

Ejemplos de Expresiones Regulares en Java

Una expresión regular define un patrón de búsqueda para cadenas de caracteres.

La podemos utilizar para comprobar si una cadena contiene o coincide con el patrón. El contenido de la cadena de caracteres puede coincidir con el patrón 0, 1 o más veces.

Algunos ejemplos de uso de expresiones regulares pueden ser:

- para comprobar que la fecha leída cumple el patrón dd/mm/aaaa
- para comprobar que un NIF está formado por 8 cifras, un guión y una letra
- para comprobar que una dirección de correo electrónico es una dirección válida.
- para comprobar que una contraseña cumple unas determinadas condiciones.
- Para comprobar que una URL es válida.
- Para comprobar cuántas veces se repite dentro de la cadena una secuencia de caracteres determinada.
- Etc. Etc.

El patrón se busca en el String de izquierda a derecha. Cuando se determina que un carácter cumple con el patrón este carácter ya no vuelve a intervenir en la comprobación.

Ejemplo:

La expresión regular "010" la encontraremos dentro del String "010101010" solo dos veces: "010101010"

Símbolos comunes en expresiones regulares

Expresión	Descripción
.	Un punto indica cualquier carácter
^expresión	El símbolo ^ indica el principio del String. En este caso el String debe contener la expresión al principio.
expresión\$	El símbolo \$ indica el final del String. En este caso el String debe contener la expresión al final.
[abc]	Los corchetes representan una definición de conjunto. En este ejemplo el String debe contener las letras a ó b ó c.
[abc]{1,2}	El String debe contener las letras a ó b ó c seguidas de 1 ó 2
[^abc]	El símbolo ^ dentro de los corchetes indica negación. En este caso el String debe contener cualquier carácter excepto a ó b ó c.
[a-z1-9]	Rango. Indica las letras minúsculas desde la a hasta la z (ambas incluidas) y los dígitos desde el 1 hasta el 9 (ambos incluidos)
A B	El carácter es un OR. A ó B
AB	Concatenación. A seguida de B

Meta caracteres

Expresión	Descripción
\d	Dígito. Equivale a [0-9]
\D	No dígito. Equivale a [^0-9]
\s	Espacio en blanco. Equivale a [\t\n\r\f]
\S	No espacio en blanco. Equivale a [^\s]
\w	Una letra mayúscula o minúscula, un dígito o el carácter _ Equivale a [a-zA-Z0-9_]

Expresión	Descripción
\W	Equivale a [^\w]
\b	Límite de una palabra.

En Java debemos usar una doble barra invertida \

Por ejemplo para utilizar \w tendremos que escribir \\w. Si queremos indicar que la barra invertida es un carácter de la expresión regular tendremos que escribir \\\.

Quantificadores

Expresión	Descripción
{X}	Indica que lo que va justo antes de las llaves se repite X veces
{X,Y}	Indica que lo que va justo antes de las llaves se repite mínimo X veces y máximo Y veces. También podemos poner {X,} indicando que se repite un mínimo de X veces sin límite máximo.
*	Indica 0 ó más veces. Equivale a {0,}
+	Indica 1 ó más veces. Equivale a {1,}
?	Indica 0 ó 1 veces. Equivale a {0,1}

Para usar expresiones regulares en Java se usa el package [java.util.regex](#)

Contiene las clases [Pattern](#) y [Matcher](#) y la excepción [PatternSyntaxException](#).

Clase **Pattern**: Un objeto de esta clase representa la expresión regular. Contiene el método [compile\(String regex\)](#) que recibe como parámetro la expresión regular y devuelve un objeto de la clase Pattern.

La clase **Matcher**: Esta clase compara el String y la expresión regular. Contienen el método [matches\(CharSequence input\)](#) que recibe como parámetro el String a validar y devuelve true si coincide con el patrón. El método [find\(\)](#) indica si el String contienen el patrón.

Ejemplos de Expresiones Regulares en Java:

1. Comprobar si el String *cadena* contiene exactamente el patrón (matches) "abc"

```
Pattern pat = Pattern.compile("abc");
Matcher mat = pat.matcher(cadena);
if (mat.matches()) {
    System.out.println("SI");
} else {
    System.out.println("NO");
}
```

2. Comprobar si el String *cadena* contiene "abc"

```
Pattern pat = Pattern.compile(".*abc.*");
Matcher mat = pat.matcher(cadena);
if (mat.matches()) {
    System.out.println("SI");
} else {
    System.out.println("NO");
}
```

También lo podemos escribir usando el método find:

```
Pattern pat = Pattern.compile("abc");
Matcher mat = pat.matcher(cadena);
if (mat.find()) {
    System.out.println("Válido");
} else {
    System.out.println("No Válido");
}
```

3. Comprobar si el String *cadena* empieza por "abc"

```
Pattern pat = Pattern.compile("^abc.*");
Matcher mat = pat.matcher(cadena);
if (mat.matches()) {
    System.out.println("Válido");
} else {
    System.out.println("No Válido");
}
```

4. Comprobar si el String *cadena* empieza por "abc" ó "Abc"

```
Pattern pat = Pattern.compile("[aA]bc.*");
Matcher mat = pat.matcher(cadena);
if (mat.matches()) {
    System.out.println("SI");
} else {
    System.out.println("NO");
}
```

5. Comprobar si el String *cadena* está formado por un mínimo de 5 letras mayúsculas o minúsculas y un máximo de 10.

```
Pattern pat = Pattern.compile("[a-zA-Z]{5,10}");
Matcher mat = pat.matcher(cadena);
if (mat.matches()) {
    System.out.println("SI");
} else {
    System.out.println("NO");
}
```

6. Comprobar si el String *cadena* no empieza por un dígito

```
Pattern pat = Pattern.compile("^[^\\d].*");
Matcher mat = pat.matcher(cadena);
if (mat.matches()) {
    System.out.println("SI");
} else {
    System.out.println("NO");
}
```

7. Comprobar si el String *cadena* no acaba con un dígito

```
Pattern pat = Pattern.compile(".*[^\\d]$");
```

```

Matcher mat = pat.matcher(cadena);
if (mat.matches()) {
    System.out.println("SI");
} else {
    System.out.println("NO");
}

```

8. Comprobar si el String *cadena* solo contienen los caracteres a ó b

```

Pattern pat = Pattern.compile("(a|b)+");
Matcher mat = pat.matcher(cadena);
if (mat.matches()) {
    System.out.println("SI");
} else {
    System.out.println("NO");
}

```

9. Comprobar si el String *cadena* contiene un 1 y ese 1 no está seguido por un 2

```

Pattern pat = Pattern.compile(".*1(?!2).*");
Matcher mat = pat.matcher(cadena);
if (mat.matches()) {
    System.out.println("SI");
} else {
    System.out.println("NO");
}

```

Ejemplo: expresión regular para comprobar si un email es válido

```

package ejemplo1;
import java.util.Scanner;
import java.util.regex.Matcher;
import java.util.regex.Pattern;
public class Ejemplo1 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String email;
        System.out.print("Introduce email: ");
        email = sc.nextLine();
        Pattern pat = Pattern.compile("^[\w-]+(\.[\w-]+)*@[A-Za-z0-9]+(\.[A-Za-z0-9]+)*(\.[A-Za-z]{2,})$");
        Matcher mat = pat.matcher(email);
        if(mat.find()){
            System.out.println("Correo Válido");
        }else{
            System.out.println("Correo No Válido");
        }
    }
}

```

Hemos usado la siguiente expresión regular para comprobar si un email es válido:

```

"^[\w-]+(\.[\w-]+)*@[A-Za-z0-9]+(\.[A-Za-z0-9]+)*(\.[A-Za-z]{2,})$"

```

La explicación de cada parte de la expresión regular es la siguiente:

<code>[\\w-]+</code>	<p>Inicio del email</p> <p>El signo + indica que debe aparecer uno o más de los caracteres entre corchetes:</p> <p>\\w indica caracteres de la A a la Z tanto mayúsculas como minúsculas, dígitos del 0 al 9 y el carácter _</p> <p>Carácter –</p> <p>En lugar de usar \\w podemos escribir el rango de caracteres con lo que esta expresión quedaría así:</p> <p><code>[A-Za-z0-9- _]+</code></p>
<code>(\\.[\\w-]+)*</code>	<p>A continuación:</p> <p>El * indica que este grupo puede aparecer cero o más veces. El email puede contener de forma opcional un punto seguido de uno o más de los caracteres entre corchetes.</p>
<code>@</code>	<p>A continuación debe contener el carácter @</p>
<code>[A-Za-z0-9]+</code>	<p>Después de la @ el email debe contener uno o más de los caracteres que aparecen entre los corchetes</p>
<code>(\\.[A-Za-z0-9]+)*</code>	<p>Seguido (opcional, 0 ó más veces) de un punto y 1 ó más de los caracteres entre corchetes</p>
<code>(\\.[A-Za-z]{2,})</code>	<p>Seguido de un punto y al menos 2 de los caracteres que aparecen entre corchetes (final del email)</p>

Usar expresiones regulares con la clase String. Métodos matches y splits.

`String.matches(regex)`

Podemos comprobar si una cadena de caracteres cumple con un patrón usando el método matches de la clase String. Este método recibe como parámetro la expresión regular.

```
if (cadena.matches(".*1(?:2).*")) {  
    System.out.println("SI");  
} else {  
    System.out.println("NO");  
}
```

`String.split(regex)`

El método split de la clase String es la alternativa a usar StringTokenizer para separar cadenas. Este método divide el String en cadenas según la expresión regular que recibe. La expresión regular no forma parte del array resultante.

Ejemplo 1:

```
String str = "blanco-rojo:amarillo.verde_azul";
String [] cadenas = str.split("[-:._]");
for(int i = 0; i<cadenas.length; i++){
    System.out.println(cadenas[i]);
}
```

Muestra por pantalla:

```
blanco
rojo
amarillo
verde
azul
```

Ejemplo 2:

```
String str = "esto es un ejemplo de como funciona split";
String [] cadenas = str.split("(e[s|m])|(pl)");
for(int i = 0; i<cadenas.length; i++){
    System.out.println(cadenas[i]);
}
```

Salida:

```
to
un ej
o de como funciona s
it
```