Expresiones regulares. (I)

Las expresiones regulares permiten comprobar si una cadena sigue o no un patrón preestablecido. Son un mecanismo para describir esos patrones, y se construyen de una forma relativamente sencilla. Existen muchas librerías diferentes para trabajar con expresiones regulares, y casi todas siguen, más o menos, una sintaxis similar, con ligeras variaciones. Dicha sintaxis nos permite indicar el patrón de forma cómoda, como si de una cadena de texto se tratase, en la que determinados símbolos tienen un significado especial. Por ejemplo "[01]+" es una expresión regular que permite comprobar si una cadena conforma un número binario.

Las reglas generales para construir una expresión regular son:

- Podemos indicar que una cadena contiene un conjunto de símbolos fijo, simplemente poniendo dichos símbolos en el patrón, excepto para algunos símbolos especiales que necesitarán un carácter de escape como veremos más adelante. Por ejemplo, el patrón "`aaa`" admitirá cadenas que contengan tres aes.
- "`[xyz]`". Entre corchetes podemos indicar opcionalidad. Solo uno de los símbolos que hay entre los corchetes podrá aparecer en el lugar donde están los corchetes. Por ejemplo, la expresión regular "`aaa[xy]`" admitirá como válidas las cadenas "aaax" y la cadena "aaay". Los corchetes representan una posición de la cadena que puede tomar uno de varios valores posibles.
- "`[a-z]`" "`[A-Z]`" "`[a-zA-Z]`". Usando el guión y los corchetes podemos indicar que el patrón admite cualquier carácter entre la letra inicial y la final. Es importante saber que se diferencia entre letras mayúsculas y minúsculas, no son iguales de cara a las expresiones regulares.
- "`[0-9]`". Y nuevamente, usando un guión, podemos indicar que se permite la presencia de un dígito numérico entre 0 y 9, cualquiera de ellos, pero solo uno. Con las reglas anteriores podemos indicar el conjunto de símbolos que admite el patrón y el orden que deben tener. Si una cadena no contiene los símbolos especificados en el patrón, en el mismo orden, entonces la cadena no encajará con el patrón. Veamos ahora como indicar repeticiones:
- "`a?`". Usaremos el interrogante para indicar que un símbolo puede aparecer una vez o ninguna. De esta forma la letra "a" podrá aparecer una vez o simplemente no aparecer.
- "`a*`". Usaremos el asterisco para indicar que un símbolo puede aparecer una vez o muchas veces, pero también ninguna. Cadenas válidas para esta expresión regular serían "aa", "aaa" o "aaaaaaaa".
- "`a+`". Usaremos el símbolo de suma para indicar que otro símbolo debe aparecer al menos una vez, pudiendo repetirse cuantas veces quiera.
- "`a{1,4}`" . Usando las llaves, podemos indicar el número mínimo y máximo de veces que un símbolo podrá repetirse. El primer número del ejemplo es el número 1, y quiere decir que la letra "a" debe aparecer al menos una vez. El segundo número, 4, indica que como máximo puede repetirse cuatro veces.
- "`a{2,}`". También es posible indicar solo el número mínimo de veces que un carácter debe aparecer (sin determinar el máximo), haciendo uso de las llaves, indicando el primer número y poniendo la coma (no la olvides).
- "`a{5}`". A diferencia de la forma anterior, si solo escribimos un número entre llaves, sin poner la coma detrás, significará que el símbolo debe aparecer un número exacto de veces. En este caso, la "a" debe aparecer exactamente 5 veces.
- "`[a-z]{1,4}[0-9]+`". Los indicadores de repetición se pueden usar también con corchetes, dado que los corchetes representan, básicamente, un símbolo. En el ejemplo anterior se permitiría de una a cuatro letras minúsculas, seguidas de al menos un dígito numérico.

Con lo visto hasta ahora ya es posible construir una expresión regular capaz de verificar si una cadena contiene un DNI o un NIE. La expresión regular que permite verificar si una cadena contiene un DNI o un NIE es la siguiente: "`[XYxy]?[0-9]{1,9}[A-Za-z]`"; aunque no es la única solución.

Expresiones regulares. (II)

Para usar expresiones regulares Java ofrece las clases `Pattern` y `Matcher`, contenidas en el paquete `java.util.regex.*`. La clase `Pattern` se utiliza para procesar la expresión regular y "compilarla", lo cual significa verificar que es correcta y dejarla lista para su utilización. La clase `Matcher` sirve para comprobar si una cadena cualquiera sigue o no un patrón. Veámoslo con un ejemplo:

```
Pattern p=Pattern.compile("[01]+");
Matcher m=p.matcher("00001010");
if (m.matches()){
      System.out.println("Si, contiene el patrón");
}else{
      System.out.println("No, no contiene el patrón");
}
```

En el ejemplo, el método estático compile de la clase `Pattern` permite crear un patrón, dicho método compila la expresión regular pasada por parámetro y genera una instancia de `Pattern` (p en el ejemplo). El patrón p podrá ser usado múltiples veces para verificar si una cadena coincide o no con el patrón, dicha comprobación se hace invocando el método `matcher`, el cual combina el patrón con la cadena de entrada y genera una instancia de la clase `Matcher` (m en el ejemplo). La clase `Matcher` contiene el resultado de la comprobación y ofrece varios métodos para analizar la forma en la que la cadena ha encajado con un patrón:

- `m.matches()`. Devolverá true si toda la cadena (de principio a fin) encaja con el patrón o false en caso contrario.
- `m.lookingAt()`. Devolverá true si el patrón se ha encontrado al principio de la cadena. A diferencia del método `matches()`, la cadena podrá contener al final caracteres adicionales a los indicados por el patrón, sin que ello suponga un problema.
- `m.find()`. Devolverá true si el patrón existe en algún lugar la cadena (no necesariamente toda la cadena debe coincidir con el patrón) y false en caso contrario, pudiendo tener más de una coincidencia. Para obtener la posición exacta donde se ha producido la coincidencia con el patrón podemos usar los métodos `m.start()` y `m.end()`, para saber la posición inicial y final donde se ha encontrado. Una segunda invocación del método `find()` irá a la segunda coincidencia (si existe), y así sucesivamente. Podemos reiniciar el método `find()`, para que vuelva a comenzar por la primera coincidencia, invocando el método `m.reset()`.

Veamos algunas construcciones adicionales que pueden ayudarnos a especificar expresiones regulares más complejas:

- "`[^abc]`". El símbolo "^", cuando se pone justo detrás del corchete de apertura, significa "negación". La expresión regular admitirá cualquier símbolo diferente a los puestos entre corchetes. En este caso, cualquier símbolo diferente de "a", "b" o "c".
- "`^[01]+$`". Cuando el símbolo "^" aparece al comienzo de la expresión regular, permite indicar comienzo de línea o de entrada. El símbolo "$" permite indicar fin de línea o fin de entrada. Usándolos podemos verificar que una línea completa (de principio a fin) encaje con la expresión regular, es muy útil cuando se trabaja en modo multilínea (cadenas que contienen en su interior más de un salto de línea a veces no se procesan bien con las expresiones regulares, dado que estas se limitan a verificar generalmente solo la primera línea. El modo multilínea permite buscar líneas completas que encajan con un patrón dentro de una cadena que contiene varias líneas) y con el método `find()`.
- "`.`". El punto simboliza cualquier carácter.
- "`\\d`". Un dígito numérico. Equivale a "`[0-9]`".
- "`\\D`". Cualquier cosa excepto un dígito numérico. Equivale a "`[^0-9]`".
- "`\\s`". Un espacio en blanco (incluye tabulaciones, saltos de línea y otras formas de espacio).
- "`\\S`". Cualquier cosa excepto un espacio en blanco.
- "`\\w`". Cualquier carácter que podrías encontrar en una palabra. Equivale a "`[a-zA-Z_0-9]`".

- "\\W". Cualquier carácter que no sea alfanumérico. Equivale a "[^a-zA-Z_0-9]".
- "\\b". Límite de palabra.
- "\\B". No límite de palabra.

Los paréntesis permiten indicar repeticiones para un conjunto de símbolos, por ejemplo: "(#[01]){2,3}". En el ejemplo anterior, la expresión "#[01]" admitiría cadenas como "#0" o "#1", pero al ponerlo entre paréntesis e indicar los contadores de repetición, lo que estamos diciendo es que la misma secuencia se tiene que repetir entre dos y tres veces, con lo que las cadenas que admitiría serían del estilo a: "#0#1" o "#0#1#0".

Cuando en una expresión regular necesitamos especificar que lo que tiene que haber en la cadena es un paréntesis, una llave, o un corchete, tenemos que usar una secuencia de escape, dado que esos símbolos tienen un significado especial en los patrones. Para ello, simplemente antepondremos "\\" al símbolo. Por ejemplo, "\\(" significará que debe haber un paréntesis en la cadena y se omitirá el significado especial del paréntesis. Lo mismo ocurre con "\\[", "\\]", "\\)", etc. Lo mismo para el significado especial del punto, éste, tiene un significado especial salvo que se ponga "\\.", que pasará a significar “un punto” en vez de “cualquier carácter”. La excepción son las comillas, que se pondrían con una sola barra: "\"".

Más información sobre expresiones regulares en:

http://download.oracle.com/javase/tutorial/essential/regex/

Páginas para hacer pruebas on line:

https://www.regexpal.com/

https://regex101.com/