

# Unidad 5:

## Estructuras básicas de datos en el lenguaje Java

---

**Fundamentos de Programación.**  
**1º de ASI**



Esta obra está bajo una licencia de Creative Commons.  
Autor: Jorge Sánchez Asenjo (año 2009) <http://www.jorgesanchez.net>  
e-mail: [info@jorgesanchez.net](mailto:info@jorgesanchez.net)

---

Esta obra está bajo una licencia de Reconocimiento-NoComercial-CompartirIgual de Creative Commons  
Para ver una copia de esta licencia, visite:  
<http://creativecommons.org/licenses/by-nc-sa/2.5/es/legalcode.es>  
o envíe una carta a:  
Creative Commons, 559 Nathan Abbot





## Reconocimiento-NoComercial-CompartirIgual 2.5 España

### Usted es libre de:



copiar, distribuir y comunicar públicamente la obra



hacer obras derivadas

### Bajo las condiciones siguientes:



**Reconocimiento.** Debe reconocer los créditos de la obra de la manera especificada por el autor o el licenciador (pero no de una manera que sugiera que tiene su apoyo o apoyan el uso que hace de su obra).



**No comercial.** No puede utilizar esta obra para fines comerciales.



**Compartir bajo la misma licencia.** Si altera o transforma esta obra, o genera una obra derivada, sólo puede distribuir la obra generada bajo una licencia idéntica a ésta.

- Al reutilizar o distribuir la obra, tiene que dejar bien claro los términos de la licencia de esta obra.
- alguna de estas condiciones puede no aplicarse si se obtiene el permiso del titular de los derechos de autor
- Apart from the remix rights granted under this license, nothing in this license impairs or restricts the author's moral rights.

Advertencia

Los derechos derivados de usos legítimos u otras limitaciones reconocidas por ley no se ven afectados por lo anterior.  
Esto es un resumen legible por humanos del texto legal (la licencia completa) disponible en los idiomas siguientes:

Catalán Castellano Euskera Gallego

Para ver una copia completa de la licencia, acudir a la dirección <http://creativecommons.org/licenses/by-nc-sa/2.5/es/legalcode.es>



# (5)

## estructuras básicas de datos en Java

### esquema de la unidad

(5.1) arrays	5
(5.1.1) introducción	5
(5.1.2) unidimensionales	6
(5.1.3) arrays multidimensionales	11
(5.1.4) longitud de un array	12
(5.1.5) la clase Arrays	12
(5.1.6) el método System.arraycopy	14
(5.2) clase String	14
(5.2.1) introducción. declarar e iniciar textos	14
(5.2.2) comparación entre objetos String	15
(5.2.3) String.valueOf	16
(5.2.4) métodos de los objetos String	16
(5.3) arrays de Strings	22
(5.4) parámetros de la línea de comandos	22

### (5.1) arrays

#### (5.1.1) introducción

Los tipos de datos que conocemos hasta ahora no permiten solucionar problemas que requieren gestionar muchos datos a la vez. Por ejemplo, imaginemos que deseamos leer las notas de una clase de 25 alumnos. Desearemos por tanto almacenarlas y para ello, con lo que conocemos hasta ahora, no habrá más remedio que declarar 25 variables.

Eso es tremendamente pesado de programar. Manejar esos datos significaría estar continuamente manejando 25 variables. Por ello en casi todos los lenguajes se pueden agrupar una serie de variables del mismo tipo en



una misma estructura que comúnmente se conoce como array<sup>1</sup>. Esa estructura permite referirnos a todos los elementos, pero también nos permite acceder individualmente a cada elemento.

Los arrays son una colección de datos del mismo tipo al que se le pone un nombre (por ejemplo *nota*). Para acceder a un dato individual de la colección hay que utilizar su posición. La posición es un número entero, normalmente se le llama *índice* (por ejemplo *nota[4]* es el nombre que recibe el cuarto elemento de la sucesión de notas).

Hay que tener en cuenta que en los arrays el primer elemento tiene como índice el número cero. El segundo el uno y así sucesivamente; es decir *nota[4]* en realidad es el quinto elemento del array.

Esto (con algunos matices funciona igual en casi cualquier lenguaje). Sin embargo en Java los arrays (como casi todo) son *objetos*, y aunque la programación orientada a objetos de Java será abordada más adelante, ya ahora conviene tenerlo en cuenta.

Por otro lado mientras en lenguajes como C los arrays son *estructuras estáticas* (significa que su tamaño queda definido en cuanto se declara el array), en Java son *estructuras dinámicas* (sí que es posible modificar su tamaño en tiempo de ejecución).

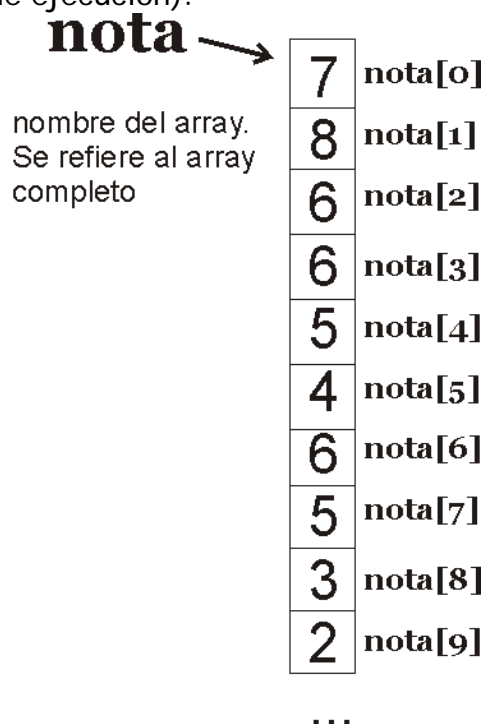


Ilustración 4-1, Ejemplo de array de notas

### (5.1.2) unidimensionales

Los arrays habituales son los llamados unidimensionales. Coinciden con la idea comentada en el apartado anterior. Aglutinan datos del mismo tipo y un

<sup>1</sup> Otros nombres habituales además de *arrays* son: listas, matrices, arreglos,... Estimo que array es el más aceptado en la actualidad, los otros términos son más ambiguos.

**índice** encerrado entre corchetes nos permite acceder a cada elemento individual.

La **declaración de un array unidimensional** se hace con esta sintaxis.

```
tipo nombre[];
```

Ejemplo:

```
double cuentas[]; //Declara un array que almacenará valores  
                  // doubles
```

Declara un array de tipo double. Esta declaración indica para qué servirá el array, pero no reserva espacio en la RAM al no saberse todavía el tamaño del mismo. En este sentido hay una gran diferencia entre Java y la mayoría de lenguajes clásicos. No es necesario conocer el tamaño del array en el momento de la declaración. Simplemente se avisa de que aparecerá un futuro array.

Tras la declaración del array, se tiene que **iniciar**. Eso lo realiza el operador **new**, que es el que realmente crea el array indicando un tamaño. Cuando se usa new es cuando se reserva el espacio necesario en memoria. **Un array no inicializado no se puede utilizar en el código.**

Ejemplo de iniciación de un array:

```
int notas[]; //sería válido también int[] notas;  
notas = new int[3]; //indica que el array constará de tres  
                  //valores de tipo int
```

También se puede hacer ambas operaciones en la misma instrucción:

```
int notas[] = new int[3];
```

En el ejemplo anterior se crea un array de tres enteros (con los tipos básicos se crea en memoria el array y se inicializan los valores, los números se inician a 0).

Los valores del array se asignan utilizando el índice del mismo entre corchetes:

```
notas[2]=8;
```

También se pueden asignar valores al array en la propia declaración:

```
int notas[] = {8, 7, 9};  
int notas2[] = new int[] {8,7,9}; //Equivalente a la anterior
```

Esto declara e inicializa un array de tres elementos. En el ejemplo lo que significa es que **notas[0]** vale **8**, **notas[1]** vale **7** y **notas[2]** vale **9**.

En Java (como en otros lenguajes) el primer elemento de un array es el cero. El primer elemento del array `notas`, es `notas[0]`. Se pueden declarar arrays a cualquier tipo de datos (enteros, booleanos, doubles, ... e incluso objetos como se verá más adelante).

La ventaja de usar arrays (volviendo al caso de las notas) es que gracias a un simple bucle `for` se puede recorrer fácilmente todos los elementos de un array:

```
//Calcular la media 18 notas  
suma=0;  
for (int i=0;i<=17;i++){  
    suma+=nota[i];  
}  
media=suma/18;
```

A un array se le puede inicializar las veces que haga falta:

```
int notas[]=new notas[16];  
...  
notas=new notas[25];
```

Pero hay que tener en cuenta que el segundo `new` hace que se pierda el contenido anterior. De hecho elimina ese contenido.

En la perspectiva de Java, un array es una referencia a una serie de valores que se almacenan en la memoria. El operador `new` en realidad lo que hace es devolver una **referencia** a los valores a los que ha asignado el hueco en memoria. Es decir el array es la manera de poder leer y escribir en esos valores.

Veamos paso a paso estas instrucciones y su efecto en la memoria:



## instrucción

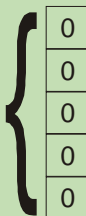
```
int notas[];
```

```
notas=new[5];
```

## efecto

se crea la referencia **notas**

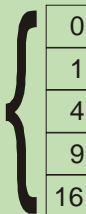
se crea un array de cinco números enteros y **notas** hace referencia a ellos:

**notas** → 

0
0
0
0
0

```
for(int i=0;i<5;i++){  
    nota[i]=i*i;  
}
```

se asignan valores en el array

**notas** → 

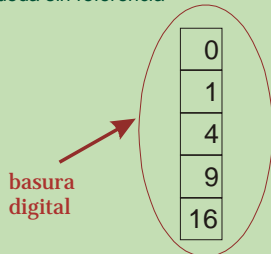
0
1
4
9
16

## instrucción


```
notas=new[8];
```

## efecto

se crea un nuevo array de ocho enteros **notas** le hace referencia, el anterior se queda sin referencia



basura digital

**notas** → 

0
0
0
0
0
0
0
0

**Ilustración 4-2**, efecto del uso del operador **new** en los arrays

La imagen anterior trata de explicar el funcionamiento del operador **new** (no sólo para los arrays). Utilizar dos veces el operador new con la misma referencia de array, provoca que haya valores que se quedan **sin referencia**. A eso se le suele llamar **basura digital**, es espacio que se consume en la memoria y que no se puede utilizar.

En Java un recolector de basura se encarga cada cierto tiempo de eliminar la basura (a diferencia de lo que ocurre en lenguajes menos seguros como C). Desde el código podemos forzar la recolección de basura mediante:

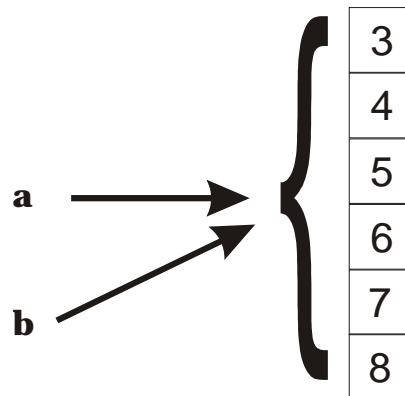
```
System.gc();
```

Aunque dicha instrucción no garantiza la recolección de basuras. Sólo se suele utilizar cuando nuestro código genera basura más rápidamente de lo que el recolector puede eliminar.

Un array se puede asignar a otro array (si son del mismo tipo):

```
int a[];  
int b[]=new int[]{3,4,5,6,7,8};  
a=b;
```

A diferencia de lo que podríamos intuir, lo que ocurre en el código anterior es que tanto *a* como *b* hacen referencia al mismo array. Es decir el resultado sería:



Esta asignación provoca que cualquier cambio en *a* también cambie el array *b* (ya que, de hecho, es el mismo array). Ejemplo:

```
int a[]={3,3,3};  
int b[];  
b= a;  
b[0]=8;  
System.out.println(a[0]);//Escribirá el número 8
```

Finalmente, como detalle final a este tema, el operador de igualdad (==) se puede utilizar con arrays, pero nuevamente no compara el contenido sino si las referencias señalan al mismo array. Es decir:

```
int a[]={3,3,3};  
int b[]={3,3,3};  
System.out.println(a==b); //escribe false, aunque ambos arrays  
                           //tienen el mismo contenido  
int c[]=b;  
System.out.println(b==c); //escribe true
```

### (5.1.3) array multidimensionales

Los arrays además pueden tener varias dimensiones. Entonces se habla de arrays de arrays (arrays que contienen arrays). Un uso podría ser representar datos identificables por más de un índice. Por ejemplo:

```
int nota[][];
```

*nota* es un array que contiene arrays de enteros. La primera dimensión podría significar el número de un aula y el segundo el número del alumno. De modo que, por ejemplo, *nota[2][7]* significaría la nota del alumno número ocho del tercer aula.

Como ocurre con los arrays unidimensionales, hasta que no se define el array con el operador *new*, no se puede utilizar el array en el código.

```
notas = new int[3][12];    //notas está compuesto por 3 arrays
                           //de 12 enteros cada uno
notas[0][0]=9;             //el primer valor es un 9
```

Los arrays multidimensionales se pueden inicializar de forma más creativa incluso. Ejemplo:

```
int notas[][]=new int[5][]; //Hay 5 arrays de enteros (aún por definir)
notas[0]=new int[100];      //El primer array es de 100 enteros
notas[1]=new int[230];      //El segundo de 230
notas[2]=new int[400];      //...
notas[3]=new int[100];
notas[4]=new int[200];
```

Hay que tener en cuenta que en el ejemplo anterior, *notas[0]* es un array de 100 enteros. Mientras que *notas*, es un array de 5 arrays de enteros. Esta forma irregular de arrays da muchas posibilidades y permite optimizar el gasto de espacio de almacenamiento; la idea es recordar que los arrays multidimensionales en Java son manejados como arrays de arrays.

Se pueden utilizar más de dos dimensiones si es necesario. Por ejemplo:

```
int nota[][][]=new int[5][7][90];
```

Esta instrucción crea cinco arrays de siete arrays de noventa números enteros. Es un array de arrays de arrays (un tanto complicado de describir) o también un array tridimensional. Es raro necesitar más de tres dimensiones, pero sería posible hacerlo.

#### (5.1.4) longitud de un array

Los arrays poseen un método que permite determinar cuánto mide un array; es decir, de cuántos elementos consta. Se trata de **length**. Ejemplo:

```
int a[]=new int [17];  
int b[][]=new int [30][7];  
  
System.out.println(a.length);    //Escribe: 17  
System.out.println(b.length);    // Escribe: 30  
System.out.println(b[7].length); // Escribe: 7
```

Gracias a este método el bucle de recorrido de un array (para cualquier tipo de array) es (ejemplo para escribir cada elemento del array **x** por pantalla):

```
for (int i = 0; i < x.length; i++) {  
    System.out.print(x[i]+ " ");  
}  
System.out.println();
```

#### (5.1.5) la clase Array

En el paquete **java.util** se encuentra una **clase estática** llamada **Arrays**. Una clase estática permite ser utilizada como si fuera un objeto (como ocurre con **Math**), es decir que para utilizar sus métodos hay que utilizar simplemente esta sintaxis:

```
Arrays.método(argumentos);
```

##### **fill**

Permite rellenar todo un array **unidimensional** con un determinado valor. Sus argumentos son el array a rellenar y el valor deseado:

```
int a[]=new int[23];  
Arrays.fill(valores,-1); //Todo el array vale -1
```

También permite decidir desde qué índice hasta qué índice rellenamos:

```
Arrays.fill(valores,5,8,-1); //Del elemento 5 al 7 valdrán -1
```

### **equals**

Compara dos arrays y devuelve true si son iguales. Se consideran iguales si son del mismo tipo, tamaño y contienen los mismos valores. A diferencia del operador de igualdad (==), este operador sí compara el contenido. Ejemplo (comparación entre el operador de igualdad y el método **equals**):

```
int a[] = {2,3,4,5,6};
int b[] = {2,3,4,5,6};
int c[] = a;
System.out.println(a==b);           //false
System.out.println(Arrays.equals(a,b)); //true
System.out.println(a==c);           //true
System.out.println(Arrays.equals(a,c)); //true
```

### **sort**

Permite ordenar un array en orden ascendente. Se pueden ordenar todo el array o bien desde un elemento a otro:

```
int x[] = {4,5,2,3,7,8,2,3,9,5};
Arrays.sort(x,2,5);    //El array queda {4 5 2 3 7 8 2 3 9 5}
Arrays.sort(x);        //Estará completamente ordenado
```

### **binarySearch**

Permite buscar un elemento de forma ultrarrápida en un array ordenado (en un array desordenado sus resultados son impredecibles). Devuelve el índice en el que está colocado el elemento. Ejemplo:

```
int x[] = {1,2,3,4,5,6,7,8,9,10,11,12};
Arrays.sort(x);
System.out.println(Arrays.binarySearch(x,8));    //Escribe: 7
```

### **copyOf**

Disponible desde la versión 1.6 del SDK, obtiene una copia de un array. Recibe dos parámetros: el primero es el array a copiar y el segundo el tamaño que tendrá el array resultante. De modo que si el tamaño es menor que el del array original, sólo obtiene copia de los primeros elementos (tantos como indique el tamaño); si el tamaño es mayor que el original, devuelve un array en el que los elementos que superan al original se rellenan con ceros o con datos de tipo **null** (dependiendo del tipo de datos del array).

```
int a[] = {1,2,3,4,5,6,7,8,9};
int b[] = Arrays.copyOf(a, a.length); //b es {1,2,3,4,5,6,7,8,9}
int c[] = Arrays.copyOf(a, 12);       //c es {1,2,3,4,5,6,7,8,9,0,0,0}
int d[] = Arrays.copyOf(a, 3);        //d es {1,2,3}
```

### copyOfRange

Funciona como la anterior (también está disponible desde la versión 1.6), sólo que indica con dos números de qué elemento a qué elemento se hace la copia:

```
int a[] = {1,2,3,4,5,6,7,8,9};  
int b[]=Arrays.copyOfRange(a, 3,6); //b vale {4,5,6}
```

#### (5.1.6) el método System.arraycopy

La clase System también posee un método relacionado con los arrays, dicho método permite copiar un array en otro. Recibe cinco argumentos: el array que se copia, el índice desde que se empieza a copia en el origen, el array destino de la copia, el índice desde el que se copia en el destino, y el tamaño de la copia (número de elementos de la copia).

```
int uno[]={1,1,2};  
int dos[]={3,3,3,3,3,3,3,3};  
System.arraycopy(uno, 0, dos, 0, uno.length);  
for (int i=0;i<=8;i++){  
    System.out.print(dos[i]+" ");  
} //Sale 1 1 2 3 3 3 3 3 3
```

La ventaja sobre el método copyOf de la clase Arrays está en que este método funciona en cualquier versión de Java-

## (5.2) clase String

### (5.2.1) introducción. declarar e iniciar texto

Una de las carencias más grandes que teníamos hasta ahora al programar en Java, era la imposibilidad de almacenar textos en variables. El texto es uno de los tipos de datos más importantes y por ello Java le trata de manera especial.

Para Java las cadenas de texto son objetos especiales. Los textos deben manejarse creando objetos de tipo String (String se suele traducir como cadena; cadena de texto).

Declarar e iniciar un texto se suele hacer de esta forma:

```
String texto1 = "¡Prueba de texto!";
```

Las cadenas pueden ocupar varias líneas utilizando el operador de concatenación "+".

```
String texto2 = "Este es un texto que ocupa " +  
                "varias líneas, no obstante se puede " +  
                "perfectamente encadenar";
```



Otras formas de También se pueden crear objetos String sin utilizar constantes entrecomilladas, usando otros constructores.

Por ejemplo podemos utilizar un array de caracteres, como por ejemplo:

```
char[] palabra = {'P','a','l','a','b','r','a'}; //palabra es un array de caracteres  
  
//no es lo mismo que un String
```

a partir de ese array de caracteres podemos crear un String:

```
String cadena = new String(palabra); //mediante el operador new  
//podemos convertir el array de caracteres en  
//un String
```

De forma similar hay posibilidad de convertir un array de bytes en un String. En este caso se tomará el array de bytes como si contuviera códigos de caracteres, que serán traducidos por su carácter correspondiente al ser convertido a String. Hay que indicar la tabla de códigos que se utilizará (si no se indica se entiende que utilizamos el código ASCII:

```
byte[] datos = {97,98,99};  
String codificada = new String (datos, "8859_1");
```

En el último ejemplo la cadena *codificada* se crea desde un array de tipo byte que contiene números que serán interpretados como códigos Unicode. Al asignar, el valor *8859\_1* indica la tabla de códigos a utilizar.

### (5.2.2) comparación entre objetos String

Los objetos *String* (como ya ocurría con los arrays) no pueden compararse directamente con los operadores de comparación. En su lugar se deben utilizar estas expresiones:

- ♦ *cadena1.equals(cadena2)*. El resultado es *true* si la *cadena1* es igual a la *cadena2*. Ambas cadenas son variables de tipo *String*.
- ♦ *cadena1.equalsIgnoreCase(cadena2)*. Como la anterior, pero en este caso no se tienen en cuenta mayúsculas y minúsculas. Por cierto en cuestiones de mayúsculas y minúsculas, los hispanohablantes (y el resto de personas del planeta que utilice otro idioma distinto al inglés) podemos utilizar esta función sin temer que no sea capaz de manipular correctamente caracteres como la eñe, las tildes,... Funciona correctamente con cualquier símbolo alfabético de cualquier lengua presente en el código *Unicode*.
- ♦ *s1.compareTo(s2)*. Compara ambas cadenas, considerando el orden alfabético. Si la primera cadena es mayor en orden alfabético que la segunda devuelve *1*, si son iguales devuelve *0* y si es la segunda la

mayor devuelve **-1**. Hay que tener en cuenta que el orden no es el del alfabeto español, sino que usa la tabla ASCII, en esa tabla la letra **ñ** es mucho mayor que la **o**.

- ♦ `s1.compareToIgnoreCase(s2)`. Igual que la anterior, sólo que además ignora las mayúsculas (disponible desde Java 1.2)

### (5.2.3) **String.valueOf**

Este método pertenece no sólo a la clase String, sino a otras y siempre es un método que convierte valores de una clase a otra. En el caso de los objetos String, permite convertir valores que no son de cadena a forma de cadena. Ejemplos:

```
String numero = String.valueOf(1234);  
String fecha = String.valueOf(new Date());
```

No vale para cualquier tipo de datos, pero sí para casi todos los vistos hasta ahora. No valdría por ejemplo para los arrays.

### (5.2.4) **métodos de los objetos String**

Cada nuevo String que creamos posee, por el simple hecho de ser un String, una serie de métodos que podemos utilizar para facilitar nuestra manipulación de los textos. Para utilizarlos basta con poner el nombre del objeto (de la variable) String, un punto y seguido el nombre del método que deseamos utilizar junto con los parámetros que necesita. método y sus parámetros después del nombre de la variable String. Es decir:

```
variableString.método(argumentos)
```

#### **length**

Permite devolver la longitud de una cadena (el número de caracteres de la cadena):

```
String texto1="Prueba";  
System.out.println(texto1.length()); //Escribe 6
```

#### **concatenar cadenas**

Se puede hacer de dos formas, utilizando el método **concat** o con el operador suma (+). Desde luego es más sencillo y cómodo utilizar el operador suma. Ejemplo:

```
String s1="Buenos ", s2=" días", s3, s4;  
s3 = s1 + s2; //s3 vale "Buenos días"  
s4 = s1.concat(s2); //s4 vale "Buenos días"
```

## charAt

Devuelve un carácter de la cadena. El carácter a devolver se indica por su posición (el primer carácter es la posición 0) Si la posición es negativa o sobrepasa el tamaño de la cadena, ocurre un error de ejecución (se pararía el programa), una excepción tipo **IndexOutOfBoundsException**. Ejemplo:

```
String s1="Prueba";  
char c1=s1.charAt(2); //c1 valdrá 'u'
```

## substring

Da como resultado una porción del texto de la cadena. La porción se toma desde una posición inicial hasta una posición final (sin incluir esa posición final). Si las posiciones indicadas no son válidas ocurre una excepción de tipo **IndexOutOfBoundsException**. Se empieza a contar desde la posición 0. Ejemplo:

```
String s1="Buenos días";  
String s2=s1.substring(7,10); //s2 = "día"
```

## indexOf

Devuelve la primera posición en la que aparece un determinado texto en la cadena. En el caso de que la cadena buscada no se encuentre, devuelve -1. El texto a buscar puede ser **char** o **String**. Ejemplo:

```
String s1="Quería decirte que quiero que te vayas";  
System.out.println(s1.indexOf("que")); //Escribe: 15
```

Se puede buscar desde una determinada posición. En el ejemplo anterior:

```
System.out.println(s1.indexOf("que",16)); //Ahora escribe: 26
```

## lastIndexOf

Devuelve la última posición en la que aparece un determinado texto en la cadena. Es casi idéntica a la anterior, sólo que busca desde el final. Ejemplo:

```
String s1="Quería decirte que quiero que te vayas";  
System.out.println(s1.lastIndexOf("que")); //Escribe: 26
```

También permite comenzar a buscar desde una determinada posición.

## endsWith

Devuelve true si la cadena termina con un determinado texto. Ejemplo:

```
String s1="Quería decirte que quiero que te vayas";  
System.out.println(s1.endsWith("vayas")); //Escribe true
```

### startsWith

Devuelve **true** si la cadena empieza con un determinado texto.

### replace

Cambia todas las apariciones de un carácter por otro en el texto que se indique y lo almacena como resultado. El texto original no se cambia, por lo que hay que asignar el resultado de **replace** en otro String para no perder el texto cambiado:

```
String s1="Mariposa";  
System.out.println(s1.replace('a','e')); //Da Meripose  
System.out.println(s1); //Sigue valiendo Mariposa
```

### replaceAll

Modifica en un texto cada entrada de una cadena por otra y devuelve el resultado. El primer parámetro es el texto que se busca (que puede ser una expresión regular), el segundo parámetro es el texto con el que se reemplaza el buscado. La cadena original no se modifica.

```
String s1="Cazar armadillos";  
System.out.println(s1.replaceAll("ar","er")); //Escribe: Cazer ermedillos  
System.out.println(s1); //Sigue valiendo Cazar armadillos
```

### toUpperCase

Obtiene la versión en mayúsculas de la cadena. Es capaz de transformar todos los caracteres nacionales:

```
String s1 = "Batallón de cigüeñas";  
System.out.println(s1.toUpperCase()); //Escribe: BATALLÓN DE CIGÜEÑAS  
System.out.println(s1); //Escribe: Batallón de cigüeñas
```

### toLowerCase

Obtiene la versión en minúsculas de la cadena.

### toCharArray

Consigue un array de caracteres a partir de una cadena. De esa forma podemos utilizar las características de los arrays para manipular el texto, lo cual puede ser interesante para manipulaciones complicadas.

```
String s="texto de prueba";  
char c[]=s.toCharArray();
```

## matches

Es una función muy interesante disponible desde la versión 1.4. Examina la expresión regular que recibe como parámetro (en forma de String) y devuelve verdadero si el texto que examina cumple la expresión regular.

Una expresión regular es una expresión textual que utiliza símbolos especiales para hacer búsquedas avanzadas.

Las expresiones regulares pueden contener:

- ♦ **Caracteres.** Como *a, s, ñ*,... y les interpreta tal cual. Si una expresión regular contuviera sólo un carácter, **matches** devolvería verdadero si el texto contiene sólo ese carácter. Si se ponen varios, obliga a que el texto tenga exactamente esos caracteres.
- ♦ **Caracteres de control** (*\n, \\, ...*)
- ♦ **Opciones de caracteres.** Se ponen entre corchetes. Por ejemplo *[abc]* significa *a, b* ó *c*.
- ♦ **Negación de caracteres.** Funciona al revés impide que aparezcan los caracteres indicados. Se pone con corchetes dentro de los cuales se pone el carácter circunflejo (^). *[^abc]* significa ni *a* ni *b* ni *c*.
- ♦ **Rangos.** Se ponen con guiones. Por ejemplo *[a-z]* significa: cualquier carácter de la *a* la *z*.
- ♦ **Intersección.** Usa *&&*. Por ejemplo *[a-x&&r-z]* significa de la *r* a la *x* (intersección de ambas expresiones).
- ♦ **Substracción.** Ejemplo *[a-x&&[^cde]]* significa de la *a* la *x* excepto la *c, d* ó *e*.
- ♦ **Cualquier carácter. Se hace con el símbolo punto (.)**
- ♦ **Opcional.** El símbolo *?* sirve para indicar que la expresión que le antecede puede aparecer una o ninguna veces. Por ejemplo *a?* indica que puede aparecer la letra *a* o no.
- ♦ **Repetición.** Se usa con el asterisco (\*). Indica que la expresión puede repetirse varias veces o incluso no aparecer.
- ♦ **Repetición obligada.** Lo hace el signo *+*. La expresión se repite una o más veces (pero al menos una).
- ♦ **Repetición un número exacto de veces.** Un número entre llaves indica las veces que se repite la expresión. Por ejemplo *\d{7}* significa que el texto tiene que llevar siete números (siete cifras del 0 al 9). Con una coma significa *al menos*, es decir *\d{7,}* significa *al menos* siete veces (podría repetirse más veces). Si aparece un segundo número indica un máximo número de veces *\d{7,10}* significa de siete a diez veces.

Ejemplo, se trata de comprobar si el texto que se lee mediante la clase `JOptionPane` empieza con dos guiones, le siguen tres números y luego una o más letras mayúsculas. De no ser así, el programa vuelve a pedir escribir el texto:

```
import javax.swing.JOptionPane;

public class ExpresionRegular1 {

    public static void main(String[] args) {
        String respuesta;
        do {
            respuesta=JOptionPane.showInputDialog("Escribe un texto");
            if(respuesta.matches("--[0-9]{3}[A-Z]+")==false) {
                JOptionPane.showMessageDialog(null,
                    "La expresión no encaja con el patrón");
            }
        }while(respuesta.matches("--[0-9]{3}[A-Z]+")==false);
        JOptionPane.showMessageDialog(null,"Expresión correcta!"
    }
}
```

### lista completa de métodos

método	descripción
<code>char charAt(int index)</code>	Proporciona el carácter que está en la posición dada por el entero <i>index</i> .
<code>int compareTo(String s)</code>	Compara las dos cadenas. Devuelve un valor menor que cero si la cadena <i>s</i> es mayor que la original, devuelve <i>0</i> si son iguales y devuelve un valor mayor que cero si <i>s</i> es menor que la original.
<code>int compareToIgnoreCase(String s)</code>	Compara dos cadenas, pero no tiene en cuenta si el texto es mayúsculas o no.
<code>String concat(String s)</code>	Añade la cadena <i>s</i> a la cadena original.
<code>String copyValueOf(char[] data)</code>	Produce un objeto <b>String</b> que es igual al array de caracteres <i>data</i> .
<code>boolean endsWith(String s)</code>	Devuelve <b>true</b> si la cadena termina con el texto <i>s</i>
<code>boolean equals(String s)</code>	Compara ambas cadenas, devuelve <b>true</b> si son iguales
<code>boolean equalsIgnoreCase(String s)</code>	Compara ambas cadenas sin tener en cuenta las mayúsculas y las minúsculas.
<code>byte[] getBytes()</code>	Devuelve un array de bytes que contiene los códigos de cada carácter del String



método	descripción
<code>void <b>getBytes</b>(int srcBegin, int srcEnd, char[] dest, int dstBegin);</code>	Almacena el contenido de la cadena en el array de caracteres <i>dest</i> . Toma los caracteres desde la posición <i>srcBegin</i> hasta la posición <i>srcEnd</i> y les copia en el array desde la posición <i>dstBegin</i>
<code>int <b>indexOf</b>(String s)</code>	Devuelve la posición en la cadena del texto <i>s</i>
<code>int <b>indexOf</b>(String s, int primeraPos)</code>	Devuelve la posición en la cadena del texto <i>s</i> , empezando a buscar desde la posición <i>PrimeraPos</i>
<code>int <b>lastIndexOf</b>(String s)</code>	Devuelve la última posición en la cadena del texto <i>s</i>
<code>int <b>lastIndexOf</b>(String s, int primeraPos)</code>	Devuelve la última posición en la cadena del texto <i>s</i> , empezando a buscar desde la posición <i>PrimeraPos</i>
<code>int <b>length</b>()</code>	Devuelve la longitud de la cadena
<code>boolean <b>matches</b>(String expReg)</code>	Devuelve verdadero si el String cumple la expresión regular
<code>String <b>replace</b>(char carAnterior, char ncarNuevo)</code>	Devuelve una cadena idéntica al original pero que ha cambiando los caracteres iguales a <i>carAnterior</i> por <i>carNuevo</i>
<code>String <b>replaceFirst</b>(String str1, String str2)</code>	Cambia la primera aparición de la cadena <i>str1</i> por la cadena <i>str2</i>
<code>String <b>replaceFirst</b>(String str1, String str2)</code>	Cambia la primera aparición de la cadena uno por la cadena dos
<code>String <b>replaceAll</b>(String str1, String str2)</code>	Cambia la todas las apariciones de la cadena uno por la cadena dos
<code>String <b>startsWith</b>(String s)</code>	Devuelve <i>true</i> si la cadena comienza con el texto <i>s</i> .
<code>String <b>substring</b>(int primeraPos, int segundaPos)</code>	Devuelve el texto que va desde <i>primeraPos</i> a <i>segundaPos</i> .
<code>char[] <b>toCharArray</b>()</code>	Devuelve un array de caracteres a partir de la cadena dada
<code>String <b>toLowerCase</b>()</code>	Convierte la cadena a minúsculas
<code>String <b>toLowerCase</b>(Locale local)</code>	Lo mismo pero siguiendo las instrucciones del argumento <i>local</i>
<code>String <b>toUpperCase</b>()</code>	Convierte la cadena a mayúsculas
<code>String <b>toUpperCase</b>(Locale local)</code>	Lo mismo pero siguiendo las instrucciones del argumento <i>local</i>
<code>String <b>trim</b>()</code>	Elimina los blancos que tenga la cadena tanto por delante como por detrás
<code>static String <b>valueOf</b>(<i>tipo</i> elemento)</code>	Devuelve la cadena que representa el valor <i>elemento</i> . Si <i>elemento</i> es booleano, por ejemplo devolvería una cadena con el valor <i>true</i> o <i>false</i>

### (5.3) arrays de Strings

Los arrays se aplican a cualquier tipo de datos y eso incluye a los Strings. Es decir, se pueden crear arrays de textos. El funcionamiento es el mismo, sólo que en su manejo hay que tener en cuenta el carácter especial de los Strings.

Ejemplo:

```
String[] texto=new String[4];
texto[0]="Hola";
texto[1]=new String();
texto[2]="Adiós";
texto[3]=texto[0];
for (int i = 0; i < texto.length; i++) {
    System.out.println(texto[i]);
}
/*se escribirá:
    Hola

    Adiós
    Hola
*/
```

### (5.4) parámetros de la línea de comandos

Uno de los problemas de trabajar con entornos de desarrollo es que nos abstrae quizá en exceso de la realidad. En esa realidad un programa java se ejecuta gracias a la orden `java programa`; orden que hay que ejecutar en la línea de comandos del sistema.

En cualquier momento podemos hacerlo. Pero es más, podemos añadir parámetros al programa. Por ejemplo supongamos que hemos creado un programa capaz de escribir un texto un número determinado de veces. Pero en lugar de leer el texto y el número por teclado, queremos que lo pasen como parámetro desde el sistema. Eso significa añadir esa información en el sistema operativo, por ejemplo con:

```
java Veces Hola 10
```

Eso podría significar que invocamos al programa llamado `Veces` y le pasamos dos parámetros: el primero es el texto a escribir () y el segundo el número de veces.

El método `main` es el encargado de recoger los parámetros a través de la variable `args`. Esta variable es un array de Strings que recoge cada uno de los parámetros. De tal modo que `args[0]` es un String que recoge el primer parámetro, `args[1]` el segundo, etc. Si no hay parámetros, `args.length` devuelve cero.

Así el programa **Veces** comentado antes quedaría:

```
public class Veces {  
    public static void main(String[] args) {  
        if(args.length>=2) {  
            int tam=Integer.parseInt(args[1]);  
            for (int i = 1; i <=tam; i++) {  
                System.out.println(args[0]);  
            }  
        }  
        else {  
            System.out.println("Faltan parámetros");  
        }  
    }  
}
```

En casi todos los IDE se pueden configurar los parámetros sin necesidad de tener que ejecutar el programa desde la línea de comandos. Por ejemplo en Eclipse se realiza desde el menú **Ejecutar-Configuración de ejecución**:

