



UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA  
Escuela Técnica Superior de Ingeniería Informática



**Procesadores de Lenguajes**

# Práctica de Procesadores de Lenguajes

Análisis Léxico con JLex

*Javier Vélez Reyes*  
[jvelez@lsi.uned.es](mailto:jvelez@lsi.uned.es)

*Javier Vélez Reyes* [jvelez@lsi.uned.es](mailto:jvelez@lsi.uned.es)

## Objetivos

- Funcionamiento del generador de escáneres JLex
  - Introducción
  - Utilización
- Especificación de escáneres en JLex
  - Especificación de código de ayuda
  - Especificación de patrones
  - Especificación de acciones
- Integración con el generador de parsers Cup

# Índice

- Introducción
- Especificación
  - Código de Usuario
  - Directivas JLex
  - Reglas patrón-acción

# Introducción

- Pasos
  - Especificar **scanner.lex** en JLex
  - Lanzar JLex a ejecución con **scanner.lex**

```
java -classpath %classpath%;dir Main scanner.lex
```

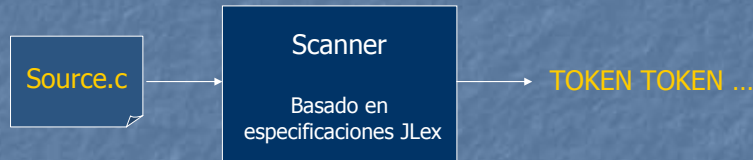
- Renombrar a **Yylex.java** (opcional si se configura JLex)
- Compilar **Yylex.java** a **Yylex.class**
- Interpretar **Yylex.class**



# Introducción

## ■ Utilización

- Fichero fuente de entrada **source.c**
- Llamada a `yylex()`
- Obtener secuencia de tokens



# Introducción

## ■ Funcionamiento

### Scanner.lex

```
import java.lang.System;

class Sample {
    ...
}

class Utility {
    ...
}

class Yytoken {
    ...
}

%%

%line
%char
%state COMMENT

ALPHA      = [A-Za-z]
DIGIT      = [0-9]
WHITE_SPACE = [\ \t\b\012]
WHITE_SPACE_CHAR = [\b \t\b\012]
...
%%

<YYINITIAL> " " ( return (new Yytoken(...));
<YYINITIAL> ":" ( return (new Yytoken(...));
<YYINITIAL> ";" ( return (new Yytoken(...));
<YYINITIAL> "(" ( return (new Yytoken(...));
<YYINITIAL> ")" ( return (new Yytoken(...));
<YYINITIAL> "[" ( return (new Yytoken(...));
...
```



### Scanner.lex.java

```
import java.lang.System;

class Sample {
    ...
}

class Utility {
    ...
}

class Yytoken {
    ...
}

class Yylex {

    public Yylex (java.io.Reader reader) {
        ...
    }

    public Yylex (java.io.InputStream instream) {
        ...
    }

    ...
    ...
    ...

    public Yytoken yylex ()
        throws java.io.IOException {
        ...
        return Yytoken(...);
    }
}
```

# Especificación

## ■ Tres partes separadas por %%

- Código de Usuario
  - Código de ayuda
  - Se copia al inicio
  - Import, package, helper class
- Directivas JLex
  - Macros
  - Ordenes a JLex
  - Configurar comportamiento
- Reglas patrón-acción
  - Determinar el DT del scanner
  - Determinar acciones a tomar
  - Determinar patrones
  - Gestionar estados

Scanner.lex

```
import java.lang.System;

class Sample {
    ...
}

class Utility {
    ...
}

class Yytoken {
    ...
}

%%

%line
%char
%state COMMENT

ALPHA      = [A-Za-z]
DIGIT      = [0-9]
WHITE_SPACE = [\t\b\012]
WHITE_SPACE_CHAR = [\n\ \t\b\012]
...

%%

<YYINITIAL> ">

```

# Código de Usuario

- Código de usuario
  - Código de ayuda utilizado dentro de las acciones
  - Se copia al inicio del programa
  - Suelen aparecer
    - Importación de clases
    - Declaración de paquetes
    - Clases de ayuda
    - Definición de la clase Token

No se recomienda definir la clase Token dentro del fichero jlex. Para mayor claridad es preferible implementarlo en un fichero aparte e importarlo desde aquí con una cláusula import.

Para la practica, sin embargo, se recomienda utilizar la clase Symbol de la herramienta Cup

```
import java.lang.System;

class Sample {
    ...
}

class Utility {
    ...
}

class Yytoken {
    ...
}

%%
```



```
import java.lang.System;

class Sample {
    ...
}

class Utility {
    ...
}

class Yytoken {
    ...
}
```

# Directivas JLex

- Código Interno a Clase de Analizador
  - El código entre `%{` y `%}` se copia al inicio de YyLex

```
%  
  
%{  
    private int comment_count = 0;  
%}  
  
%line  
%char  
%state COMMENT  
  
ALPHA=[A-Za-z]  
DIGIT=[0-9]  
NONNEWLINE_WHITE_SPACE_CHAR=[\ \t\b\012]  
WHITE_SPACE_CHAR=[\n\ \t\b\012]  
  
%}
```



```
...  
class Yylex {  
    private final int YY_BUFFER_SIZE = 512;  
    private final int YY_F = -1;  
    private final int YY_NO_STATE = -1;  
    private final int YY_NOT_ACCEPT = 0;  
    private final int YY_START = 1;  
    private final int YY_END = 2;  
    private final int YY_NO_ANCHOR = 4;  
    private final int YY_BOL = 128;  
    private final int YY_EOF = 129;  
  
    private int comment_count = 0;  
    private java.io.BufferedReader yy_reader;  
    private int yy_buffer_index;  
    private int yy_buffer_read;  
    private int yy_buffer_start;  
    private int yy_buffer_end;  
    private char yy_buffer[];  
    private int yychar;  
    private int yyline;  
    private boolean yy_at_bol;  
    private int yy_lexical_state;  
    ...  
}
```

# Directivas JLex

- Init
  - El código entre `%init{` y `%}` se copia al constructor

```
%  
  
%{  
    private int comment_count = 0;  
%}  
  
%init{  
    << init code >>  
%}  
%line|  
%char  
%state COMMENT  
  
ALPHA=[A-Za-z]  
DIGIT=[0-9]  
NONNEWLINE_WHITE_SPACE_CHAR=[\ \t\b\012]  
WHITE_SPACE_CHAR=[\n\ \t\b\012]  
  
%}
```



```
...  
Yylex (java.io.Reader reader) {  
    << init code >>  
    this ();  
    if (null == reader) {  
        throw (new Error("Error: Bad input stream in  
    })  
    yy_reader = new java.io.BufferedReader(reader);  
}  
  
Yylex (java.io.InputStream instream) {  
    << init code >>  
    this ();  
    if (null == instream) {  
        throw (new Error("Error: Bad input stream in  
    })  
    yy_reader = new java.io.BufferedReader(new java.io.  
    )  
}
```

# Directivas JLex

## ■ Initthrow

- **%initthrow{** excepción, excepción,... **%}** define excepciones que se lanzan en constructor

```
%
%{
    private int comment_count = 0;
%}

%initthrow {
    IOException
%}

%line
%char
%state COMMENT

ALPHA=[A-Za-z]
DIGIT=[0-9]
NONNEWLINE_WHITE_SPACE_CHAR=[\ \t\b\012]
WHITE_SPACE_CHAR=[\n\ \t\b\012]

%%
```



```
...
Yylex (java.io.Reader reader)
    throws IOException {
    this ();
    if (null == reader) {
        throw (new Error("Error: Bad
    )
    yy_reader = new java.io.BufferedRea
}

Yylex (java.io.InputStream instream)
    throws IOException {
    this ();
    if (null == instream) {
        throw (new Error("Error: Bad
    )
    yy_reader = new java.io.BufferedRea
}
...

```

# Directivas JLex

## ■ Yylex y Excepciones

- **%yylexthrow{** excepción , ... **%}** define excepciones lanzadas por la función yylex()

```
%
%{
    private int comment_count = 0;
%}

%yylexthrow{
    IOException
}

%line
%char
%state COMMENT

ALPHA=[A-Za-z]
DIGIT=[0-9]
NONNEWLINE_WHITE_SPACE_CHAR=[\ \t\b\012]
WHITE_SPACE_CHAR=[\n\ \t\b\012]

%%
```



```
...
public Ytoken yylex ()
    throws java.io.IOException {
    int yy_lookahead;
    int yy_anchor = YY_NO_ANCHOR;
    int yy_state = yy_state_dtrans[yy_lexical_state];
    int yy_next_state = YY_NO_STATE;
    int yy_last_accept_state = YY_NO_STATE;
    boolean yy_initial = true;
    int yy_this_accept;

    yy_mark_start();
    yy_this_accept = yy_acpt[yy_state];
    if (YY_NOT_ACCEPT != yy_this_accept) {
        yy_last_accept_state = yy_state;
        yy_mark_end();
    }
}

```

# Directivas JLex

- End of file y excepciones

- `%eof{ ... %}` se ejecuta tras encontrar EOF
- `%eofval{ ... %}` especifica el valor devuelto al EOF
- `%eofthrow{ excepción , ... %}` lanza al encontrar EOF

```
%
%
%{
%   private int comment_count = 0;
%}

%eof{
%   << eof-code >>
%}

%eofthrow{
%   IOException
%}

%line
%char
%state COMMENT

ALPHA=[A-Za-z]
DIGIT=[0-9]
NONNEWLINE_WHITE_SPACE_CHAR=[\ \t\b\O12]
WHITE_SPACE_CHAR=[\n \t\b\O12]

%
%
```



```
private yy_do_eof()
throws IOException {
    << eof-code >>
}
```

# Directivas JLex

- Definiciones de macros

- **Nombre = definición** define una macro
- La macro sirve para dulcificar la sintaxis en las reglas
- Debe estar definida en una sola línea
- Una macro puede usar otra **nombre = ...{nombre}...**

```
ALPHA=[A-Za-z]
DIGIT=[0-9]
NONNEWLINE_WHITE_SPACE_CHAR=[\ \t\b\012]
WHITE_SPACE_CHAR=[\n\ \t\b\012]
```



# Directivas JLex

- Definiciones de estados
  - **%state nombre, nombre,...** define estados
  - Deben estar contenidos en una línea
  - Puede hacer varias líneas definiendo estados
  - **YYINITIAL** es un estado definido por defecto

```
%state COMMENT
...
%%

<YYINITIAL> " " { return (new Ytoken(0,yytext(),yyline,yychar,yychar+1)); }
<YYINITIAL> ";" { return (new Ytoken(1,yytext(),yyline,yychar,yychar+1)); }
<YYINITIAL> ":" { return (new Ytoken(2,yytext(),yyline,yychar,yychar+1)); }
<YYINITIAL> "(" { return (new Ytoken(3,yytext(),yyline,yychar,yychar+1)); }
...

<COMMENT> "/*" { comment_count = comment_count + 1; }
<COMMENT> "*/" {
    comment_count = comment_count - 1;
    Utility.assert(comment_count >= 0);
    if (comment_count == 0) {
        yybegin(YYINITIAL);
    }
}
<COMMENT> (COMMENT_TEXT) { }
...|
```

# Directivas JLex

- Configuración
  - **%ignorecase** no diferencia entre mayúsculas y minúsculas
  - **%char** define la variable **yychar** para contar caracteres
  - **%line** define la variable **yyline** para indicar la línea
  - **%notunix** reconoce `\r\n` y `\n` como nueva línea
  - **%full** utiliza el código ASCII extendido 8 bits
  - **%unicode** utiliza codificación UNICODE de 16 bits



## Directivas JLex

- Configuración
  - **%class name** define el nombre de la clase del scanner
  - **%public** define la clase del scanner como pública
  - **%function name** define el nombre de la función yylex()
  - **%interface name** la clase implementa el interface name
  - **%type name** define el tipo de retorno de yylex()
  - **%integer** establece el tipo de retorno de yylex() a **int**
  - **%intwrap** establece el tipo de retorno de yylex() a **Integer**
  - **%yyeof** define la constante **Yylex.YYEOF**
    - Debe estar presente la directiva **%integer**
    - Al encontrar EOF se devuelve **Yylex.YYEOF**

## Directivas JLex

- Configuración
  - **%cup** configura JLex para integrarse con Cup
    - **%implements java\_cup.runtime.Scanner**
    - **%function next\_Token**
    - **%type java\_cup.runtime.Symbol**

# Reglas patrón-acción

- Tres partes
  - Estados léxicos
  - Expresión regular
  - Acciones java a ejecutar

```
<COMMENT> "/*" { comment_count = comment_count + 1; }
<COMMENT> "*/" {
    comment_count = comment_count - 1;
    Utility.assert(comment_count >= 0);
    if (comment_count == 0) {
        yybegin(YYINITIAL);
    }
}
<COMMENT> {COMMENT_TEXT} { }

<YYINITIAL> "," { return (new Yytoken(0, yytext(), yyline, yychar, yychar+1)); }
<YYINITIAL> ":" { return (new Yytoken(1, yytext(), yyline, yychar, yychar+1)); }
<YYINITIAL> ";" { return (new Yytoken(2, yytext(), yyline, yychar, yychar+1)); }
<YYINITIAL> "(" { return (new Yytoken(3, yytext(), yyline, yychar, yychar+1)); }
<YYINITIAL> ")" { return (new Yytoken(4, yytext(), yyline, yychar, yychar+1)); }
<YYINITIAL> "[" { return (new Yytoken(5, yytext(), yyline, yychar, yychar+1)); }
<YYINITIAL> "]" { return (new Yytoken(6, yytext(), yyline, yychar, yychar+1)); }
<YYINITIAL> "{" { return (new Yytoken(7, yytext(), yyline, yychar, yychar+1)); }
...
```

# Reglas patrón-acción

- Conflictos
  - Si más de una regla puede lanzarse ante una cadena de entrada el analizador escoge la regla que se ajusta a una cadena más larga
  - Ante igualdad de longitud el analizador escoge la regla que se encuentre definida en primer lugar. Por tanto el orden de ubicación de las reglas es importante
  - Si el analizador recibe una cadena que no se ajusta a ningún patrón se produce un error. Por tanto todas las entradas deben tener una regla. Para garantizar esto existe el patrón comodín `.` que se ajusta cualquier patrón no definido antes

# Reglas patrón-acción

## ■ Estados

- Una lista de estados puede preceder a cualquier regla
- Existe un estado por defecto llamado **YYINITIAL**
- Funcionamiento
  - El analizador esta en el estado **A**
  - Sólo se comprueban las reglas etiquetadas con A
  - Si no hay ajuste se produce un error
  - **yybegin(estado)** cambia de estado

# Reglas patrón-acción

## ■ Patrón

- No pueden contener espacios en blanco puesto que son interpretados como el final del patrón. Para representar espacios en blanco deben entrecomillarse
- Metacaracteres

\$	?	*	+	\		(	)	{	^	}
----	---	---	---	---	--	---	---	---	---	---

## ■ Composición

- Concatenación **e f**
- Opcionalidad **e | f**

# Reglas patrón-acción

## ■ Patrón

### ■ Secuencias de escape

- `\b` retroceso
- `\n` nueva línea
- `\t` tabulador
- `\f` avance página
- `\r` retorno de carro
- `\ddd` número octal
- `\xdd` número hexadecimal
- `\xdddd` número hexadecimal de 4 dígitos
- `\^C` carácter de control
- `\c` backslash seguido del carácter c

# Reglas patrón-acción

## ■ Patrón

### ■ Otros metacaracteres y metaexpresiones

- `$` fin de fichero
- `.` cualquier carácter menos `\n` (igual a `[^\n]`)
- `"..."` ...
- `{name}` expansión de una macro
- `*` clausura de Kleene
- `+` una o más repeticiones
- `?` Cero o una repetición
- `(...)` agrupar expresiones regulares
- `[...]` conjunto de caracteres
  - `{name}` expansión de una macro
  - `^` niega los siguientes caracteres
  - `a-b` rango de caracteres desde a hasta b

# Reglas patrón-acción

## ■ Acciones

- Consisten en bloques de código java
- Si la acción no devuelve un valor se llama a `yylex()`
- Una acción puede hacer recursión llamando a `yylex()`
- La función `yybegin(s)` salta al estado `s`
- En las acciones se suele consultar
  - `yyline` devuelve la línea
  - `yychar` devuelve el número del carácter
  - `yytext()` devuelve el lexema

# Bibliografía

[AJO] AHO, SETHI, ULLMAN: *Compiladores: Principios, técnicas y herramientas*, Addison-Wesley Iberoamericana, 1990



[GARRIDO] A. Garrido, J. Iñesta, F. Moreno y J. Pérez. 2002. *Diseño de compiladores*. Universidad de Alicante.





UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA  
Escuela Técnica Superior de Ingeniería Informática



**Procesadores de Lenguajes**

# Práctica de Procesadores de Lenguajes

Análisis Léxico con JLex

*Javier Vélez Reyes*  
[jvelez@lsi.uned.es](mailto:jvelez@lsi.uned.es)