Práctica: Expresiones Regulares en Java

Introducción

Una **expresión regular** es un patrón en el que se utilizan una secuencia de caracteres y símbolos para definir un conjunto de cadenas. Estas pueden usarse para comparar (y reemplazar) texto en un archivo, datos suministrados, capturados, etc.

El patrón definido por una expresión regular se aplica a una cadena de izquierda a derecha. Una vez que un carácter de la cadena "se ha usado", no puede reutilizarse. Por ejemplo la expresión regular "oso" se ajusta a la cadena "osososos" solo dos veces (oso_oso).

¿Qué se puede hacer con las expresiones regulares? ¿Qué utilidad práctica tienen las expresiones regulares? ¿Para qué sirven las expresiones regulares?

- **Buscar** un patrón y decir si se ha encontrado o no
- **Remplazar** lo que ha encontrado un patrón por otra cosa
- **Separar** partes y guardarlas en variables
- **Contar** todas las coincidencias que encuentra un patrón
- **Posición** de cada coincidencia que haya encontrado el patrón

JAVA

La API de Java contiene una serie de clases que nos permiten determinar si una secuencia de caracteres se ajusta a un patrón definido por una expresión regular. El paquete `java.util.regex` contiene dos clases: la clase `Matcher` y la clase `Pattern`.

La clase `Pattern` es la representación compilada de una expresión regular, o lo que es lo mismo, representa a la expresión regular, que en el paquete `java.util.regex` necesita estar compilada. En castellano significa patrón.

La clase `Matcher` es un tipo de objeto que se crea a partir de un patrón mediante la invocación del método `Pattern.matcher`. Este objeto es el que nos permite realizar operaciones sobre la secuencia de caracteres que queremos validar o la en la secuencia de caracteres en la que queremos buscar. En castellano lo mas parecido a esto es la palabra encajador.

Las expresiones regulares se rigen por una serie de normas, y para la construcción de cualquier patrón de caracteres se utilizan (además de letras y números) unos caracteres especiales. A lo largo de esta práctica se explicarán algunos de ellos, pero en el documento "Regular-expressions.pdf" aparece un listado con una breve explicación de su uso.

Veamos a través de ejemplos, el uso de estas clases y cómo se han de definir los patrones. Lo primero que hay que hacer es incluir el paquete `java.util.regex` al inicio de nuestras clases.

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");
}
```

El objeto `pat`, de la clase Pattern, representa la expresión regular. Este contiene el método `compile(String regex)` que recibe como parámetro la expresión regular. El objeto `mat`, de la clase Matcher, compara el String y la expresión regular. Contiene el método `matches(CharSequence input)` que recibe como parámetro el String a validar y devuelve true si coincide con el patrón.

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

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

El patrón definido en la expresión regular contiene los caracteres "`.*`". El carácter "`.`" indica que puede aparecer cualquier carácter menos el comienzo de una nueva línea (\n). El carácter "`*`", es un cuantificador, e indica que, de lo que hay antes, pueden haber o cero o más ocurrencias.

La clase `Matcher` contiene más métodos para trabajar con las expresiones regulares:

- `boolean find()` Busca en la cadena de texto a analizar, a partir del último matching, la secuencia de caracteres que puede concordar con la expresión regular (si hay varias coincidencias dentro del mismo texto, cada vez que llamemos a find(), nos devolverá la siguiente coincidencia).
- `boolean find(int start)` Lo mismo que el método anterior, pero empieza a buscar a partir de un índice que se le introduce como parámetro.
- `String replaceAll(String cad)` Devuelve una cadena, que es la alteración de la cadena original, en la que se han sustituido todas subsecuencias de caracteres que coinciden con la expresión regular "cad" pasada por parámetro.

El listado completo de métodos de la clase `Matcher` pueden encontrarse en:
https://docs.oracle.com/javase/tutorial/essential/regex/matcher.html

Por otro lado, la clase `Pattern` permite trabajar en su método compile con flags. Dependiendo de cuales se usen, esto afectará a la forma en la que se busquen las coincidencias de patrones. El listado de flags que pueden emplearse con esta clase se encuentra en:
https://docs.oracle.com/javase/tutorial/essential/regex/pattern.html

Método `split(String)`

Este método puede ser de gran utilidad a la hora de trabajar con expresiones regulares. Nos permite dividir una cadena en base a las ocurrencias de una expresión regular definida dentro. Véase ejemplo:

```
import java.util.regex.Pattern;
import java.util.regex.Matcher;
public class SplitDemo {
    private static final String REGEX = ":";
    private static final String INPUT =
        "one:two:three:four:five";

    public static void main(String[] args) {
        Pattern p = Pattern.compile(REGEX);
        String[] items = p.split(INPUT);
        for(String s : items) {
            System.out.println(s);
        }
    }
}
OUTPUT:
one
two
three
four
five
```

Ejercicios

Una vez explicados los conceptos básicos de las expresiones regulares en Java, los afianzaremos y ampliaremos realizando ejercicios. Para comprobar si las expresiones regulares funcionan tal y como se desea, existen plataformas on-line, que nos ayudan a validarlas:

- txt2re: http://txt2re.com/
- Regex Pal: http://www.regexpal.com/
- RegExr: http://www.regexr.com/
- Regexper: https://regexper.com/
- Regular expressions 101: https://regex101.com/

Puedes utilizarlas para validar y evaluar tus ejercicios.

1. Comprobar si una cadena empieza por “abc”
2. Comprobar si una cadena empieza por “abc” o “Abc”
3. Comprobar si una cadena no empieza por un dígito
4. Comprobar si una cadena no acaba con un dígito
5. Comprobar si una cadena solo contiene los caracteres “l” o “a”
6. Comprobar si una cadena contiene un 2 y ese no está seguido por un 6
7. Comprobar si una cadena está formada por un mínimo de 5 letras mayúsculas o minúsculas y un máximo de 10
8. Comprobar si una cadena es una dirección web que comience por www y sea de un servidor español
9. Comprobar si una cadena es una fecha dd/mm/yy. Comprueba que tu patrón coincida con las siguientes fechas: 25/10/83, 4/11/56, 30/6/71 y 4/3/85
10. Comprobar si una cadena contiene una dirección IP. Comprueba que tu patrón coincida con las siguientes IP: 192.168.1.1, 200.36.127.40 y 10.128.1.253
11. ¿Qué expresión regular utilizarías para comprobar si un número de teléfono fijo es español? Ten en cuenta el siguiente ejemplo para realizar el patrón: +34 95 6030466
12. ¿Qué expresión regular utilizarías para comprobar el número de pedido de una empresa cuyo ID puede tener los siguientes:
    - P nn-nnnnn
    - P-nn-nnnn
    - P# nn nnnn
    - P#nn-nnnn
    - P nnnnnn

    Siendo P el comienzo del ID, y n un número.
13. Para evitar el spam, intenta localizar posibles alteraciones que se utilizan para saltarse los filtros de correo. Por ejemplo, la palabra “viagra” podría encontrarse:
    - vi@gra

- v1agra
- v1@gra
- v!@gr@

14. Descarga la página principal de la UCA y localiza a través de una expresión regular en el fichero html almacenado, todas las imágenes de la página web. Busca cómo se incluyen imágenes en html.
15. Dada la siguiente cadena:

    “<a>unodos<c>tres</c><d>cuatro</d><e>cinco</e>” Extrae los caracteres escritos entre los <tags></tags> utilizando el siguiente patrón: <[^>]*>([^<]*)</[^>]*> ¿es correcto? Si no es así, corrígelo. Intenta ahora utilizar el siguiente patrón: <.*>(.*)<\/.*> ¿funciona? ¿qué diferencia hay con el patrón anterior? Finalmente utiliza el siguiente patrón: <.*?>(.*?)<\/.*?> ¿Obtienes los mismos resultados? ¿Qué diferencia hay entre unos y otros?
16. Elimina los símbolos (:,.;?¿¡!…”’<<>>) del texto que aparece en el fichero “EjercicioExpresiones.txt”
17. Quita las tildes del texto obtenido en el ejercicio anterior; reemplaza por la letra no acentuada
18. Reemplaza, del resultado obtenido en el ejercicio anterior, las palabras formadas únicamente por números (ojo, no las palabras con números como H2O, R2-D2, etc) por un espacio
19. Convierte el texto anterior a mayúsculas
20. Reemplaza los dobles espacios que se hayan podido crear por un único espacio.