Expresiones Regulares - Java

Matías Rojas

Las **expresiones regulares** son patrones que se utilizan para buscar, extraer o manipular texto. Se utilizan en una amplia gama de aplicaciones, como la validación de datos, el procesamiento de texto y la búsqueda de patrones. Dicho patrón funciona encontrando una determinada combinación de caracteres dentro de una cadena de texto. Estas expresiones son útiles debido a la flexibilidad que proporcionan a la hora de buscar o reconocer cadenas de texto.

Las expresiones regulares, si bien son utilizadas para verificar el formato de diversos datos, no pueden ser utilizados para verificarlos. Como podría ser verificar la existencia de un email o de un DNI. Es decir, que podrá verificar que un email tenga el formato “[usuario@dominio.com](mailto:usuario@dominio.com)” pero no podrá comprobar que el email realmente exista.

Una expresión regular puede estar formada por caracteres normales, como letras y números, o por metacaracteres, que son caracteres especiales que tienen un significado específico. Puede ser simple como “a” o “Celular”. Expresiones que buscarán una coincidencia **exacta** dentro de la cadena de texto. Sin embargo, las expresiones de este tipo no son más que básicas. Para realizar algunas más complejas, existen diversos constructores que podemos utilizar, siendo algunos:

Coincidencias Básicas:

. - Cualquier Carácter, excepto salto de línea

\\d - Cualquier Dígitos (0-9)

\\D - No es un Digito (0-9)

\\w - Carácter de Palabra (a-z, A-Z, 0-9, \_)

\\W - No es un Carácter de Palabra.

\\s - Espacios de cualquier tipo. (espacio, tab, salto de línea)

\\S - No es un Espacio, Tab o nueva linea.

Limites:

\b - Limite de Palabra

\B - No es un Límite de Palabra

^ - Inicio de una cadena de texto

$ - Final de una cadena de texto

Cuantificadores:

\* - 0 o Más repeticiones del elemento

+ - 1 o Más repeticiones del elemento

? - elemento opcional (0 o 1 repeticiones)

{3} - repeticiones del elemento

{3,4} - Rango de repeticiones del elemento (Minimo, Maximo)

Conjuntos de Caracteres

[] - Caracteres dentro de los brackets

[^ ] - Caracteres que NO ESTAN dentro de los brackets

Grupos

( ) - Grupo

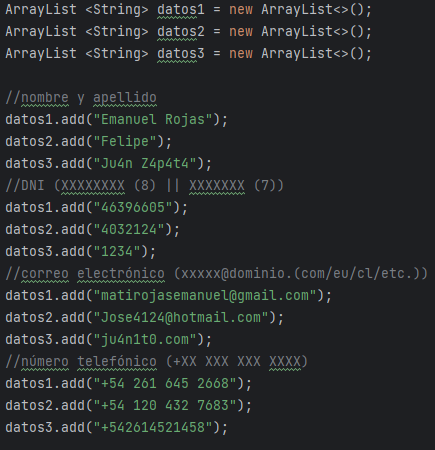
| - Uno u otro

Siendo, por ejemplo, “[0-9]{7,8}” una expresión que verifica que una cadena contenga únicamente dígitos del 0 al 9 y que repita de 7 a 8 veces dichos dígitos.  
Algunas de las expresiones regulares comúnmente utilizadas son:

* Coincidencia nombre de usuario: ^[a-z0-9\_-]{3,16}$
* Coincidencia contraseña: ^[a-z0-9\_-]{6,18}$
* Coincidencia número hexadecimal: ^#?([a-f0-9]{6}|[a-f0-9]{3})$
* Coincidencia Slug: ^[a-z0-9-]+$
* Coincidencia correo electrónico: ^([a-z0-9\_\.-]+)@([\da-z\.-]+)\.([a-z\.]{2,6})$

En java, podemos utilizar funciones para realizar las comprobaciones con expresiones regulares. Por ejemplo:

Creamos ArrayLists y las llenamos con los datos a verificar:



Creamos las funciones de comprobación con sus respectivas expresiones regulares. Teniendo las expresiones para los datos:

Nombre:

**^([A-Z][a-z]+)\\s([A-Z][a-z]+)$**

^…$: Empieza y termina (Indica orden).

Toda cadena que empiece con una letra mayúscula y contenga 1 o más letras minúsculas **^([A-Z][a-z]+)** seguido de un espacio en blanco **\\s** y termine con una letra mayúscula y contenga 1 o más letras minúsculas **([A-Z][a-z]+)$**.

DNI:

**^[0-9]{7,8}$**

Toda cadena que empiece y termine con dígitos del 0 al 9 **[0-9]** entre 7 u 8 veces **{7,8}**.

Correo:

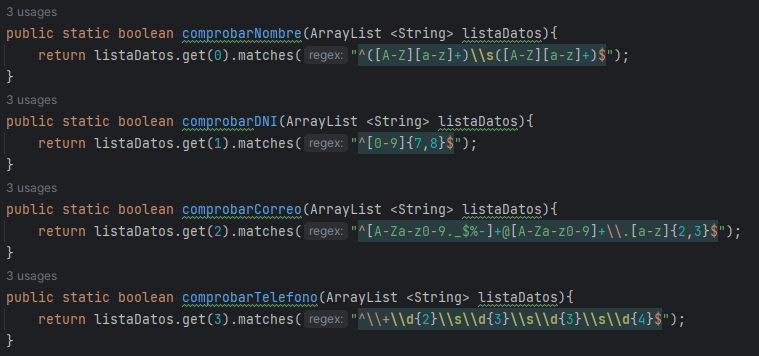
**^[A-Za-z0-9.\_$%-]+@[A-Za-z0-9]+\\.[a-z]{2,3}$**

Toda cadena que empiece con letras minúsculas, mayúsculas, números o caracteres especiales 1 o más veces **^[A-Za-z0-9.\_$%-]+** seguido de un @ seguido de letras minúsculas, mayúsculas o números 1 o más veces **[A-Za-z0-9]+** seguido de un punto **\\.** y termine con 2 o 3 letras minúsculas **[a-z]{2,3}$**.

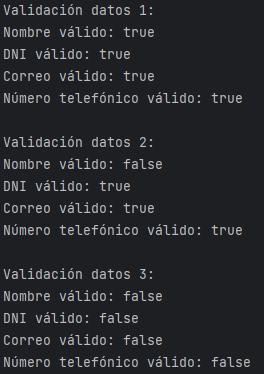
Teléfono:

**^\\+\\d{2}\\s\\d{3}\\s\\d{3}\\s\\d{4}$**

Toda cadena que empiece con un “+” **^\\+** seguido de 2 dígitos y un espacio **\\d{2}\\s** seguido de 3 dígitos y un espacio (2 veces) **\\d{3}\\s\\d{3}\\s** y que termine en 4 dígitos **\\d{4}$**.

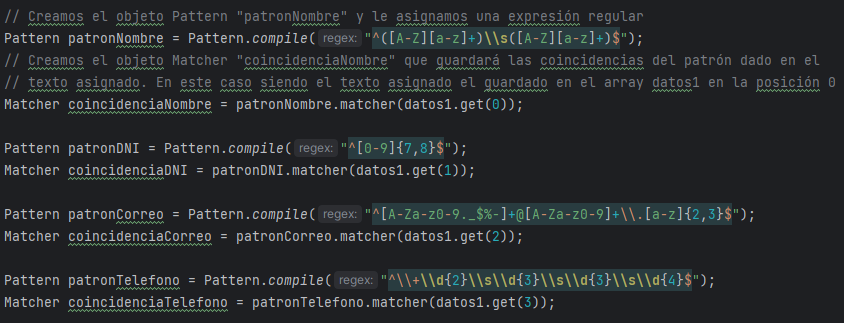


Llamamos a las funciones y ejecutamos el código, obteniendo como resultado:



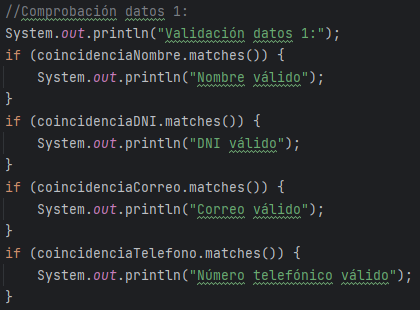
También, podemos hacer el mismo programa pero, sin utilizar funciones y en su lugar utilizar clases de las librerías java.util.regex.Pattern y java.util.regex.Matcher. Estas bibliotecas nos permiten realizar lo mismo que hicimos con las funciones, pero utilizando las clases Pattern y Matcher.

Para utilizar estas clases, primero, debemos importar las librerías. Luego, definimos los objetos:

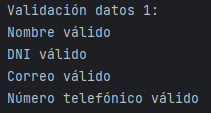


Al objeto de la clase Pattern se le asigna la expresión regular con la que deseamos verificar un valor. Y, al objeto de la clase Matcher, se le asigna el texto que deseamos comparar con el patrón. Siendo en el caso del ejemplo, el elemento en la posición 0 del array. Cabe aclarar que el objeto de la clase Pattern puede ser reutilizado por varios Matcher de ser necesario. Sin embargo, los objetos de la clase Matcher reciben textos únicos por lo que, si se desea verificar varias cadenas, se deberá crear un objeto para cada una de las cadenas.

Ahora, con los mismos datos del programa anterior, solicitamos los datos de la siguiente forma:



Obteniendo los siguientes resultados:



Un dato importante a destacar es que la función “matches()” busca en la cadena las coincidencias **específicas** con el patrón dado. En este caso, ya que definimos un patrón que verifica la totalidad de la cadena, no hay problema en que usemos matches(). Pero, si quisiéramos buscar en el nombre la cadena “Ema”, la función anterior no funcionaría ya que la cadena “Emanuel Rojas” no es exactamente igual a la cadena “Ema”. En este caso, deberíamos utilizar la función “find()” la cuál busca la inclusión del patrón dado dentro de la cadena.