Expresiones regulares

Las expresiones regulares se usan para validar patrones de cadenas de forma rápida. Para ello tenemos que saber cómo construir el patrón a validar y luego cómo llevar esa validación a Java.

Por ejemplo, imagina que quieres validar que el usuario introduce bien en tus programas su DNI, o correo electrónico.

Una forma rápida de hacerlo es, identificar el patrón que debe seguir el DNI o mail, y a continuación construir la expresión regular.

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

Por ejemplo, el patrón "010" la encontraremos dentro del String "010101010" solo dos veces:

"010101010"

Tenemos que aprender entonces a construir expresiones regulares. Para ello usaremos los elementos de las siguientes tablas.

Símbolos

Expresión	Descripción
.	Indica cualquier carácter
^expresión	Indica el principio del String. En este caso el String debe contener la expresión al principio.
expresión$	Indica el final del String. En este caso el String debe contener la expresión al final.
[abc]	Indican un conjunto. En este ejemplo el String debe contener las letras a, b ó c.
[abc][12]	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, 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\x0b\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_]
\W	Equivale a [^\w]
\b	Límite de una palabra.

Cuantificadores

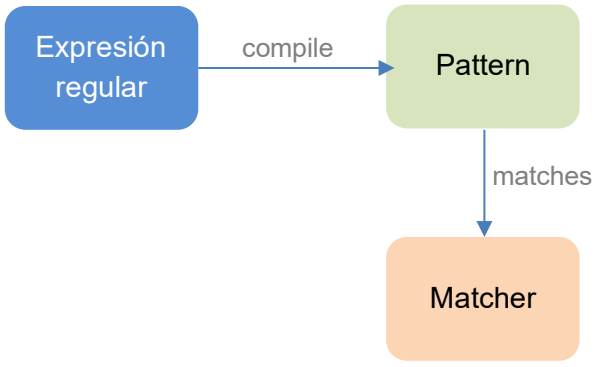
Expresión	Descripción
{X}	Indica que lo que va justo antes de las llaves se repite X veces
{X,Y}	Indica que lo que va justo antes de las llaves se repite mínimo X veces y máximo Y veces. También podemos poner {X,} indicando que se repite un mínimo de X veces sin límite máximo.
*	Indica 0 ó más veces.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

Java

Para usar expresiones regulares en Java usaremos las siguientes clases:

- **Pattern**: objeto que representa la expresión regular. El método *compile(String regex)* recibe como parámetro la expresión regular y devuelve un objeto de la clase Pattern.
- **Matcher** : compara el String y la expresión regular. El método *matches(CharSequence input)* 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.

Las excepciones que se puedan producir serán del tipo **PatternSyntaxException.**



Veamos algunos ejemplos con el String **cadena**:

```
String cadena = "contenido a comprobar";
Pattern pat = Pattern.compile("patron");
Matcher mat = pat.matcher(cadena);
if (mat.matches()) {
    System.out.println("SI COINCIDE");
} else {
    System.out.println("NO COINCIDE");
}
```

En todos los casos, los objetos serán estos. Irá cambiando la cadena y como es lógico la expresión regular o patron.

1. Comprobar si contiene exactamente el patrón "abc":
   ```
   Pattern.compile("abc");
   ```
2. Comprobar si contiene "abc":
   ```
   Pattern.compile(".*abc.*");
   ```
3. Comprobar si empieza por "abc":
   ```
   Pattern.compile("^abc.*");
   ```
4. Comprobar si empieza por "abc" ó "Abc":
   ```
   Pattern.compile("^[Aa]bc.*");
   ```
5. Comprobar si está formada por un mínimo de 5 letras mayúsculas o minúsculas y un máximo de 10.
   ```
   Pattern.compile("[a-zA-Z]{5,10}");
   ```
6. Comprobar si no empieza por un dígito:
   ```
   Pattern.compile("[^\\d].*");
   ```
7. Comprobar si no acaba con un dígito:
   ```
   Pattern.compile("[^\\d].*");
   ```
8. Comprobar si solo contiene los caracteres a o b:
   ```
   Pattern.compile("(a|b)+");
   ```
9. Comprobar si tiene un 1, y no está seguido por un 2:
   ```
   Pattern.compile(".*1(?!2).*");
   ```
10. Comprobar si es un mail válido:
    ```
    Pattern.compile("^[\\w.-]+@[\\w.-]+\\.[a-zA-Z]{2,}$");
    ```