Ejemplos de Expresiones Regulares en Java

Una expresión regular define un patrón de búsqueda para cadenas de caracteres.

La podemos utilizar para comprobar si una cadena contiene o coincide con el patrón. El contenido de la cadena de caracteres puede coincidir con el patrón 0, 1 o más veces.

Algunos ejemplos de uso de expresiones regulares pueden ser:

- para comprobar que la fecha leída cumple el patrón dd/mm/aaaa
- para comprobar que un NIF está formado por 8 cifras, un guión y una letra
- para comprobar que una dirección de correo electrónico es una dirección válida.
- para comprobar que una contraseña cumple unas determinadas condiciones.
- Para comprobar que una URL es válida.
- Para comprobar cuántas veces se repite dentro de la cadena una secuencia de caracteres determinada.
- Etc. Etc.

El patrón se busca en el String de izquierda a derecha. Cuando se determina que un carácter cumple con el patrón este carácter ya no vuelve a intervenir en la comprobación.

Ejemplo:

La expresión regular "010" la encontraremos dentro del String "010101010" solo dos veces: "010101010"

Símbolos comunes en expresiones regulares

Expresión	Descripción
.	Un punto indica cualquier carácter
^expresión	El símbolo ^ indica el principio del String. En este caso el String debe contener la expresión al principio.
expresión$	El símbolo $ indica el final del String. En este caso el String debe contener la expresión al final.
[abc]	Los corchetes representan una definición de conjunto. En este ejemplo el String debe contener las letras a ó b ó c.
[abc][12]	El String debe contener las letras a ó b ó c seguidas de 1 ó 2
[^abc]	El símbolo ^ dentro de los corchetes indica negación. En este caso el String debe contener cualquier carácter excepto a ó b ó c.
[a-z1-9]	Rango. Indica las letras minúsculas desde la a hasta la z (ambas incluidas) y los dígitos desde el 1 hasta el 9 (ambos incluidos)
A|B	El carácter | es un OR. A ó B
AB	Concatenación. A seguida de B

Meta caracteres

Expresión	Descripción
\d	Dígito. Equivale a [0-9]
\D	No dígito. Equivale a [^0-9]
\s	Espacio en blanco. Equivale a [\t\n\x0b\r\f]
\S	No espacio en blanco. Equivale a [^\s]
\w	Una letra mayúscula o minúscula, un dígito o el carácter _ Equivale a [a-zA-Z0-9_]
\W	Equivale a [^\w]
\b	Límite de una palabra.

En Java debemos usar una doble barra invertida \\

Por ejemplo para utilizar \w tendremos que escribir \\w. Si queremos indicar que la barra invertida en un carácter de la expresión regular tendremos que escribir \\\\.

Cuantificadores

Expresión	Descripción
{X}	Indica que lo que va justo antes de las llaves se repite X veces
{X,Y}	Indica que lo que va justo antes de las llaves se repite mínimo X veces y máximo Y veces. También podemos poner {X,} indicando que se repite un mínimo de X veces sin límite máximo.
*	Indica 0 ó más veces. Equivale a {0,}
+	Indica 1 ó más veces. Equivale a {1,}
?	Indica 0 ó 1 veces. Equivale a {0,1}

Usar expresiones regulares con la clase String.

Métodos matches y split.

String.matches(regex)

Podemos comprobar si una cadena de caracteres cumple con un patrón usando el método matches de la clase String. Este método recibe como parámetro la expresión regular.

Ejemplos de Expresiones Regulares en Java:

1. Comprobar si el String *cadena* contiene exactamente el patrón (matches) “abc”

```
System.out.println(s.matches("abc"));
```

2. Comprobar si el String *cadena* contiene “abc”

```
if (s.matches(".*abc.*"))  System.out.println("Sí");
else  System.out.println("No");
```

3. Comprobar si el String *cadena* empieza por “abc”

```
String s="En la casa es mayor abc que def";
if (s.matches("^abc.*"))  System.out.println("Sí");
else  System.out.println("No");
```

4. Comprobar si el String *cadena* empieza por “abc” ó “Abc”

```
String s="abc En la casa es mayor abc que def";
System.out.println(s.matches("^[aA]bc.*")?"SI":"NO");
```

5. Comprobar si el String *cadena* está formado por un mínimo de 5 letras mayúsculas o minúsculas y un máximo de 10.

```
System.out.println(s.matches("[a-zA-Z]{5,10}")?"SI":"NO");
```

6. Comprobar si el String *cadena* no empieza por un dígito

```
System.out.println(s.matches("^[^\\d].*")?"SI":"NO");
```

7. Comprobar si el String *cadena* no acaba con un dígito

```
System.out.println(s.matches(".*[^\\d]$")?"SI":"NO");
```

8. Comprobar si el String *cadena* solo contienen los caracteres a ó b

```
System.out.println(s.matches("(a|b)+")?"SI":"NO");
```

9. Comprobar si el String *cadena* contiene un 1 y ese 1 no está seguido por un 2

```
System.out.println(s.matches(".*1(?!2).*")?"SI":"NO");
```

String.split(regex)

El método split de la clase String es para separa cadenas. Este método divide el String en cadenas según la expresión regular que recibe. La expresión regular no forma parte del array resultante.

Ejemplo 1:

```
String str = "blanco-rojo:amarillo.verde_azul";
String [] cadenas = str.split("[-:._]");
for(int i = 0; i<cadenas.length; i++){
  System.out.println(cadenas[i]);
}
```

Muestra por pantalla:

```
blanco
rojo
amarillo
verde
azul
```

Ejemplo 2:

```
String str = "esto es un ejemplo de como funciona split";
String [] cadenas = str.split("(e[s|m])|(pl)");
for(int i = 0; i<cadenas.length; i++){
  System.out.println(cadenas[i]);
}
```

Salida:

```
to
 un ej
o de como funciona s
it
```

Ejemplo: expresión regular para comprobar si un email es válido

```
import java.util.Scanner;

public class Ej1 {
   public static void main(String[] args) {
       Scanner teclado=new Scanner(System.in);
       String email;

       System.out.print("Introduce el email: ");
       email=teclado.nextLine();
        if (email.matches("^[\\w-]+(\\.[\\w-]+)*@[A-Za-z0-9]+(\\.[A-Za-z0-9]+)*(\\.[A-Za-z]{2,})$"))
           System.out.println("Válido");
       else
           System.out.println("No Válido");
   }
}
```

Hemos usado la siguiente expresión regular para comprobar si un email es válido:

`"^[\\w-]+(\\.[\\w-]+)*@[A-Za-z0-9]+(\\.[A-Za-z0-9]+)*(\\.[A-Za-z]{2,})$"`

La explicación de cada parte de la expresión regular es la siguiente:

[\\w-]+	Inicio del email El signo + indica que debe aparecer uno o más de los caracteres entre corchetes: \\w indica caracteres de la A a la Z tanto mayúsculas como minúsculas, dígitos del 0 al 9 y el carácter _ Carácter – En lugar de usar \\w podemos escribir el rango de caracteres con lo que esta expresión quedaría así: [A-Za-z0-9-_]+
(\\.[\\w-]+)*	A continuación: El * indica que este grupo puede aparecer cero o más veces. El email puede contener de forma opcional un punto seguido de uno o más de los caracteres entre corchetes.
@	A continuación debe contener el carácter @
[A-Za-z0-9]+	Después de la @ el email debe contener uno o más de los caracteres que aparecen entre los corchetes
(\\.[A-Za-z0-9]+)*	Seguido (opcional, 0 ó más veces) de un punto y 1 ó más de los caracteres entre corchetes
(\\.[A-Za-z]{2,})	Seguido de un punto y al menos 2 de los caracteres que aparecen entre corchetes (final del email)

Ejercicios de Expresiones Regulares

1. **Nota**: Este ejercicio se puede probar utilizando el formulario.

Validación de entrada de texto

Escribe algo:

Enviar Borrar

Al pulsar el botón Enviar, debe mostrarse una lista donde se indique si la expresión introducida por el usuario es:

a. vacía
b. una única palabra que contenga solamente letras
c. un único número que contenga solamente cifras (sin decimales ni signo)
d. número de teléfono (9 cifras, empezando por 6 o 9)
e. DNI (de 1 a 8 números, con letra o sin ella)
f. código postal (cinco cifras, empezando por 0, 1, 2, 3 o 4)
g. dos palabras (sólo letras, separadas por uno o varios espacios)
h. varias palabras (sólo letras, separadas por uno o varios espacios)
i. fecha de nacimiento: dd/mm/aaaa
j. un único número sin signo y con como mucho dos decimales (con separador punto o coma, sólo si hay decimales)
k. un único número con signo (más o menos) y con decimales (con separador punto o coma, sólo si hay decimales)
l. contraseña (que tenga al menos seis caracteres, y que contenga números y letras y algún símbolo, pero no espacios)
m. dirección de correo electrónico (con la arroba y que acabe con un dominio)

2. Verificar el NIF.

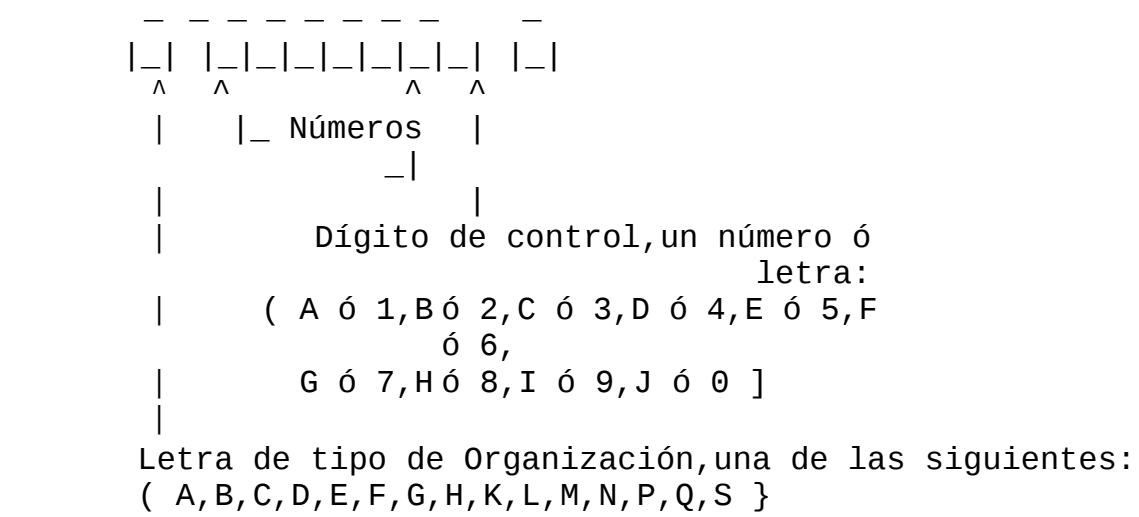
Si utilizamos 10 caracteres, 8 pertenecen al Dni, 1 para '-' y 1 para la letra. Los 8 dígitos del Dni se utilizan para el cálculo de la letra. Para ello, es necesario calcular el resto de dividir el Dni entre 23. Siendo la secuencia: **T R W A G M Y F P D X B N J Z S Q V H L C K E**

3. Verificar el N.I.E. Numero de Identificacion de Extranjeros.

El NIE siempre comienza por una X, después un guión, luego 7 números y finalmente termina con guión y una letra como el DNI. Se utiliza el mismo algoritmo pero con sólo la introducción de los 7 dígitos numéricos, y con la misma cadena.

4. Verificar el C.I.F. - Código de Identificación Fiscal (Empresas - Organizaciones).

El CIF (Código de Identificación Fiscal) es un elemento de identificación administrativa
para organizaciones y consta de 9 dígitos:



El primer dígito es una letra que indica el tipo de la
organización y puede ser una de los siguientes:

A - Sociedad Anónima.
B - Sociedad de responsabilidad limitada.
C - Sociedad colectiva.
D - Sociedad comanditaria.
E - Comunidad de bienes.
F - Sociedad cooperativa.
G - Asociación.
H - Comunidad de propietarios.
K - Formato antiguo.
L - Formato antiguo.
M - Formato antiguo.
N - Formato antiguo.
P - Corporación local.
Q - Organismo autónomo.
S - Organo de la administración.

Los siete dígitos siguientes son números y el último es el dígito de control que puede ser un número ó una letra.

Las operaciones para calcular el dígito de control se realizan sobre los siete dígitos centrales y son las siguientes:

- Sumar los dígitos de la posiciones pares. Suma = A

- Para cada uno de los dígitos de la posiciones impares, multiplicarlo por 2 y sumar los dígitos del resultado.

Ej.: (8 * 2 = 16 --> 1 + 6 = 7)

Acumular el resultado. Suma = B

- Sumar A + B = C

- Tomar sólo el dígito de las unidades de C y restárselo a 10.
Esta resta nos da D.
- A partir de D ya se obtiene el dígito de control.
Si ha de ser numérico es directamente D y si se trata de una letra se corresponde con la relación: A = 1, B = 2, C= 3, D = 4, E = 5, F = 6, G = 7, H = 8, I = 9, J = 0

Ejemplo para el CIF: A58818501

- Utilizamos los siete dígitos centrales = 5881850

- Sumamos los dígitos pares: A = 8 + 1 + 5 = 14

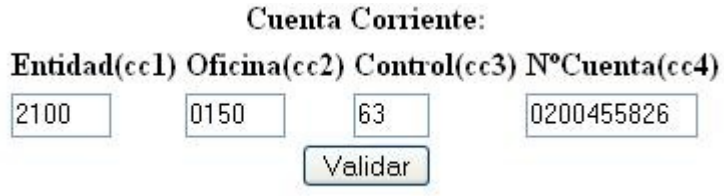
- Posiciones impares:
5 * 2 = 10 -> 1 + 0 = 1
8 * 2 = 16 -> 1 + 6 = 7
8 * 2 = 16 -> 1 + 6 = 7
0 * 2 = 0 -> = 0
Sumamos los resultados: B = 1 + 7 + 7 + 0 = 15

- Suma parcial: C = A + B = 14 + 15 = 29

- El dígito de las unidades de C es 9.
Se lo restamos a 10 y nos da: D = 10 - 9 = 1

- Si el dígito de control ha de ser un número es 1 y si ha de ser una letra es la "A"

5. Validación de una cuenta bancaria correspondiente a un Código de Cuenta de Cliente (CCC) de alguna entidad financiera que opere en España una cadena numérica de 20 caracteres de longitud. De la siguiente forma:



CÓDIGO DE ENTIDAD	CÓDIGO DE OFICINA	DÍGITOS DE CONTROL	NÚMERO DE CUENTA
1234	5678	06	1234567890

Código de Entidad: lo forman los primeros cuatro dígitos del Código de Cuenta de Cliente (CCC), y representa el Banco, Caja de Ahorros o Entidad Financiera donde radica la cuenta.

Código de Oficina: lo forman los siguientes cuatro dígitos del CCC, y es un número que asigna cada Entidad para identificar a cada una de su sucursales, donde un cliente mantiene su cuenta.

Dígitos de Control: lo forman los dígitos noveno y décimo del CCC. El primer dígito (el que ocupa la novena posición) verifica los Códigos de Entidad y Oficina; el segundo dígito (el que ocupa la décima posición en el CCC) verifica el Número de Cuenta.

Número de Cuenta: lo forman los últimos diez dígitos del CCC, donde la Entidad puede incluir identificadores para individualizar cada número de cuenta en particular

Tenemos los Pesos = array(6, 3, 7, 9, 10, 5, 8, 4, 2, 1);

Dígito de control de la entidad y sucursal

A continuación se muestra los factores para su cálculo y un ejemplo para los siguientes datos:

Entidad 2077
Sucursal 0338

			Ejemplo	
Posición	**Factores**	**Datos**	**Operación**	**Resultado**
Unidad:	**6**	**8**	**6*8**	**48**
Decena:	**3**	**3**	**3*3**	**9**
Centena:	**7**	**3**	**7*3**	**21**
Unidad de millar:	**9**	**0**	**9*0**	**0**
Decena de millar:	**10**	**7**	**10*7**	**70**
Centena de millar:	**5**	**7**	**5*7**	**35**
Unidad de millón:	**8**	**0**	**8*0**	**0**
Decena de millón:	**4**	**2**	**4*2**	**8**
			Suma	**191**

Cálculos:

191/11 = 17; Resto = 191-(17*11) = 4; Dígito = 11-4 = 7

- Si el resultado es 11, el dígito buscado es 0
- Si el resultado es 10, el dígito buscado es 1

Dígito de control de la cuenta

A continuación se muestran los factores para el cálculo del dígito de control de una cuenta, que siempre ha de estar compuesta por 10 dígitos, la cuenta del ejemplo es : 3100254321

			Ejemplo	
Posición	**Factor**	**Datos**	**Operación**	**Resultado**
Unidad:	**6**	**1**	**6*1**	**6**
Decena:	**3**	**2**	**3*2**	**6**
Centena:	**7**	**3**	**7*3**	**21**
Unidad de millar:	**9**	**4**	**9*4**	**36**
Decena de millar:	**10**	**5**	**10*5**	**50**
Centena de millar:	**5**	**2**	**5*2**	**10**
Unidad de millón:	**8**	**0**	**8*0**	**0**
Decena de millón:	**4**	**0**	**4*0**	**0**
Centena de millón:	**2**	**1**	**2*1**	**2**
Unidad de millar de millón:	**1**	**3**	**1*3**	**3**
			Suma	**134**

Cálculos:
134/11 = 12; Resto = 134-(12*11) = 2; Dígito = 11-2 = 9

Así en nuestro ejemplo el Código Cuenta Cliente completo será:

Entidad	Sucursal	Dígitos de Control	Cuenta
2077	**0338**	**7,9**	**3100254321**

6. Cómo validar el número de la tarjeta de crédito mediante el algoritmo ISO 2894 El algoritmo es:

1. Calcular el peso para el primer dígito: si el número de dígitos es par el primer peso es 2, de lo contrario es 1. Después los pesos alternan entre 1,2,1,2,1 …
2. Multiplicar cada dígito por su peso
3. Si el resultado del 2º paso es mayor que 9, restar 9
4. Sumar todos los dígitos
5. Comprobar que el resultado es divisible por 10

7. Validación de un número de la Seguridad Social española

El número de afiliación a la Seguridad Social en España, lleva asociados un par de dígitos de control que sirven para evitar errores de transcripción.

El número de afiliación a la seguridad social consta de tres partes:

aa/bbbbbbbb/cc

Los primeros dígitos (a) son un indicativo de la provincia. Los dígitos centrales (b) son el nº del asegurado dentro de cada provincia. Los dígitos finales son dígitos de control.

A menudo, el número de la seguridad social se escribe sin barras.

Los dígitos de control (c) se obtienen a partir de las otras dos partes (a) y (b) de la siguiente forma:

si (b<10000000)
entonces d=b+a*10000000
si no d=valor de (a concatenado con b) //con b sin ceros a la izquierda
c=d mod 97 //resto de la división entera

8. Validación de códigos de barras en formato EAN13 y EAN8

El codigo EAN (European Article Number) es un sistema de código de barras para asignar un número único a cada producto. Los códigos más comunes tienen 8 o 13 dígitos, especialmente 13 (sistemas conocidos como EAN8 y EAN13). En ellos van codificados el pais de origen del producto, la empresa y el propio producto. El último de los dígitos es un dígito de control para evitar errores de transcripción.

Podemos describirlo algoritmicamente de esta manera:

Comprobar que el código tiene 8 o 13 dígitos. De no ser así, no es correcto.
Sumar los dígitos de lugares pares por un lado y los de los impares por otro, pero sin incuir el último dígito.
Si el código es EAN13, multiplicar la suma de los pares por 3.
Si el código es EAN8, es la suma de los impares la que se multiplica por 3.
Sumar el resultado de los pares y el de los impares y hallar el resto de la división por 10.
Realizar la operación 10 menos ese resto y ese es el dígito de control. Si como resultado sale 10, entenderemos que el dígito de control es 0.

Comprobar que el dígito de control que hemos calculado y el último dígito del código EAN coinciden

Por ejemplo, para validar el código EAN8 "12345678" (Obviamente es inventado)

1.Separar el dígito de control. Nos quedamos con "1234567" y "8"
2.Sumar pares: sumapares=2+4+6=12
3.Sumar impares: sumaimpares=1+3+5+7=16
4.Como es EAN8, multiplicamos los impares por 3.
5.sumaimpares=16*3=48
6.Sumar el resultado de pares e impares: 12+48=60
7.Hallar el resto de la division por 10: 60 mod 10 = 0
8.Hacer 10-resto: 10-0=10
9.Como nos ha salido 10, el dígito de control es 0.
10. Comparar el dígito de control que hemos calculado con el que tenía el código: Nos sale 0 y el código tenía un 8. Es incorrecto.

Para usar expresiones regulares en Java se usa el package **java.util.regex**

Contiene las clases **Pattern** y **Matcher** y la excepción **PatternSyntaxException**.

Clase **Pattern:** Un objeto de esta clase representa la expresión regular. Contiene el método compile(String regex) que recibe como parámetro la expresión regular y devuelve un objeto de la clase Pattern.

La clase **Matcher:** Esta clase compara el String y la expresión regular. Contienen el método matches(CharSequence input) que recibe como parámetro el String a validar y devuelve true si coincide con el patrón. El método find() indica si el String contienen el patrón.

Ejemplos de Expresiones Regulares en Java:

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");
  }
```

2. Comprobar si el String *cadena* contiene “abc”

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

También lo podemos escribir usando el método find:

```
    Pattern pat = Pattern.compile("abc");
  Matcher mat = pat.matcher(cadena);
  if (mat.find()) {
    System.out.println("Válido");
  } else {
    System.out.println("No Válido");
  }
```

3. Comprobar si el String *cadena* empieza por “abc”

```
  Pattern pat = Pattern.compile("^abc.*");
  Matcher mat = pat.matcher(cadena);
  if (mat.matches()) {
    System.out.println("Válido");
  } else {
```

```
    System.out.println("No Válido");
  }
```

4. Comprobar si el String *cadena* empieza por “abc” ó “Abc”

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

5. Comprobar si el String *cadena* está formado por un mínimo de 5 letras mayúsculas o minúsculas y un máximo de 10.

```
  Pattern pat = Pattern.compile("[a-zA-Z]{5,10}");
  Matcher mat = pat.matcher(cadena);
  if (mat.matches()) {
    System.out.println("SI");
  } else {
    System.out.println("NO");
  }
```

6. Comprobar si el String *cadena* no empieza por un dígito

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

7. Comprobar si el String *cadena* no acaba con un dígito

```
  Pattern pat = Pattern.compile(".*[^\\d]$");
  Matcher mat = pat.matcher(cadena);
  if (mat.matches()) {
    System.out.println("SI");
  } else {
    System.out.println("NO");
  }
```

8. Comprobar si el String *cadena* solo contienen los caracteres a ó b

```
  Pattern pat = Pattern.compile("(a|b)+");
```

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

9. Comprobar si el String *cadena* contiene un 1 y ese 1 no está seguido por un 2

```
  Pattern pat = Pattern.compile(".*1(?!2).*");
  Matcher mat = pat.matcher(cadena);
  if (mat.matches()) {
    System.out.println("SI");
  } else {
    System.out.println("NO");
  }
```