REGEX en JAVA

Caràcters de control

Caràcter	Descripció	Exempe
\d	Qualsevol numero	[0-9]
\w	Qualsevol caràcter alfanumèric (majuscula, minuscula, número, caràcter)	[a-zA-Z0-9_]
\s	Qualsevol espai en blanc[0-9]	[\t\n\x0b\r\f]
\D	No número	[^0-9]
\W	Quasevol cos que no siga un caràcter alfanumèric	[^\w]
\S	Sense espai en blanc	[^\s]

Caràcters

Expressió	Descripció
.	Qualsevol caràcter
^	Negació de caràcter
?	El caràcter pot apareixer o no
[abc]	Indiquen Opció. La cadena té a o b o c.
[abc][12]	a o b o c seguit de 1 o 2
[^abc]	^ Qualsevol caràter excepte a o b o c
[A-Z]	Rang de caràcters, inclourà de la A fins Z
[a-z1-9]	Minúscules de la a fins la z (les dos incloses) i seguirdesd'un número de 1 a 9
A|B	| es una OR . A o B.
AB	A seguida de una B

Exemples:

```
public class ExemplesMatches {

	public static void main(String[] args) {

		String cadena="Solo se que no se nada";

		// ejemplo1: devolvera false, ya que la cadena tiene mas caracteres
		System.out.println("ejemplo1: "+cadena.matches("Solo"));

		// ejemplo2: devolvera true, siempre y cuando no cambiemos la cadena Solo
		System.out.println("ejemplo2: "+cadena.matches("Solo.*"));

		// ejemplo3: devolvera true, siempre que uno de los caracteres se cumpla
		System.out.println("ejemplo3: "+cadena.matches(".*[qnd].*"));

		// ejemplo3: devolvera false, ya que ninguno de esos caracteres estan
		System.out.println("ejemplo4: "+cadena.matches(".*[xyz].*"));

		// ejemplo4: devolvera true, ya que le indicamos que no incluya esos caracteres
		System.out.println("ejemplo4: "+cadena.matches(".*[^xyz].*"));

		// ejemplo5: devolvera true, si quitamos los caracteres delante de ? del String
		original seguira devolviendo true
		System.out.println("ejemplo5: "+cadena.matches("So?lo se qu?e no se na?da"));
```

```
                // ejemplo6: devolvera false, ya que tenemos una S mayuscula empieza en el String
                System.out.println("ejemplo6: "+cadena.matches("[a-z].*"));

                // ejemplo7: devolvera true, ya que tenemos una S mayuscula empieza en el String
                System.out.println("ejemplo7: "+cadena.matches("[A-Z].*"));

                String cadena2="abc1234";

                // ejemplo8: devolvera true, ya que minimo debe repetirse alguno de los caracteres al
                menos una vez
                System.out.println("ejemplo8: "+cadena2.matches("[abc]+.*"));

                // ejemplo9: devolvera true, ya que, ademas del ejemplo anterior, indicamos que debe
                  repetirse un valor numerico 4 veces
                System.out.println("ejemplo9: "+cadena2.matches("[abc]+\\d{4}"));

                // ejemplo10: devolvera true, ya que, ademas del ejemplo anterior, indicamos que debe
                repetirse un valor numerico entre 1 y 10 veces
                System.out.println("ejemplo10: "+cadena2.matches("[abc]+\\d{1,10}"));

        }
}
```

Clase Pattern

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

Si queremos comprobar que una cadena leída por teclado cumple ese patrón, podemos usar la clase **`Pattern`**. A la clase **`Pattern`** le decimos el patrón que queremos que cumpla nuestra cadena y nos dice si la cumple o no.

Expresión regular para fecha

Imaginemos la fecha en formato *dd/mm/yyyy*. Son grupos de dos cifras separadas por barras. En una expresión regular `\d` representa una cifra. El día pueden ser una o dos cifras, es decir `\d{1,2}`, el mes igual y el año vamos a obligar que sean cuatro cifras exactamente `\d{4}`

El siguiente ejemplo comprueba si la cadena cumple con la expresión regular. T

```
            // dd/mm/aaaa:   \d{1,2}/\d{1,2}/\d{4}

String regexp = "\\d{1,2}/\\d{1,2}/\\d{4}";

            // Lo siguiente devuelve true

            System.out.println(Pattern.matches(regexp, "11/12/2014"));
            System.out.println(Pattern.matches(regexp, "1/12/2014"));
            System.out.println(Pattern.matches(regexp, "11/2/2014"));


            // Los siguientes devuelven false

            System.out.println(Pattern.matches(regexp, "11/12/14"));
        // El año no tiene cuatro cifras

            System.out.println(Pattern.matches(regexp, "11//2014"));
        // el mes no tiene una o dos cifras

            System.out.println(Pattern.matches(regexp, "11/12/14perico"));
            // Sobra "perico"
```

Supongamos que queremos que el mes se exprese como "ene", "feb", "mar", ... en vez de como un número. Cuando hay varias posibles cadenas válidas, en la expresión regular se ponen entre paréntesis y separadas por `|`. Es decir, algo como esto `(ene|feb|mar|abr|may|jun|jul|ago|sep|oct|nov|dic)`. Si además nos da igual mayúsculas o minúsculas, justo delante ponemos el flag de *case insensitive* `(?i)` (la 'i' es de *ignore case*)

El siguiente código muestra un ejemplo completo de esto.

```
public class ExempleRegex {

public static void main(String[] args) {

String regexp = "\\d{1,2}/(?i)(ene|feb|mar|abr|may|jun|jul|ago|sep|oct|nov|dic)/\\d{4}";


       // Los siguientes devuelven true
       System.out.println(Pattern.matches(regexp, "11/dic/2014"));
       System.out.println(Pattern.matches(regexp, "1/nov/2014"));
       System.out.println(Pattern.matches(regexp, "1/AGO/2014"));   // Mes en mayúsculas
       System.out.println(Pattern.matches(regexp, "21/Oct/2014"));  // Primera letra del
mes en mayúsculas.



             //Los siguientes devuelven false
       System.out.println(Pattern.matches(regexp, "11/abc/2014"));   // abc no es un mes
       System.out.println(Pattern.matches(regexp, "11//2014"));      // falta el mes
       System.out.println(Pattern.matches(regexp, "11/jul/2014perico")); // sobra perico

       }

}
```

Expresión Regular Para DNI

`\d{8}[A-HJ-NP-TV-Z]    (no existen letras 'I', 'O' i 'U')`

```
String regexp
      = "\\d{1,2}/(?i)(ene|feb|mar|abr|may|jun|jul|ago|sep|oct|nov|dic)/\\d{4}";


             String dniRegexp = "\\d{8}[A-HJ-NP-TV-Z]";

             // Lo siguiente devuelve true
             System.out.println(Pattern.matches(dniRegexp, "01234567C"));

             // Lo siguiente devuelve faslse

             System.out.println(Pattern.matches(dniRegexp, "01234567U"));
             // La U no es válida

             System.out.println(Pattern.matches(dniRegexp, "0123567X"));
              // No tiene 8 cifras
```

Expresión regular para email

Suponemos que:

`[^@]+@[^@]+\.[a-zA-Z]{2,}`.

- `[^@]+` cualquier caracter que no sea @ una o más veces seguido de
- `@` una @ seguido de
- `[^@]+` cualquier caracter que no sea @ una o más veces seguido de
- `\.` un punto seguido de
- `[a-zA-Z]{2,}` dos o más letras minúsculas o mayúsculas

```
public class ExempleRegex {

	public static void main(String[] args) {

		String emailRegexp = "[^@]+@[^@]+\\.[a-zA-Z]{2,}";

		// Lo siguiente devuelve true
		System.out.println(Pattern.matches(emailRegexp, "a@b.com"));
		System.out.println(Pattern.matches(emailRegexp, "+++@+++.com"));

		// Lo siguiente devuelve faslse
		System.out.println(Pattern.matches(emailRegexp, "@b.com")); // Falta el
nombre
		System.out.println(Pattern.matches(emailRegexp, "a@b.c")); // El dominio
final debe tener al menos dos letras

	}
```