JLex

Salvador Sánchez, Daniel Rodríguez
Departamento de Ciencias de la Computación
Universidad de Alcalá

JLex

- Lex en Java
  - JLex es un analizador Lexico en Java.
  - JLex fue desarrollado por Elliot Berk (Princeton University).
    - Mantenido por C. Scott Ananian.

`http://www.cs.princeton.edu/~appel/modern/java/JLex/`

Instalación JLex

- Java debe estar instalado.
  - **`http://java.sun.com/`**
  - J2SE(TM) Development Kit
  - (JRE – Java Runtime environment no es suficiente)
- Escoger el directorio p.e., [**`c:\java\`**]
- Creamos un subdirectorio llamado JLex en el directorio escogido en el punto 1. [**`c:\java\JLex`**]
- Copiamos el fichero Main.java en este subdirectorio. [**`c:\java\JLex\Main.java`**]
- Compilamos Main.java: **`javac Jlex\Main.java`**
- Añadir JLex a **CLASSPATH**

Generación y Ejecución del Analizador Léxico

- Generación del explorador (JLex -> Java):
  - **`java JLex.Main fichero.lex`**
- Compilación del explorador (Java -> Byte-Code):
  - **`javac fichero.lex.java`**
- Ejecurtar explorador
  - **`java Yylex`**
    - (**`Yylex`** es el nombre por defecto)

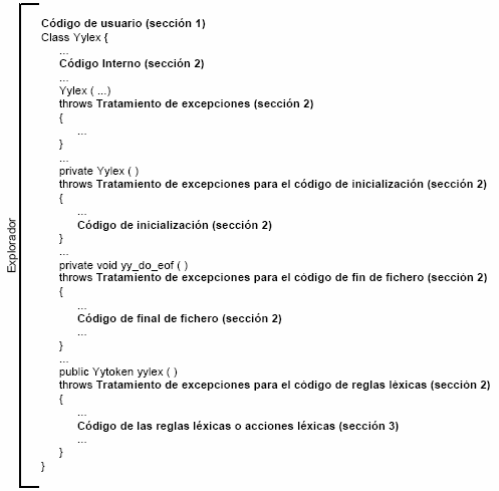
Estructura de un fichero JLex

```
Código de usuario
%%
Directivas de JLex
%%
Reglas de expresiones regulares
```

- **Código de usuario**, se copia directamente en el explorador generado. Esta sección se utiliza para incluir sentencias Java de importación y la definición de clases y tipos de datos que puedan ser de interés para la aplicación a generar.
- **Sección de directivas** se utiliza para particularizar algunas características del explorador generado y, también, es donde se declaran las macros y estados que se usarán en la definición de las reglas léxicas.
- **Sección de reglas léxicas**, contiene las reglas léxicas que se utilizarán para generar el explorador.

Código Generado

- El explorador se implementa con una clase que, por defecto, tiene el nombre **Yylex**.



Función *yylex()*

- La función **`yylex( )`** implementa el autómata de la gramática.
- Identifica el patrón que encaja con la entrada actual y retorna el testigo correspondiente al lexema identificado que, por defecto, es de tipo **`Yytoken`**.

Función *yylex( )* considerando la entrada por defecto

```
import java.io.*;
// Sección de código de usuario
(...)
%%
// Sección de directivas
%{
public static void main (String argv[])
throws java.io.IOException {
  Yylex yy = new Yylex(System.in);
  while (yy.yylex() != -1) ; }
%}
(...)
%%
// Sección de reglas léxicas
(...)
```


Sección de reglas léxicas

- La 3ª sección del fichero de especificación contiene las reglas que extraen tokens del fichero de entrada. Cada regla consta de:
-
  1. Lista opcional de estados
  2. Expresión regular
  3. Acción léxica

  [Lista_de_Estados] [Expresión_Regular] [Acción_Léxica]

- Ej:

```
[a-zA-Z][a-zA-Z_0-9]*  {System.out.println
                              ("Identificador");)
```


Metacaracteres

- Metacaracteres: **? * + | ( ) ^ $ [ ] { } " \ .**
- Para utilizarlos se puede:
  - Delimitarlos con comillas. Ej.: "*"
  - Anteponer la barra invertida: Ej.: *
  - Usar su código de carácter Ej.:
    - \ooo (3 dígitos en octal)
    - \x hh (2 dígitos hexadecimales)
    - \u hhhh (código unicode)
  - Caracteres especiles con secuencias de control:
    - \b Retroceso.
    - \n Nueva línea.
    - \t Tabulación.
    - \f Avance de línea.
    - \r Retorno de carro.
    - \^C Carácter de control

Expresiones Regulares

- Todo carácter del lenguaje es una expresión regular que se representa a sí mismo, excepto el espacio en blanco (ya que es un delimitador) y los metacaracteres.
  - Ej.: **a {System.out.println("Caracter a encontrado");}**
- carácter dólar (**$**) denota el final de una línea.
- El carácter (**^**) denota el inicio de una línea.
- Dos expresiones regulares consecutivas representan su concatenación. E.j.: **ab** es '**a**' concatenado con '**b**'
- Dos expresiones regulares separadas por la barra vertical (**|**), representa la opción entre ambas
- Los paréntesis se utilizan para agrupar expresiones regulares. E.j. **agu(a|v)ino** encaja '**aguaino**' o '**aguvino**'.

Expresiones Regulares II

- El asterisco (*****) representa la clausura de Klenne
- El signo suma (**+**) encaja con una o más repeticio
- El interrogante (**?**) encaja una o ninguna repetición
- El punto . encaja con cualquier carácter excepto nueva línea
- Los corchetes (**[...]**) denotan una clase de caracteres
  - **[abc]**
- El guión (-) se utiliza para definir un rango de caracteres
  - **[a-z]**
- El circumflejo (**^**), situado después de la apertura del corchete indica la negación de la clase.
  - **[^abc]** encaja con cualquier carácter que no sea '**a**', ni '**b**', ni '**c**'.

Ejemplos

- **[^a-zA-z]** Cualquier carácter que no sea letra mayúscula ni minúscula.
- **["A-Z"]** Los caracteres '**A**', '**Z**' o '**-**' (guión).
- **[A-Z][a-z]*** Texto que empieza por letra mayúscula.
- **[[A-Z][a-z]]** Expresión **ilegal**.
- **[\[a-z\]]** Letra minúscula, o apertura o cierre de corchetes.
- **[a-z*]** Letra minúscula o asterisco.
- **[+-]** El carácter '+' o el '-'.
- **[+-/]** Los caracteres entre '**+**' (ASCII 43) y '**/**' (ASCII 47).
- **[(ab)]** Los caracteres '**a**', '**b**', apertura '**(**' o cierre de paréntesis '**)**'

Ambigüedad

- En una especificación léxica, pueden darse las siguientes situaciones:
  - Dos o más reglas encajan con una misma entrada o lexema.
  - Una regla encaja con un lexema y una segunda regla encaja con un fragmento de este lexema.
- JLex resuelve esta ambigüedad aplicando los siguientes criterios:
  - Si más de una regla encaja con un fragmento de la entrada, se elige una regla que encaja con la cadena más larga.
  - Si más de una regla encaja con la misma cadena, se elige la regla que se encuentra en primera posición de entre ellas.

Acciones Léxicas

- Las acciones léxicas consiste en código Java entre llaves:

```
{ Código_Java }
```

- Valores léxicos:
  - **yytext()**: Función que nos retorna el fragmento de entrada identificado.
  - **yychar**: Entero que contiene la posición, dentro el fichero de entrada, del primer carácter del lexema actual.
  - **yyline**: Entero con el número de línea donde se encuentra
  - Ej.: **[a-zA-Z][a-zA-Z_0-9]***

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


Recursividad

- Las acciones léxicas pueden incluir la sentencia

```
return(...)
```

para que la función **yylex(...)** retorne el valor o testigo que deseamos a la función que la ha invocado

(en un compilador, el analizador sintáctico).

Recursividad – Ejemplo

```
import java.io.*;
import java.lang.Integer;
%%
%{
public static void main (String argv[]) throws java.io.IOException {
  Integer resultado ;
  Yylex yy = new Yylex(System.in);
  do {
    resultado=yy.yylex();
    System.out.println ("yylex() retorna el dígito: " + resultado);
  } while (resultado != null) ;
}
%}
%intwrap
%%
[\t\r\n ]+      {       }
[0-3]           { }
[4-6]           { return yylex(); }
[7-9]           { return Integer.valueOf( yytext()); }
[^0-9]+         { System.out.println(yytext() +
                        " -> No es un dígito!! " );}
```


Estados

- Es posible encontrar varios componentes con diferentes significados en función del lugar donde aparezcan dentro del fichero de entrada. Ej., en C: **`int var1; /* var1 */`**

  `[<ESTADO1 [,ESTADOn ...]]>] regla_con_estado { acción }`

- El comportamiento del analizador léxico será el siguiente:
  - Si el analizador se encuentra en el **`ESTADO`**, la regla regla_con_estado estará activada. Esto es, se comparará su correspondencia con los lexemas de entrada.
  - Si el analizador no se encuentra en ninguno de los estados **`<ESTADO`**$_i$**`>`**, la regla **`regla_con_estado`** no estará activada. Es decir, no se comparará su correspondencia con los lexemas de entrada.
  - Si una regla no contiene ningún estado en su lista previa de estados, la regla estará siempre activa.

Estados - Ejemplo

```
<YYINITIAL>"/*"     { yybegin (CCOMMENT); }
<CCOMMENT>[^*]*"*/" { System.out.println ("Comentario: <" +
                    yytext().substring(0,yytext().length()-
                    2)+">");
                    yybegin (YYINITIAL); }
<YYINITIAL>[^/]*    { }
```

- Este analizador detecta comentarios del estilo **`/* … */`**

Sección Directivas

1. Directivas. Se inician "%" seguidas de un identificador de directiva. Hay diferentes tipos siguientes:
   - Directivas que afectan a las reglas léxicas.
   - Directivas que afectan al formato de la entrada o salida
   - Directivas que particularizan las propiedades de la clase del explorador
2. Código añadido a la clase. Lo clasificaremos en los tipos siguientes:
   - Código que se añade al interior de la clase.
   - Código de inicialización de la clase.
   - Código para el tratamiento del final del fichero.
   - Código para el tratamiento de excepciones.
3. Declaración de macros.
   - Las macros utilizadas en la sección de reglas léxicas tienen que declararse en esta sección.

Declaración de macros

- La declaración de una macro es una secuencia de la forma:
  - nombre = expresión_regular
- Ejemplo:

```
digito = [0-9]
alfa = [a-zA-Z]
entero = <digito>+
```

  - Notas:
    - El (**=**) puede, ir precedido y seguido por espacios y tabuladores.
    - Cada definición de macro debe caber en una sola línea.
    - El nombre de la macro debe ser un identificador válido (secuencia de letras, dígitos y subrallados, iniciada con una letra o subrallado).
    - La regla que define la macro debe ser una expresión regular válida.
    - Las definiciones de macros pueden contener otras macros, delimitando los nombres con los "**<**" y "**>**". No se permite que una macro se incluye a sí misma en su definición.

Directivas sobre reglas léxicas

- Declaración de estados:

```
%state ESTADO1 [, ESTADO2[, ESTADO3 ...]]]
```

- Sensibilidad a mayúsculas

```
%ignorecase
```

```
//Ejemplo 1
%%
%%
[a-zA-Z]+ {...}
```

```
//Ejemplo 2
%%
%ignorecase
%%
[a-z]+ {...}
```

- Activacion de **`int yychar`** e **`int yyline`**

```
%char
%line
```


Directivas de tipo retornado

- La función **`yylex()`** retorna, por defecto, un valor de tipo **`Yytoken`**.

```
class Yylex {...
    public Yytoken yylex () {
... }
```

- Para modificar el tipo retornado se dispone de:
  - **`%integer`**: la función **`yylex()`** generada retorna un valor de tipo int:

```
class Yylex {...
    public int yylex () {
... }
```

  - **`%intwrap - java.lang.Integer`**

```
class Yylex {...
    public java.lang.Integer yylex () {
... }
```

- **`%type`** - puede definirse cualquier tipo

Valor final del fichero

- El explorador generado por JLex necesita un símbolo para representar el valor de final de fichero (**EOF**).
  - Por defecto, este valor es: **`null`**.
- **`%yyeof`** - declara la constante entera **`Yylex.YYEOF`**
  - **`public final int YYEOF = -1;`**
- **`%eofval{ ...%eofval}`**
  - permite al usuario escribir código Java que será copiado en la función **`yylex ()`** y que se ejecutará cuando llegueal final de fichero.

Ejemplo %eofval

```
// Código de usuario ...
%%
%{
// Código interno ...
public static void main (String argv[])
throws java.io.IOException {
  Yylex yy = new Yylex(FicheroEntrada);
    while (yy.yylex()!= 0) ; }
%}

%eofval{
  return 0;
%eofval}
%type double

%%
// Reglas léxicas .
```


Directivas modificadoras de la clase

- Por defecto, JLex genera una clase que
  - Se llama **Yylex**.
  - No tiene ningún modificador de acceso.
  - La función principal tiene el nombre **yylex()**
- **%class** nombreClase
  - cambia el nombre de la clase
- **%public**
  - La hace publica
- **%function** nombreFunción
  - Cambia el nombre de la funcion `yylex()`
- **%implements** nombreInterface
  - **class Yylex implements *nombreInterface* { ...**

Ejemplo – Directivas de Clase

Directivas	ninguna	%class Scanner %function Tokenizer %public
Código Generado	class Yylex { ... public Yytoken yylex ()	public class Scanner { ... public Yytoken Tokenizer ()

Sección de código de usuario

- El código de usuario precede a la primera directiva %%. Este código se copia literalmente al principio del código generado por JLex.
- Se suele incluir:
  - Las declaraciones de importación de clases que hagan falta.
  - Una declaración package.
  - El código de les clases auxiliares que se precisen.

- Código añadido a la clase

```
%{
   ...<Código>...
%}
```

- Código de inicialización de la clase (añade al constructor):

```
%init{
  ...<Código>...
%init}
```

- Código de tratamiento de excepciones

```
%initthrow{
  excepción1 [, excepción2 [,...]]
%initthrow}
```


Formatos de Ficheros y Texto

- **`%full`** Con esta directiva se pueden utilizar los 255 caracteres de ASCII
  - Por defecto, el explorador generado lee y escribe en ficheros de texto ASCII. Esta tabla de caracteres contiene los códigos, con tamaño de carácter de siete bits, comprendidos entre 0 (NUL) y 127 (sin representación).
- **`%unicode`** – caracteres unicode
  - Debe usarse el constructor **`java.io.Reader`**
- **`%notunix`**
  - retorno de carro (**`\r`**) y el de nueva línea (**`\n`**) se reconocen como un solo carácter de nueva línea.
    - Unix (**`\n`**) Windows (**`\r\n`**)