

Hoja resumen del tema 2: Expresiones Regulares

Nota: esta hoja es tan solo una hoja-resumen con parte de los contenidos de la asignatura. No viene explicada la teoría, tan solo es un esquema para que puedas apoyarte a la hora de hacer los ejercicios.

Las expresiones regulares es lo que se denomina comúnmente como “patrones”.

A la hora de trabajar con Strings, hay una serie de ejercicios que implique extraer un patrón o identificar patrones se resuelven de una forma mucho más cómoda, rápida y sencilla mediante expresiones regulares.

Las expresiones regulares se componen de dos partes: el patrón y las coincidencias.

El patrón o *Pattern* es la ristra concreta de caracteres que estamos buscando.

Las coincidencias o *Matches* son ristas que contienen o cumplen el patrón.

Ejemplo: “Queremos buscar todas las palabras de una String s que contengan la letra ‘a’”

El *Pattern* será la letra ‘a’

Los *Matches* serán las distintas palabras (si las hay) que contengan la letra ‘a’

Esto en java sería tal que:

```
//Declaramos el Pattern
Pattern p = Pattern.compile("a");
//Asociamos a la String
Matcher m = p.matcher(s);
```

Si te fijas, tanto el *Pattern* como el *Matcher* son **variables** como las que hemos estado trabajando y se declaran igual:

TipoVariable NombreVariable = Valor;

La diferencia es que estas variables son mucho más complejas que un int o una String, por lo que hace falta utilizar unos métodos para declararlas.

Para trabajar con expresiones regulares debes saber que no están en la librería por defecto de java, esto quiere decir, que hace falta importarlo:

```
package ejercicio2;

//Importamos el Matcher y el Patterns para que funcionen
import java.util.regex.Matcher;
import java.util.regex.Pattern;
```

Una vez tenemos importada las dos librerías, ya podemos trabajar.

Para declara un patrón, primero debemos saber que patrón buscamos exactamente: no es lo mismo buscar si aparece una serie de caracteres o una palabra entera.

Así pues, primero debemos definir bien el patrón y para ello, java dispone de una serie de caracteres especiales que nos permite crearlos. Para utilizar dichos caracteres hace falta colocar una doble \\ para “*escaparlos*”.

[La lista de expresiones está en la otra hoja]

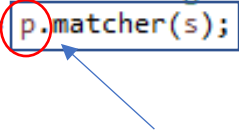
Una vez tenemos el patrón definido, éste se tiene que compilar; es decir, que java lo lea y detecte los caracteres especiales además de asegurarse de que has introducido un patrón válido y es por ello que utilizamos el *.compile()*

```
//Declaramos el Pattern
Pattern p = Pattern.compile(patron);
```

Y con esto ya tendríamos declarado el patrón. En nuestro caso lo hemos llamado *p*.

Ahora debemos asociar ese patrón con la String y para ello tenemos la segunda parte, el *Matcher*. Veamos cómo se declara:

```
//Asociamos a la String
Matcher m = p.matcher(s);
```



La primera parte es la típica de cualquier variable “TipoVariable NombreVariable” que podemos leer tal que: “hemos declarado una variable de tipo *Matcher* llamada *m*.”

En la segunda parte ocurre lo mismo que con el *Pattern*, hace falta un método para declararla. En este caso, como tenemos que *asociar* el patrón con la String, necesitamos ambas variables. Es por ello que el método, que en este caso es *.matcher()*, lo invoca la variable *Pattern* (*p*) y recibe por parámetro la String (*s*).

Así pues, nos queda:

```
//Declaramos el Pattern
Pattern p = Pattern.compile("a");
//Asociamos a la String
Matcher m = p.matcher(s);
```

Aunque son dos líneas de código, es muy importante que se entienda que es lo que ha pasado:

- 1) Se ha creado un *Pattern* (*p*)
- 2) Se ha asociado con una String y dicha asociación se ha guardado en una variable *Matcher* (*m*)

Por lo que, a la hora de trabajar con las expresiones regulares, lo que vamos a utilizar es la variable *Matcher* (*m*).

Existe un método llamado *.find()* de la clase *Matcher* que devuelve verdadero (*true*) si se encuentra el patrón en la String.

Esté método combinado con un *while()* te permite obtener **todas las veces** que se detecta el *Pattern* en la String.

Por otro lado, existe otro método llamado *.group()* que, por un lado, permite coger al *Pattern* y guardarlo en una String, y por otro nos permite subdividir el *Pattern* en grupos.

Analicemos la siguiente estructura para poder afianzar estos dos nuevos conceptos:

```
String palabrasConA = ""; //String donde guardar el resultado.
//Declaramos el Pattern
Pattern p = Pattern.compile("a");
//Asociamos a la String
Matcher m = p.matcher(s);

while (m.find()){ //mientras encontremos palabras con 'a'.
    palabrasConA += m.group() + " "; //guardo todas las palabras con 'a' separadas por espacios
}
```

Fíjate que tanto *.group()* como *.find()* son invocados desde la variable *m* (la variable *Matcher*) puesto como se ha comentado, estos métodos pertenecen a la clase *Matcher* y solo pueden ser invocados por objetos / variables de dicha clase.

Este código, a simple vista, lo que hace es guardar todas las palabras que contengan al menos una 'a'. Sin embargo, veremos que no ocurre así, ¿sabrías por qué?

Veamos que ocurre si ejecutamos dicho código y le pasamos la String "¡Hola gente! ¿Qué tal vamos?"

(consola)

```
String : ¡Hola gente! ¿Qué tal vamos?  
  
Las palabras con 'a' son:  
  
>> a a a
```

Si analizamos la frase, vemos que hay 3 palabras con 'a': "¡Hola", "tal" "vamos?".

Recuerda: las palabras son todo el contenido que hay entre dos espacios o marca de fin de cadena. Es por eso que se coge el interrogante y la exclamación.

Pero hemos visto que la respuesta ha sido 3 'a' separadas por espacios. ¿Sabrías decir por qué?

Si sabemos que en esta instrucción...

```
palabrasConA += m.group() + " "; //guardo todas las palabras con 'a' separadas por espacios
```

...es cuando se guardan las palabras, significa que el *m.group()* es lo que está cogiendo solo 'a'.

group() por defecto carga todo el patrón, pero si utilizamos paréntesis en el *Pattern*, podemos hacer subgrupos y mediante *group(int)*, ir tratando cada grupo por separado.

Lo que ocurre en este caso es que hemos dicho que **SOLO** coja 'a', y hemos obviado el resto de la String.

He introducido este fallo porque, a parte de que sirve para entender mejor todo el tema de como funcionan los métodos y demás, sepas por donde tirar cuando falla un programa. A parte, este error es muy típico a la hora de trabajar expresiones regulares y ocurre porque presuponemos que el patrón funciona como un **detector**.

La solución a nuestro problema sería algo tal que:

```
String palabrasConA = ""; //String donde guardar el resultado.  
//Declaramos el Pattern  
Pattern p = Pattern.compile("\\w*a\\w*");  
//Asociamos a la String  
Matcher m = p.matcher(s);  
  
while (m.find()){ //mientras encontremos palabras con 'a'.  
    palabrasConA += m.group() + " "; //guardo todas las palabras con 'a' separadas por espacios  
}  
  
System.out.print("\n >> " + palabrasConA); //devuelve todas las palabras que contengan una 'a'.
```

(consola)

```
String : ¡Hola gente! ¿Qué tal vamos?  
  
Las palabras con 'a' son:  
  
>> Hola tal vamos
```