UD06 - Apuntes - Ficheros. Flujos de entrada y salida. Regex

REGEX

Una expresión regular o expresión racional (también son conocidas como regex o regexp, por su contracción de las palabras inglesas **reg**ular **exp**ression) es una secuencia de caracteres que conforma un patrón de búsqueda. Se utilizan principalmente para la búsqueda de patrones de cadenas de caracteres u operaciones de sustituciones.

Podemos definir una expresión regular como una secuencia de caracteres que forman una secuencia o patrón que puede ser automatizada de alguna manera.

Por ejemplo, dentro de un texto queremos eliminar todas las palabras que sean «la» podemos usar un patrón para eliminarlas. Una expresión regular nos va a permitir buscar o reemplazar una secuencia.

Podemos usar regex para:

- Buscar patrones concretos de caracteres.
- Validar el texto para garantizar que coincide con un patrón predefinido (como una dirección de correo electrónico).
- Extraer, editar, reemplazar o eliminar subcadenas de texto.
- Agregar cadenas extraídas en una colección para generar un informe.

¿Qué necesitamos en Java para crear una expresión regular?

Para poder hacer uso de expresiones regulares en Java necesitamos importar el paquete regex, el cual fue introducido en la versión 1.4 de Java.

La clase regex nos aporta las siguientes clases:

- Matcher: Esta clase nos permite hacer match sobre la secuencia de caracteres que nos define el Pattern.
- MatchResult: El resultado de la operación al hacer match
- Pattern: Es la representación de la expresión regular.
- PatternSyntaxException: Se lanza una Unchecked Exception para indicar un error de expresión en el patrón.

Cómo crear expresión regular en Java

Para crear una expresión regular haremos uso de los cuantificadores y meta-caracteres.

Cuantificadores para una expresión regular

Tenemos caracteres especiales que nos van a indicar el número de repeticiones de la expresión, la siguiente tabla muestra los caracteres:

Cuantificador	Descripción
n+	Encuentra cualquier string con al menos un «n»
n*	Encuentra cero o más ocurrencias de n
n?	Encuentra en el string la aparición de n cero o una vez

n{x}	Encuentra la secuencia de n tantas veces como indica x.
n{x,}	Encuentra una secuencia de X tantas veces como indica n

Metacaracteres en una expresión regular

Metacaracter	Descripción
|	Símbolo para indicar OR.
.	Encuentra cualquier carácter
^	Sirve para hacer match al principio del string
$	Hace match al final de un String
\d	Encuentra dígitos
\s	Busca un espacio
\b	Hace match al principio de una palabra.
\uxxxx	Encuentra el carácter Unicode especificado por el número hexadecimal xxxx

Metacaracteres y ejemplos con expresiones regulares

Expresión regular	Descripción
.	Hace match con cualquier caracter
^regex	Encuentra cualquier expresión que coincida al principio de la línea.
regex$	Encuentra la expresión que haga match al final de la línea.
[abc]	Establece la definición de la expresión, por ejemplo la expresión escrita haría match con a, b o c.
[abc][vz]	Establece una definición en la que se hace match con a, b o c y a continuación va seguido por v o por z.
[^abc]	Cuando el símbolo ^ aparece al principio de una expresión después de [, negaría el patrón definido. Por ejemplo, el patrón anterior negaría el patrón, es decir, hace match para todo menos para la a, la b o la c.

Expresión regular	Descripción
[e-f]	Cuando hacemos uso de -, definimos rangos. Por ejemplo, en la expresión anterior buscamos hacer match de una letra entre la e y la f.
Y|X	Establece un OR, encuentra la Y o la X.
HO	Encuentra HO
$	Verifica si el final de una línea sigue.

Reglas generales para construir patrones:

Cadena	Coincidencia
a	El carácter a. Podemos indicar que una cadena coincide con un conjunto fijo de caracteres simplemente buscando dichos caracteres en la cadena de entrada.
[abc]	Busca cualquiera de los caracteres incluidos entre corchetes
[a-z]	Busca cualquiera de los caracteres entre la a y la z
[A-Za-z]	Busca cualquiera de los caracteres entre la A y la Z o entre a y la z
[0-9]	Busca cualquiera de los caracteres entre 0 y 9
A?	A, una o ninguna vez
A*	A, cero o más veces
A+	A, una o más veces
A{2}	AA, A exactamente 2 veces
A{3,}	A tres o más veces
A{2, 4}	A al menos 2 veces, pero no más de 4
[a-z]{1,4}[0-9]+	Combinación de 1 a 4 caracteres entre la a y la z seguido de un grupo de número
[^abc]	Cualquier carácter que no sea a, b ó c.
.	Cualquier carácter
\d	Un dígito: [0-9]
\D	Cualquier carácter que no sea dígito: [^0-9]
\s	Un espacio en blanco, incluidos \t, \n, \r, \f

\S	Cualquier carácter que no sea un espacio en blanco [^\s]
\w	Cualquier carácter de palabra: [a-zA-Z_0-9]
\W	Cualquier carácter que no sea de una palabra: [^\w]
Operador Lógico OR	
X|Y	Se cumple X ó Y
Límites	
^	Comienzo de la cadena
$	Final de la cadena
\b	Límite de palabra
\B	Un carácter que no es un límite de palabra

Uso de Pattern y Matcher para crear expresiones regulares

Ya hemos visto como podemos crear o qué necesitamos para crear expresiones regulares, ahora vamos a ver como podemos crear las expresiones regulares haciendo uso de Pattern y Matcher.

```
Pattern pattern = Pattern.compile("\\w+");
Matcher matcher = pattern.matcher("Say Hi");
```

Lo primero que vamos a hacer es hacer uso de Pattern en donde definimos la expresión regular, y a partir de aquí haremos uso de Matcher con el Pattern creado pasando por parámetro un String.

Ejemplos de Expresiones regulares en Java

A continuación vamos a ver diferentes ejemplos de expresiones en Java:

Encontrar todas las palabras de una cadena de texto:

A continuación vamos a ver cómo sería el patrón dado

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class Main {
    public static void main(String[] args) {

            String text = "Say Hi";
            Pattern pattern = Pattern.compile("\\w+");
            Matcher matcher = pattern.matcher(text);
            System.out.println(matcher.find());

        }
    }
```

Encontrar todas las palabras que acaben en id seguido de null

```
public class Main {
    public static void main(String[] args) {

        String text = "carId:null";
        Pattern pattern = Pattern.compile("\\w*Id:null");
        Matcher matcher = pattern.matcher(text);
        System.out.println(matcher.find());

        }
    }
```

Encontrar fechas con expresión regular

```
  String text = "2014-02-02";
  Pattern pattern = Pattern.compile("\\d{4}-\\d{2}-\\d{2}");
  Matcher matcher = pattern.matcher(text);
  System.out.println(matcher.find());
```

Encontrar la secuencia de caracteres que cumpla con un formato UUID en forma canónica.

Un UUID es un número de 16 bytes (128 bits). Contiene 32 caracteres hexadecimales. Puede expresarse en forma no canónica (series de 32 letras [A-F, a-f] y/o números [0-9], por ejemplo 550e8400e29b41d4a716446655440000) o en forma canónica (grupos de 8,4,4,4,12, por ejemplo 550e8400-e29b-41d4-a716-446655440000).

```
  String text = "aa8fe95b-db40-480b-b47c-949d5340d9df";
  Pattern pattern = Pattern.compile( "[a-fA-F0-9]{8}-[a-fA-F0-9]{4}-
[a-fA-F0-9]{4}-[a-fA-F0-9]{4}-[a-fA-F0-9]{12}");
  Matcher matcher = pattern.matcher(text);
  System.out.println(matcher.find());
```

Encontrar la secuencia de caracteres que cumpla con un correo electrónico:

```
  String text = "refactorizando.web@gmail.com";
  Pattern pattern = Pattern.compile("^[A-Z0-9._%+-]+@[A-Z0-9.-]+\\.[A-
Z]{2,6}$", Pattern.CASE_INSENSITIVE);
  Matcher matcher = pattern.matcher(text);
  System.out.println(matcher.find());
```

Encontrar palabras duplicadas

```
  String text = "Hola que que";
  Pattern pattern = Pattern.compile("\b(\w+)\s+\1\b",
Pattern.CASE_INSENSITIVE);
  Matcher matcher = pattern.matcher(text);
  System.out.println(matcher.find());
```

Agrupar grupos en la expresión regular

Podemos agrupar una parte de nuestra expresión regular con paréntesis. Además de poder agrupar, podemos crear una referencia posterior de la expresión, es decir, una referencia que se hace más tarde almacena la parte de string que coincide con el grupo.

Para poder referirnos a un grupo en concreto haremos uso del símbolo del $, por ejemplo:

Eliminar las palabras «hola» y «adios»:

```
String pattern = "(\\hola)(\\qué)(adios)";

System.out.println(TEXT.replaceAll(pattern, "$1$3"));
```

Tratamiento del backslash en las expresiones regulares

El uso de backslash es algo que hay que tener en cuenta a la hora de definir nuestra expresión regular. El backslash \ es un carácter de escape en los Strings de Java. Por ejemplo si queremos definir en nuestra expresión \w tendríamos que usar \\w en nuestra expresión.

Usar Or en un patrón

Para hacer uso de Or en un patrón, utilizamos | por ejemplo:

```
.*(coche|azul).*
```

Negación de un patrón

Si necesitamos la negación de un patrón podemos hacer uso de (?!pattern), por ejemplo si queremos encontrar la palabra plátano que no esté seguida por la ‘s’:

```
plátano(?!s)
```

Acciones específicas en un patrón

Podemos añadir tres diferentes acciones o modos para nuestra expresión regular:

- (?i) nuestra expresión será sensible a mayúsculas y minúsculas.
- (?m) o Pattern.MULTILINE para modo de múltiple línea, hace que la intercalación y el dolar coincida al principio y al final.

- (?s) o Pattern.DOTALL para modo de una única línea hace que el punto coincida al final de la línea.

Si queremos aplicar todos los modos sería de la siguiente manera: (?ismx).