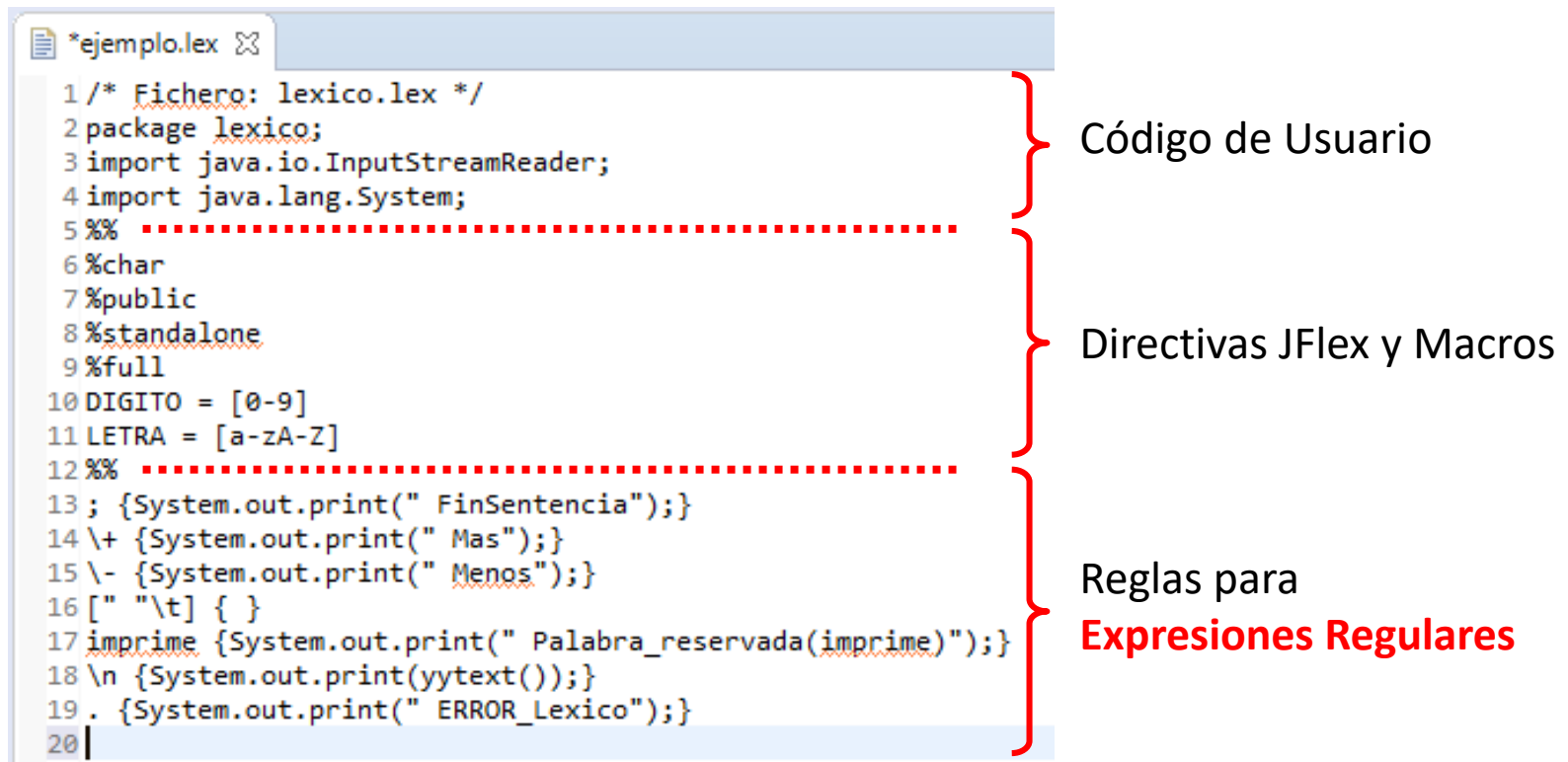


← A la izquierda de Eclipse, debería quedar así.

Avisa al profesor en caso contrario

# El fichero de especificación

El fichero de especificación (ejemplo.lex) se divide en 3 secciones:



The image shows a text editor window titled '\*ejemplo.lex'. The code is as follows:

```
1 /* Fichero: lexico.lex */
2 package lexico;
3 import java.io.InputStreamReader;
4 import java.lang.System;
5 %% .....
6 %char
7 %public
8 %standalone
9 %full
10 DIGITO = [0-9]
11 LETRA = [a-zA-Z]
12 %% .....
13 ; {System.out.print(" FinSentencia");}
14 \+ {System.out.print(" Mas");}
15 \- {System.out.print(" Menos");}
16 [" "\t] { }
17 imprime {System.out.print(" Palabra_reservada(imprime)");}
18 \n {System.out.print(yytext());}
19 . {System.out.print(" ERROR_Lexico");}
20 |
```

Red curly braces on the right side of the code group the lines into three sections:

- Código de Usuario**: Lines 1 through 4.
- Directivas JFlex y Macros**: Lines 5 through 11.
- Reglas para Expresiones Regulares**: Lines 12 through 19. The text 'Expresiones Regulares' is written in red.



# El fichero de especificación

## Código de usuario:

- Esta sección se copia directamente al fichero java generado.
- Suele utilizarse para incluir los “import” necesarios
- En esta práctica, **dejaremos esta sección como está.**

## Directivas JFlex y Macros:

- Directivas nativas para configurar el funcionamiento de JFlex.
- Las macros facilitan la definición de Expresiones Regulares
- Se definen siguiendo el siguiente formato:

`<nombreMacro> = (<definiciónMacro>)`

- donde la definición debe ser una Expresión Regular

# El fichero de especificación

## Reglas para Expresiones Regulares:

- Especifica el formato que tiene cada lexema y le asocia una acción (por ejemplo, asignar un token)
- Cada lexema está definido en una línea mediante una Expresión Regular
- A la derecha de cada E.R. se define su acción asociada entre llaves.
- Si una cadena de texto encaja en más de una E.R.:
  - Se elige aquella E.R. que lee más caracteres (la expresión más larga)
  - En caso de empate, la que aparezca antes en el fichero de especificación.
- Toda entrada debe ser reconocida por alguna E.R.

# El fichero de especificación

## Expresiones Regulares en JFlex:

- La unión se representa con el operador |
  - En lugar del operador + que usamos en clase
- La concatenación se representa juntando expresiones
  - Como en clase
- El cierre de Kleene se representa con \*
  - El símbolo + representa el cierre positivo: un cierre de Kleene donde la palabra vacía queda excluida. ( $a+ = aa^*$ )
- Para especificar alguno de los siguientes símbolos, debe colocarse el símbolo \ delante, o ponerse entre comillas dobles:

+ - \* / ? | ( ) ^ \$ . [ ] { } “ \

# El fichero de especificación

## Expresiones Regulares en JFlex:

- El operador [ ] permite abreviar la unión de conjuntos

0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 = [0123456789]

- El carácter '.' representa "cualquier carácter".
  - Excepto el salto de línea
- Las macros deben escribirse entre llaves
- La interrogación indica que la expresión regular es opcional.
  - Ejemplo: (ab)? Representa el lenguaje  $\{\Lambda, ab\}$
- El símbolo '-' permite expresar rangos de valores
- '^' especifica el complementario

[0123456789] = [0-9]    [^0-9]    (cualquier carácter que no sea dígito)

[abcde...xyz] = [a-z]    [^a-zA-Z] (cualquier carácter que no sea letra)

[ABCDE...XYZ] = [A-Z]

# Nuestro analizador

¿Qué hay ya hecho de nuestro analizador?

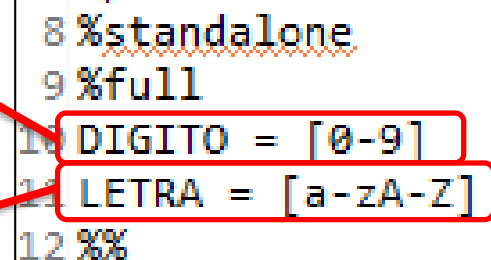
**Macros:**

- Se incluyen dos macros:

**Macro DIGITO:** Expresión Regular que identifica un dígito

**Macro LETRA:** Expresión Regular que identifica una letra mayúscula o minúscula

```
5 %%  
6 %char  
7 %public  
8 %standalone  
9 %full  
10 DIGITO = [0-9]  
11 LETRA = [a-zA-Z]  
12 %%
```



# Nuestro analizador

**¿Qué hay ya hecho de nuestro analizador?**

## **Expresiones regulares:**

- Punto y coma. Muestra por consola “FinSentencia”.
- Operador +: Muestra por consola “Mas”.
- Operador -: Muestra por consola “Menos”.
- Palabra clave “imprime”.
- Expresiones para filtrar espacios en blanco y saltos de línea (16 y 18)
- La última línea acepta cualquier cadena no reconocida anteriormente y la da como error de compilación.

```
13 ; {System.out.print(" FinSentencia");}  
14 \+ {System.out.print(" Mas ");}  
15 \- {System.out.print(" Menos ");}  
16 [" "\t"] { }  
17 imprime {System.out.print(" Palabra_reservada(imprime)");}  
18 \n {System.out.print(yytext());}  
19 . {System.out.print("ERROR Lexico");}
```

# Analizador léxico para Aemedede

## ➤ Ejercicio:

- A partir de este fichero, vamos a generar el analizador léxico.
- Para ello, vete a Run → Run Configurations → Java Application → GeneraLexico. Haz click en Run.
- ¿Qué es lo que ha salido por la consola de Eclipse?
- ¿Qué crees que ha hecho el JFlex con nuestras Expresiones Regulares?
- Vete a la ventana izquierda, selecciona el paquete “léxico” y presiona F5. Se actualizará el fichero llamado Yylex.java
- **Yylex.java**: Analizador léxico programado en Java

# Analizador léxico para Aemedede

## ➤ Ejercicio:

- Ahora probemos si nuestro analizador funciona.
- En la ventana izquierda, abre el fichero testEjemplo.txt.
  - Contiene un programa escrito en Aemedede
- Vete a Run → Run Configurations → Java Application → EjecutaLexico. Haz click en Run.
- ¿Qué es lo que ha salido por la consola de Eclipse?
- ¿Qué ha hecho el analizador léxico con nuestro código fuente?
- ¿Por qué hay tantos errores léxicos?



# Analizador léxico para Aemedede

## ➤ Ejercicios:

Modifica el fichero de especificación para que reconozca los siguientes lexemas:

1. Operador \*: Debe mostrar por consola “ Por ”.
2. Operador /: Debe mostrar por consola “ Entre ”.
3. Paréntesis: Al abrir un paréntesis debe mostrar “ Abre-Par”, y al cerrarlo debe mostrar “ Cierra-Par”
4. Números **enteros**: Reconoce cualquier número entero siguiendo el formato dado en la definición del lenguaje. Se les debe asociar la siguiente acción:

```
{System.out.print(" Entero("+yytext()+")");}
```

5. Números **reales**: Reconoce cualquier número real.

```
{System.out.print(" Real("+yytext()+")");}
```

# Analizador léxico para Aemedede

## ➤ Ejercicios:

Modifica el fichero de especificación para que reconozca los siguientes lexemas:

1. UOs de la Universidad de Oviedo: Cadenas que comienzan por UO en cualquier combinación de mayúsculas y minúsculas, seguidas de 4 a 6 dígitos.
2. Direcciones de correo uniovi: Nombre de usuario formado por letras, dígitos y puntos, pero ha de comenzar por una letra y no puede tener dos puntos seguidos ni acabar en punto. El dominio será uniovi.es ó uniovi.com, separado del nombre de usuario por una @

# Analizador léxico para Aemedede

## ➤ Ejercicios:

Modifica el fichero de especificación para que reconozca los siguientes lexemas:

6. Palabras clave “inicio” y “fin”. Debe mostrar un mensaje similar al de la palabra clave “imprime”.
7. Operador de asignación :=. Debe mostrar por consola “Asigna”
8. Variable: El nombre de la variable debe seguir las reglas dadas en la definición del lenguaje. Debes asignarle la siguiente acción

```
{System.out.print(" Variable("+yytext()+")");}
```

9. Cadena de texto. Debe seguir el formato dado en la definición del lenguaje. Debes asignarle la siguiente acción:

```
{System.out.print(" Cadena("+yytext()+")");}
```

# Analizador léxico para Aemedede

## ➤ Ejercicios:

10. Añade los siguientes elementos para definir la expresión regular que reconoce los comentarios:
  - a) Añade la macro SIMBOLO que reconozca todos los símbolos que puedan aparecer en un comentario
  - b) Añade la macro LINEA que reconozca una secuencia de uno o más SIMBOLOS.
  - c) Añade la macro DOBLEBARRA que reconozca la combinación “//”
  - d) Utiliza estas macros para definir la expresión regular que reconozca un comentario.
  - e) Se le debe asignar la acción que muestre por consola “Comentario”.
11. Actualiza la E.R. que reconoce las cadenas de texto para que aproveche las macros creadas en el ejercicio 10

# Analizador léxico para Aemedede

Para probar el nuevo analizador léxico:

1. Ir a Run → Run History → GeneraLexico
2. Vete a la ventana izquierda, selecciona el paquete “léxico” y presiona F5 (Actualiza el fichero Yylex.java)
3. Ir a Run → Run History → EjecutaLexico
4. Comprueba que no hay errores léxicos en el programa
  - En ese caso, quiere decir que hay alguna expresión regular mal.